PhD Thesis

PhD in Arts and Computational Technology

Working Performatively with Interactive 3D Printing

An artistic practice utilising interactive programming for computational manufacturing and livecoding

Evan Saul Raskob

October 13, 2022

Goldsmiths, University of London

PhD Thesis

Copyright
©This book is copyright Evan S. Raskob.

License
Shared under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International (CC BY-NC-ND 4.0) License:
https://creativecommons.org/licenses/by-nc-nd/4.0/

First printed on October 13, 2022

A special thank you to my PhD advisors, Prof. Mick Grierson and Prof. Rebecca Fiebrink, for all their support, great discussions, and helpful feedback on this project over the years. Thanks for seeing the potential of a somewhat ridiculous idea! To Selena, for being my part-time editor and full-time support; to Oz, for trailblazing and helping me stay on the path; Ezra Teboul, for great chats about printers and musicality; to Nick, for being the occasional guinea pig and a good friend; Alex McClean and the TOPLAP community for all the organising that made most of these performances possible; the ICLC and broader livecoding community for all your support and interest and great performances; Simon, for all your listening and also distracting; my colleagues at Goldsmiths, and current and past, for all your support.

Contents

List of Figures

List of Tables

Abstract | 1

This thesis explores the liminal space where personal computational art and design practices and mass-manufacturing technologies intersect. It focuses on what it could look and feel like to be a computationally-augmented, creative practitioner working with 3D printing in a more programmatic, interactive way. The major research contribution is the introduction of a future-looking practice of *Interactive 3D Printing (I3DP)*. I3DP is articulated using the Cognitive Dimensions of Notations in terms of associated user activities and design trade-offs. Another contribution is the design, development, and analysis of a working I3DP system called LivePrinter. LivePrinter is evaluated through a series of qualitiative user studies and a personal computational art practice, including livecoding performances and 3D form-making.

Introduction 2

"... art is a province in which one finds all the problems of form (e.g. proportion and balance) but also spiritual problems (e.g. of philosophy, of religion, of sociology, of economy)"

— Josef Albers

2.1 Interactive 3D Printing

There is a liminal space at the threshold of robotic and human labour, where personal practice and mass-manufacturing technology intersect. Computer-numerically-controlled (CNC) machines that apply digital control to physical fabrication, such as 3D printers, are now ubiquitous in community maker spaces, educational settings, and some small offices and homes. These machines have captured people's imaginations with new possibilities of bespoke, small-scale manufacturing, or "personal fabrication" (Baudisch and Mueller, 2017). 3D printing is so integrated into some people's practices that it is part of their identities as creative professionals, product designers, computational designers, artist activists, lab technicians, and social-media "vloggers".

This thesis explores this new sense of identity by asking a question that is both pragmatic and speculative in nature: "What would it look and feel like to combine interactive programming and augmented manufacturing into an artistic practice?" In particular, it focuses on the experience of current and future artists and designers working in the relatively new territories of the computationally-augmented creative arts. What if they could work with 3D printing in a more programmatic and yet direct, interactive, and immediate way?

This main question provides a "lens" through which to view an abductive research process, heavily driven by intuition but guided by empirical experiments and reflective practice. Looking through that lens into the near future, we see the beginnings of programmatic dialogue with "intelligent" machines for personal fabrication, those descendants of modern-day digital plotters, laser cutters, and 3D printers.

Looking closer, we zoom in on an individual artistic practice embedded in a like-minded community, revealing a previously unseen ontology of performance and computational manufacturing. This new practice is built around this thesis's author, Evan Raskob, sometimes A.K.A. pixelpusher, BITPRINT, or BITLIP when making art or performing live music. Evan is at once a teacher of computational art and design; an audio-visual performer; digital artist working with image and sculpture; and also a researcher. His attempts to make use of new techniques of interactive 3D printing in his work are integral to this thesis.

Four questions were used to further guide the project's research activities towards more concrete outcomes:

1. **Can interactive programming for 3D printers help participants understand how the process of manufacturing using 3D printers relates to their discipline**, so they can start to experiment usefully with it (or not)?
2. **Can interactive programming for 3D printers allow users to create physical forms using novel functions that take into account physical properties like speed and temperature,** instead of the usual method of beginning with 3D modelling and automating fabrication?
3. **How can the combination of visual aesthetics and musical concepts lead to knowledge about new digital manufacturing toolpaths, and vice versa?** (e.g. exploring how 3D printing toolpaths can be influenced by concerns other than optimising for speed and strength)
4. **How can a livecoding environment be useful for 3D printing?**

Inherent in those questions is the possibility of a new system for 3D fabrication that transcends the current limitations and arbitrary constraints of current 3D printing software. Why not combine livecoding with 3D printing? What new and useful possibilities for creative fabrication arise when livecoding is combined with computational manufacturing?

2.2 Contributions

A number of concrete research contributions are built on top of this rather broad foundation. Firstly, we introduce a new practice of *Interactive 3D Printing (I3DP)*. Drawing on various first-hand experiences and studies of interactive programming and livecoding, I3DP is articulated as a few "typical" user activities for interactively programming 3D printers. This design blueprint is introduced, discussed and evaluated using a system derived from theories of cognition and computer interaction called the Cognitive Dimensions of Notations (the CDNs) (T. R. G. Green and Petre, 1996) in Chapter 5 (Designing for Interactive 3D Printing).

A second major contribution is the design, development, and analysis of a working I3DP system called *LivePrinter.* The details of LivePrinter's development process contain new techniques for designing and developing interactive programming environments designed to work specifically with 3D printers. The specific implementation details of the LivePrinter system, such as the software development process and accompanying hardware integration, and their relationship to individual CDNs, is discussed in Chapter 6 (Implementing an I3DP system: LivePrinter).

LivePrinter is then evaluated through a series of qualitative user studies. Building on the theoretical cognitive trade-offs discussed in Chapter 5 (Designing for Interactive 3D Printing), we held individual interviews with prospective users to determine some conceptual boundaries for a real I3DP system, and later ran a series of user workshops to evaluate the experience of learning and using it. These user interviews and user studies are discussed in detail in Chapter 7 (User Studies and Analysis).

These two contributions help establish a new field of computationally-augmented fabrication, but despite the time and effort put into them, they are just a beginning. The second part of the initial research question, of what people might actually do with I3DP, forms the rest of the thesis. This is explored through a personal computational art practice, including livecoding performances and 3D form-making.

Towards that end, we introduce new techniques for interactively fabricating a variety of shapes and patterns, including the use of space-filling curves such as the Hilbert Curve (Hilbert, 1891). These techniques may have some use as precursors to new metamaterials with interesting mechanical properties. They may also lead to more sustainable 3D prints that use less material and energy in their construction. Some artefacts resulting from these experiments found their way into public exhibitions, such as at the London Design Festival in 2019.

We also demonstrate how such curves can be useful in musical livecoding performances through reflections on live performances where 3D printers were controlled by interactive programming. These performances, and artefacts derived from the computational sculpting experiment, are informed by the practice of generative art, which uses computational processes to compose works of art in part or even whole, and livecoding, which is related in its use of programmatic constructs to create experiences for live audiences, usually in the form of ephemeral software programs that exist mainly in the for the duration of performances. The artistic and design outcomes can be found in Chapter 8 (Filling space, filling time), and videos of some early live performances where a 3D printer is interactively controlled by code in LivePrinter to make sound and sculpture can be seen in Section A.1 (Selected Performances).

Taken together, these outcomes paint a picture of a new form of practice based around writing computer code to creatively control computation fabrication machines. They show how we can use practitioner-led design research to begin influencing the direction of innovation in personal manufacturing technology.

The rest of the research process, along with its theoretical underpinnings are discussed in more detail in the next chapter, Chapter 4 (Methods).

2.3 Key influences and inspirations

This thesis was particularly inspired by the "Six Challenges for Personal Fabrication" described by Baudisch and Mueller (2017), especially the two challenges of improving the interactivity and user experience of personal fabrication, and increasing practitioners' material- and machine-specific knowledge. In terms of interactivity, the project was influenced by the technique of *interactive programming*, where programmers continually add code a running computer program to see its effects in real-time. This technique has been adopted by a community of self-proclaimed *livecoders*, who have elevated the public display of programming to a form of performance art.

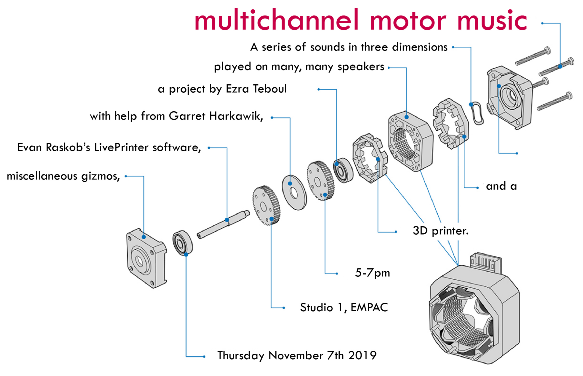

multichannel motor music
A series of sounds in three dimensions
played on many, many speakers
a project by Ezra Teboul
with help from Garret Harkawik,
Evan Raskob's LivePrinter software,
miscellaneous gizmos,
and a
3D printer.
5-7pm
Studio 1, EMPAC
Thursday November 7th 2019

Figure 2.1: Poster for Ezra Teboul's "Multichannel Motor Music" performance using 3D printers. Used with permission of the author (Ezra Teboul).

This study, as well as the livecoding community, have been deeply influenced by contemporary artists and artistic movements that have their more recent roots in *Minimalism* and *Conceptualism*. The influence of these movements on this study can be seen in the practice of I3DP, which shifts the focus from the physicality of hands-on making to a descriptive and textual forms of making that use code and digital fabrication.

This echoes the evolution of Conceptualism, which followed on from Minimalism. Minimalism's original emphasis on material structure inspired Conceptualist artists to rebel against to and focus instead on text, words and linguistic structure. As Sol LeWitt, a Minimalist artist who later became an influential Conceptualist, famously pronounced in his 1967 *Paragraphs on Conceptual Art*, "the idea becomes the machine that makes the art" (Lewitt, 1967).

2.4 The influence of Generative and Audiovisual Art

The *Generative art* movement built further on these fertile ideas by focusing on the aesthetic arising from the use of repetitive, algorithmic techniques for constructing images and forms. The ideals of the movement, and techniques associated with practitioners of it such as *cellular automata* and space-filling curves, has been an inspiration for many of the sculptural forms created during this study.

The term *generative art* roughly dates back to the mid-20th-century, around when the term *generative aesthetic* was first used by German philosopher of science Max Bense in 1954 to describe (then) new experiments in computer-rendered art (Dohm and Hoffmann, 2008, p. 31). This practice often placed minimalist forms inside a flurry of repetitive structures, and was enabled by machines and computer rendering, as can be seen in the pioneering works of Georg Nees and Vera Molnar and more recent practitioners like Tyler Hobbes and Jared Tarbell.

Generative artists often borrow technical approaches from (or inspired by) physics and mathematics in the creation of many of their works, as seen in works such as Herbert W. Franke and Andres Hübner's *Pendulum Oscillogram*. This places them within a long tradition of using mathematics and science as a muse for art-making (Bak, 2019). It also makes generative art an interesting technique for helping people to visualise and understand scientific and mathematical ideas by abstracting them and thus focusing on them removed from their messy, real-world context.

Importantly, the movement began as a rebuke to photojournalism. It challenged the idea that the purpose of photography was to be somehow more honest and truthful than other forms of art. The trust that photojournalists had in their machines, i.e. their cameras, to faithfully capture their realities was directly challenged by the non-representational photographs of early practitioners of the *Generative Aesthetic* like Gottfried Jäger, Herbert W. Franke, Andres Hübner, and Hein Gravenhorst. Their use of mechanical algorithms and chemical processes (later run on early computers and plotters) were intended to produce their alternate abstract realities, or "aesthetic states."

From its pre-digital beginnings, Generative Art has continued to co-evolve along with computational tools. As a practice, it is closely tied to tools for computational fabrication, as they are programmable and mechanically-inexhaustible platforms for repetitive form-making. Many artists co-evolve their own tools, like digital plotters and even robot arms with bespoke software for human-like graffiti drawing (D. Berio, Calinon, and Fol Leymarie, 2016; Daniel Berio, 2020).

1: `https://github.com/tidalcycles/sounds-repetition`

Many works of both Generative Art and Livecoding can also be considered to belong to the genre of *Audiovisual Art*, which is more explicitly focused on the audience or participant's *experience* of the work than the implicit "aesthetic states" represented in generative imagery or animations. This related artistic tradition comes from a more mixed background of fine artists and audiovisual performers, including John Whitney Sr., Vibeke Sorenson, and Ryoji Ikeda. There is certainly a porous border between these genres, but it is helpful to review Grierson (2018, p. 15)'s observations on Audiovisual Art as similarly incorporating technological innovation and employing abstraction as a method for focusing on non-narrative experience. According to Grierson (2018, p. 15), Audiovisual artists:

- ► Focus on "audiovisual experience over the musical or visual, for the purposes of specifically and directly modulating attention through multisensory stimulation"
- ► Use "abstraction to focus the work on these experiential qualities, as opposed to other qualities such as story, characters, context"
- ► Develop and apply "new technological approaches to more effectively explore these principles in practice"

2.5 The influence of machine art

Another important point of reference for this project was the long tradition of "machine-as-art" and machinic art. Both of these genres focus on the machines *making* the art as a work in their own right, instead of simply as a means to an end. This is especially important when looking at 3D printers are both platforms for manufacturing but also performative objects in their own right that are interesting to look at, listen to, and to play with.

In some artists' views, the "haphazard" mark of the artist could be replaced by the "rational" one of the machine, turning art into a process rather than an individual striving. In the words of Marcel Duchamp (1955), "The mechanical aspect of it influenced me then, or at least that was also the point of departure of a new technique. I couldn't go into the haphazard drawing or the paintings, the splashing of the paint. I wanted to go back to a completely *dry* drawing, a *dry* conception of art. . . And the mechanical drawing was for me the best form for that dry conception of art." (Sweeny, 1973, p. 127)

Artists Jean Tinguely and Moholy-Nagy also both used machines as works of art, in addition to using them for making art, exhibited in their own aesthetic right. They also often used these works as critiques of the perceived power of the artistic individual and the unrepeatable expression of their individuality. Some of their contemporaries thought that the then-dominant idea of the "heroic artist" struggling to create a masterwork was elitist, and mass-manufacturing and machines could be a means for ushering in a new era of democracy by enabling the public to participate in that mysterious and mystical process of making artistic objects (Dohm and Hoffmann, 2008, p. 18).

For example, we can look at Jean Tinguely's *Méta-Matic* drawing machines, especially *Méta-Matic No. 14*. This work was operated by a hand-crank attacked to a series of gears, springs and a pencil that scratched lines across a blank page. The randomness of the drawing mechanism made achieving precise strokes and forms difficult, not impossible. It also removed the pressures of creative failure, because, once set in motion, the machine was always playfully creating *something*.

Removing the precision of the art-making process made this device more accessible to the general public as a means for making art. In a sense, this can be viewed as an ironic take on the view of art as the "heroic struggle" of the artistic individual. It also highlights the *process* of making the art, rather than the uniqueness and value of the artefacts produced (Dohm and Hoffmann, 2008).

The artist's role in producing these works of machinic art are similar to a computer programmer's: they create the framework (the machine; the program) and establish a context for the art (the gallery; the Internet) and then leave it to run and produce some kind of "artistic" outputs. In the process, they also implicate the "audience" in the creation of the work by either having them directly take part, as with Tinguely, or to simply be present during the act of creation.

For example, since the 1990s Andrea Bulloch has worked on a series of machines that use motors, controlled or effected by sensors, to draw lines on white gallery walls in front of the viewing public. Similarly to Tinguely and Moholy-Nagy and their ironic take on the singularity of authorship, she describes the way it works as questioning the artist as the creator of something original (Dohm and Hoffmann, 2008; Bulloch, 1998). Yet she also engages the viewer in watching the work unfold, deciding how, when and how long this viewing takes place: ". . .I've set the whole thing up. I've decided how it works and looks and I leave only a small part of the structure open-ended. . . I make the process of viewing visible."(Bulloch, 1998)

This line of thought has much in common with this project's goals of demystifying the process of 3D printing by making the means of artistic production visible but also creatively constrained. It also challenges the idea of repetition as being in opposition to individual creative expression and devoid of drama, a contrary view surely shared by Generative and Audiovisual artists. The idea that machines are dry, rational, and dehumanising in relation to the dramatic, emotional and individual humans that they were set to replace must contend with Tinguely's works, especially the very public and dangerously self-destructing *Homage to New York* (1960), which had plenty of dramatic presence.

2.6 Repetition, repeatability, error and creativity

What is truly interesting about machine art, and both Generative and Audiovisual art, is that they force us to directly confront the difference between "repetition" and "repeatability." That is one of the concepts at the heart of this project. Just because a machine repeats an action doesn't necessarily mean it will produce the same result; often, errors and defects creep into the manufacturing process, and these can be leveraged in creative ways.

Some artists explicitly embrace "errors" through techniques of *non-determinism*, partially as a means to critique the illusion of complete control that we assume digital machines have given us over the messiness of the fabrication process. Roxy Paine's *Scumak 2*, for example, was an industrial-looking 3D printing machine, situated in an art gallery, that slowly built turd-like piles of plastic over time (Goodman, 2014). The mechanism for this work was a simple spout, continually releasing flows of downwards-dripping molten material onto a moving conveyor belt. The "sculptural" results are very hard to predict, owing to the complexity of fluid dynamics; they vary greatly in diameter, height, and texture. It shares many of the same mechanisms of 3D printers, but lacks their precision and controllability.

Innovators in clay and architecture such as the team of Ronald Rael, Virginia San Fratello, Kent Wilson, Alex Schofield also embraced "controlled error" by modifying GCode, the set of instructions that most 3D printers use to create objects. Their forms are usually generated initially as nearly-Platonic shapes, and then their fabrication code is manipulated to create uneven and often accidental patterns in their ceramic vessels (Rael et al., 2016). Whilst leveraging "error" was a creative way of investigating the limitations of current ceramic fabrication technologies to create complex textures, it was also a way of highlighting those same limitations and how they affect artistic process in unexpected but ultimately recognisable ways.

2.7 Manufacturing as performance art

As Tinguely surely would have appreciated, a digital/mechanical system like I3DP can take on a dramatic, performative nature, and just watching a 3D printer in action on its own can be a fascinating aesthetic experience.

In performances with the printer, we were surprised how captivated audiences were by its simple, repetitive and often music movements. The aesthetic possibilities of performing with machines inspired us to think of future manufacturing as a more shared, even public practice.

There also exist works art and manufacturing integrated together that are more explicitly performative, and might act as a guide for future works of *performative manufacturing*. Similarly, *performative food preparation* can be found in open-plan kitchens in restaurants, especially ones where the Japanese art of Yakiniku is practised. In Yakiniku (or in the West, simply "Japanese Barbecue"), ingredients are cooked as a form of fiery theatre in front of diners. In certain high-end shops, artisan chocolate- and bread-making takes place in glass-enclosed food preparation spaces adjacent to cafés and retail spaces.

Especially with food production, the art of making dishes is often a public demonstration of skill exposing the craft of the artisan chefs. It could be seen as an act of storytelling, linking the journey from raw material (ingredients) to manufacturing methods (chocolate preparation, sushi rolling, barbecue); making edibles in a narrative-driven demonstration of skill that becomes part of the customer's experience of the manufactured object.

In industrial product manufacturing, where assembly lines are often hidden inside vast factories, performance aesthetics are less of a consideration. Performance in this technical sense refers to equipment effectiveness and reliability in producing something. The worker is mainly there to make objects to the greatest extent of their ability, consistently, not to perform for others' enjoyment. Yet, the assembly line is often referred to as "a ballet" or "dance" and its actions are "choreographed" in the sense that they are designed to be closely performed followed by human workers.

Some artists have experimented with the concept of assembly line as performance, as with **75 Watts**[*]. This performance design piece by Reyvital Cohen and Tuur Van Balen with Alexander Whitley re-appropriated a production line factory in the epicentre of *Made in China* as a choreographed performance making an object that had no use other than to be "manufacturable." Few, if any, other artists have gone so far as to make a functional industrial manufacturing method into a creative experience, either for the workers and craftspeople themselves, or for an outside audience.

2.8 Augmentation as taking back control

So much of people's lives are already dictated by algorithms – Facebook feeds, Amazon purchases, AI-assisted cars, track-and-trace apps on our devices monitoring deadly contagions, decisions about whom gets to speak during video conference calls. People have been trusting in AI and automation to manage the messiness of human society more than they trust other people to do it. Some people trust AI so much more that we now have companies made up of people *pretending* to be computer

[*] https://www.cohenvanbalen.com/work/75-watt

programs, a term coined as *fauxtomation* by Astra Taylor in her article "The Automation Charade" (Taylor, 2018).

These general misunderstandings, that automation is either more efficient, safer, or somehow better than human labour, present us with a false choice. We can either choose to continue losing human jobs to automation, or discard digital technology altogether and work Luddite-like in manual, craft-centred, time- and labour-intensive disciplines. We may want to reject machine labour, because it is arguable that full automation has its dark sides, such as high energy costs, and a tendency to reduce employment (Acemoglu and Restrepo, 2018) that puts more power in the hands of already wealthy factory owners and threatens workers' control over the means of production. Yet, that doesn't tell the whole story about *partial* automation, or *augmentation*.

Acemoglu and Restrepo (2018) demonstrated that the process of **augmentation**, where technological advances are integrated into human work to make more complex tasks, has had the opposite effect on employment as full automation. Computational augmentation can create new roles, new ways of working. Instead of StyleGAN replacing artists, we might soon see new disciplines of art that work specifically with them to create new forms of images. As the techniques of "realistic" photography were repurposed towards creating abstract Generative Art, so too can robots and computational manufacturing devices be repurposed towards unforeseen artistic ends.

With digital manufacturing, it becomes more clear what (Acemoglu and Restrepo, 2018) meant when he proposed that the process of augmentation was really a driver for **increasing** the complexity of human labour. For example, the development of 3D printing created new roles for people using new computer software to design 3D objects, replacing the traditional draughts-person who used to sketch plans for objects on paper. It also created new technical roles for 3D printer tooling, operating, repairing, and for object post-processing that had no equivalent before.

2.9 Towards a future of human/computational ecosystems

One way of resolving the conflict between humans and automation is to more deeply embed humans in the process of making. This has been the approach of this thesis, to avoid designing away the complexity of human interaction in the name of false efficiencies of time. It is hoped that projects like this will help to influence developers of future human/machine manufacturing systems purporting to be part of a holistic "Integrated Computational Materials Engineering (ICME) paradigm" bringing together process, structure, material properties, and structural performance into a single design process (Jared et al., 2017).

The same conclusion about the usefulness of augmentation over automation was reached by the pioneering computational artist Ernest Edmonds. In his process of creating complex visual forms, the computer was less useful as automation device working on its own, and more useful as an assistant that he could work alongside. Part of this new process of working alongside and communicating with digital "assistants" was to create new "specifications" for works of art in computer code: "Though the computer can replace man in the production of graphic images, its function in the arts is seen as assisting in the specification of art systems and in their subsequent real-time management" (Edmonds, 2018, p. 5).

In more general language, these "specifications" can be though of as user interfaces, or user-facing forms of *notation*. These new forms of notation both describe and specify the process of creating new works, acting as a common language shared between people and the machines that assist them. By looking at them as notation, we can use a framework like the Cognitive Dimensions of Notations to discuss their designs and create evaluative tools that we can test with potential users.

Currently, the state of human/machine manufacturing systems sits somewhere near the junction of mass-manufacturing, artisanal craft, Do It Yourself (DIY) design, and personal fabrication. Machines are not yet intelligent enough to be creative partners, and may never be, but they still present opportunities for collaboration and creative and technical assistance that never existed before.

Before we embark on our journey towards a future of holistic, practitioner-led computational fabrication, we must first reflect on the paths that have led up to our current position. These interweaving paths map out the complex interplay of industry with creative practices. Taken together, they provide some possible routes for practitioner-led projects to incorporate 3D printing processes into the aesthetics in their work without compromising their core social and artistic values.

Literature review of interative programming and computational manufacturing | 3

3.1 Introduction

Computerised manufacturing and interactive forms of computer programming have both existed since at least the 1960s. Semi-automated manufacturing platforms run on software and can often be maintained interactively, but few, if any, systems for live, improvisational control of these systems exist. In industry, it looks unlikely that the situation will change. Recent studies show that the current trend is towards fully-automated computational manufacturing that replaces people rather than computationally augmenting human workers and craftspeople (Acemoglu and Restrepo, 2018; Abraham and Kearney, 2018). The driving forces behind this trend are complex and varied; it isn't just a matter of safety constraints or the limits of technological complexity but a blend of social, governmental and historical forces such as national politics, educational systems, historical precedents and even tax laws favouring full automation (Acemoglu, Manera, and Restrepo, 2020), as discussed in the Introduction.

An exploration of how this separation of computational design and computational fabrication evolved into its current state would be a large task. Instead, we will focus on some key areas of interest relative to a holistic practice of augmented computational manufacturing, such as Interactive 3D Printing (I3DP). In such a system, practitioners would use interactive programming techniques and livecoding practices manipulating computer-numerical-controlled (CNC) machines like 3D printers "on the fly." In theory, this would allow them to experiment with complex forms more quickly than with more traditional, manual ways of making and open up new practices of artisan computer-augmented manufacturing that include livecoding performances.

Such a system could adapt techniques and aesthetics from the artistic practices of Generative Art and Audiovisual Art to the interactive fabrication of 3D objects. It is not inconceivable to imagine this practice as a blend of the current practice of *livecoding* and *3D printing*, both occupying interdisciplinary spaces marked by porous conceptual borders between related areas of practice and research.

As we will discuss, livecoding is a practice that mainly uses the established technique of *interactive programming* towards various performative ends, sharing conceptual DNA with other disciplines that allow programmers to change running programs without stopping and restarting them. The evolution of interactive programming and its relationship to these other disciplines will be explored, along with design strategies for creating such systems.

The term "3D printing" itself refers to a wide range of industrial processes from the expensive laser fusion processes that produce precision aerospace parts to the cheaper but more error-prone, melted-plastic-layering technique used in rapid desktop prototyping. We will attempt to describe the relevant types of 3D printing techniques, and the workings of 3D printers associated with them. Then, we will look at the current challenges facing 3D printing and how programmatic augmentation of that process might offer up some new insights into potential solutions for them.

3.2 Interactive programming and real-time feedback

The use of programming languages for experimenting with making multimedia art, such as sound and visuals, is commonly identified as beginning soon after the transistor-based computer was first developed, in the 1950s. From the beginning, rapid experimentation and timely results were important to computer musicians and artists, who often needed to create a series of variations on a theme and learn from their attempts. Max Mathews, one of the first pioneers of music computing, developed musical programming systems starting in the 1950s at Bell Labs. Some of his work was fully textual and based on the syntax of the C language, such as Csound (Mathews, 1963; Aaron and Alan Blackwell, 2013), but, starting around 1968, he was integrating textual programming and physical controls with analogue sound generators (Mathews and Moore, 1970) in order to get real-time feedback.

It was also during this early era that one of the most basic techniques for interactive programming, the Read-Evaluate-Print-Loop (REPL), was first developed. The traditional REPL supported an interactive workflow where lines of code are entered, compiled, and then executed as soon as possible, with results made visible (e.g. "printed") on the screen. This empowered a programmer to edit, extend, or otherwise change a running program and experience the result almost immediately afterwards. It has been a key part of interactive environments for variants of the LISP programming language since their inception in the 1960s, with the *Emacs* editor being a prime example (Tanimoto, 2013; McCarthy, 1981).

Technically speaking, the opposite of interactive programming can be thought of as the so-called "edit, compile, link, run" (Tanimoto, 2013) workflow. In this mode of development, a programmer writes code that is parsed by a traditional compiler or interpreter into machine-specific instructions. The previous code is thrown out before the start of this process, leaving no record of historical changes. Each development cycle, the resulting program is run anew as if it were the first time.

3.3 Interactive programming versus livecoding

If the term *interactive programming* can be thought of as referring to a specific set of techniques for modifying running programs, the term *livecoding* refers to something altogether less technical and more conceptual in nature. As Roberts and Wakefield (2018) point out, there are a number of definitions, of which they add that livecoding is simply "when performers create time-based works by programming them while these same works are being executed." In our example, we expand this definition beyond the intentions of the performer to include the audience, and describe the general process of a livecoding performance as:

> a process of *algorithmic communication* between two or more entities that is potentially perceivable to others, usually between at least one performer and one or more *livecoding software systems* running on a computer and broadcasted or made available in some way to an audience.

In this description, the term *algorithmic communication* usually refers to the sharing of computer code, but can also incorporate any symbolic communication that is intended to be used as notation for a computational process. The term *livecoding software system* is also broadly defined here as any system that is intended to be used to receive *algorithmic communication* and interpret that communication as a form of output that audiences can experience.

The process of livecoding thus depends on the communication between performer(s) and audience, making it fundamentally a subjective experience. This distinction of what constitutes livecoding is mainly based on the personal intention of all the participants, and also on the experience of the audience of this performance. Livecoding, like pornography, appears to follow more of an "I know it when I see it" (Wikipedia, 2021) definition that makes it hard to precisely pin down in concrete terms. It is this tension between the precise technical language we usually use to refer to computing and programming, versus the looseness of the more social and conceptual term "livecoding" that is what makes it so interesting.

3.4 The practice of livecoding

The exhibition changing grammars at the University of Fine Arts of Hamburg (HfbK), in 2004, has been described by Julien Rohrhuber as a formative moment for today's understanding of how the term "live" modifies the traditional meaning of "coding" (McLean and Dean, 2018)(Alan Blackwell, McLean, et al., 2014, p. 147). To Rohrhuber, "liveness" in coding meant:

1. programming as public thought
2. programming as a rewriting of running programs (Alan Blackwell, McLean, et al., 2014, p. 147)

In the beginning, this mainly manifested as "laptop composers. . .building their own custom software, tweaking or writing the programs themselves as they perform" (N. Collins et al., 2003, p. 321). This was often seen as a political act of extreme transparency in software development, a response to the trend of hiding code and complexity from the end user. Often this involved projecting the textual programs onto the performers, or in the performance space – thus the motto of some livecoders became "show us your screens!"

LiveCoding is, of course, not the only form of creative expression to incorporate audience-facing or publicly available text. Both Sol LeWitt and Jenny Holtzer used instructions and text extensively in their Conceptualist periods of working. This was a shift from the art movement Minimalism's emphasis on bare physicality to a focus on language and more cognitive structures and effects (Hughes, 2006, p. 426).

This practice also references more ancient practices of "text-as-art", seen in traditional Islamic works and many contemporary works of illustrated poetry, as well as the practice of *concrete poetry*. These practices expose the underlying conceptual structure of a piece, such as LeWitt's technique of elevating the importance of the instructions for making a piece of work to the level of the work itself as in his *Wall Drawings* (Kent, 2007; Lewitt, 1967). Similarly, the motto of "process not product" has been uttered by many modern artists since LeWitt's predecessor Marcel Duchamp arrived on the scene in the early 20th century to question the primacy of a finished work of art over the process that spawned it.

In this sense, livecoding follows a tradition of taking an inward-facing, intellectual act that is usually hidden from public view (in this case programming, versus Duchamp's art-making) and turning it outwards as a form of public performance (Alan Blackwell, McLean, et al., 2014). It differs from Duchamp's questioning of how the heroic role of the artist is celebrated in the production of art, in that it acts to re-empower the anonymous programmer, whose role is usually obscured from view and whose name is often unknown in the commercial software production process. In livecoding performances, the programmer is foregrounded instead of their software; their work placed at centre stage instead of hiding them and their code away to focus on a seamless, "transparent" aesthetic experience. According to Wieser (2018):

> In IT, the word transparency is used in an unexpected way: it here describes the relation between 'user-friendly' graphical user interface and the hidden program code. The program is transparent if it is like a clear window, which is not to be noticed, a notion which is as common as it is misleading. This strange disappearance of the work of the programmers might be an important reason for the widespread anxieties with regard to programming. Of course, many people share strong disinclinations against such strategies of disempowerment.

Livecoding is an interesting practice that is both intellectual and meta-cognitive in nature but also fully *bodied*. It involves thinking of programmatic, algorithmic systems of heuristics (e.g. computer processes) that then control physical systems (e.g. audio generating hardware, projection devices, computerised 3D printing machinery, directing human performers). Livecoding often involves an immediate physicality and personal presence that borrows from theatre and other forms of live arts, whether that is directly addressed or not by the performer. There is even a livecoding language for live choreography of dancers (Sicchio, 2014; Sicchio, 2016; Sicchio, 2019).

3.5 Livecoding and cybernetics

It is interesting to compare livecoding's innate performative aspect with the cybernetic vision, Cybernetics, is sometimes referred to as a science of control and "steering" (Wiener, 1961; Rose, 1974; Ashby, 1956). The cybernetic method of inquiry is "evolutionary, rather than causal or calculative" (Andrew Pickering, 2011). It is made up of processes of action and performance that evolve over time (Andrew Pickering, 2011, p. 6).

Knowledge, in a cybernetic system, "is a part of performance rather than an external controller of it". It refuses the conceptual split between "people" and "things", instead focusing on the interaction between "black-boxes", those ultimately unknowable entities enmeshed in complex systems that can only really be understood through experiencing them (Andrew Pickering, 2011; Ashby, 1956). Thus, Andrew Pickering (2011) states that "the stance of cybernetics was a concern with performance as performance, not as a pale shadow of representation". A cybernetic system contains an internal ontology that evolves over time out of the relationship between its entities as they perform with one another in a "dance of agency" (Andrew Pickering, 2012; A. Pickering, 1995).

This interpretation of cybernetics as a system of non-deterministic patterns evolving over time has been influential in some areas of art, science, and music for a few decades now. There is a distinct lineage of cybernetics-inspired works, including architect Christopher Alexander's Notes on the Synthesis of Form (Alexander, 1966) and Brian Eno's evolving musical ecosystems starting in the mid-1970s. In the world of computational art and design, software-based L-Systems and cellular automata borrowed ideas of simple rules playing out over time to create unpredictable, iterative patterns. For example, Meinhardt (2009) showed how to algorithmically generate seashells by modelling reaction-diffusion equations in iterative passes. In the livecoding community, Leitão et al. (Leitão and Martins, 2010) proposed a version of the LISP-like language Scheme for livecoding generative architectural models, inspired by the livecoding environment Fluxus (Griffiths, 2019), and a livecoding performance group at McMaster University has signalled its debt to cybernetics in its name, *the Cybernetic Orchestra* (Ogborn, 2012).

According to Sorenson et al. (A. Sorensen, Swift, and Riddell, 2014), meaning in livecoding has similarly cybernetic attributes: it is symbolic, computational, but also simultaneously embodied and thus must be physically experienced. They describe three basic principles inherent to livecoding, which build on Rohrhuber's notions of "public" and the concept of modifying a "running" program:

- **program-process semantics**: the formal system that defines the livecoding language
- **the process-task semantics**: the computational tasks performed by the computer as a result of running code
- **"cyber-physicality"**: the physical "perturbations" in the world caused by running the code, e.g. sound or visuals (A. Sorensen, Swift, and Riddell, 2014)

That is, the difference between livecoding and non-performative coding is in this interpretation of tokens (e.g. computer code) by both a computer and also by a human audience, with the added layering of the physical experience of that code when it is executed: "'meaning' can happen through a semantic interpretation of tokens by a human interpreter or through the mechanical transduction of formal tokens into the physical environment" (A. Sorensen, Swift, and Riddell, 2014, p. 68). Code is something to be interpreted by the audience, but also to be literally felt through its audio/visual/mechanical effects in the physical world.

In A. Sorensen, Swift, and Riddell (2014, p. 67)'s example, executing the line of code "**drums –> | s k |**" in the livecoding system IXI Lang (Thor Magnusson, 2011) might trigger a snare drum sound followed by a silence and then a kick-drum sound followed by a silence. The meaning of those tokens can potentially reveal itself as the code is executed and the sound is heard by an audience. Of course, any internal translation of tokens to significance and meaning by the audience members depends on a basic understanding of text and exposure to certain textual conventions that are very much dependent on context (Meyer, 1961, p. 60).

3.6 Comparing practices of liveness in programming

There are plenty of reasons why people manipulate code live that have little to do with performance, sound production or audiovisual art. For example, live programming can help us understand the effects of code during the software development process. As Tanimoto observed, we might simply benefit from "minimizing the latency between a programming action and seeing its effect on program execution" (Tanimoto, 2013, p. 31). For example, artist Vera Molnar describer her "conversational method" in a 1975 essay in Leonardo titled "Towards Aesthetic Guidelines for Paintings with the Aid of a Computer" as an iterative process of tweaking code and viewing the results on her computer monitor. This form of interactive software development "dialogue" was essential to her graphical form-finding process (Molnar, 1975).

According to Rein et al. (2019, p. 1), different motivations, programming contexts and communities have spawned three main families of related approaches to liveness in software development: livecoding, live programming, and exploratory programming. These approaches are commonly described as "programming environments and tools that can provide the impression of changing a program while it is running".

Compared to livecoding, which Rein et al. (2019, p. 2) describes somewhat reductively in terms of its visible nature as "often concerned with the creation of art through changing source code as a performance in front of an audience", live programming and exploratory programming can be defined as follows:

> Live programming. . .put[s] the very activity of programming in its focus (Hancock, 2002; Tanimoto, 2013). Correspondingly, the term seems to be used when describing programming tools which provide immediate feedback on the dynamic behavior of a program even while programming. The term exploratory programming often refers to a particular workflow during programming whenever requirements are not fully defined but are yet to be discovered (Sheil, 1983; Trenouth, 1991). It is supported by exploratory programming environments that incorporate changing a running system to make exploration of unknown domains or of design alternatives easier. (Rein et al., 2019, p. 2)

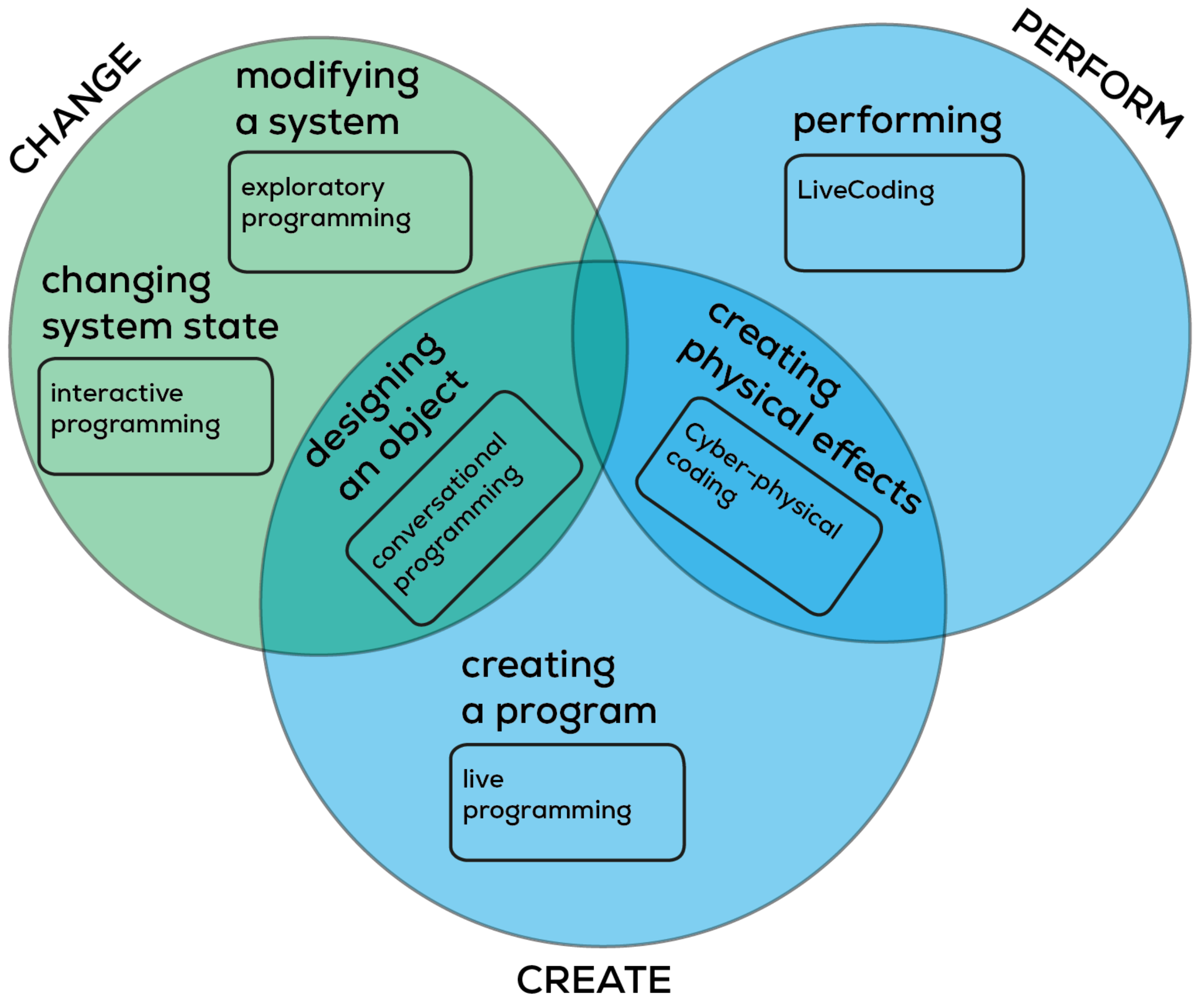


Figure 3.1: Relationships between common terms for modes of liveness in coding after Rein et al. (2019), A. C. Sorensen (2018), and Tanimoto (2013)

Livecoding, live programming and exploratory programming papers were categorised by their "intended outcomes" in Rein et al. (2019)'s review. Interestingly, they found that live programming practitioners more often had the intention of creating a computer program; exploratory programming practitioners more often had the intention of modifying or evolving a running system (such as a database or simulation); and livecoding practitioners more often had the intention of "creating an effect" where the program itself is "only secondary and might be discarded after the computation" (Rein et al., 2019, p. 19).

Yet, the "program" in livecoding is more than an abstract concept or disposable piece of software. A program, in livecoding, unfolds over the entire time of a performance and forms an essential part of that performance's experience. It might be better stated that livecoders who are in the act of livecoding are often more interested in the experience of their systems and programs than either writing programs for future use or maintaining or evolving an ongoing system.

A. Sorensen and Gardner (2010) goes even further to define another type of live programming of which livecoding is a part, called "cyber-physical programming." In cyber-physical programming, the state of the physical environment also becomes a meaningful component of the program's state as it is edited live (A. C. Sorensen, 2018).

In Alan Blackwell and Aaron (2015), livecoding and live programming cross over in the development of Sonic Pi. Aaron's practicing livecoding often leads to him developing software artefacts in an iterative cycle of performance–reflection development. Livecoding in Sonic Pi was used as means of "sketching" and musical practice, leading Sam to reflect on the performative limitations of the Sonic Pi system, leading him to further development of Sonic Pi:

> Often during a practice session [with Sonic Pi] Sam will be in a musical position and have an idea of where he'd like to go, but run into syntactic or semantic barriers in the language that prevent him getting there easily. Or, he'll find himself writing boilerplate code again and again and therefore start imagining a nicer way of automating the repeated task. Ideas that would spring into his head he'd just jot down.
>
> Occasionally, when there is no obvious solution to a problem, he might allow himself to interrupt a practice session (if the opportunity has sufficient merit) and then spend a bit of time imagining he has already implemented the solution, so that he can see if he could sketch the solution to the idea in Sonic Pi. This sketching activity does not seem like development work, but has far more the character of reflective craft – design ideation that must be captured in the performance moment (Alan Blackwell and Aaron, 2015, p. 8).

Surprisingly, Rein et al. (2019) found no overlap between livecoding and exploratory programming. This is particularly interesting because Sorensen (after Perera (2013)) describes live programming similarly to Rein et al. (2019)'s exploratory programming, i.e. as a form of "computational unfolding" achieved through the live editing of source code (i.e. modifying or evolving a running system) that has *real-world effects* (A. C. Sorensen, 2018, p. 24). Sorensen, as both a livecoder and live programmer, is less concerned with creating a "software artefact" than with producing "some physical effect, and through that effect some greater comprehension of the physical world" (A. C. Sorensen, 2018, p. 25). The focus is on the experience and process, rather than outputting a finished program-as-artefect.

The term *interactive programming* is used similarly to exploratory programming to refer a dialogue between a programmer and the changing state of a system. An example is the Unix shell, where a user interrogates the operating system of a running computer program in order to understand and change it using "shell" commands. "Shells" are live interpreters, often taking a line of dialogue from a user, performing an operation and responding back in typical REPL fashion.

On Unix and related operating systems, shells such as *zsh, bash, sh, tcsh* have their own syntax and control structures and are thus considered a form of programming. They are also found in many software programs, such as all major web browsers (*Firefox, Edge, Chrome*) where each has an interactively programmed *Console* for displaying web page status and also manipulating and interrogating web elements and processes.

Clearly, the distinctiveness of the definitions of categories of liveness in software development is relative to the communities that practice them. Professional software developers work to develop finished software artefacts. Engineers and designers use software to produce physical artefacts and are often concerned with finding out where a structure might collapse or otherwise fail. Livecoders perform live at concerts in front of audiences, some called "Algoraves" as a somewhat earnest take on algorithmic music and post-1990's rave culture. They may share labels for their live practice, but do not necessarily share the same context or goals for their practice.

3.7 Designing livecoding systems

It is important to reinforce that, despite the definition from N. Collins et al. (2003) that links livecoding to "laptop composers," the current practice of livecoding is not constrained to the genre of "laptop music." More generally, livecoding systems can be thought of as forms of performative notation supporting interactive, public workflows that "[expose] the functional abstractions. . .that form the underlying basis of aesthetic . . .experiences" and allow them to be modified in a live setting (Aaron and Alan Blackwell, 2013, p. 35).

One helpful tool for designing the notion of such systems if T. R. G. Green and Petre (1996)'s Cognitive Dimensions of Notations Framework (CDNs). The CDNs are meant as an evaluation tool for interactive systems and their notation, and as a guide to designers of such systems. For example, *Hidden dependencies* occur when parts of a system have a relationship between them that isn't visible to the user of, or participant in, that system (T. R. G. Green and Petre, 1996; Alan Blackwell and Thomas Green, 2003; A. F. Blackwell et al., 2001). This hints at the difficulty of designing a livecoding system where symbols are stand-ins for actual computing processes. A livecoding system designer must consider both the physical and cognitive ergonomics of the graphical (or physical) interface and of the livecoding language itself, or, as the CDNs might describe it, the *notation*.

The difficulties of designing cognitive trade-offs between elements of live notation exposes livecoding system design as a kind of *wicked problem* (Rittel and Webber, 1973), where one of the main design complications of trying to solve the problem of providing immediate control over a time-sensitive physical performance that is also balanced with the slower, meta-cognitive process of symbolic composition. Inevitably, there will be parts of the system that are hidden, abbreviated or otherwise abstracted away to increase the reaction time of the performer, but at the expense of legibility and very likely capability. It is part software engineering, part user experience design, and part experiment in audiovisual notation. Ultimately, a successful system must be both *usable* as a live programming system but also *useful* for creating interesting-enough live performances that people will want to continue using it.

Designers of livecoding systems have a difficult task ahead of them. They need to learn specialist domain knowledge so that they can effectively generate *real-time* audio, video or drive other physical outputs. They must be aware of the visual aesthetics of the system and the syntax of the chosen livecoding language, which are directly exposed to the audience and form part of the performance. Their new livecoding system might usefully interact with various other pieces of software, from time-synchronisation systems like Ableton Sync to full livecoding ecosystems that combine software synthesizers, sequencers, effects, and even other livecoding systems using network protocols like OSC or websockets.

A further complication is that livecoding is often collaborative in nature, and performances will likely encompass other computers, devices, and groups of performers. In many performances audio and visual performers communicate across networks of laptops, such as Algobabes, SLUB, PowerBooks Unplugged, and the Sheffield Livecoding Ensemble, to name a few. These remote systems and persons must somehow be represented in the interface and possibly the language itself.

Designers have the opportunity to make their system accessible to a range of other potential users. Care must be taken to make the installation process easy enough for a diverse group of people to attempt. Basic usage guides and tutorials must be written for others to learn the system. Since this is social software, there needs to be channels of communication where users can discuss problems, ask and answer questions and become part of this new system's community.

It is hard to define what a "successful" livecoding system might look like. Some might consider it to be the one adopted by numerous performers, others might be satisfied with a system used only once for a particularly memorable performance and then never seen again. It is no wonder that many of them are relatively unique in implementation: according to Rittel and Webber (1973), "Every solution to a wicked problem is a 'one shot operation."'

As with any wicked problem, a solution generally arrives before the problem is fully formulated in an inductive process of experimentation. That is, the "problem" or the gap in the livecoding ecosystem that the livecoding practitioner was intuitively aiming to address by creating their system might not be clearly understood until a working version of their system demonstrates a "solution" in the form of a performance.

Framing these systems as wicked problems helps explain the uniqueness of most livecoding systems. They vary widely in structure, livecoding language, and general philosophy. No one design pattern describes how they should be put together. Some livecoding systems use a variety of non-textual symbols and tokens to represent computational actions, such as Dave Griffiths' music system for PS4 called Al Jazari (blocks), and Kate Sicchio's system for live choreography called Terpsicode (pictures and annotations).

Another system, Dave Griffith's augmented reality/physical livecoding system called "Pattern Matrix"[1] used augmented reality and carved wooden blocks that represent computational operations. There even have existed livecoding systems that didn't use digital computers like Evan Raskob's "Human Patching" (2007)[2] and many experimental pieces

1: `https://penelope.hypotheses.org/598`

2: Human Patching (2007) was a livecoding system for music performance using a human-based network of program flow. It assigned logical and computational operations to people (e.g. count, increment, match a number) using handwritten signs hung around their necks and by physically connecting each person to one another using bungee cords. The design was loosely based on the visual patching environments MaxMSP and Puredata.

by Nick Collins[3] where people themselves (i.e."wetware") took on the computational tasks normally reserved for software.

3: https://composerprogrammer.com/music/livecodingforensemble.pdf

3.7.1 The Cognitive Dimensions of Notations

The Cognitive Dimensions of Notations (CDNs) (T. R. G. Green and Petre, 1996; Alan Blackwell and Thomas Green, 2003; A. Blackwell, 2005) were developed as a method, grounded in cognitive psychology, to help designers and HCI practitioners articulate and discuss the cognitive trade-offs inherent in the designs of notational systems. Such systems typically rely on some sort of textual or pictorial visual notation as a user interface, as with text- and flow-chat-based programming languages, IDEs, and other interactive development environments. The framework is not limited to text, icons, or screens – the dimensions are more generally concerned with questions of what happens when information is represented, whether interactive, static, or even physical forms.

The ultimate goal of the CDNs is not to be an exhaustive checklist of potential usability issues but to help designers who are creating activity-specific interactive systems to understand some basic dimensions of usability, and to balance these dimensional effects against one another whilst supporting the user's main activities.

If this all sounds a bit vague, that is by design. The CDNs were designed to be general enough to describe issues applicable to a broad range of notational systems, but with enough detail to make those descriptions useful to designers. In the words of T. R. G. Green and Petre (1996, p. 132), "by introducing a defined vocabulary for such ideas [that have previously gone unnamed in HCI discourse], the framework. . . makes it easier to converse about cognitive artefacts without having to explain all the concepts," or in A. Blackwell (2005)'s later revision of the CDNs they are a *set of discussion tools* that "improve the quality of the discussion". In both views, they provide a nearly-complete set of terms for discussing known usability issues and trade-offs in notational systems for designers to refer to across the entirety of the design process, from initial concept to development to testing and final delivery.

The framework has often been applied to visual programming languages (T. R. G. Green and Petre, 1996) but can be applied to word processors, information tables, and even physical control panels (T. R. G. Green and Petre, 1996; A. Blackwell, 2005). It has also been widely used in a number of different research contexts: commercial products from Microsoft (A. Blackwell, 2005), musical composition software (Bellingham, Holland, and Mulholland, 2014) and music notation (A. Blackwell, 2005), UML (Britton et al., 2002), programming library design (S. Clarke and Becker, 2003), machine learning systems (Bernardo et al., 2020), and previously to discuss livecoding activities (Alan Blackwell, 2015), amongst others.

There are 14 current CDNs which have been revised and added to since their introduction by Green (T. R. G. Green and Petre, 1996; A. F. Blackwell et al., 2001; Alan Blackwell and Thomas Green, 2003). Each individual CDN relates to some others in ways that makes a user's experience "easier", "harder", or maybe not at all, in plain language. They are designed so that dimensions can be paired up and varied in their relation to one another (inverse, direct, none) whilst looking at the effects on others.

A. Blackwell (2005) provides a helpful diagram of some possible relationships between some individual CDNs.

Importantly, they must be considered within the context of their notation-editing and -interacting environment, and seen as part of specific activities that users will be tasked with when working in that environment. Changing activities will likely change the balance of the desired traits, such as designing for highly-structured user activities like spreadsheet editing which rely on data entry, data modification and copying versus programming environments that support more free-form exploration that requires less structural constraints.

The CDNs can be particularly helpful when designing and implementing a new notational system by giving the designer(s) a checklist of cognitive variables that are relevant to interaction design. It also provides some discussion of the trade-offs between these variables, since they are usually interdependent on one another and can have complex relationships. The CDNs have also been used at the end of design processes to evaluate the finished design, sometimes in the form of user-facing questionnaires as in Clarke (2010) and Bernardo et al. (2020, pp. 545–565).

Table 3.1: The Cognitive Dimensions of Notations

Dimension	Description
Abstraction Types	The spectrum of possible combinations and groupings for elements in a system (T. R. G. Green and Petre, 1996, p. 144) that makes up its notation, plus the mechanisms provided for changing that notation (Alan Blackwell and Thomas Green, 2003, p. 9).
Closeness of Mapping	The conceptual distance between a system's notation in the "program world" and its equivalent in the "problem world" (Mayer, 1987; Pennington, 1987)(T. R. G. Green and Petre, 1996, p. 144). With textual programming languages, this becomes a measure of differences between programming language semantics, user's mental models, and the properties exposed by the "real world" or "target system".
Consistency	T. R. G. Green and Petre (1996, pp. 147–148) equate this to *guessability*, or what a user can infer about the notation once they know some of the functional notation
Diffuseness / Terseness	Verbosity / succinctness of language
Error-Proneness	When syntax leads to unwanted consequences
Hard Mental Operations	High demands on cognitive resources
Hidden Dependencies	Relationships between elements that are obscured or invisible to the user. A classic example: spreadsheets containing cells that perform calculations based on the contents of other cells, a process which isn't exposed to the user without further actions on their part (A. Blackwell, 2005).
Premature Commitment	Constraints on the order of doing things that often force the user to stop and think ahead of their actions
Progressive Evaluation	Work-to-date can be checked at any time
Provisionality	Degree of commitment to actions or marks
Role Expressiveness	The purpose of a component is readily inferred
Secondary Notation	Extra information in means other than formal syntax, such as comments in code or user sketches on a paper note pad
Viscosity	How much a system's notation permits or resists the user's actions to change the system. Sometimes these actions slow down change by making users repeat them, called *repetition viscosity*, other times they may force the user into a chain of actions of the same type, with a *knock-on viscosity*.
Visibility, juxtaposibility	The degree of visibility of information about elements in the notational system, presented to and accessible by the user.

3.7.2 Abduction as theory generator

Introducing new ideas into the scientific and academic body of knowledge often requires an *abductive* approach, attributed originally to Charles Peirce and further refined over the years by others (Dunne and Dougherty, 2016, p.170). Pierce called abduction "the process of forming an explanatory hypothesis. It is the only logical operation which introduces any new ideas; for induction does nothing but determine a value, and deduction merely involves the necessary consequences of a pure hypothesis" (Peirce, 1934, p. 171). It is a synthetic research process of where a researcher's perceptions are iteratively analysed by generating hypotheses about related phenomena, hidden phenomena, or even new categories of phenomena (Dunne and Dougherty, 2016; Fann, 1970).

Somewhat problematic was Pierce's innate "tendency towards a positive truth" (Peirce, 1934, p. 591). Rather than present abduction as an instinct, it is more useful to consider the role of the researcher in the world and how their "socially cultivated and cultivatable ways of seeing become preconditions for abductive reasoning" Dunne and Dougherty (2016). Bringing the rest of the world into the frame of the research process is unfortunately inevitable, in the sense that the researcher is a product of a specific time and place with a particular view on events. This brings up questions of scientific authority and representation (Atkinson 1990; Marcus and Fischer 1986) that must be interrogated inside the research process.

Taking an abductive approach, many research projects rely on a mix of qualitative collection methods and iterative theorising that Vaugn and Timmerman call *alternative casing*. *Alternative casing* is a recursive process of trying to fit the data to different theoretical frameworks, looking to see if it has been fully accounted for or if there are misguided preconceptions, different or incompatible circumstances Dunne and Dougherty (2016).

3.7.3 Livecoding systems design as research

The design process for a livecoding system can also be framed as a form of artistic or designerly research. This traditional "craft practice" is documented in Alan Blackwell and Aaron (2015, p. 1) in the development of both Blackwell's language *Palimpsest* and Aaron's *Sonic Pi*. Neither developer followed a conventional software engineering process nor adhered to conventional HCI techniques. In Blackwell's view their processes much more resembled an extended form of practice-led research that he likened to Fallman (2003)'s "pragmatic account" of design as "a hermeneutic process of interpretation and creation of meaning" (Fallman, 2003).

In this mode of knowledge creation, practice-led craft research does not aim to exhibit scientific falsifiability nor replicability. As Gaver (2012, p. 940) argued, theory underspecifies design, making theories about designs unfalsifiable. Instead, the role of the designer-academic is to bring their critical and analytical tools to bear on the design process itself, or as Alan Blackwell, McLean, et al. (2014) puts it:

> The key consideration is to maintain a critical technical practice in which technical activity is embedded within a recognized and acknowledged tradition, and subjected to rigorous reflection (Alan Blackwell and Aaron, 2015, p. 1).

This "technical activity" must produce some kind of artefacts to be reflected upon. Consider Philip Agre's concept of a *critical technical practice* that arose from his research into AI (A. Blackwell, 2018; Agre, 1997). Research questions, like those around AI, are often both philosophical and practical but "true" AI research involves both or else it is more *science fiction* than science, in A. Blackwell (2018)'s view.

The artefacts produced by this technical activity are meant to reveal a "theory nexus": the overlapping area between the choices made by designers, the issues that they think are important, and their beliefs about the right way to address those issues (Gaver, 2012; R. Collins, 1994, p. 944). Gaver (2012, p. 944) proposes that, instead of formalised theories, we can map out the boundaries of these theory spaces through a collection of "annotated portfolios" of designed artefacts that embody both "pure" theories and the "messy" practical decisions arising from functional constraints that designers need to contend with in order to make something *real*. In this space, "artefacts do not address these issues analytically, but represent the designer's best judgement about how to address the particular configuration of issues in question" (Gaver, 2012, p. 944).

According to Gaver, the criteria for annotating these projects could range from "the philosophical (what values should designs serve?) to the functional (how should those values be achieved in interaction) to the social (what will the people who use this be like?) to the aesthetic (what form and appearance is appropriate for the context?)" (Gaver, 2012, p. 944).

3.7.4 Design patterns for livecoding software systems

Livecoding systems are often combinations of two approaches to developing software applications: using pre-compiled libraries and frameworks in an "intra-process" manner to provide domain-specific functionality such as network protocols (OpenSoundControl or OSC)(Wright, 2005) or audio out (Jack), or to incorporate other running applications that provide that functionality and communicate with them in an "inter-process" manner. Intra-process applications share computing resources across all their running processes whereas inter-process applications run with separate resources that are inaccessible to each other.

Both A. C. Sorensen (2018, p. 48) with *Extempore* and McCartney (2002) with SuperCollider started with approaches that tightly coupled their livecoding languages (*XTLang* and *SCLang*, respectively) to intra-process, high-performance, domain-specific code libraries. Both systems eventually evolved into more decoupled designs.

A common decoupled design pattern for a livecoding software system is to use a collection of specialised programs communicating in an inter-process manner over a computer network[4]. A livecoder writes code in one program which then communicated with one or more using set

4: A "network" can encompass one or more computers, meaning that a computer can talk to itself over a network made of just itself.

Application Programming Interfaces (APIs). Sorensen (A. C. Sorensen, 2018) calls this "half-stack" livecoding, because typically the livecoder is limited to editing only part of the computing environment at any time. The livecoder can effect their environment up until the networked communication is initiated, but no further. Through the API, they can interact with *but not change* lower-level operations of the OS. In contrast, a *full-stack* approach would allow them to construct their entire real-time environment live during a performance.

The network of inter-process applications can be heterogeneous in its composition of those applications. Some livecoding environments, like ORCA, are for audiovisual performance but don't themselves generate sound. They are designed as exclusively a language interpreter that sends another form of standardised data, in this case MIDI, to either a hardware synthesizer or to other programs which generate the audio (Linvega, 2019). A livecoding system can be a fractal-like construct where one livecoding system communicates with another embedded livecoding system which itself is a collection of subsystems, as with Sonic Pi and TidalCycles. Both use SuperCollider internally, which itself is binary networked system consisting of a language interpreter program loosely coupled with a separate audio-generating server program. Other setups blend multiple livecoding systems together, using TidalCycles to generate audio from SuperCollider and also to generate visuals from Processing[5].

5: An early example is Alex McLean and Rodrigo Velasco's 2015 composition s2hs2, sharing OSC data between Tidal and Processing at https://toplap.org/s2hs2/

These systems are often networked to other performers running the same or different systems. Since these computer programs are processes that communicate over networks, they don't necessarily need to run on the same computer or even in the same physical location. Some performances take place across continents and time zones with different local and remote audiences, as with the annual TOPLAP anniversary streams mentioned previously (McLean, 2019).

There is a major benefit to an inter-process architecture, specifically in the case of live performance. For reasons of reliability, a livecoding system with more independent parts is less likely to fully fail at once, helping a performer keep at least part of the performance going through inevitable technical failures. Unstable, experimental parts can crash and then potentially be safely re-booted without destroying the entire performance.

Additionally, the design principle of "separation of concerns" can help make development quicker and easier because off-the-shelf software projects can be combined using common messaging protocols like OSC and MIDI. This is especially useful as domain-specific, highly technical systems like real-time audio generation can require specialist technical knowledge, making them difficult and time-consuming to create for most programmers.

One downside is that a separate protocol or API needs to be developed and implemented for each inter-process communication, which may be more work than simple incorporating a domain-specific library or framework. Another downside is the complexity of the software ecosystem itself, where it may be hard to for a performer to know what exactly is doing what, if anything, and when, in a live setting.

There are other potential pitfalls as well. As the component software systems evolve, they may change API or protocol and break the livecoding system. Also, timing may be an issue as inter-process communication is inherently slower than intra-process communication because it relies on extra layers of message passing. For many performances, this won't be an issue, however, as the limitation of the user's typing speed is more likely to be the slowest part of the system.

3.8 3D printing: From industry niche to mass consumer product

Believing the machine to be our modern medium of design, we sought to come to terms with it (Gropius, 1965).

Table 3.2: Common manufacturing processes

Technique	Description	Ex. Processes
Additive	Deposit or solidify material in layers	light (SLA), lasers (SLS), Inkjet 3D printing, powder bed fusion
Subtractive	Cut, carve or scrape to remove material	CNC router, plasma cutter, laser cutter, hand tools
Weaving	Combine threads of material in interleaving patterns to create relatively flexible forms	Jacquard loom, knitting, sewing machine
Welding	Strips or pieces of material are combined by applying intense heat (or electric current) at mechanical joins	arc welding, mig welding
Thermoforming	Heat is used to shape a material into a desired form, usually with a mould	vacuum forming, thermoplastic casting
Moulding/Casting	Material is shaped by pouring or injecting it into a desired form	blow moulding, injection moulding, lost-wax casting

Computer controlled fabrication has been in use for decades. More specifically, Computer Numerical Control (CNC) machines started being put into use for 'rapid prototyping' in the 1960s. 3D printing (3DP) technologies joined that field in the 1980s and 1990s with the invention and then patenting of key technologies for rapidly combining plastics and metals. Companies owning those patents, such as StrataSys, dominated the field until 2009 when the original patents began expiring.

The technology behind the expired patents was simple enough to work with that the DIY, hacker spaces, and maker community began to tinker with it on their own terms. Their relatively inexpensive, often open source 3DP designs spread widely in the form of major commercial enterprises like MakerBot (later sold to StrataSys) Prusa, and Ultimaker, to name a few (Baudisch and Mueller, 2017; Hoskins, 2014).

It is important to note this "desktop" 3D printing is quite a different industry than 3D printing for aerospace and other precision forms of manufacturing. In industry, 3D printing often results in parts that will be additional processed (further machined, finished up by skilled technicians) before they are shipped to consumers. Outside industry, its general unreliability, high cost per unit in terms of time and material, and inability to achieve required precision and surface finishes in finished models make it unattractive for manufacturing at scale on its own (Gao et al., 2015).

This is evident in many of the mostly "decorative" uses of non-industry 3D printing concentrating on *appearance* and *feel* (Baudisch and Mueller, 2017, p. 200). Architects use it to make one-off models for client presentations. Hobbyists, crafters or artists might use it for one-off uses like shelving, doorstops, sculptures and bespoke toys. The majority of downloadable models for 3D printing on the popular website Thingiverse[6] in 2017 could be classified as decorative, lacking any sort of *functional* behaviour (Baudisch and Mueller, 2017, p. 203).

6: http://thingiverse.com

With respect to professional product design, 3D printing is also rarely used to create finished products because if its cost. It instead has a niche in the rapid prototyping and early-stage testing of designs as with "product artist" Lionel Dean: ". . .it's got to be right the first time and I've got to be able to sell it. So I need to cheaply visualise it first." In his case, "cheap" was using a $15,000 Z Corp printer (Hoskins, 2014, p. 110).

Some product and industrial designers are satisfied with FDM printing and have found a place for it within their practice, like Tim Rundle and James Lamb, two designers with their own studio practices and clients who also teach part-time on MA Design Products at the Royal College of Art. They found the most value in FDM rapid prototyping was in testing out designs in an intermediate stage: after creating prototypes by hand using blue foam or other quick materials but before the costly investment in one or more higher-quality SLS (selective laser sintering or powder bed fusion) prototypes. Those SLS prototypes would normally lead directly to the tooling and setup costs of mass manufacturing a final product.

The process of 3D printing can also be used for professional research. Product designers are used to designing parts and products around practical manufacturing constraints, so the similar constraints of designing for FDM can help them learn more about the manufacturability of their designs. James found it helpful that this process sometimes forced him into creating more detailed and complex designs for the sake of better fabrication, such as splitting a lever in half and printing each side separately and then combining them. This helped him double check that his initial CAD design was valid[7].

7: Based on conversations, some recorded, both formal and informal, with Tim Rundle and James Lamb and other working product designers and students from the author's time at the Royal College of Art on the MA Design Products programme from 2016-2018.

3.8.1 An Example of Diffuse Innovation

The emergent ecosystem of open source 3D printing designs is particularly interesting in that it is an example of a *bottom-up* or *diffuse* form of innovation. Standardised parts, the expiry of key patents, open source software and hardware designs, and a global supply chain made it possible for tinkerers across the world to develop their own 3D printers. At this point, there are hundreds of bespoke, unbranded parts one can buy off eBay to either make their own printer or to modify their current one.

Similarly, one can choose to build their DIY printer around a number of mature, open source firmware projects like Marlin and bespoke controller boards like RAMPS that form the computerised control centre for each machine. The firmware is a computer program loaded onto a digital controller board that controls the operation of the printer. Firmware makes it straightforward to build a printer: simply assemble the right parts (motors, controller board, chassis, heaters and hot ends), load a firmware by connecting a standard computer and some other open source software, customise a few options, and the printer is ready to be used or modified and extended. Modified printers can produce moulds for lost wax casting, fabricate sugary treats, form ceramic structures ready for firing, and even print conductive parts and integrate electronics into designs.

This market is an example of the innovative potential of amateurs, semi-professionals and others working outside of industry. In Eric Von Hipple's words, it is the "free innovation" of the masses that has captured the global imagination: a decentralised, international ecosystem centred around small-batch and experimental manufacturing. (Hippel, 2016) As seen in MakerBot's acquisition by StrataSys and Ultimaker and others' continued support of open source projects like Marlin and printing software Cura, these innovative non-professionals have a symbiotic relationship with industry.

The recent experiments with customised, DIY (Do It Yourself) 3D printing and CNC workflows by digital artists and the ceramics and pottery community illustrates both the future potential of bottom-up innovation, but also the ways in which computational manufacturing is fundamentally altering the practice of making art and crafts. As Wenger argued in "Communities of Practice", "the structure of practice is emergent, both highly perturbable and highly resilient, always reconstituting itself in the face of new events. . .In a world that is not predicable, improvisation and innovation are more than desirable, they are essential." (Wenger, 1999)

In ceramics, one example of this individual-led innovation is Jonathan Keep, a potter and sculptor. Keep built his own clay 3D printer and used it to evolve a hybrid printing process that combined this automated system with a manual assembly process informed by his deep knowledge of his craft (Keep, 2020; Keep, 2014). He calls this process of "computerised coil building" (Keep, 2014, p. 32) a new, fourth way of making ceramics beyond traditional methods of hand-building, throwing, and moulding.

With all this *diffuse innovation* going on across many different disciplines, the future potential of 3D printing as a method of artistic and designerly working is unclear. Many people see it as a technology indistinguishable from magic, fulfilling the RepRap Project's dream of a machine capable of producing practically anything of unlimited complexity, including a copy of itself. (Various, 2019a) Baudisch and Mueller (2017) see it as the start of a new revolution towards *personal fabrication*, where 3DP usage scales from hundreds of thousands of users to hundreds of millions of users. Giving consumers this ability and precise control over physical matter itself would have the effect of "democratizing a whole range of fields preoccupied with physical objects, from product design to interior design, to carpentry, and to some areas of mechanical and structural engineering."

3.8.2 Desktop FDM 3D printing

The form of 3D printing that we are concerned with belongs to a family of manufacturing processes called additive manufacturing (AM). A list of common manufacturing processes can be seen in Table 3.2. (Thompson, 2007) The particular AM method used in desktop printers is referred to in industry and the literature as Fused Deposition Modelling (FDM) or sometimes Fused Filament Deposition (FFD) (Livesu et al., 2017; Hoskins, 2014; Gao et al., 2015), but here it will be referred to as FDM. FDM is popular with DIY, home and for education because of its flexibility, affordability, relative safety, and relative simplicity compared to other computerised techniques. (Gao et al., 2015)

The FDM process has been described many times in the literature, especially in CIRP publications as noted by Bourelli et al. (Bourell et al., 2017). FDM works by melting a material (e.g. a thermoplastic, liquid chocolate) or working with an already liquid material (e.g. clay or biogel) and depositing it continuously along vector paths. The material is first forced through an extruder component that has an extrusion nozzle of a particular size (0.2 mm for the Ultimaker 2 series). It is important that the material retains a high-viscosity during this entire process so that the extruded forms retain their shape when deposited. It is also important that the flow of material through the extruder remains constant so that the deposited paths are consistent in shape and don't have gaps or uneven sections. (Bourell et al., 2017; Sung-Hoon et al., 2002; Livesu et al., 2017)

A *print head* typically contains the extruder, heater, temperature sensors and cooling fans. The print head is attached to either a 2-axis mechanism moving in perpendicular horizontal directions (commonly referred to as the *x-axis* moving perpendicular to the printer front and the *y-axis* moving towards and away from the printer front) or in a *delta* configuration with 3 motors arranged in a triangular formation controlling the printhead which is suspended on 3 strings below. With some variations on the gantry system, the printer bed may move towards or away from the user along the y-axis with the print head moving both up and down along the z-axis and side-to-side along the x-axis. In all of these configurations, the print head will deposit layers on a flat surface that moves up and down (e.g. the *z axis*) called the printer *bed* or *build plate*. The bed is often heated so the material sticks to it during the *build process*.

With thermoplastics, each layer of deposited material is bonded to previously deposited layers when it makes contact. This creates a 'staircase effect' (Livesu et al., 2017) where the height of each 'stair' is related to how far the extruder nozzle is above the previous layer when the molten plastic is squirted out. The heat of the newly extruded material bonds with the previous layer(s) because it is above the material's melting point temperature and is making either loose surface contact (sintering) or by being pushed slightly into the previous layer (diffusion). (Sung-Hoon et al., 2002)

3.8.3 The Process of 3D Printing

A 3D printer, for all its complexity, is normally a standalone device that take a digital model as an input and eventually outputs a physical model, if all goes according to plan. This is neither the true start of the design process nor is it necessarily the end of the manufacturing process, as there is often some post-production involved.

The overall process of 3D printing can be thought of as a series of phases, starting with the creation of a digital design and following through a series of processes until a finished object is produced. For an industrial product, these phases are typically called *conception, design, realisation* (Lutters et al., 2014). Here we break these phases into two parts, *conception – design* and *design – realisation* which is called *Process Planning* (PP). (Livesu et al., 2017)

3.8.3.1 From concept to design

Here we focus on how a concept (idea or sketch) is translated or transformed into a digital design (e.g. a 'model') that can be 3D printed. One conceptually straightforward way to turn a physical model into a digital one is by scanning it using an optical technology like structured light or lasers and using software to interpret the data as a 3D model. Baudisch and Mueller argue that the 3D printing process can be understood as fundamentally an 'AD/DA' or 'analogue-to-digital then digital-analogue converter' technology similar to a photocopier for physical objects.

Using a general-purpose 3D scanner as an input, one could theoretically create a digital model of an existing physical object and then make multiple copies of that object using a 3D printer. The digital copy could also be modified, after scanning, much the same way as video is captured and then edited. As of now, the tools for scanning models and then editing them are not widely available nor usable by most consumers, but as Baudisch and Mueller note, the same could have been said for the similar AD/DA technology of digital video capture and editing. They propose a number of ways this could be advanced in the future (Baudisch and Mueller, 2016).

The future aside, using design software (CAD) is currently the main way that models for 3D printing are created. For professionals, software from AutoDesk and SolidWorks is often used because of their built-in structural and mechanical engineering knowledge. They require specific *domain knowledge* of physical and materials to operate properly. Other tools exist for casual users, either online running in a web browser (*TinkerCAD*) or free to download (*AutoDesk Fusion360* and *SketchUp*) but for non-professionals the lack of domain knowledge is a major barrier to creating working designs. As Baudisch and Mueller note, "The majority of 3D models found in the online database *Thingiverse* for example, are *decorative* objects, i.e., objects that are desirable because of their shape and appearance, but that exhibit no *functional* behaviour."

3.8.3.2 From designed model to realisation

For the purposes of this review, we will focus on the *design – realisation* or *process planning* phases of the FDM process, which are similar for other forms of digital fabrication.

Typically, this process has 3 phases. Firstly, after a digital model is created on the computer using CAD or some other software means, the model is *tessellated* to turn it from an ideal mathematically-described model into a finite mesh of triangular pieces that represent exposed surfaces. Secondly, this mesh is rendered or *sliced* into a number of *tool paths* that describe the actions the 3DP must take to physically build the model, such as print head movements, temperature changes, and filament extrusions. These tool paths represent a specific sequence of machine instructions and are often rendered into a textual and human-readable format called *GCode*. Finally, the GCode is executed step-by-step on the 3D printer to build the object (Livesu et al., 2017). The software controlling the fabrication process is often included in the 3D printer itself in the form of firmware, like the popular, open-source Marlin (Various, 2019b).

Describing these phases in terms of the different pieces of software involved:

1. First, digital models are exported from CAD software as triangular mesh geometry, typically in the form of STL files (Wikipedia, 2019b; Gao et al., 2015). They are often run through intermediate software packages to check for holes in the mesh or other irregularities such as MeshLab or MeshMixer.
2. Second, software packages such as the open source Cura are used to load these geometry files and slice them into machine-specific tool paths in a particular GCode *flavour* (a variant of standard GCode that varies with printer model and firmware).
3. Third, this file is then either loaded directly into the 3D Printer itself via a SD Card inserted into the printer and then printed via the printer's GUI, or streamed to the printer using a USB connection to the host computer and a program like Octoprint[8] or Ultimaker's Cura[9] which monitors and controls the printing process.

8: `https://octoprint.org/`

9: `https://ultimaker.com/software/ultimaker-cura`

As Baudisch and Mueller (and many others) have observed, in terms of technical progress relative to mainstream computing, this process "lives roughly in the 1960s" in that it resembles the punch-card era when a computer program had to be loaded and then run without much of a chance for user intervention once the process is started. Once the GCode has been loaded onto the printer, there is little the user can do to influence the fabrication process other than to make sure that nothing unexpected goes wrong during the hours-to-days-long process of fabricating the object (Baudisch and Mueller, 2017).

Even in the previous slicing stage, there is often little chance for meaningful user input. This prevents users from catching and solving problems at early stages of the process, meaning that they have to go through the entire process before they can fix even simple errors. For example, many rudimentary problems that the slicing software can catch, such as excessive overhangs or holes in the mesh, can't be fixed without going back to the beginning of the design–realisation process and re-working the original CAD model for export.

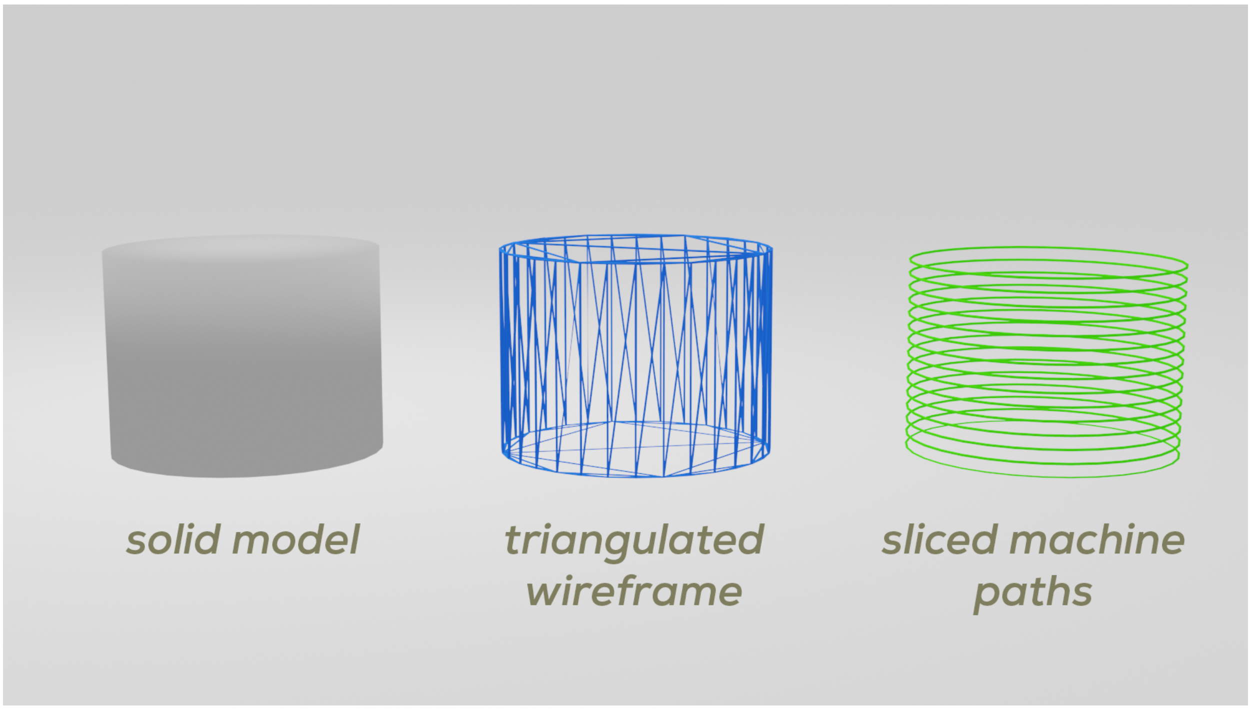


Figure 3.2: Three main phases in process planning (as described by Livesu et al. (2017)) of 1) solid modelling of object geometry, leading to 2) tessellation (surface triangulation) and then to 3) slicing where machine tooling paths are determined, usually as a series of stacked horizontal layers.

3.9 Challenges and opportunities facing 3D printing

Baudisch and Mueller (2017) divided the main challenges to desktop 3DP into general themes organised around three levels of scale, from hardware to users to society. This research project focuses on the second set of challenges (numbers 2,3,4) in the yellow areas of Figure 3.3[10]:

10: these later map to the design goals for LivePrinter in Subsection 7.3.2 (Tasks and goals)

2 Domain knowledge: the physical knowledge of materials, mechanical and structural engineering

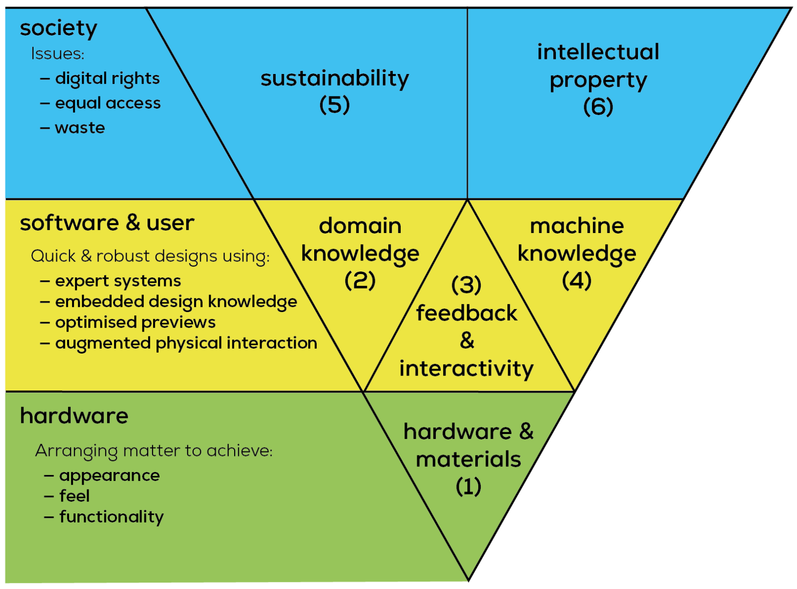


Figure 3.3: The Six challenges for fabrication research, and their levels of specificity, informed by (Baudisch and Mueller, 2017)

3 Visual feedback and interactivity: working interactively within the (currently) long fabrication times of 3D printing
4 Machine-specific knowledge: understanding and controlling computerised manufacturing systems, including tooling options

To briefly summarise these challenges (each of which is large enough that Baudisch and Mueller (2017) give them entire chapters in their book) we start with their concept of *domain knowledge*. This term can be paraphrased as "what the software needs to *know* in order to help users design an object that is structurally sound, including solving technical problems" (emphasis added). The question of what it means for software "to know" and how that manifests in a computational fabrication system is illustrated in specific examples in the rest of their chapter, such as the system's ability to recognise common situations and offer potential solutions, to offer libraries of interchangeable parts, and to incorporate simulations of physical structures.

The challenge of *visual feedback and interactivity* centres around how to help the user iteratively make aesthetic judgements about their work-in-progress. The core of the issue is mainly about what form the software/hardware interface takes, and how that form mediates the user's actions in the exploratory, trial-and-error process of making. Baudisch and Mueller (2017) recognise that all of these new digital fabrication technologies fit in between the person and the physical material, adding new capabilities but also creating an increasing distance between the maker and their materials. This places extra physical and conceptual constraints on the act of making, as well as increasing the overall complexity of the process.

The feedback and interactivity in digital making processes can be compared to the more traditional crafts, where a craftsperson has near-direct contact with their raw materials. For example, a hand-held chiselling tool works as a direct extension of the human hand, but harder and more specialised for chipping away at a stone block. A person using a chisel has immediate control over the tool and making process, to the point where they can directly feel and hear the vibrations of the tool scraping the stone.

Then there are 3D printers, where the designs stage often happens in the computer, far removed from the printer that will eventually manufacture the object in a semi-automated fashion, with little or no direct contact from the designer. This process offers precision, repeatability, and new aesthetic possibilities, but at a cost of added complexity, constraints on what forms are possible, and an increased distance from the act of making.

One of the practical challenges of feedback and interactivity is concerned with how to bridge the conceptual gap between the user's design and the final fabricated object. These *previews* of the in-progress results of the user's design range from static images on a screen to fully interactive previews that sometimes have tangible results. A preview might be a static image showing a "snapshot" of the design as it might look after manufacture, or a 3D model in itself with some digital tools for interacting with its form and structure. When previews become less virtual and more physical, they can incorporate the tangible interface concepts of Shneiderman (1983)'s *direct manipulation* and Willis et al. (2011)'s *interactive fabrication* where making operations are performed on a physical form in real-time (or thereabouts), or *continuously*.

In the next challenge Baudisch and Mueller (2017) introduce the idea of *Machine-specific knowledge* as "knowledge about and. . .specific to the fabrication machine at hand". The examples they give are mainly practical examples, such as how software might automatically split larger models into smaller ones so they fit properly into the build volume of a particular 3D printer, or how specific printing motions can be optimised to produce better models, such as using slower tool movement speeds when needed or printing at angles for sturdier parts.

The challenge area of *Machine-specific knowledge* is conceptually awkward as a top-level challenge when the examples given by Baudisch and Mueller (2017) clearly require the software to have some encoded understanding of physical and mechanical systems, which is part of the other top-level challenge of *domain knowledge*. For software to help a user to understand a machine, which is quite literally a mechanical system, the software needs to encode relevant physical models describing how the machine works. Since we live in a physical world, any computational system for making physical objects will need to encode some understanding of physics.

Drawing a boundary between the knowledge of materials and mechanics in the world, and the inner workings of mechanical fabrication systems, appears somewhat arbitrary. What is likely meant by this challenge is that the workings of the machine are as important as a basic understanding of the physics of the structures that it makes, but perhaps the underlying challenge for both these areas might be better articulated as a question: What levels of physical abstraction are useful for computational fabrication systems? Rigid objects? Soft bodies? Fluids? Or even at atomic and subatomic levels, leading to quantum mechanics?

This remains an open question, but an imminently practical one for the designers of these systems and for the practitioners working with them. When a systems designer creates software and hardware that hides away complexity or makes complex systems difficult to access and change, they might be creating *viscous* constraints for others that they wouldn't appreciate in their own practice: "the same desirable flexibility in changing information structures should be offered to all users. . .designers should be alert to the possibility they might be imposing a high level of viscosity on users that they would not accept for themselves." (Alan Blackwell and Fincher, 2010) Perhaps the conclusion we can draw from Mueller (2017)'s challenge is that 3D printing cannot be treated as a black box, where designs go in and objects come out. Put another way, designers and makers cannot ignore the physicality of the fabrication machine they are using and just concern themselves with the objects that come out.

3.9.1 Manual computational fabrication

For more manual, direct physical control over fabrication machines, there have been some promising experiments in *interactive fabrication*. These experiments represent a hybrid form of computational machine, blending physical and computational interfaces in the computer-controlled production of physical objects (Willis et al., 2011). (Baudisch and Mueller, 2017, p. 230) go further differentiate the turn-taking or asynchronous aspect of some forms of interactive fabrication with *continuous interaction* defined as "systems that fabricate continuously while the user interacts with the workpiece." For both cases, their book collects a number of promising experiments that blend the precision and embedded domain knowledge of computerised tools with people's creative intentions during the real-time fabrication process.

Other studies, such as Mueller, Lopes, and Baudisch (2012), Peng et al. (2018), and Li et al. (2017) have looked at Augmented Reality (AR) or computer vision as interfaces for quickly creating on-the-fly tool paths for machines. In Mueller, Lopes, and Baudisch (2012)'s *constructable*, a laser pointer was used to "trace" cutting paths in materials for an augmented laser cutter. The system followed the general area of where the user specified a cut, but often substituting the judgement of the system in place of the user's wishes by making sure that edges lined up and shapes were filled. The system helpfully compromised between the user's intentions and the structural integrity of the material, as well as the safety of the cutting process.

In Peng et al. (2018)'s RoMA, a similar approach is taken using an AR system to allow the user to specify and then pre-visualise 3D printing operations on an actual physical model. This 3D visualisation, combined with haptic controls, helps direct a robot arm to perform the desired fabrication. Again, some control is given up by the user in exchange for speed of modelling, structural integrity and safety.

For more manual, direct physical control over fabrication machines, there have been some promising experiments in *interactive fabrication*. These experiments represent a hybrid form of computational machine, blending physical and computational interfaces in the computer-controlled production of physical objects (Willis et al., 2011). (Baudisch and Mueller, 2017, p. 230) go further differentiate the turn-taking or asynchronous aspect of some forms of interactive fabrication with *continuous interaction* defined as "systems that fabricate continuously while the user interacts with the workpiece." For both cases, their book collects a number of promising experiments that blend the precision and embedded domain

knowledge of computerised tools with people's creative intentions during the real-time fabrication process.

3.9.2 CAD tools and lost information

Current tools (e.g. CAD tools designed for 3D fabrication like Fusion360, SolidWorks, Blender and Microsoft's 3D Builder) trying to accommodate Mueller (2017)'s *interactivity and feedback* and *domain-specific knowledge* often bolt on many of the elements described above by Tompson et al. (2016) to a classic CAD geometry-oriented workflow, like physics and generators for different types of microstructures. Whilst it can be helpful to have general knowledge of the physics of models using Fixed Element Analysis (FEA) and some rudimentary understanding of material properties, these packages are still a long way from being holistic design tools that can manage a process from start to finish (Livesu et al., 2017). A few software packages working in series are required to get to a final result, as seen in

This has a lot to do with historical, pre-AM workflows. In the recent past, industrial designers would create a product model (virtual or physical) and then use *Process Planning* (PP) to develop its manufacturing process. Products were often manufactured in multiple stages using multiple techniques, such as Subtractive Manufacturing (SM) and other traditional techniques that were "so complex that the experience of a skilled manufacturer [was] unavoidable." With AM, the manufacturing process happens (mostly) in a single machine, and so the *process planning* is mostly about build orientation and machine-specific settings (Livesu et al., 2017; Edwards, J. Chen, and Warth, 2016).

Many contemporary professional CAD tools were originally designed for this multi-stage, pre-AM workflow, where each software tool is a single link in a software chain, designed for a specific stage of manufacturing, meant to be utilised by specialised technicians. For reasons of basic compatibility, these tools adhere to lowest-common-denominator data sharing formats that allow only limited amounts of information to be shared with other software packages, even with ones directly adjacent in the workflow chain.

This can be seen in common file formats used in CAD like STL that only retain geometric information about shape and say nothing about tooling, materials, or texture (Wikipedia, 2019b; P. F. Jacobs, 1994). This is beginning to change with formats like 3MF, but at the time of this writing the interoperability of these formats between software and machines is still limited (Wikipedia, 2019a). Again, this problem is recognised in Mueller (2017)'s challenges, where the *domain knowledge* and *machine-specific knowledge* of each design is lost when it is shared in lossy file formats (like STL) between designers and makers. A software package might run a complex FEA simulation on a part, but that information almost never finds it way into the output file when it gets passed to another software tool.

3.9.3 Filling up space with plastic

Ordinarily the 3D printing process starts with a digital model created in a Computer Aided Design (CAD) program that defines the geometry of the object that is to be printed. This geometry mostly defines the outer and inner surfaces of the object, leaving the process of describing how the internals are to be manufactured to another piece of software called the "slicer". The slicer determines the layer-by-layer construction of the object based on the properties of the model of printer to be used in that construction process. The process of starting with a model, optimizing the geometry, and preparing the machine instructions for manufacturing is often referred to as "Process Planning" (Thompson, 2007; Livesu et al., 2017) and has been previously discussed in Section 3.8.3 (The Process of 3D Printing).

The problem of how to efficiently generate 2D tool paths that fill up 3D space in a structurally sound way whilst minimising time, material and movement remains an open area of research (Jin, He, J.-Z. Fu, et al., 2014; Jin, He, G. Fu, et al., 2017; Ding et al., 2016). Current software packages offer a variety of filling patterns, from linear stripes to diamonds to vector patterns. Recent research has also looked at the properties of the Hilbert curve for filling spaces both rectangular and irregular (Ding et al., 2016; Papacharalampopoulos, Bikas, and Stavropoulos, 2018), which is discussed in some detail in a later section.

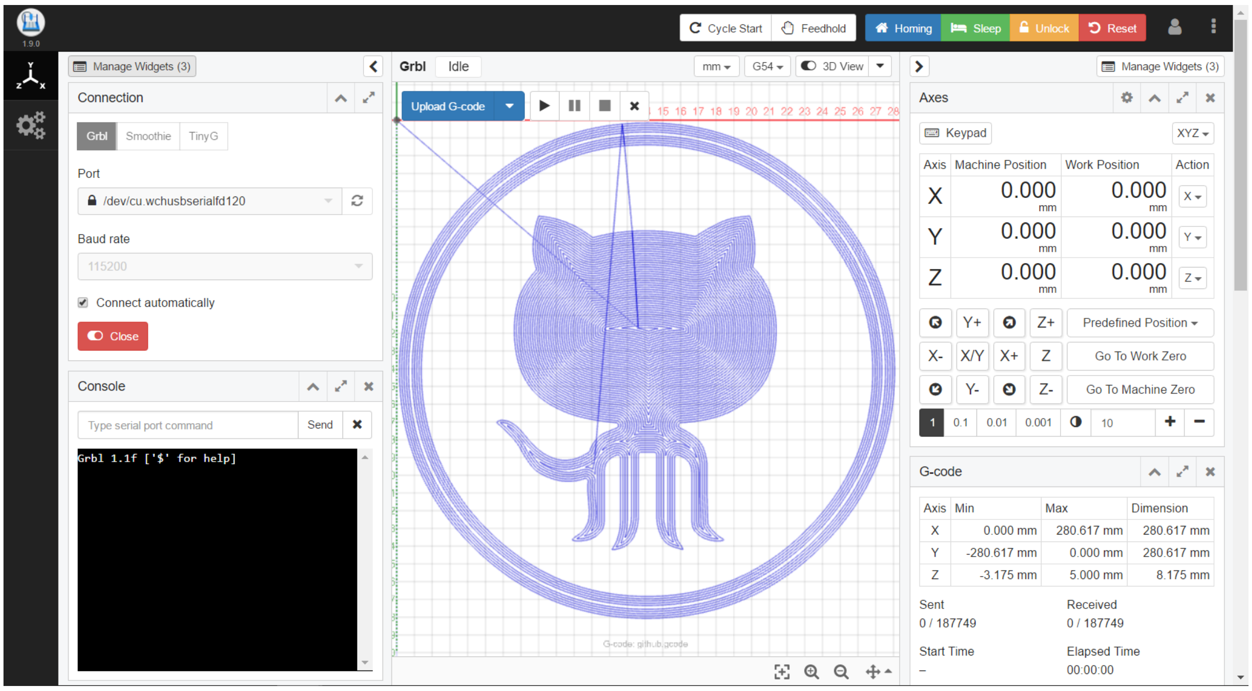
Filling patterns that take more time to move across a space will not only waste machine and staffing time but also have higher energy costs. Tool paths that can fill spaces in continuous lines are preferred as they result in less extraneous movements of the print head as it jumps around from the start of one filling movement to another. These extra moves are called "travel moves" and can add quite a lot of time as the tool head raises up from the end of one movement, retracts the hot filament to prevent leakage during travel, moves to another area, lowers down again and feeds the filament forward to prime it for the next drawing operation.

3.9.4 Controlling 3D printing with code

One solution for adding back in more detailed knowledge of material physics and manufacturing might be to use a more descriptive format, like computer code, to represent manufacturable objects. The logistics of adopting such a format are not simple, as they would rely on competing software and hardware manufacturers, amongst others, coming to a public agreement. For now, this could be somewhat achieved through adding textual interfaces to existing CAD software. Graphical interfaces have been a useful way of interacting with software, but they have limited screen space in which to represent complex processes with many options, with the result being that parameters and processes inevitably get left off the screen.

As a quick and easy way of giving advanced users control to extend the software, textual interfaces like REPLs have already been added to popular 3D printing rendering, slicing and monitoring software. In combination with GUIs, they enable users to do more advanced tasks like manipulating models programmatically and directly controlling 3D printers. REPLs allow users to customise many aspects of the software program, including the GUI itself.

In the *design – realisation* phase of 3DP REPLs are used in both slicing and print managing and monitoring software. Sometimes these functions are combined, as with the REPL in the slicing and monitoring software Cura which provides a way to interrogate the slicing process and send simple GCode commands to the printer during its operation.



Connection: Grbl | Smoothie | TinyG
Port: /dev/cu.wchusbserialfd120
Baud rate: 115200
☑ Connect automatically
Close

Console: Type serial port command — Send

```
Grbl 1.1f ['$' for help]
```

Grbl Idle

Axes

Axis	Machine Position	Work Position
X	0.000 mm	0.000 mm
Y	0.000 mm	0.000 mm
Z	0.000 mm	0.000 mm

G-code

Axis	Min	Max	Dimension
X	0.000 mm	280.617 mm	280.617 mm
Y	-280.617 mm	0.000 mm	280.617 mm
Z	-3.175 mm	5.000 mm	8.175 mm

Sent 0 / 187749 Received 0 / 187749
Start Time – Elapsed Time 00:00:00

Figure 3.4: The CNCjs interface, from https://github.com/cncjs/cncjs

11: https://github.com/cncjs/cncjs

Similarly, the open source printer monitoring software Octoprint and CNCjs[11] provide a REPL for tweaking parameters and sending simple commands. This is of limited use in interactive programming because of the size of the REPL area, and the lack of a functional text editor and line-by-line execution, as evident in Figure 3.4.

Interestingly, there is little evidence on the online forums that REPLs are used for anything in 3D printing except setting up prints using machine-specific settings and tweaking prints-in-progress by using GCode to alter settings not found in the GUI. Part of the reason for that could be the design of the REPLs, which often only allow a user to type in a single line of code at a time, thus preventing them from easily writing and running programs, as seen in Figure 3.4 under the “Console” area in the bottom left.

Code can also be used at the start of the design process, before a model is manufactured or *realised*. During the *concept/design* stage of 3DP, code is sometimes used to generate a range of models based on different mechanical or physical procedures. In parametric and algorithmic 3D modelling software packages, like Rhino/Grasshopper and Blender, code allows the designer create, alter and add geometry to models with precision and repetition, both of which are difficult to represent through manual onscreen controls. For example, code allows model geometry to be expressed precisely (specifying (x, y, z) vertex data as explicit numbers like $[2.0, 4.5, 0.0]$) and is often combined with recursive and iterative processes like procedures, loops and functions that can describe a method for generating millions, even billions of data points in just a few lines. Again, this is not yet a very portable way of working because it requires designer-developers to work with multiple files across multiple pieces of software, different versions of that software, and even across multiple programming languages!

Code is also important in the relatively new field of *metamaterials*, which are materials that deform in useful ways due to the complexities of their structure. A variety of these mechanisms exist, such as simple "walking machines" and one-part pliers and door handles (H.-T. Chen et al., 2006). Their repeating structures are often generated programmatically, meaning that they require software to design and generate the physical geometry that will be used for 3D printing (Florijn, Coulais, and Hecke, 2014). This means that all the information that parameterises the model's mechanism is lost when the CAD file is shared between users, unless they also share the original software that generated it and all the relevant files to recreate the model.

An example of this in practice is seen in Amorim, Nachtigall, and Alonso (2019), where they experimented with algorithms and a series of software processes for generating bespoke footwear. The team used a series of software packages to design 3D printed metamaterials with different mechanical properties (e.g. stiffness, softness) for use in the soles of the footwear. The process was complex, and one of the key findings was that current software packages are too limited to support designers of metamaterials, even though the potential gains in sustainability and customised material properties could be significant.

3.9.5 Programmatic tool paths

For many practitioners, computational fabrication (as the broader category encompassing AM) has been understood as a holistic process integrating materials knowledge, mechanics and software in ways that are not easily untangled. Some of the main aspects of this process were laid out by Tompson et al. (2016):

> To receive the full benefits of AM, designers must learn to think differently while focusing on creating robust industrial solutions with added value. Design theories, processes, methods, tools, and techniques (Lutters et al., 2014) must be adapted or developed to address the inherent coupling between material, geometry, and quality in these systems. Specialized and application-specific tools must be developed to support the design of cellular structures, metamaterials, heterogeneous artifacts, biological scaffolds (e.g. (Podshivalov et al., 2013)), and more. Finally, it must be acknowledged that each build is a design artifact with its own requirements and constraints, and its own features (e.g. support structures, part layout, etc.) to be designed and optimized. Thus, DfAM must extend beyond the product to the production system and consider the entire value chain.

In 3D printing, at the junction of "material, geometry, and quality" lies the concept of the tool-path. The path that the print head takes during the fabrication process determines much of the success of the final outcome. In field of computationally fabricated ceramics, Jonathan Keep, a potter and sculptor, built his own version of a clay 3D printer so that he could experiment with his tool paths directly. He used his printer to evolve a hybrid way of working with his machine; a printing process combining an automated material deposition system with a more manual assembly process, informed by his deep knowledge of his craft (Keep, 2020; Keep, 2014). He calls this process of "computerised coil building" (Keep, 2014, p. 32) a new, fourth way of making ceramics beyond traditional methods of hand-building, throwing, and moulding.

Keep's reference to "coils" reveals the depth of his relationship between the shape of the fabrication process and the ultimate form of his pieces. As obvious as this might seem to any craftsperson used to working with their hands, it should be just as obvious to computational designers using 3D printers that rely on a single, moving tool head to build their forms up line-by-line, and layer-by-layer. It is also another example of an area of the fabrication process that could benefit from the integration of code with geometry data, resulting in a computationally-augmented manufacturing process.

The application of computer programming to the shape of the fabrication process itself can be seen in the growing body of research into ways of procedurally optimising the tool paths in 3D printing to help increase part strength, lower printing speeds, and create different textures across the surface of the finished objects. Continuous deposition, where the tool head moves throughout the entire manufacturing process without pausing or stopping, and the material flow is kept steady without any retractions, is one such promising area of research. Zhao et al. (2016) worked with spiral fills, showing that optimising the tool path could decrease printing time and also increase the quality of the final print in certain circumstances. Gupta, Krishnamoorthy, and Dreifus (2020) created a framework for generating a single, continuous tool path to fabricate a 3D model that doesn't cross over itself. Papacharalampopoulos, Bikas, and Stavropoulos (2018) and Bertoldi et al. (1998) looked at using continuous curves (such as Hilbert curves) for path planning. These are just a small selection of other published algorithms, each with their real-world drawbacks depending on the specific methods chosen (Ding et al., 2016).

Others go beyond optimising paths for manufacturing pre-rendered objects. Programmatic paths can become an integral part of design, allowing a designer to create forms that adapt to their surroundings. In Pattinson et al. (2019), for example, an explicitly programmed tool path results in flexible mesh materials with bespoke mechanical properties and geometry. These materials shaped to conform to the soft tissue of living bodies where they are designed to be implanted, creating new kinds of "wearable" devices.

3.9.6 3D printing as sonic performance

Three motors drive the print head in x, y, z directions where x is side-to-side, y is front to back, and z is up and down (often moving just the print surface, or "bed"). Another motor feeds the plastic filament through the head, also pulling it back at times to prevent unwanted material leakage in a process referred to as "retraction".

As these motors spin, they vibrate and make sound. People have used 3D printers to make music, notably the Imperial March from Star Wars and, less notably, Nickelback (Milkert, 2014). Also, performer and researcher Ezra Teboul composed a series of works called "Music of the spheres" consisting of 3D-objects-as-scores for a MakerBot 3D printer using his knowledge of the sonic properties of 3D printed forms and musical structures (Teboul, 2020).

Helpfully, a library for the Python language is available to convert MIDI note numbers to motor frequencies (Westcott, 2015). The sounds of the motors are relatively quiet but can be captured using contact microphones attached directly to the printer motors and then amplified using audio amplifiers, as was done during the experiments in this chapter.

MIDI (Musical Instrument Digital Interface) is an industry standard music technology protocol for connecting musical devices. The MIDI standard includes a definition of note numbers mapped to standard Western musical pitches that I used in the design of LivePrinter. MIDI is quite a commonly-supported standard, and has been around in some form since about 1983 (see `https://www.midi.org/`)

This leads us to the possibility of livecoding performances using 3D printers. As A. Sorensen, Swift, and Riddell (2014) pointed out, livecoding has an inherent *cyber-physicality* in that it causes "physical perturbations in the world" through executing computer code. These perturbations are often experienced in the vibrant humming of speakers or the blinding light of projection, but they can also have more direct effects in the physical world.

Livecoding can be a means for directing the physical movements of machines, such as robots and 3D printers. This change in media from sound and screen to machine presents some unique challenges. For 3D printing, one of the biggest challenges is how improvise when making new forms out of extruded lines of plastic, without accidentally destroying those forms in the process. This means moving the printing tool head safely and without hitting previously constructed structures.

Also of concern is the structural soundness of the printed fill patterns. Some have better shear strength, others better tensile strength. It is not strictly necessary to fill up all the empty space inside a print if the geometry of the fill can support the forces applied to the object from outside. For example, diamond-shaped fill patterns result in diamond-shaped air gaps inside the printed object, which is then covered with an outside "wall" so that the outer surface is smooth and unbroken. In the unique use case of 3D printing as audio/visual/sculptural performance, the strength of the printed objects is not necessarily as important as other aesthetic properties of the performance like sound and visual form.

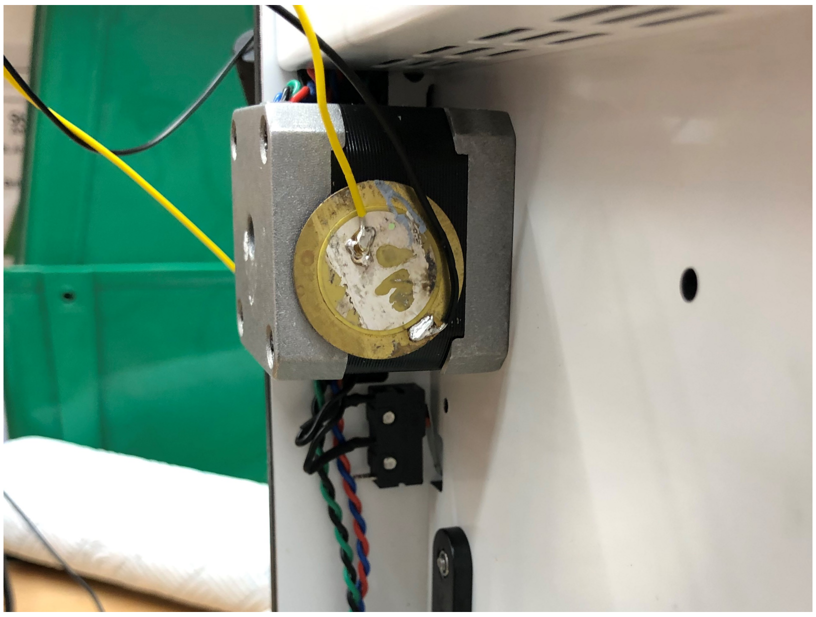


Figure 3.5: A well-worn piezo-electrical disc is attached to the z-axis motor of a 3D printer, acting as a contact microphone that picks up the sound of the motor vibrating as it is used in printing operations.

3.10 Conclusions

Clearly, there are many opportunities in the computational design process and 3D printing in particular where code could be used to describe forms, or families of forms, instead of using static geometry. As researchers have shown, current software packages may have some support for using code to generate forms, but they currently lack standardised support for using code to directly control manufacturing processes (i.e. tool paths). This limits our ability to experiment with material properties and adaptive geometry, which has implications in the fields of metamaterials and bioimplants, to name two areas that could directly benefit from more programmatic manufacturing.

At the same time, the ways that users currently interact with the 3D printing process also leaves much to be desired. New areas of research like *interactive fabrication* and augmented reality fabrication have some potential to get users directly involved with the entire fabrication process, from start to finish. What remains under-explored is the role that interactive programming can take in the interactive fabrication process.

Perhaps livecoding systems can be helpful points of reference in such a practice-led research process. A number of useful examples of such research-through-design models exist, like A. Blackwell (2018)'s "craft practices," along with a growing number of livecoding and interactive programming systems that can be used as examples of technical implementation patterns.

Methods 4

4.1 About

This chapter explains the research methods used in this thesis and how they relate to the nature of the research goals. It discusses how these methods and their associated outcomes were influenced by the abductive theories of *Research-through-design*, *practice-led craft research* and other qualitative research methods. Finally, it lists the main research activities and outcomes, and links them to key influences from the "design canon" of livecoding and interactive programming projects.

4.2 Introduction

The initial work on the thesis began in 2015 with a series of experiments exploring computationally-designed, semi-generative models optimised for 3D printing. When these experiments hit up against the limited capabilities of current 3D printing software and the difficulty of optimising prints for FDM 3D printers, the project pivoted into trying to answer the short question, "What would it look and feel like to combine livecoding and augmented manufacturing into an artistic practice?"

The process of trying to answer this question was abductive, iterative, and varied in approach. As with many practice-led research activities, it is hard to separate the research outputs from research activities because they are so intrinsically intertwined. The underlying development process employed a mixture of empirical and reflective methods, whose activities could be categorised as: unstructured interviews for background research; structured interviews; user studies; reflective software development; reflective production of physical artefacts for exhibitions; and reflective production of live performances.

As part of the background research we conducted interviews with a variety of practitioners who had some experience of 3D printing, especially with interactive or computational fabrication. These interviews helped us understand how others approach 3D printing in their practice, and what they hope to get out of it. These interviews, along with other user studies, are discussed in detail in Chapter 7 (User Studies and Analysis).

Insights from the interviews, combined with reflective experience from our own experiments, led to the development of a bespoke software and hardware system for experimenting with an interactive programming approach to computational manufacturing, ultimately called *LivePrinter*. This system embodies a new, more general approach to interactive 3D fabrication that we call *Interactive 3D Printing (I3DP)*. The design of a "typical" I3DP system is discussed in the context of various cognitive and usability theories in Chapter 5 (Designing for Interactive 3D Printing).

The specific design details and technical challenges inherent in implementing the LivePrinter system can be found in Chapter 6 (Implementing an I3DP system: LivePrinter). Later on, When self-testing proved that the LivePrinter system was stable enough as a hardware and software system, we ran a series of structured user workshops to test the usability of the system and to explore the concept of livecoding 3D printing in general, which are also recorded in Chapter 7 (User Studies and Analysis).

The data from the user studies and interviews were mainly evaluated through a thematic analysis. The results of this evaluation, combined with a reflection on the *Cognitive Dimensions of Notations (CDNs)* (T. R. G. Green and Petre, 1996; Alan Blackwell and Thomas Green, 2003; A. F. Blackwell et al., 2001), helped us analyse details of the design of the interactive programming systems.

Whilst the user tests were underway, LivePrinter was being used for a series of performances and as a vehicle for creating physical sculptures. These activities, along with reflective examples from the researcher's practice are discussed in Chapter 8 (Filling space, filling time).

4.2.1 User studies undertaken

The key user research activities of each type that were undertaken during the project were:

- **2017–8:** Initial exploratory interviews with 3D printing end users and practitioners
- **2019 Jan. 9–10, Goldsmiths:** 4 workshop sessions on LivePrinter (2 per day), approx. 24 participants of mixed ages, technical literacy and professional backgrounds
- **2019 Jan. 16, ICLC at MediaLab Prado, Madrid**: workshop session with about 8 livecoders attending the conference
- **2019 Feb. 19, Goldsmiths**: workshop session with 6 MA/MFA Computational Art students with experience of physical computing
- **2019 June 3, Brooklyn Research, Brooklyn**: workshop session with 7 participants with backgrounds in music production, art, or hardware/software development

4.2.2 Exhibitions and presentations participated in

The key performances, exhibitions and other events from this thesis were:

1. 2018, early experiments printing on materials; initial software development for LivePrinter
2. 2018 Sept. 1, TOPLAP Moot Algorave, Sheffield, UK[1]: first public performance
3. 2019 Feb. 15, Goldsmiths Algorave for TOPLAP's 15th birthday[2]: first full performance
4. 2019 May 5-6 Expressive '19 Art exhibition at Eurographics 2019: exhibition of generative sculptures created using LivePrinter

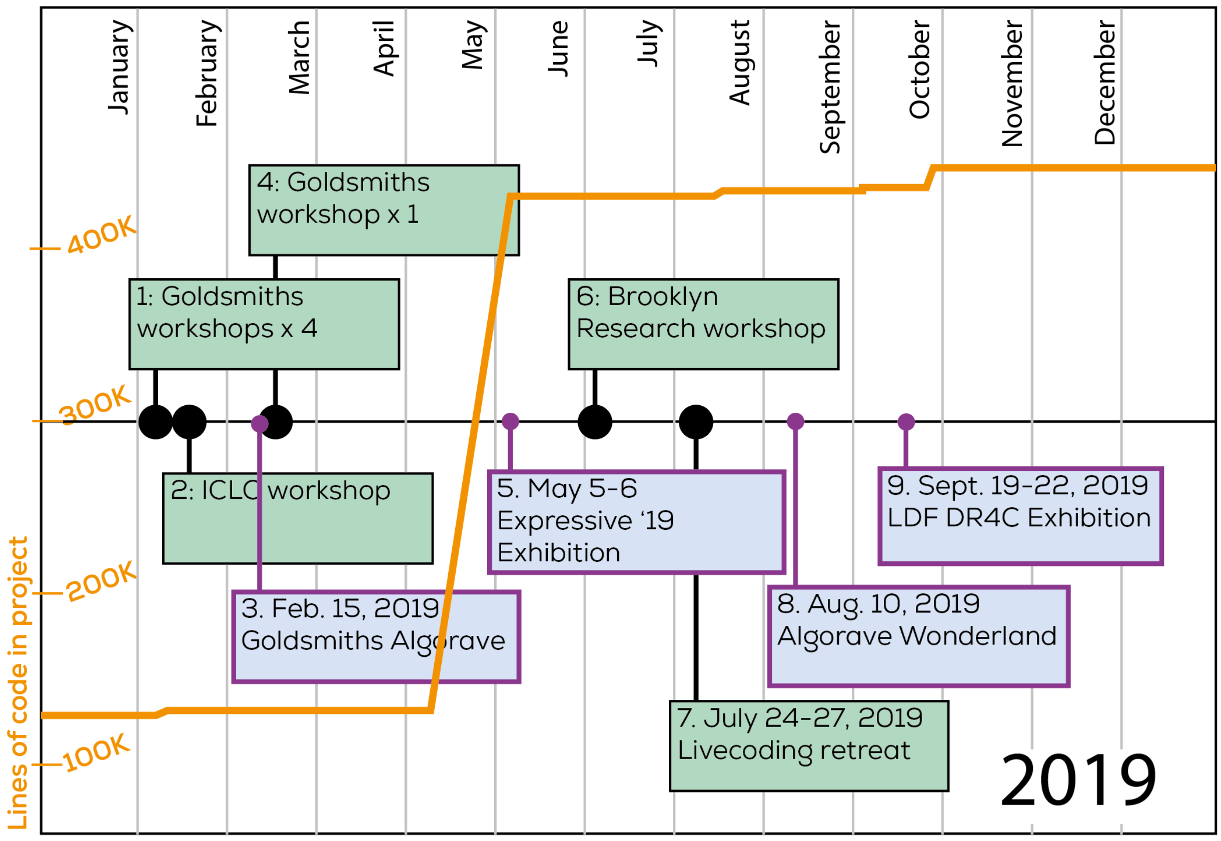
1: `http://livecode.toplap.org/2018/events/algorave/`

2: See Subsection A.1.1 (Goldsmiths Algorave 2019)

5. 2019 Jul. 24–27, the Mimic/SEMA project's machine learning Live-coding retreat and performance in Brighton, UK[3]: minigrammar syntax for Interactive 3D Printing developed
6. 2019 Aug. 10, LiveCodeNYC Algorave at Wonderland, Brooklyn, NY: second full public performance
7. 2019 Sept. 19–22, London Design Festival Design Research for Change (DR4C) Exhibition (Rodgers, 2020): exhibition of generative artefacts, code, techniques
8. 2020 Jan., ICLC in Limerick, IR: poster presentation about LivePrinter and using Hilbert curves for physical techno music performance
9. 2020 Apr. 28, *CyberYachtVR Algorave* in VR on Mozilla Hubs: third public performance
10. 2020 May 8, Live-streamed performance for the *Quarantine Stream* event: fourth public performance

3: `http://www.emutelab.org/blog/summerworkshop`

One can get a sense of how these activities progressed by looking at a timeline of the code development over the main period of the thesis, 2018-2019, seen in Figure 4.1.



User Research Workshops

1. 2019 Jan. 9-10 Goldsmiths (4 sessions, 2 per day, mixed ages and backgrounds, approx. 24 ppl)
2. 2019 Jan. 16 ICLC Madrid (MediaLab Prado with about 8 livecoders)
4. 2019 Feb. 19 Goldsmiths (6 MA/MFA students)
6. 2019 June 3 Brooklyn Research, Brooklyn (approx 7 with backgrounds in music or tech)

Exhibitions, Performances and Events

3. Feb. 15, 2019, Goldsmiths Algorave -- first live performance
5. May 5-6 Expressive '19 Exhibition -- sculptures and rendered gcode for exhibition
7. July 24-27, 2019, Livecoding retreat -- minigrammar added
8. Aug. 10, 2019, Algorave Wonderland -- public performance of Hibert techno
9. Sept. 19-22, 2019, LDF DR4C Exhibition -- sculptures and fill patterns

Figure 4.1: A timeline of key research and development events during the project. To view these changes in more detail, download the project via GitHub and run: git log stat since 2019-01-01 until 2020-01-01

4.3 The research process

The shape of this research process could best be described as belonging to the *Research-through-design* family of approaches. These processes can often be described as wicked problems, where success is hard to define, and the design process is inductive, iterative, and prototype-led, arriving at a potential solution first before the problem can be fully understood (Rittel and Webber, 1973). In the words of Koskinen et al. (2012, p.6) this process is about the creation of ". . .vehicles for research about, for and through [art &] design" (Matthews and Wensveen, 2015), sometimes called “constructive design research” or research that “imagines and builds new things and describes and explains these constructions”.

In particular, the *cybernetic model of design research* Jonas, 2007; Jonas, 2015 was an important conceptual guide. It situates the designer–researcher inside their community, creating artefacts and interventions for both themselves and their community in an abductive, iterative fashion. There is no boundary or hierarchy between researcher and “users” in the community, which was appropriate in this situation because many of the participants in this study were livecoders and researchers in their own rights, or experts in their design practices, with important insights to contribute.

Each experimental artefact and intervention was guided by the *cybernetic model's* (Jonas, 2015, p. 26) process of “internal or external perturbations (called ideas, creativity, intuition, accidents, environmental changes, etc.)” to create variations to aspects of the LivePrinter system (and it's associated outputs) leading to “stabilizations (negative feedback)” in the evolution of different versions and outputs, or “amplifications and evolutionary developments (positive feedback)” based on analysis of the results.

The co-evolution of new computational tools for a design process, as part of a design process, is also a form of research about *design itself* as noted in Gaver (2012), Jonas (2007), and Jonas (2015). This part of the process is evident in Chapter 5 (Designing for Interactive 3D Printing) on I3DP systems design, and in Chapter 6 (Implementing an I3DP system: LivePrinter) on the implementation of the LivePrinter system, which have as much to say about the process of designing such systems as they do about using them.

In terms of specific research outputs, Alan Blackwell and Aaron (2015)'s notion of *practice-led craft research* was also a helpful guide. This process, which is arguably a re-framing of *design-as-research* lists some key components of the process that are more specific to creating interactive programming systems for practice-led research, especially for livecoding:

> i) understanding of the design canon – a body of exemplars that are recognised by our community of practice as having classic status;
>
> ii) critical insight derived from theoretical analysis and from engagement with audiences and critics via performance, experimentation, and field work;
>
> iii) diligent exploration via “material” practice in the craft of programming language implementation;
>
> iv) reflective critical assessment of how this new work should be interpreted in relation to those prior elements.
>
> *from Alan Blackwell and Aaron (2015, p. 3)*

Mapping Alan Blackwell and Aaron (2015)'s framework to this thesis, the *understanding of the design canon* can be found in the Chapter 3 (Literature Review) and briefly below; the *critical insight. . .* can be seen in both the engagement with users in Chapter 7 (User Studies and Analysis);

the *exploration of material practice* in literal form in the material and form studies in Chapter 8 (Filling space, filling time) and in software implementation in Chapter 5 (Designing for Interactive 3D Printing) on I3DP systems design, and in Chapter 6 (Implementing an I3DP system: LivePrinter); finally, *reflective critical assessment* can be found throughout, and particularly in the last chapter that reflects on this process and looks towards the future.

4.3.1 A selection from the "canon of influences"

The LivePrinter project is part of a lineage of computational and programmatic tools for creative human-computer expression and augmented physical making. In more contemporary terms, it exists inside the open source, livecoding community. Some major livecoding influences were:

1. Sonic Pi[*] (Aaron and Alan Blackwell, 2013): Livecoding music for a wide audience using an all-in-one system and friendly syntax
2. TidalCycles[†] (and underlying Haskell): combining patterns of movements and chaining together operations to (mainly) make music
3. Livecodelab[‡]: Quick and playful audio/visual livecoding in the web browser
4. ixilang: livecoding system with an abbreviated syntax, designed to be fast with "a maximum 5-second wait before some sound is heard" (Thor Magnusson, 2011)

Except for TidalCycles, which unfortunately relies on a large Haskell language installation, the other two influences are fairly trivial to install (in LiveCodeLab that means visiting a webpage) and start to use. All of them derive their livecoding syntax from major programming or scripting languages (Ruby, Haskell, or Typescript) but present a somewhat simplified or bespoke syntax to the beginning livecoder that can still be extended by experts.

Other influences were more about constructing visual images or 3D forms:

1. Turtle graphics & Logo: for simple drawing and movement syntax (Papert, 1980).
2. Processing[§]: providing basic graphic design tools in a programmatic way using a simplified editor and syntax
3. OpenSCAD[¶]: programmatic solid 3D modelling

These were not livecoding tools per se, but still interactive enough in nature so that programs were relatively easy to run and see the results.

Another influence was the *Assembler language* for its brevity and conciseness of language and limiting of the basic operations available to

[*] https://sonic-pi.net/
[†] https://tidalcycles.org/index.php/Welcome
[‡] https://livecodelab.net/
[§] http://processing.org
[¶] http://www.openscad.org/documentation.html

programmers, all of which are close to the underlying mechanical model of the computer, such as setting individual bits in registers. The other software influences dealt in more higher-level abstractions, which would hide some underlying complexity of 3D printing that LivePrinter sought to expose.

Similarly, the concept of *L-systems* (Prusinkiewicz and Lindenmayer, 2012) influenced some experiments in creating more concise and potentially recursive descriptions of printer operations, as well as inspiring some artistic explorations in form conducted with LivePrinter.

In 3D printing research, Mueller, Im, et al. (2014)'s *WirePrint* was influential in that it showed possible ways of making that could be explored with bespoke software, outside the boundaries of the usual 3D printing workflows of modelling, rendering and tool path generations. WirePrint's translation of solid forms into physical, 3D "wireframes" that could be printed quickly on desktop 3D printers opened up new possibilities for shapes and form-making, as well as in making existing form-making more efficient in its use of material and printing time.

Finally, Ezra Teboul's performances using 3D printers and composing for 3D printers were influential in that they brought to the forefront the musicality of this manufacturing system, and how that could become a part of humanising its operation (Teboul, 2020, p. 106).

4.4 Reflecting on the design process

The ultimate goal of this process was to find novel relationships between elements of interactive programming and livecoding practice (code syntax, performance settings), 3D printing, and other creative practitioners (product and textile designers) that might provide us with direction for future research into augmented making. The evolution of this livecoding computational manufacturing system was thus guided by an iterative process of *abductive analysis*, trying in each development cycle to cultivate surprising and "anomalous" empirical findings leading to the discovery of novel theories about aspects of computational communication and collaboration in manufacturing (Dunne and Dougherty, 2016).

Regardless of what we label it as, this iterative, practitioner-led, pattern of interaction design is seen in a number of other livecoding projects such as the aforementioned TidalCycles, Hydra and Sonic PI (Aaron and Alan Blackwell, 2013; McLean and Dean, 2018). Programmer–artists who are instigating these projects have the means for bottom-up innovation of their own bespoke computational tools, and they are re-imagining these tools as unpredictable and improvisational vehicles for constructive design research in their practices. New livecoding practices are thus invented, and take root in the community.

3D printing and other forms of CNC manufacturing are not simply new ways of manufacturing – they are new ways of thinking and making, with the potential to have profound effects on our environment and society. They can fundamentally change the way that we design new physical forms, use plastic material and transfer technical skill from humans to machines and vice-versa. As a computational designer myself, my experience using this tool and the questions it raised for my own practice and that of the community needed to be taken into account, as do the artefacts and experience that arose from this practice.

Designing for Interactive 3D Printing using the Cognitive Dimensions of Notations 5

5.1 About

This chapter introduces a general type of text-based, interactive programming system, or "notational system", specifically for controlling 3D printers (but extendable to other CNC machines) in a new practice called *Interactive 3D Printing (I3DP)*. A "typical" I3DP system has been abstracted from the design, implementation, and user testing of the LivePrinter system, as described in the following two chapters. The discussion of the cognitive trade-offs inherent in designing a mainly textual, interactive, real-time, 3D printing environment is supported by applying some general theories of human cognition in relation to the Cognitive Dimensions of Notations framework (the CDNs). Specifically, it identifies key user activities that support both intuitive and deliberative modes of thinking in interactive programming systems. These activities are analysed using the CDNs to highlight trade-offs inherent in designing appropriate notational systems that supports them. This discussion should be relevant to anyone designing interactive programming systems, and especially livecoding environments.

5.2 Introduction

Most popular 3D printing software packages present graphical interfaces to users that let them do basic modification tasks like selecting and loading 3D models, changing various numerical parameters, and loading and saving presets, as well as visually monitoring printer properties and print jobs. In addition, they sometimes have some basic textual input, usually in the form of a Read-Evaluate-Print-Loop or "Console", as discussed in Subsection 3.9.4 (Controlling 3D printing with code). Here we introduce the concept of an Interactive 3D Printing (I3DP) system, where the emphasis on graphical vs. textual control of the printing process is flipped. These systems can be thought of as a new class of interactive programming environments, specifically for textually interactively programming 3D printers.

As we have previously observed, designing interactive programming (and especially livecoding) systems is a kind of *wicked problem* (Rittel and Webber, 1973), a framing device introduced in Section 3.7 (Designing livecoding systems). There are many possible software architectures for interactive programming and livecoding systems, as well as a number of trade-offs in how they can be visually or otherwise presented to a user. In the next chapter, Chapter 6 (Implementing an I3DP system: LivePrinter) we will discuss the specific software implementation details for our working I3DP system, called *LivePrinter*. First, by abstracting our design of an I3DP system to a "typical" one, we can focus more on the cognitive trade-offs inherent in system's notation rather than implementation-specific details.

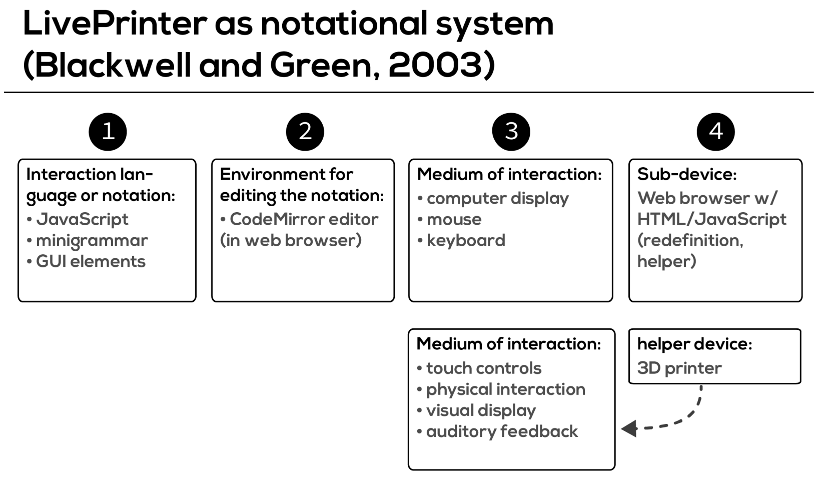


Figure 5.1: A "typical" I3DP system, in this case the LivePrinter system developed as part of this thesis, interpreted as a notational system according to Alan Blackwell and Thomas Green (2003)

5.3 I3DP systems as notational systems

An I3DP system can be thought of as a type of *notational system*. According to Alan Blackwell and Thomas Green (2003, p. 7), there are four aspects of a notational system (illustrated in Figure Figure 5.1):

1. Interaction language or notations
2. Notation-editing environment
3. Medium of interaction
4. Sub-devices (embedded systems, helpers, redefinitions)

In the case of our prototypical system, the *interaction language(s) and notation* would be realised through:

1. A general programming language – an all-purpose language for users to interact with the overall system, such as JavaScript
2. An *Application Programming Interface* or *API* – basic functionality provided through a number of pre-defined textual abstractions
3. A *minigrammar* – a domain-specific syntax more specific to describing I3DP operations, built on top of the general. In LivePrinter, this is called the *minigrammar*.
4. A *Graphical User Interface* or *GUI* – the user-facing GUI elements in a web browser described [1]

As a practical example, the *notation-editing environment* could be realised as a web application viewable in a standard web browser containing a text editor component and a variety of basic interactive controls and informational displays. Most GUI elements could then be standard HTML elements, with some additions from popular fast-prototyping libraries like the *Bootstrap* framework.[2]

This notation-editing environment could include a few *sub-devices* of note, such as the web browser itself. It might be strange to think of the browser as a *sub-device*, but the nature of web browsers is that they are mostly obscured from the user's view by whichever web application they

1: Examples of how web components like the *CodeMirror* text editor could be used in the GUI of an I3DP system like LivePrinter can be as seen in Section 6.4 (The LivePrinter Graphical Interface), and specifically Figure Figure 6.2 on page 72 for an overview.

2: Bootstrap can be found at `https://getbootstrap.org` are running, until the user decides to use the built-in Developer Tools commonly found in major web browsers. These tools reveal that the web application and even basic browser properties can be modified, to an extent, by a REPL built into them.

Other helpful *sub-devices* would be pens and paper provided for users to sketch out ideas in free-form and visual ways not possible in the textual interface[3]. They could also function as way for users to collect and share their designs and methods. Extra notation for quick sketching could potentially make up for a lack of built-in 3D or even 2D visualisation notations for system elements and properties. This was observed in the user studies discussed in Subsection 7.4.10 (User's perspective vs. method naming), where a lack of built-in visualisations for print head position and printing direction was a source of user frustration when using the LivePrinter I3DP system.

3: See Subsection 7.4.8 (Task 8: Using LivePrinter (freestyle drawing)) for examples of this.

Lastly, another *sub-device* for displaying notation would be the documentation. To get users involved in maintaining and extending this documentation it could be written in a format like Markdown that affords easy reading and editing, and hosted on a public git repository such as GitHub or GitLab. Users could view the documentation in their web browser and use git "pull requests" to directly submit changes, or propose changes to it through GitHub's "issues" system.

The secondary *sub-device* of the 3D printer provides some lesser means of interaction, in the form of a basic GUI on the printer itself that is often accessible via a rotary switch and button, and a power switch. This secondary interaction allows users to perform a number of basic functions and modify the properties of the printer during an operation, such as changing the bed or head temperature.

The *medium of interaction* for the system would primarily be visual and on-screen through interacting with the GUI, using the keyboard and mouse for coding and manipulating GUI elements. The software that bridges the user-facing GUI with the 3D printer would not be accessible to the user, however. This software connection allows the devices to communicate with the printer's firmware, which cannot be directly modified by a user during printing operations. This is because most 3D printers are designed to connect to other devices via a built-in serial connection, often over USB, which would then provide their own user-facing interface to printing firmware functions. In practice, a standard web server with access to the computer's serial port can be used for this two-way interface between the web app and the printer firmware. For example, in LivePrinter the software interface between the I3DP system's user-facing GUI and the Marlin printer firmware was implemented as a web server running on a separate computer, built on the Python library *Tornado*, with a physical USB serial connection running directly to the printer.

Additionally, it is possible to make sounds using the printer in various ways, both mechanically through the movement of the printer motors, and digitally through the built-in electronic buzzer acting as a speaker. These sounds give users audible feedback on the printer's speed of movement and the density of current operations simply by listening to it. More explicit sounds, using the built-in electronic buzzer, can be triggered via GCode. Importantly, both of these forms of audio notation can be controlled using interactive programming, albeit in different ways.

5.4 Aligning user understanding and notational systems

Fundamental to a supporting high degree of the Cognitive Dimension of *role-expressiveness* in I3DP is creating an alignment between the user's growing understanding of the 3D printing process, their knowledge of interactive programming, and the system's notation[4]. This brings up the question of the frame of reference of the user when they try to interpret the purpose and meaning of the system's notation.

4: The Cognitive Dimensions of Notations are discussed and listed in Subsection 3.7.1 (The Cognitive Dimensions of Notations).

When tasked with learning a new machine or process, users form their own analogies to help them explain its behaviour. These explanatory models represent how users *think* the machine functions.

Users also rely on implicit or explicit *conceptual models* provided by designers (Norman, 1983, p. 7). These models, also called *implementation models* by Cooper et al. (2014, p. 16), are designed to "provide predictive and explanatory power for understanding the operation" of the system. As Cooper et al. (2014, p. 16) puts it, ". . .users of a user-friendly system should be able to construct a viable mental model directly from the target system". In other words, these designer-provided models help teach users how to use the system and guide them through its everyday use, although Cooper et al. (2014, p. 16) note that in practice this is difficult to achieve.

Cooper et al. (2014)'s use of the term *mental model* is more general and less rigorous than Philip Nicholas Johnson-Laird (1983)'s definition which is discussed later. Philip Nicholas Johnson-Laird (1983)'s definition relies on set theory and logical proofs, whereas Cooper et al. (2014) usage of the term acts as more of a shorthand for a collection of cognitive processes. In particular, the *Mental Models* theory from Philip Nicholas Johnson-Laird (1983) has three fundamental principles:

> First, each mental model represents what is common to a distinct set of possibilities. So, given an assertion, such as "It's raining or else it's snowing", you have two mental models to represent each of the two possibilities (on the assumption that both can't be true). Second, mental models are iconic, that is, their structure as far as possible corresponds to the structure of what they represent. So, an assertion such as, "All the artists are bakers", has a model representing the relation between the two sets of individuals. Third, mental models based on descriptions represent what is true at the expense of what is false. This principle of truth reduces the load that models place on working memory, but it can lead to predictable errors in reasoning (P. N. Johnson-Laird, 2013, p. 1).

Understanding how a system works sometimes speeds up people's ability to learn that system and retain knowledge about its operation, so providing coherent and useful model for users to internalise should help them learn faster [5](Ben-Ari and Yeshno, 2006; Kieras and Bovair, 1984). In various studies, when users are provided with explicit conceptual models of systems they rely less on trial and error and instead approach problem-solving more strategically and conceptually (Ben-Ari and Yeshno, 2006; Kieras and Bovair, 1984; Bhavnani, Reif, and John, 2001; Kirschner, John Sweller, and R. E. Clark, 2006). Similarly, experts, who are thought to have internal conceptual models based on their experience, are more likely to solve problems through the recognition and interpretation of patterns than by breaking information down into its constituent parts (Winn, 2004; Kirschner, John Sweller, and R. E. Clark, 2006).

5: Kieras and Bovair (1984) use the term *device model* instead of Norman (1983)'s *mental model* to distance themselves from the more specific term *mental model* from Philip Nicholas Johnson-Laird (1983)'s Model Theory of cognition. In Kieras and Bovair (1984) the *device model* is really the *implementation model* provided by the system's developers (or usability researchers) and the user's mental model is never explicitly constructed, only their performance is evaluated in terms of speed and information retention.

One possible explanation for the usefulness of models is Khemlani and P. N. Johnson-Laird (2013)'s theory, grounded in *Model Theory* (Philip Nicholas Johnson-Laird, 1983; Philip N. Johnson-Laird, 2004), that people try to create a revised explanation for their situation when faced with new and inconsistent facts because they mostly want to understand *why* an inconsistency happened instead of looking for ways to reject it outright. They found in their studies that individuals were quicker to resolve inconsistent details when they had a causal explanation for the situation than when presented with unconnected facts.

Implementation models appear to help move learners from random and possibly irrelevant internal models towards the more coherent and applicable models provided by the designers (Kieras and Bovair, 1984)(Ben-Ari and Yeshno, 2006, p. 1347). There is also evidence that providing users with accurate implementation models helps them to verbalise and communicate their intentions and usability issues. Accurate conceptual models give learners the language needed to discuss their design intentions with others, as opposed to giving more disconnected statements of their experience during the design (e.g. what they tried and what worked/didn't work), much in the same way that the CDNs help designers discuss their design decisions with one another using a common set of discursive tools.

In an I3DP system, the user-facing notation can be thought of as providing the main conceptual model of the system. Thus, as the systems notational terminology becomes more aligned to the *user's understanding* of the target system's terminology, or, in CDN terms, *consistent*, *role-expressive*, and *closely mapped* to the I3DP device model, the easier it should be for users to work with and talk about a system. We would expect them to make less errors, or at least to quickly recover from them because they can quickly form a plausible hypothesis of why the error occurred in the first place. In the user studies that follow in Chapter 7 (User Studies and Analysis), we used this observation to evaluate the effectiveness of the implementation model that we provided in the system itself and in the user workshops by analysing the ways that users articulated their design process and looking at where errors occurred and why.

5.5 Intuition and deliberation as modes of working

All the CDNs are relevant to designing the user's activities on an I3DP system, which incorporates a very rich range of user activities from graphical to textual to even physical manipulation. To focus the discussion, it was particularly helpful to think of groups of CDNs in terms of the speed and type of thinking that they best supported. Some activities, like livecoding performances in front of audiences, required as much speed as possible whilst still maintaining some visibility and intelligibility of performances for non-experts in the audience. Others, like private open-ended explorations of material properties, printer settings and possible forms conducted by the main researcher alone, allowed the user more time to reflect on alternative solutions to their design issues and to even take time to extend the system itself, but still within reasonable time periods and without creating too much confusing complexity in their work. It would be safe to say that livecoding is a more intuitive process that exists in the moment, whereas exploratory designing and making is slower and more deliberative in nature.

There are major differences between, as Kahneman (2011) puts it, "fast and slow thinking", otherwise known as the cognitive process of intuition and deliberation. Intuition is thought to rely on the quick retrieval of previous knowledge stored in people's long-term memory. It is a process that happens at a sub-conscious level and is understood to be a faster process than deliberation, or conscious thought. Deliberation relies on working memory, which is capable of problem-solving but limited in resources such as memory and processing bandwidth. Intuition is faster, but its speed comes at a cost: it is based on previously-learned heuristics, is slow to update, and has little to no access to conscious processes of thought and working memory. As such, intuitive thought processes are incapable of doing even simple arithmetic, such as counting (DeStefano and LeFevre, 2004; J. Sweller, 2003; P. N. Johnson-Laird, 2013).

There are competing theories about the nature and extent of these internal cognitive heuristics, but most cognitive scientists agree that there is some sort of internalised "'small-scale model' of external reality" Craik (1952, ch. 5) intrinsic to conscious beings, like humans (A. Clark, 2016; Philip N. Johnson-Laird, 2004; Philip Nicholas Johnson-Laird, 1983). In many predictive theories, these models are the ways that people internally simulate the world around them on a kinematic and conceptual level so they can act and make decisions (P. N. Johnson-Laird, 2013). Whatever the true nature of these models, the working memory of intuition has been found to be very limited, meaning that a person can cope with only one model at a time (Philip Nicholas Johnson-Laird, 1983, ch. 6)(Khemlani and P. N. Johnson-Laird, 2013).

Any attempt at switching models and looking at a situation "from different points of view" requires a more deliberate of thinking. Deliberation is a much more involved and slower cognitive process which is capable of recursion, searching for and applying alternative models, and more complex arithmetic including calculating probabilities (Kahneman, 2011). As P. N. Johnson-Laird (2013) puts it, "the distinction between intuition and deliberation is in computational power: intuitions are not recursive, but deliberations can be."

Selected challenges for personal fabrication

Challenge 2: Domain knowledge: the physical knowledge of materials, mechanical and structural engineering
Challenge 3: Visual feedback and interactivity: working interactively within the (currently) long fabrication times of 3D printing
Challenge 4: Machine-specific knowledge: understanding and controlling computerised manufacturing systems, including tooling options

Figure 5.2: Selected challenges to focus on in I3DP, from the six original challenges for personal fabrication by Baudisch and Mueller (2017).

5.6 Determining user activities

What the user is meant to be doing with the system provides the context for discussing the trade-offs between CDNs. Without them the question of which trade-off is "best" is meaningless. The relationships between pairs of CDNs only exist in terms of how they support these *user activities* (A. Blackwell, 2005, p. 7).

For the most part, *incrementation*, *modification*, and *searching* are all activities that are historically well-supported by the CodeMirror text editor, and so they were not specifically looked at in user tests, although they certainly figure into the intended use cases for the system.

Some of these activities use intuitive modes of thinking, others deliberative, and some are a back-and-forth mix of both. In any user session, a person is likely to switch back and forth between different activities as the task demands. Instead of looking at activities individually, we chose to create "activity clusters" that encapsulated a small collection of related activities into our intended use cases.

To further understand what user activities might be appropriate for an I3DP system, we looked to a sub-set of *the six challenges for personal fabrication* from Mueller, Im, et al. (2014). In particular, challenges 2, 3, and 4 were key influences on the philosophy, design and evaluation of such a system and the user workshops that are described in the following two chapters[6].

6: See also Subsection 3.9 (Challenges and opportunities facing 3D printing) for a fuller discussion of the Six Challenges.

For *Challenges 2 and 4: Domain and machine-specific knowledge,* we expected users to experiment and improve their understanding of how 3D printing works at a materials and machine level using a conceptual model explicitly defined by the notation as a guide, and in the process gain confidence in the 3D printing process.[7]

7: In our study, the higher-level machine and material concepts were user tested in Subsection 7.4.1 (Task 1: Context) and the more granular details of material handling and tooling was introduced in Subsection 7.4.5 (Task 5: Using LivePrinter (printing a square)) and Subsection 7.4.6 (Task 6: Using LivePrinter (retraction and material flow)) and continued through the rest of the study.

For *Challenge 3: Visual feedback and interactivity*, we looked at evaluating the user experience of visual feedback and interactivity through observations in workshops and higher-level discussions in some long-form interviews. A major consideration was evaluating cognitive trade-offs in the speed of programming the system versus the complexity and visibility of understanding what it was doing at any point in time. This challenge was complicated by the unavoidably long fabrication times for larger objects, noted by Baudisch and Mueller (2016).

This subset of three of the Six Challenges form a major part of the following activity clusters, keeping the focus on evaluating claims of increased transparency, interactivity and more immediate results in the livecoding 3D printing process, towards the goal of bringing more machine- and domain-specific knowledge into the 3D printing process (see Figure Figure 5.3 on page 68).

5.6.1 Activity Cluster: Experimental I3DP

It was helpful to think of what users might do in a solo or small group setting when they are investigating new forms, computational and artistic techniques, and materials. We called this cluster *Experimental I3DP*, based on it mainly incorporating the *exploratory understanding* activity in a combination of intuitive and deliberative thinking combining the intuitive mode of "on-the-fly" coding inherent in interactive programming with the more deliberate process of designing software beforehand. We would also expect users to spend time with *transcription* and *modification* activities when they are attempting to translate and modify algorithms or mathematical expressions, as in from published research, into code, and to a lesser extent all the other basic activities associated with text editing.

5.6.2 Activity Cluster: Exploratory I3DP

With more of a focus on quickly and playfully sketching ideas, *Exploratory I3DP* was thought of as supporting a mainly intuitive mode of working. We imagined participants working in a "thinking-through-making" mode (Schön, 1991) where the focus is on maintaining a "flow" of action, relying on the quicker activities of *incrementation*, *modification*, and *searching* for getting easy results without writing much code, and *transcription* activities of copying, modifying or translating an algorithm from a similar piece of code. Speed is still an issue to keep sessions short enough to be achievable, but less than with performance or experimenting. It was assumed to be a more forensic, reflective mode that placed visibility highest and left time for reflection between movements.

5.6.3 Activity Cluster: LiveCoding 3DP

The intended *livecoding 3DP* activity should be the quickest and most intuitive of the three. Users would be interactively programming in front of audiences, combining the CDN activities of *incrementation, modification, and searching* in the interests of speed and performative flow. Advanced practitioners might incorporate an exploratory design component looking towards new improvisations. Other secondary considerations for such performative "Experiences of Interaction" are echoed in Alan Blackwell (2015, p. 7).

Livecoding can be thought of as a practice that employs the technique of interactive programming: a particular application with its own particular patterns. These patterns of livecoding, including some typical activities and relations to the CDNs can be seen in detail in Alan Blackwell (2015) especially "Experiences of Creativity" ((Alan Blackwell, 2015, p. 8)) that is concerned with extending the system, redefining it, looking at it from different points of view, and "anything not forbidden is allowed" that lets the performer “revel in the glitch or crash, as a demonstration of fallibility or fragility of the machine”.

To support the exploratory activities and the spirit of livecoding performances, one of the key design principles for I3DP was that the user should be able to do *almost* anything that they want, no matter how dangerous[8] or unexpected, because of the focus on extending people’s knowledge of 3D printing – specifically, the domain-knowledge and machine-specific knowledge from the challenges above. This principle is explicitly articulated in Koskinen et al. (2012, p. 6)’s definition of the practice of *constructive design research*: “research that imagines and builds new things and describes and explains these constructions”.

8: Except in guided user workshops, where safety constraints were followed, such as limiting the position of the print head to inside the printer volume!

The first half of that definition, “research that imagines and builds new things” maps to the *exploratory design* activity of livecoding 3D printing. This activity focuses on unusual and untested 3D workflows, giving users the creative freedom to try new and unexpected operations in the pursuit of new forms, techniques, knowledge. Exploratory design activities were part of the user workshops, where users were given some basic techniques and then encouraged to use them for free-form sketching and form-finding, often with unexpected and unintended results like looping, stacked spirals and 2D line art.

The second half of the definition, research that “describes and explains these constructions”, maps to the *exploratory understanding* activity with more of a focus on taking apart existing examples, such as the “happy accidents” or selected sketches that come out of exploratory design sessions, trying to understand how they work. Our view is that this activity is more methodical, reflective, considered, and precise than the loose sketching typical to exploratory design. An example is the case study on “airprinting” as discussed in Section 8.5 (Airprinting), where this activity is applied to try and reverse-engineer a specific technique and then extend it fort new situations.

5.7 Mapping thinking modes to CDNs

Thinking in terms of general terms of intuition (faster, easy mental observations) versus deliberation (slower but not *too* slow, leaving time for comparisons and harder mental operations balanced with limiting confusion) lets us create clusters of the CDNs which are most relevant to each mode:

Intuition: viscosity, abstraction, premature commitment, consistency, diffuseness, error proneness, visibility, hard mental operations, role-expressiveness, secondary notation

Deliberation: abstraction, premature commitment, consistency, role-expressiveness, secondary notation, visibility, hard mental operations, provisionality, error-proneness, closeness of mapping, hidden dependencies

Each of these CDN clusters represents a complex, interdependent relationship of trade-offs where increasing the degree of one CDN may introduce changes in the others. To fully understanding how notation supports each way of working we will need to pick through each of these trade-offs and examine them in detail.

5.7.1 Design trade-offs in intuitive interactive programming

5.7.1.1 Abstraction and viscosity

Since intuitive working is meant to be as quick as possible, it should present the user with immediate options for getting started on complex tasks. This means a high-enough level of *abstraction* to make elements available for common tasks, implying an upper limit on the *viscosity* of the system.

Text-based languages are helpful here, especially the JavaScript that our system was ultimately built from, because they give the user a lot of freedom to redefine most of the aspects of the environment and even to write their own languages on top of it. JavaScript affords the user a great deal of abstraction mechanisms for general tasks, such as adding numbers and creating repetitive functions.

The trade-off is that defining new functions, terms and syntax for the I3DP-specific notation often means knowing something about how the functions are internally created and used, which calls for some deeper investigation. Even the main developer needed to check the source code files before extending our system's functionality, meaning that the viscosity of the system is pegged to the abstraction-level of most of the functional syntax of the system.

Beyond just providing abstractions for use on their own, a system should have some measure of *composability* (Roberts and Wakefield, 2018, p. 302) – mechanisms for fluidly and flexibly combining abstractions. In visual languages, this is usually achieved by virtual "patch cords" between abstractions. In the Haskell-based TidalCycles, the **$** operator arguably works in this way, helping users chain together functional abstractions into code sentences: `d1 $ fast 2 $ sound "Bleep"`

5.7.1.2 Premature commitment, abstraction and viscosity

Because they are assumed to have little access to working memory, the level of premature commitment should be low so that users can take action as they think of it, without having to remember whether they are following the proper sequence or not. The trade-off here is that one way of mitigating premature commitment is to break up larger operations into smaller ones (supporting *problem decomposition*), so that users have more flexibility and can get more immediate feedback (e.g. the system offers more *provisionality*). This puts pressure on the abstraction level to be higher, to support more granular and user-customised operations. In turn, this increases viscosity (both *knock on* and *repetition*) by having users type more explicit notation to get results.

5.7.1.3 Hard mental operations, premature commitment, abstraction

Hard mental operations should be avoided at all costs because they will switch the user into deliberative mode and carry a large cost in performance speed. Since a high *abstraction level* forces users to plan ahead and build their own abstractions before they can get started with basic operations, this puts pressure on the abstraction level to be lower, and thus balances the pressure to decrease *premature commitment* somewhat.

5.7.1.4 Secondary notation

Secondary notation may be helpful here as a form of externalised memory, where the user can leave comments explicitly telling them what they need to do (e.g. "Run this code!" or "Click here and type this: XXXX") or helping them remember experimental parameters and desired ranges for variables. Still, this must be kept to a minimum because too much secondary notation carries a cost in reading and understanding, as the *diffuseness* dimensions tells us.

5.7.1.5 Visibility

As discussed, *visibility* in the form of visual feedback for key elements is crucially important for beginners and learners of an I3DP system. It should be less so for experts who have a more developed mental model of the system, and can intuitively understand what is going on, when, or at least understand how to quickly query the system to find out.

Visibility comes at the expense of a considerable increase in the *diffuseness* text and graphical notational elements, which is an issue for an already information-dense, verbose text editor. Adding elements takes up precious screen space and increases the cognitive bandwidth for users to find and focus on them, or to be able to recognise and yet ignore them and focus on other elements.

More *abstractions* (e.g. more possible notational elements) can help increase visibility by bringing more elements and concepts to the forefront, which decreases viscosity by giving users more upfront choices of notation to pick from without creating new abstractions, but as a side effect this forces users to choose from longer lists and to decide which notational elements are appropriate before using them (e.g. a high level of *premature commitment*).

Additionally, in a livecoding performance mode where the interface is projected for an audience, the demands of the audience for maximum visibility (and legibility) are at odds with the user/performer who must concentrate on operating the system (Bruun, 2013; Roberts and Wakefield, 2018).

5.7.2 Design trade-offs in deliberate interactive programming

5.7.2.1 Abstraction, viscosity

The *abstraction* and *viscosity* dimensions are slightly inverted from intuitive working because the user has more time and working memory to reflect and will likely work on build their own new tools. Still, this system is designed for interactive fabrication, not as a general programming language, so users need enough abstractions to begin working quickly and to be "drawn in to play around" (Alan Blackwell and Fincher, 2010, p. 7).

5.7.2.2 Hard mental operations and provisionality

Users are also more likely to encounter some *hard mental operations* in their deliberative work – maybe they are working on a new 3D generative algorithm and need to stop and calculate the angles between movements or sketch out alternative forms in the middle of a working session. The system needs to support this kind of stop—start workflow through increased *provisionality*, letting users step through their code and pick up where they left off.

5.7.2.3 Secondary notation

In this mode of working, it is more likely that the user will have other sources of *secondary notation* at hand that would normally be disruptive when working intuitively "in the moment". They might refer to a sketch-book to make drawings, or use another piece of software to create 3D visualisations to test out an idea alongside working with the I3DP system. This constant context-shifting from notation systems might be helpful for working through *hard mental operations*, but it also might distract from the main task at hand.

5.7.3 Design trade-offs in both modes of working

5.7.3.1 Consistency, role-expressiveness, error proneness

Consistency is important so that users can make good guesses at notational elements without having to refer to the documentation mid-task. The elements should be easily visible and use similar forms of verbs, nouns and adjectives in naming so that users don't need to look up definitions and search through documentation, and can avoid having to stop and reflect on their purpose [9].

9: See Alan Blackwell and Fincher (2010, p. 6)'s "Experiences of Meaning" for a longer discussion of this topic.

Role-expressiveness fits alongside consistency in quick thinking and reducing *error-proneness*. It seems obvious that the quicker a user can recognise what a notation refers to, the quicker they can use it. Importantly, C. Hundhausen, Vatrapu, and Wingstrom (2003) found that, in a limited experiment looking at graphical and textual (pseudocode) translation sources, "the efficiency and accuracy of the translation task depend[ed] on the goodness of the match between (a) the descriptive notation of the algorithm, and (b) the target programming language."[10]

10: This is also discussed more fully as it relates to livecoding in Roberts and Wakefield (2018, p. 303).

5.8 Trade-offs between textual and visual programming

Why choose text as the primary for of notation for this interactive programming environment? Text represents a trade-off in abstractions level between allowing a fuller possibility of exploration and invention that textual languages support (e.g. creating on-the-fly entities of any conceivable type and redefining existing ones) with issues of speed caused by premature commitment in defining/redefining symbols and phrases, and a need to remember a number of symbols and syntax (i.e. hard mental operations). Thus, there is a balance between the premature commitment caused by the increased cognitive load of having to know and choose from (or invent!) functions, function arguments, and their syntax, rather than with visual programming where options appear on-screen and are easily clicked and connected. Or, in some cases, forms can be directly entered into the software by physical control or even drawing onto 3D surfaces and having them rendered by software (J. Jacobs et al., 2018; Peng et al., 2018; Willis et al., 2011).

Text also has a speed advantage over visual programming in that it is self-descriptive to read and understand at a glance, and easier to edit quickly. There is the potential of a lower-level understanding of what is happening (i.e. at a mechanical process level) when using a lower abstraction level with text that better represents the system at hand. When a balanced in abstraction level is struck, it can lead to less overall visual viscosity, as well as limiting knock on viscosity. It can also lead to a decrease in repetition viscosity because multiple operations can be aggregated and handled simultaneously, as opposed to graphical controls that require individual representations on screen and individual actions to control each one.

5.9 Conclusions

This chapter proposed three new "activity clusters", *Experimental I3DP*, *Exploratory I3DP*, and *LiveCoding 3DP* for helping designers articulate the particular user experience of working with an I3DP system. They can be seen as a response to some of the Six Challenges of Personal Fabrication for helping users of interactive fabrication systems, introduced by Baudisch and Mueller (2016). These activities exist on a cognitive spectrum ranging from mainly deliberative (Experimental I3DP) to mainly intuitive (Livecoding 3DP), with Exploratory I3DP somewhere in between, and represent a number of cognitive trade-offs between speedy intuition and longer deliberation in each unique mode of working.

Other researchers have developed related activities and design patterns for the user experience of interactive programming, especially as with Alan Blackwell and Fincher (2010)'s patterns and *Construction* activity, but none so far have been specifically related to interactive fabrication and live sculptural performance, each of which has a unique set of affordances, constraints, and contexts of use.

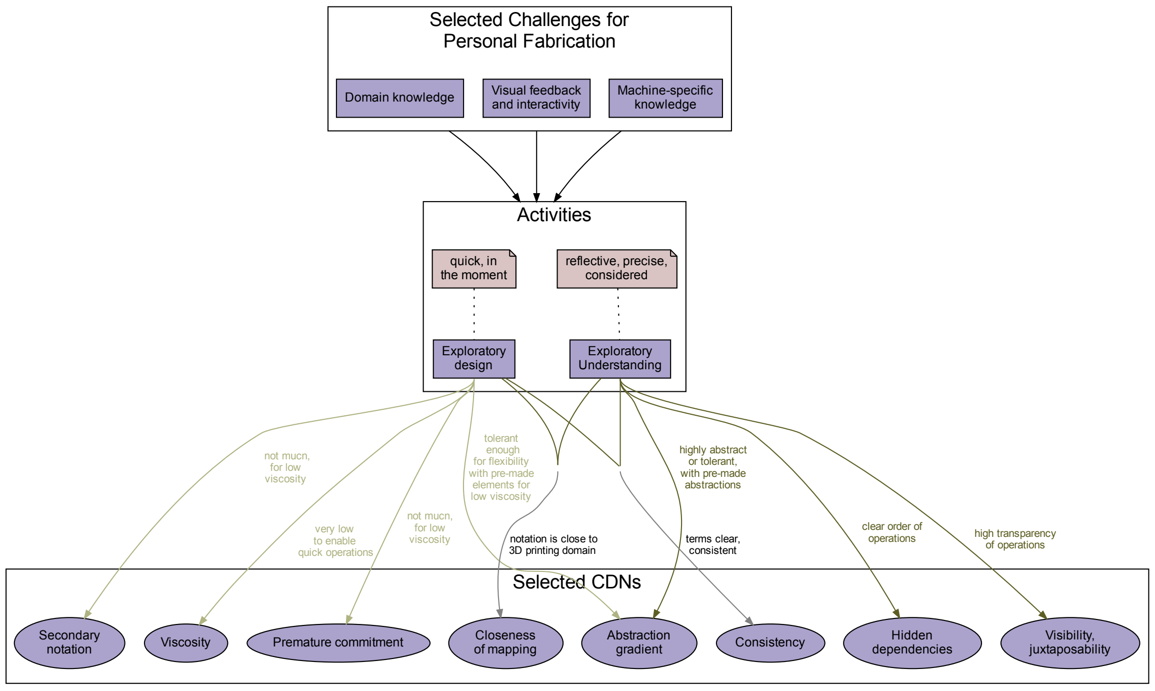

Selected Challenges for Personal Fabrication
Domain knowledge
Visual feedback and interactivity
Machine-specific knowledge
Activities
quick, in the moment
reflective, precise, considered
Exploratory design
Exploratory Understanding
not mucn, for low viscosity
very low to enable quick operations
not mucn, for low viscosity
tolerant enough for flexibility with pre-made elements for low viscosity
notation is close to 3D printing domain
highly abstract or tolerant, with pre-made abstractions
terms clear, consistent
clear order of operations
high transparency of operations
Selected CDNs
Secondary notation
Viscosity
Premature commitment
Closeness of mapping
Abstraction gradient
Consistency
Hidden dependencies
Visibility, juxtaposability

Figure 5.3: Mapping the selected Challenges of Personal Fabrication to design activites, with key trade-offs between the different CDNs that enable each activity.

Implementing an I3DP system: LivePrinter 6

6.1 About

This chapter details the design and implementation of LivePrinter, the Interactive 3D Printing (I3DP) software and hardware system used as a research vehicle for exploring the question of what a practice of live, interactively programmed 3D printing might look like in the future. To provide some context, we discuss some intended workflows and how they compare to more "traditional" 3D printing methods. These workflows frame an in-depth discussion of design decisions and software development activities, illustrating the inner workings of 3D printing hardware, firmware, and software, and describing in specific detail how the LivePrinter system acts as a conceptual wrapper around them.

6.2 Introducing the LivePrinter System

From here on, we will refer to the part of LivePrinter that users engage with when they are livecoding as the *LivePrinter System.* The *LivePrinter System* was initially developed as a working implementation of an I3DP system, to be used as a platform for general experimentation with new printed forms and for user-testing. Later, the realm of what it *should* do expanded somewhat after conversations and requests from 3D printer music composers and performers such as Ezra Teboul as discussed in Section 7.2 (Notes from Initial interviews). LivePrinter became an audiovisual performance platform that could control the sound of the printing through movement, using a hybrid syntax of musical and fabrication-specific terminology.

The LivePrinter System, as seen in Fig. 6.1, has a graphical user interface (GUI) running as a web application, written in JavaScript, HTML and CSS, and with a back-end web server written in Python that interfaces with a 3D printer via a hardware serial connection over USB. The backend is relatively small at around 1500 lines of code, whereas the front-end is a complex project at over 6000 lines of code across a number of files.

The front-end, running completely in a standard web browser, provides a livecoding editor, graphical representation of some printer properties, and some basic controls for managing the connection between the back-end and printer. The back-end receives commands from the front-end and forwards them to the printer (via a serial connection over USB), then interprets the printer responses and forwards them to the front-end. Communication between the front-end and back-end is completely asynchronous and solely in the form of AJAX-based commands and responses in the JSONRPC 2.0 format.

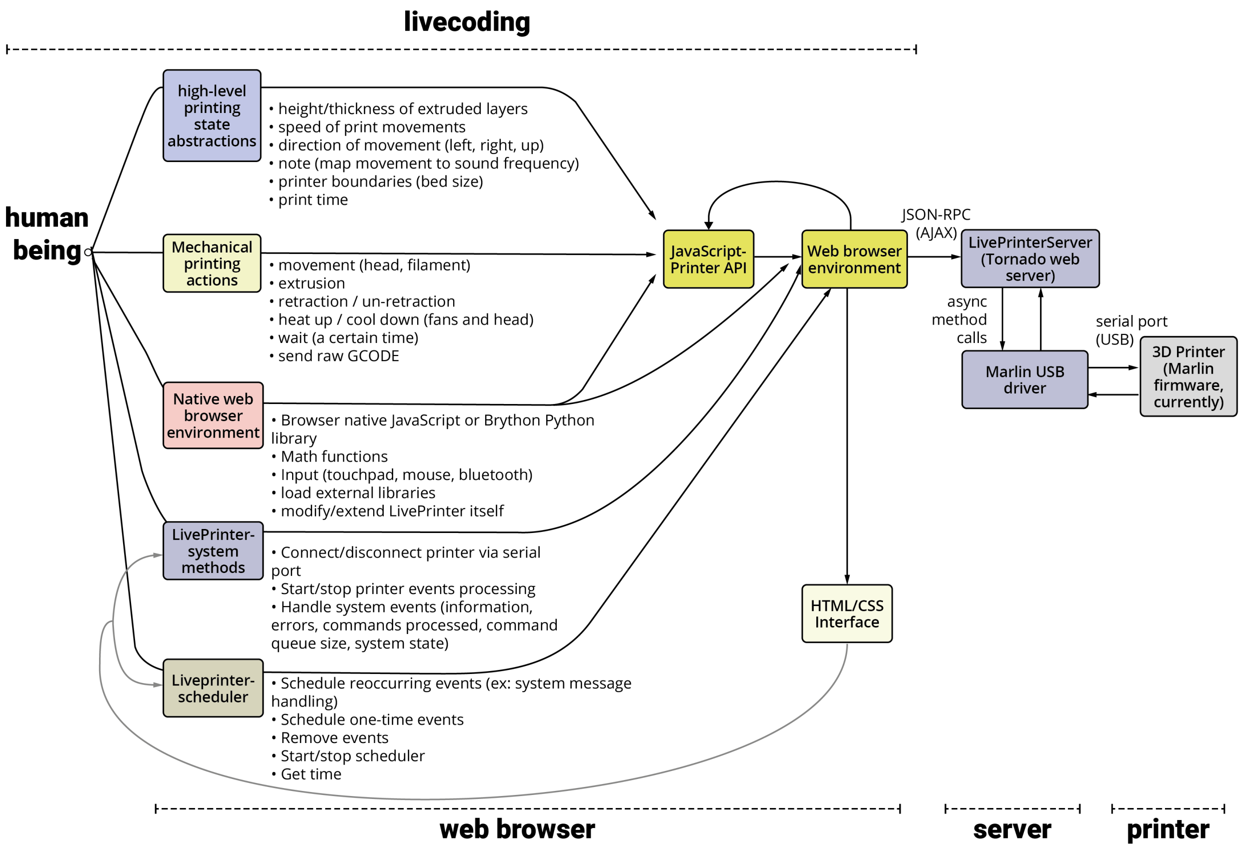

livecoding
human being
high-level printing state abstractions
• height/thickness of extruded layers
• speed of print movements
• direction of movement (left, right, up)
• note (map movement to sound frequency)
• printer boundaries (bed size)
• print time
Mechanical printing actions
• movement (head, filament)
• extrusion
• retraction / un-retraction
• heat up / cool down (fans and head)
• wait (a certain time)
• send raw GCODE
Native web browser environment
• Browser native JavaScript or Brython Python library
• Math functions
• Input (touchpad, mouse, bluetooth)
• load external libraries
• modify/extend LivePrinter itself
LivePrinter-system methods
• Connect/disconnect printer via serial port
• Start/stop printer events processing
• Handle system events (information, errors, commands processed, command queue size, system state)
Liveprinter-scheduler
• Schedule reoccurring events (ex: system message handling)
• Schedule one-time events
• Remove events
• Start/stop scheduler
• Get time
JavaScript-Printer API
Web browser environment
JSON-RPC (AJAX)
LivePrinterServer (Tornado web server)
async method calls
Marlin USB driver
serial port (USB)
3D Printer (Marlin firmware, currently)
HTML/CSS Interface
web browser
server
printer

Figure 6.1: An overview of the LivePrinter System, showing the main functional blocks provided by the system mapped to system services (e.g. web browser, server, 3D printer)

6.3 Design Goals

The design goals of the LivePrinter system came directly from the research questions at the start of this thesis, introduced in Section 2.1 (Interactive 3D Printing). From the beginning, LivePrinter aimed to be a concrete exploration of this open question of what computationally-augmented personal manufacturing might look and feel like in the future. Over the course of the project, these design goals for LivePrinter in particular coalesced into:

1. **Helping people understand how the process of manufacturing using 3D printers relates to their discipline** so they can start to experiment usefully with it (or not)
2. **To allow users to create physical forms using novel functions that take into account physical properties like speed and temperature**, instead of the usual method of beginning with 3D modelling and automating fabrication
3. **To explore how visual aesthetics and to a lesser extent, musical concepts can lead to knowledge about new digital manufacturing toolpaths and vice versa**: how 3D printing toolpaths can be influenced by concerns other than optimising for speed and strength.[1]
4. **Effectively using livecoding for 3D printing** by developing a livecoding environment for 3D printing with a usable text editor and conceptually-appropriate interface, syntax, and API

1: To an extent, this is already the case with different fill patterns, but these are usually hidden inside an object and thus invisible after the manufacturing process. They contribute little if anything to the final aesthetic of the finished artefact. More specifically, toolpaths can be influenced by the musical properties of running motors at tuned frequencies, which is explored in the chapter on Hilbert curves and techno music.

Many of these goals relate directly to the Six Challenges for personal fabrication laid out in Baudisch and Mueller (2017) and discussed in Subsection 3.9 (Challenges and opportunities facing 3D printing). Goal 2 refers to how users and further developers of the LivePrinter system might explore and extend the embedded *domain knowledge* referenced in challenge 2. Goals 3 and 4 are more complex and together reference a mix of all three challenges of *visual feedback and interactivity* for an iterative exploration of aesthetics coupled with a guided self-discovery of the properties, processes and mechanical possibilities of the 3D printer referenced in the challenges of both *machine-specific knowledge domain knowledge*.

6.3.1 The Different Parts of LivePrinter

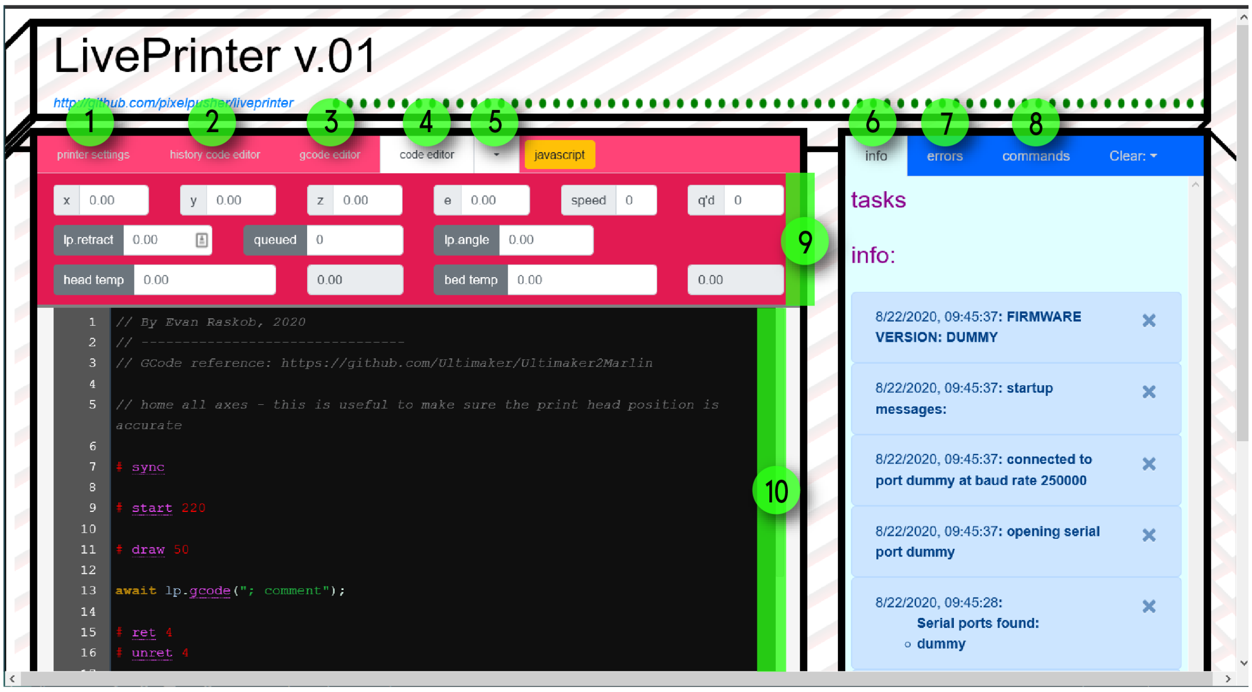
At the end of the project, LivePrinter could be described as a collection of related parts:

- A fully-functional, freely available, open source, modular, web-based editor for livecoding 3D printers written in JavaScript with a web server in Python
- An API for livecoding printers establishing a basic set of metaphors for printer functions and properties
- A "minigrammar" – a prototype programming language for programmatically describing 3D printing operations – secondary contribution

- Documentation aimed at users covering installation, 3D printer history and mechanics, use cases and step-by-step guides to the API and more advanced techniques
- Documentation aimed at future LivePrinter developers
- Records of experiments demonstrating novel uses and techniques that livecoding printers makes possible (performances, artefacts for installation, demonstration videos)

The technical system (essentially, the livecoding environment and supporting software and hardware) established a practice-led research vehicle for testing out ideas and integrating results and to reflect on at key points in the process. The project's documentation captured fundamental concepts of digital manufacturing and served as a dialogue between the designer and users, especially during research sessions. It also grounded the project in the history of augmented manufacturing in general and 3D printing in specific. Finally, both the documentation and experiments illustrated the possibilities of augmentation, both creative and mundane.

6.4 The LivePrinter Graphical Interface



```
// By Evan Raskob, 2020
// --------------------------------
// GCode reference: https://github.com/Ultimaker/Ultimaker2Marlin

// home all axes - this is useful to make sure the print head position is accurate

# sync

# start 220

# draw 50

await lp.gcode("; comment");

# ret 4
# unret 4
```

```
8/22/2020, 09:45:37: FIRMWARE VERSION: DUMMY
8/22/2020, 09:45:37: startup messages:
8/22/2020, 09:45:37: connected to port dummy at baud rate 250000
8/22/2020, 09:45:37: opening serial port dummy
8/22/2020, 09:45:28:
    Serial ports found:
    ○ dummy
```

Figure 6.2: The main GUI of LivePrinter, realised in HTML and JavaScript. It incorporates an embedded, modified CodeMirror (`http://codemirror.org`) code editor with bespoke syntax highlighting.

Livecoding, or "on-the-fly" coding, is generally described as a practice of alternating between writing, editing and running *snippets* of code (often in a performative setting) versus coding that is written as a whole program, and then run non-performatively (Wang, 2008; N. Collins et al., 2003). LivePrinter's graphical interface supports modes of livecoding workflow such as line-by-line or partial-line editing and running of code by providing the aforementioned text editor, based on the popular

CodeMirror package. It also provides different ways of viewing the results of running code through server- and client-side logging.

From Fig. 6.2:

1. Printer communications and general LivePrinter settings such as live logging of command histories
2. History Code Editor: All code that has been executed is appended to the bottom of this editor, providing a history of the session
3. GCode Editor: A history of all GCode that has been sent to the server, usually compiled from code run in other editors
4. Code Editor: the main textual livecoding editor allowing line-by-line editing and partial and asynchronous, whole- and partial-line code compiling and running
5. Examples: This lists the example provided as a part of LivePrinter to help novice users
6. Info: General informational messages and occasionally errors from the front-end and server.
7. Errors: error messages received from the server
8. Commands: a real-time view of asynchronous communication between the front-end and back end, exposing the GCode commands sent and their responses from the server
9. Visual feedback of key physical and virtual printer model properties, for quick access
10. The main livecoding editor where users can edit and run code. Any code run here is copied to the **History Code Editor** (2) and compiled into GCode and appended to the **GCode Editor** (3).

6.5 3D Printing software: under the hood

If livecoding 3D printing was likened to storytelling, then the whole 3D printing machine would be the context or situation in which the story takes place, and the protagonist or main character that the narrative follows over time would be the *tool*, or print head. The concept of a precisely controlled fabrication *tool* using computational means is key to the idea of "traditional" Computer Numerical Controlled (CNC) fabrication. There might be other actuators attached to the system that can be computer-controlled, like heaters, but the most common archetype of CNC machines is that of a computer-controlled *tool* moving at speed, cutting, extruding or otherwise shaping a material. Laser cutters, CNC routers, and CNC vinyl and paper cutters are all common examples of machines built around that general concept.

Every CNC machine has at least one tool, and in 3D printing this tool usually takes the form of one or more printing heads*, each containing one or more *hot ends*. This is the element that can be moved in x, y, z (respectively left/right, back/front, up/down from the perspective of

* There are add-ons for 3D printers that add pen plotting, cutters, and other tools as well.

a user facing the printer) directions, heats up, and extrudes the melted material out of one or more of its ends as the filament feeds into them. The filament is driven by the rotation of a filament-pushing-and-pulling motor in that is referred to moving in the *e* axis, with positive coordinates refer to extruding, and negative to rewinding.

6.6 Modes of tool movement

The 3D printing tool has two main *modes*: moving or *travelling*, and printing or *extruding*. A *travel move* is a tool operation that moves the print head *without using the filament motor*, so no material is extruded or rewound. Travel moves are mostly for moving the print head into a new drawing or printing position, and are performed at relatively high speeds. Likewise, a *printing* or *extrusion* operation uses the filament motor to extrude or rewind material, and possibly moves the tool head along a particular path at the same time.

These two modes are codified in the low-level instructions, or "Marlin-flavoured GCode"[2] that the 3D printing firmware expects to receive. GCode is internally translated by the firmware into physical movements across three mechanical axes (x, y, z) and a filament motor axis (e). These physical movements can be complex and involve acceleration or deceleration control at a firmware level. Adding to this complexity, the stepper motors in the printer are digitally controlled, and thus turn precisely according to digital (on/off) pulses determined by the driver controller microchips. Controller chips vary in the digital resolution of their operations and so different chip models can affect the smoothness, fidelity and even noise level of the printing process.

2: Named as such because of its derivation from the open source firmware project called Marlin: `https://marlinfw.org`

The printer normally expects to receive these commands from a file loaded into its SD card slot, or streamed via a serial connection (e.g. USB) to its main controlled board. An overview of how a livecoded instruction cascades down through this system, finally causing a physical printer movement, is shown in Figure 6.3.

6.6.1 Basic travel

The following are some typical use cases for travelling operations, derived from user observations in guided workshops (see last chapter) and self-reflection during this project:

1. moving the print head into position to start printing
2. moving out of the way after finishing a print, or during a series of printing operations to see the current results
3. moving in order to make a sound

These can be consolidated into three more general operations:

1. moving the print head from the current position to a specific location (x, y, z) in the print cavity
2. moving the print head a specific relative amount from the current position in the x, y, z direction (e.g. 10 mm to the right, 2 mm up)

3. moving the print head a specific relative amount from the current position and at a specific 2D angle, possibly also with an angle of elevation (i.e. either upwards or downwards)

LivePrinter's syntax for travelling comes from these three cases, with the first one mapping to the function `moveto()` and the second to the function `move()`, as demonstrated in the previous chapter. These travel moves are often done at high speed, and have a higher maximum speed than extrusion moves because the filament is more brittle and can't be moved as quickly without physically grinding off part of it. This is important to note, because LivePrinter will automatically choose either the currently set travel speed or slower printing speed based on the inferred printer operation (see Figure 6.5).

The third case maps to the more complex *chain* of functions or a sort-of *method cascading* using the LivePrinter minigrammar to specify the direction and amount to move, with the function `go()` at the end or `travel()`[3] (see Subsection 7.4.5 (Task 5: Using LivePrinter (printing a square)) for a discussion of user testing this, pre-minigrammar). Later experiments with the LivePrinter minigrammar allowed users to write briefer code sentences like `# travelspeed 80 | turn 45 | travel 20` or a more explicit but functionally-equivalent `# travelspeed 80 | turn 45 | dist 20 | go`. In this example, the travel speed is first set to 80 mm/s, the virtual heading of the print head is rotated clockwise by 45 degrees, and then moved at the travel speed by 20 mm.

3: Note: `travel(distance)` was added after the user workshops to be more conceptually-transparent, and is simply a syntactical shorthand for `dist( distance).go(0)`

Both `move()` and `moveto()` were general purpose functions, originally designed to take Cartesian coordinates as arguments. `move()` was later extended to take more cylindrical angle/direction relative coordinates in response to positive participant feedback from user tests during the initial development (see Subsection 7.4.4 (Task 4: Using LivePrinter (basics))). However, in later, less-formal single-user tests conducted in summer 2021, some participants found the angle/direction arguments confusing because they had a similar effect as the single-argument functions like `travel()` but without the possibility of a single argument.

One important caveat of `move()` and `moveto()` is that they implicitly update the current direction of travel used by `travel()` and `draw()` (discussed in the next section). Thus, a relative x, y, z movement of $(12, 8, 16)$ sets the current 2D movement direction to the x, y portion of the unit vector:

$$\frac{12}{\sqrt{12^2+8^2+16^2}}\hat{x} + \frac{8}{\sqrt{12^2+8^2+16^2}}\hat{y} + \frac{16}{\sqrt{12^2+8^2+16^2}}\hat{z}$$

or a horizontal heading of about 56.3° (i.e. from the back of the printer, towards the front and the user).

6.6.2 travelling upwards and downwards

There is also the possibility of movement in the z or upwards direction. Two quick functions are provided for convenient movement predicated on a common use cases of quickly moving the print head upwards at the end of a printed 2D layer to start the next successive layer, or moving it away at the end of a print, or moving the head down into position to start a print. Following the naming conventions of the rest of the functions, these are called:

```
// Move up or down by a vertical amount in millimeters
up(AMOUNT_IN_MM);
down(AMOUNT_IN_MM);

// Move to a specific location in the z direction.
// These two functions are synonyms for one another.
upto(HEIGHT_IN_MM);
downto(HEIGHT_IN_MM);
```

There is also a concept of moving at an angle. This upwards or downwards angle of movement, as measured between the x and z axes, is called the *elevation* in LivePrinter and set and returned using the `elevation(angle)` or `elev(angle)` functions.

The elevation *does not* update when calling the `move()` and `moveto()` along with the 2D heading because movements that continue in an upwards direction were found to be uncommon in reflective and user testing, where mainly users draw 2D layers, moving vertically upwards only when those layers are complete.

Additionally, it was not clear how to integrate more advanced concepts of full 3D spatial movement, with rotation, tumbling, and orientation into this mixed 2D/3D frame-or-reference. This points to future opportunities to develop conceptual systems for working more explicitly with 3D coordinate systems and 3D printing movements, whilst still retaining a layer-by-layer approach.

6.6.3 The minigrammar, asynchronous operations and global printer state

As mentioned, in later versions of LivePrinter a "minigrammar" or prototypical syntax designed for I3DP systems, was created using method cascading. The minigrammar was inspired in part by the brevity and clarity of the Tidal livecoding system, which was written in Haskell. This minigrammar removed most of the punctuation and symbols and hid the asynchronous syntax and functionality from the user so they could concentrate on specifying shapes and operations. For example, in the asynchronous version, **await lp.angle(45).distΔ.go(1)**, under method chaining, would have set the distance to move the print head to Δ and then extruded from the current position and at an angle of 45 degrees over that distance. Under "sort-of method cascading" by using the minigrammar syntax, this sentence becomes the more declarative **# turnto 45 | draw** Δ.

We called this minigrammar "a sort-of *method cascading*" because the "`|`" operators acts as a form of *syntactic sugar* that uses a combination of a grammar and regular expressions[4] to translate any code in between them into JavaScript functions with arguments immediately following them. In the minigrammar, these functions and arguments are often specified using object key and value pairs, such as "`functionName key: value`". What is invisible to the user, in this case, is that a single JavaScript object called "`lp`" represents the 3D printer in terms of its functionality and its current state. Thus, the code `# go 1` is nearly translated to the JavaScript `lp.go(1)`.

4: See the "liveprinter.ne" file, written in the nearleyJS format (`https://nearley.js.org`), at `https://github.com/pixelpusher/liveprinter/tree/master/js/language`

We write “nearly,” because there is an additional complexity to this system. User-facing printer operations are nearly all *asynchronous* in nature, meaning that they are called in the client-side web browser, which then triggers operations in the printer, over a network and serial connection, and then waits for the result of that operation, which will likely take a non-trivial amount of time to complete. For example, the actual transpilation of the minigrammar statement:

```
# draw 30 | retract 4 | up 50

```

is the JavaScript:

```
  await lp.draw(30); await lp.retract(4); await lp.up(50)

```

The minigrammar saves time and potential typos in typing out these more verbose commands, and also hides some of this complexity from less advanced users who might not grasp the nature of asynchronous code. Without it, the user would need to have a deeper understanding of what is involved in all the printing operations, and which are necessarily asynchronous.

Also, by scheduling this asynchronous code in an asynchronous queuing system, LivePrinter guarantees that operations will still complete in the proper order without a need for worrying about handling Promises or other asynchronous constructs manually. This does not prevent advanced users from explicitly using asynchronous code in the editor and creating their own bespoke asynchonous events and handlers, and as a further benefit it guarantees that these functions and all successive lines of code executed in the livecoding editor are executed in order, no matter what.

This being JavaScript, the “`lp`” global object is mutable and thus represents a potential global namespace for persistent functions and variables, especially ones that extend the printer object’s capabilities and might later be incorporated into its code base. These variables are often described as “polluting the global namespace” and are often frowned upon by experienced programmers for creating a proliferation of non-hierarchical or conceptually-organised variables and functions, but in livecoding, where the focus is on rapid experimentation and brevity of code, they can prove useful. Sometimes users might want to set a persistent variable or define a new function that lasts more than one operation.

To support this, LivePrinter provides a `global` keyword for defining more “traditional” global variables bound the called `window` object of the web browser, and accessible simply by name. For example, the code `global foo = 10;` transpiles to `window.foo = 10`, which isn’t much less verbose, but importantly does not require foreknowledge of the unique historical quirks that led to the `window` object being used as a global variable namespace in JavaScript.

6.6.4 Basic extrusion

The following are some typical use cases for extrusion operations, also called *printing* or *drawing* operations, derived from user observations in guided workshops (see last chapter) and self-reflection during this project:

1. Extruding filament to build up plastic forms on the print bed (e.g. building up a form 2D layer-by-layer, or in a single extruded blob), or in the air (like WirePrint)
2. Reversing (retracting) filament to prevent leakage or perform maintenance like changing the filament
3. Forwarding filament in the tube until it is inside the print head and ready for further printing operations

These can be consolidated into four more general operations:

1. Extruding a line from the current position to a specific location (x,y,z) in the print cavity
2. Extruding a line with the length of a specific amount from the current position in the x,y,z direction (e.g. 10mm to the right, 2mm up)
3. Extruding a line with the length of a specific amount from the current position and at a specific 2D angle, possibly also with an angle of elevation (i.e. either upwards or downwards)
4. Reversing the filament backwards into the tube and then changing it, or cleaning the head. Then, forwarding it until it is primed in the print head

The first operation maps to the function `extrudeto()`, with the suffix "to" again used similarly to `moveto()` as a shorthand for the user's intention to move to a specific location in the print cavity. Likewise, the second operation maps to `extrude()`, which is exactly the same as the aforementioned `move()` except that it extrudes plastic as it moves.

The third case again maps to the more complex *chain* of functions or a sort-of *method cascading* using the LivePrinter minigrammar to specify the direction and amount to extrude in, with the function `go()` at the end or `draw()`[5] Here, "draw" was chosen over "print" to emphasise the humanness of the operation and build on the metaphor or drawing or painting with plastic into space that worked well in user tests (see Subsection 7.4.5.1 (Feedback and responses)).

5: Note: As with `travel()`, the function `draw(distance)` was added after the user workshops to be more conceptually transparent, and is simply a syntactical shorthand for `dist(distance).go(1)` .

Similar to the travel functions, the LivePrinter minigrammar allowed users to write briefer code sentences like `# drawspeed 12 | turn 45 | draw 20` or a more explicit but functionally-equivalent `# drawspeed 12 | turn 45 | dist 20 | go 1`. Here, it becomes more obvious that the argument to the `go()` function tells the program whether to extrude (if **`true`**) or not (**`false`**), which can be shortened to `1` or `0` in JavaScript. In this example, the printing (or "drawing") speed is first set to 12 mm/s, the virtual heading of the print head is rotated clockwise by 45 degrees, and then the print head moves 20mm whilst extruding plastic.

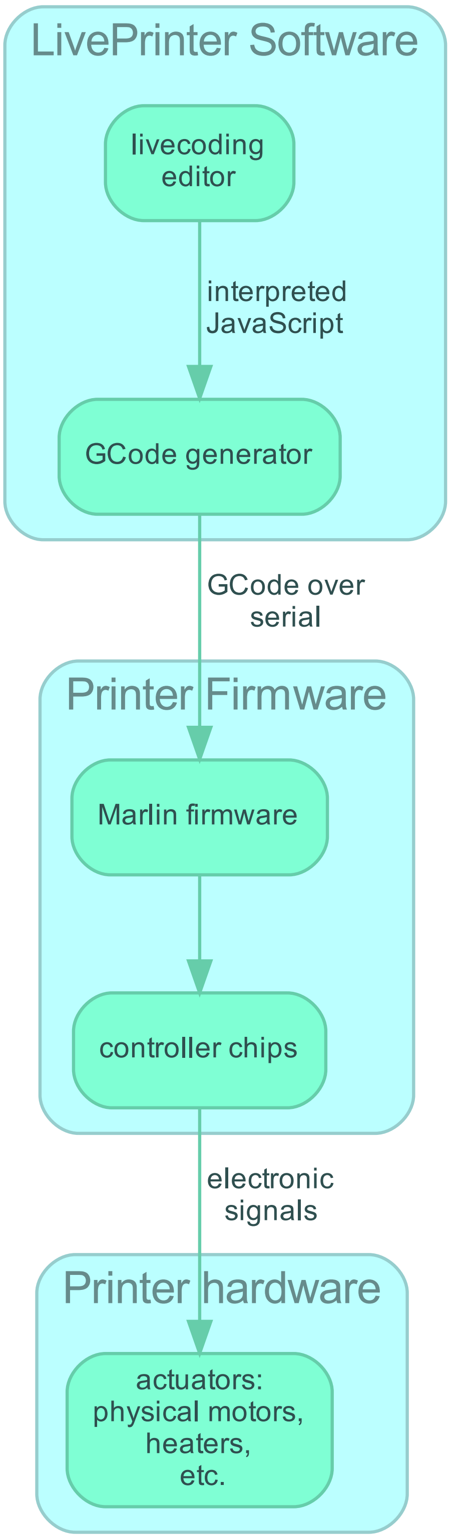


Figure 6.3: A flow chart illustrating how a LivePrinter JavaScript API function (e.g. `move(); draw(); retract();`) is translated into GCode, sent to Marlin firmware, and then turned into physical printer movements and actuation. Note that this is illustrating only one direction of this process, from user to printer hardware, for clarity; the actual system has feedback loops in each sub-system to keep track of internal states and for error-handling.

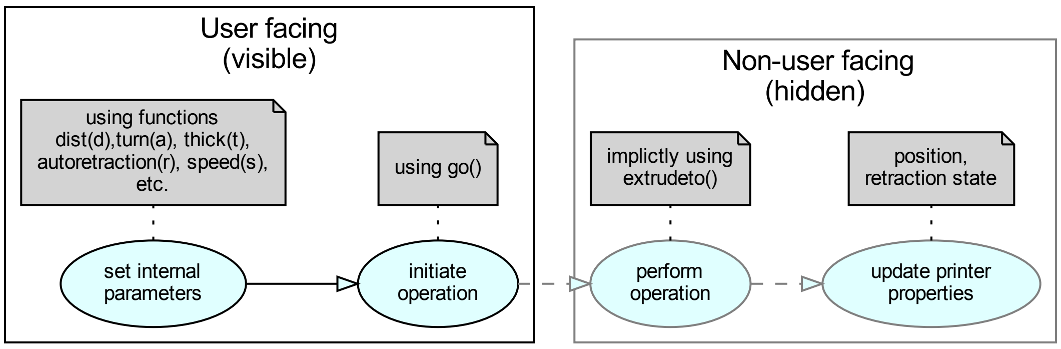

User facing
(visible)
using functions dist(d),turn(a), thick(t), autoretraction(r), speed(s), etc.
using go()
set internal parameters
initiate operation
Non-user facing
(hidden)
implictly using extrudeto()
position, retraction state
perform operation
update printer properties

Figure 6.4: A simplified overview of how method chaining and cascading look to the user, with hidden (internal) methods exposed. These are explained in more detail in other the diagrams in this chapter that explain the inner workings of the movement and printing functions.

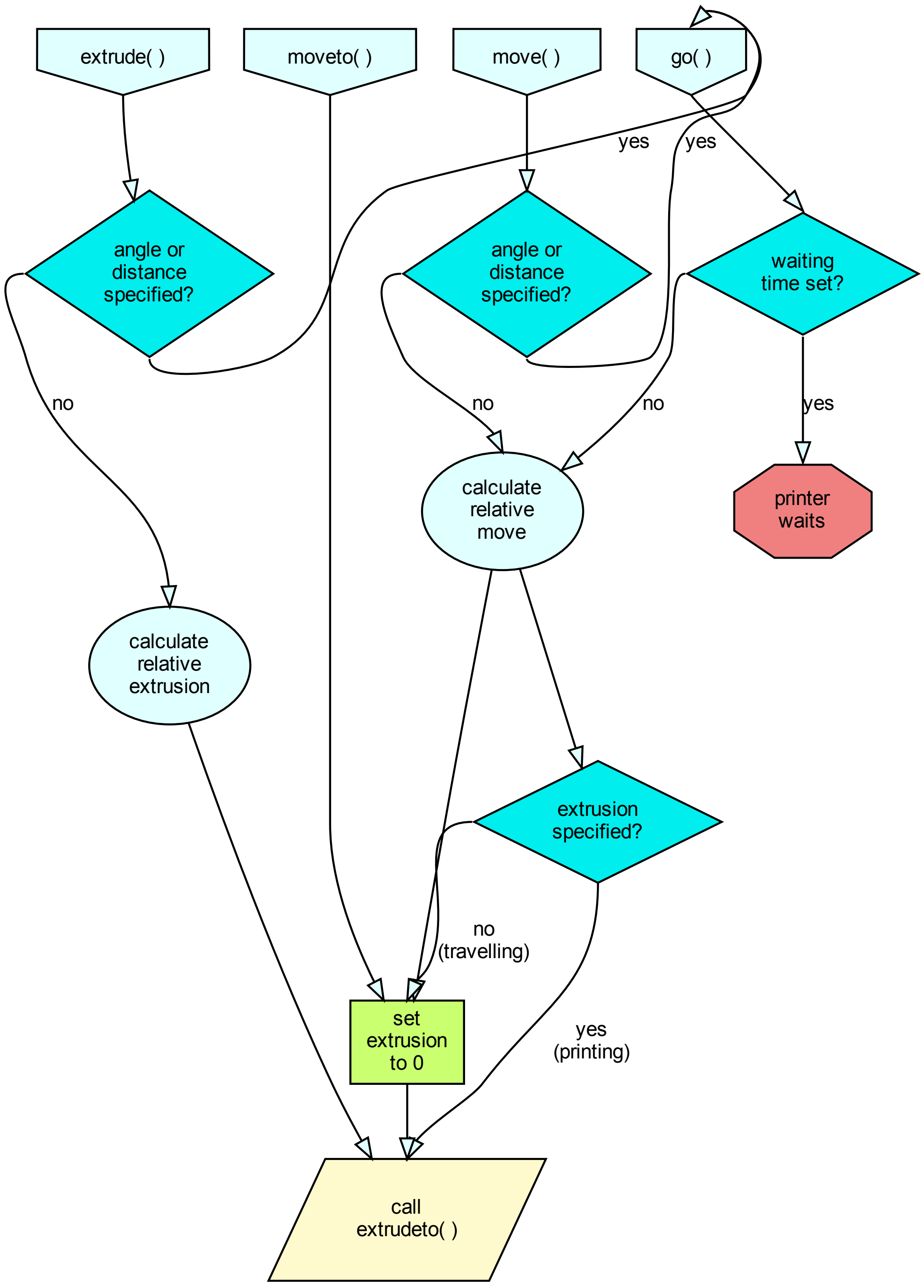


Figure 6.5: LivePrinter provides four main printing operations *move()*, *moveto()*, *extrude()* and *extrudeto()*, of which the first three map to a parameterised version of the fourth to consolidate the movement and extrusion logic into a single function. Another printing operation, *go()*, is used during either method chaining or cascading and also maps to a version of *extrudeto()* for the same reason.

6.7 A shared architecture

Figure 6.6: Relative, absolute, materials-handling and chained operations

Relative operation functions

1. move
2. extrude
3. up
4. down

Absolute operations

1. moveto
2. extrudeto
3. upto
4. downto

Chained operations (a subset)

1. travel
2. draw
3. drawdown
4. drawup
5. turn
6. turnto
7. thick
8. speed
9. elevation

Material handling operations

1. retract
2. unretract

Other operations

1. wait

By now it might seem obvious that the various travel and extrusion functions in LivePrinter have much in common, to the point where they are often near-synonyms for each other. Part of the reason for this shared architecture has much to do with the firmware of the printers themselves. Many CNC devices use the GCode command `G0` for pure travel movements and the command `G1` for extrusion or other tool operations, such as activating a cutting laser in a CNC laser cutter[†]. However, many 3D printers simply use the two commands interchangeably[‡]. That is one reason why LivePrinter uses the common function `extrudeto()` to turn every movement operation into a fully-formed `G1` command, as illustrated in Figure 6.5.

Another reason for the shared architecture was to consolidate similar movement and avoid duplicating similar code across functions, which can cause a maintenance problem when code in one function is updated and not in others, as can happen with rapidly-developing experimental code. This follows from the “don’t repeat yourself” principle of programming from Hunt and Thomas (1999, pp. 320–978). Using this principle makes it relatively easy to create new and different synonyms for functions to try out with users with little development cost because difficult code that handles complex use cases is already written.

Looking at the implementation of the core movement and extrusions function in Figure 6.5, we see that in LivePrinter the `go()` function consolidated all of the angle/elevation/distance calculations in the relative movement functions `move()` and `extrude()` and then passed on the absolute coordinates, extrusion length (if any), and any other parameters like whether-or-not to use retraction to the `extrudeto()` function where most of the movement logic and GCode generation is handled. This leaves relatively little for the `moveto()` function to handle, other than making sure that the amount of extrusion is set to 0 and then handing off to `extrudeto()` as well.

From this diagram, we can see that there are 3 main cases that the system takes into account to determine the type of operation:

1. extrusion is specified and set to 0 (travel)
2. extrusion is specified and not 0 (manual mode)
3. no extrusion is set (automatic printing mode)

In the first case, the extrusion coordinate (e) is set to the current filament position, resulting in a pure travel move with no filament movement. The retraction state stays the same – if the filament is retracted, it stay retracted, and if it is not retracted it might drip from the print head and create a string as it moves.

In the second case, the user wants to take explicit control of the filament’s movement. This is an advanced operation, because usually the filament’s new position is calculated by a combination of the LivePrinter software and the Marlin firmware. This operation also doesn’t trigger a retraction or unretraction, because the user is assumed to want complete control. It

[†] See https://marlinfw.org/docs/gcode/G000-G001.html
[‡] as is the case with the Ultimaker 2: https://github.com/Ultimaker/Ultimaker2Marlin

does, however, override the internal retraction counter so that after the operation the filament is not marked as retracted. That means that if the filament was retracted slightly before this extrusion operation is initiated then *it moves from its retracted position* and *does not* unretract first.

In the third, automatic mode, the user allows the system to control the retraction handling and volume of filament extruded, based on the intended *layerheight* specified by the *thick* property. If the printer is retracted, it is unretracted before the operation begins, and it retracts automatically at the end.

LivePrinter calculates the volume of an idealised cylinder of extruded plastic in mm^3 using the distance of the movement, desired layer height and layer width (usually the same):

$v_{filament} = d_{xyz} * 2 * lh * \pi$

where $v_{filament}$ is the volume of the filament in mm^3, d_{xyz} is the distance to travel in the x, y, z direction, and lh is the desired layer height (often 0.15mm) for each printed layer.

The Marlin firmware uses that result combined with its setting for the filament's radius to internally convert this number into a linear coordinate for the filament motor:

$l_{filament} = \frac{v_{filament}}{2\pi * r_{filament}}$

where $l_{filament}$ is the length of the filament to extrude, in mm, and $r_{filament}$ is the radius of the filament (2.85mm for an Ultimaker model, or often 1.75mm for other major types).

There is also a fourth case, which is a modification of the last automatic filament-handling case but without automatic retraction handling. The user can let the system automatically determine the amount of material to extrude, again based on the intended layer height (or thickness), but *mostly*without retraction handling.

This was one of the most complex use cases and was developed from observations during the main user workshops, as discussed in detail in Subsection 7.4.6 (Task 6: Using LivePrinter (retraction and material flow)). The main usage scenario for this case is when a user intends to draw a continuous series of lines that will make up a larger form, like a vertical stack of squares that will make up an open-top cube. In this case, the user wants manual control of when and where the retraction and unretraction happen, which usually means an unretraction at the start of the operation to prime the filament for drawing and then a retraction at the end to prevent leakage onto the printed shape. The `autoretract(true/false)` function is provided for easily turning the automatic retraction off before or during a series of such operations. This is outlined in Figure 6.7.

6.7.1 Waiting

Another important mode of operation for 3D printers is simply to do nothing for a specific amount of time. There are two key use cases for this non-operation:

1. waiting for the extruded plastic to cool after a print move

2. in a musical sense, waiting a few “beats” as part of a melody

LivePrinter provides a function aptly named `wait(time)` where `time` is the number of milliseconds to wait for. This function was specially developed as part of the musical livecoding performances with the system which required pauses in between the musical “notes” played by changing the printer travelling and drawing speeds so that movements made sounds at roughly musical pitches. It also proved useful with experiments in airprinting, where the printer draws lines of plastic in the air, supported only by their own surface tension after cooling. Such operations start with drawing a line on the printer bed, then drawing upwards at an angle of elevation with a pause to let the filament cool and harden before continuing downwards and completing the shape, as described in the next section.

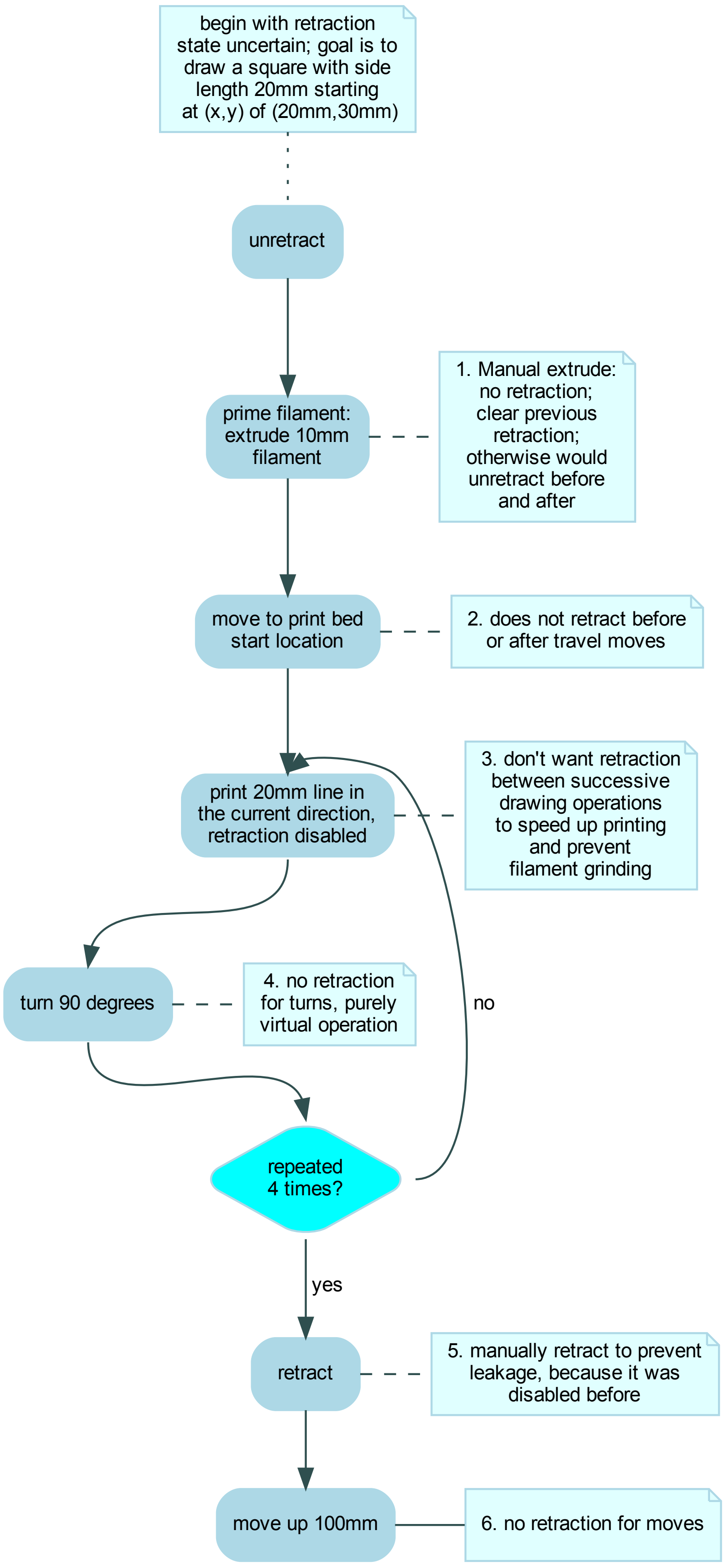

begin with retraction state uncertain; goal is to draw a square with side length 20mm starting at (x,y) of (20mm,30mm)
unretract
prime filament: extrude 10mm filament
1. Manual extrude: no retraction; clear previous retraction; otherwise would unretract before and after
move to print bed start location
2. does not retract before or after travel moves
print 20mm line in the current direction, retraction disabled
3. don't want retraction between successive drawing operations to speed up printing and prevent filament grinding
turn 90 degrees
4. no retraction for turns, purely virtual operation
no
repeated 4 times?
yes
retract
5. manually retract to prevent leakage, because it was disabled before
move up 100mm
6. no retraction for moves

Figure 6.7: Retraction use case for a 2D square

6.8 Issues of hidden dependencies

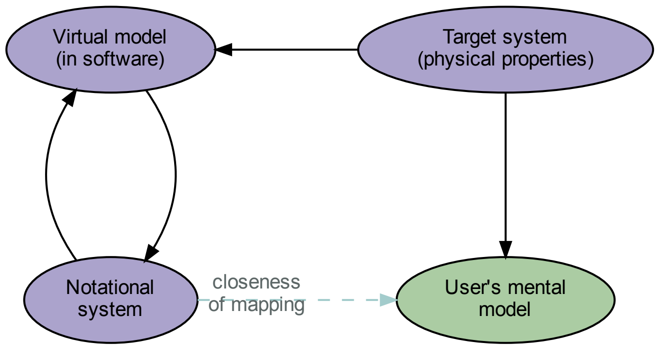


Figure 6.8: The CDN term *Closeness of Mapping* refers to the distance between the notational system (e.g. "Programming language semantics") in this case mostly the language syntax and to a lesser degree the GUI display, and the user's mental model, both of which are informed by their interactions with the exposed properties of an underlying *target system*. The notational system itself influences the software's underlying virtual model of the system, but within the limitations of the frameworks and data structures chosen to implement that model. Because language and symbolic representation is always imprecise, to a degree, there will always be a difference between how the user understands the problem world and how the system describes the problem world to the user through its notation.

In the CDNs, *hidden dependencies* are defined as "relationships between elements that are obscured or invisible to the user." With so much of the 3D printing process happening inside the printer, away from the screen, it was difficult to establish and then maintain a relationship between different screen-based and physical processes, as well as virtual-only processes that were often happening asynchronously.

For example, the original version of LivePrinter used WebSockets for realtime, persistent communications across the front end (HTML page) and the threaded python web server in the backend which communicated with the 3D printer via serial. This model proved difficult to work with, because of python's inherently single-threaded model and the difficulty of matching printer commands from the web client to server responses on the back end across websockets and serial.

Later versions, after the initial user testing, switched to a more conceptually-transparent asynchronous model, using asynchronous channels on the client side over asynchronous AJAX calls to the backend server which were handled over an asynchronous serial connection to the printer. This system was much quicker, because of the elimination of possible thread race conditions, and easier to debug because of the lack of potential thread memory collisions which sometimes were not caught by python exceptions nor the logging system.

The result was a change to the underlying implementation of the LivePrinter API that explicitly made most of the functions asynchronous because they involved communicating with the printer. These often involved print head movements, extrusions, and temperature changes. Only a few operations remained instantaneous, mainly setting virtual properties such as travel angle and speed, both of which are only finalised when they are sent to the printer as part of a travel or print operation.

Whilst understanding this distinction was useful for developing the system, it was cumbersome to livecoding it because of the added some difficult syntax needed to specify asynchronous functions, such as the *async* keyword, and even more complex code structures necessary to handle asynchronous results properly. The need for brevity and readability led to the development of the LivePrinter minigrammar as both a prototype language and a "syntactic sugar" for folding the verbosity and complexity of asynchronous operations into simpler textual statements.

6.8.1 Hidden dependencies in retraction

In previous versions of LivePrinter used in the initial user workshops, there was no `autoretract`. Successive versions experimented with a spectrum of completely automatic retraction to completely manual retraction, each with their associated problems. It was clear from user observations that completely automatic retraction caused problems when users wanted to extrude filament without moving the print head because the printer would unexpectedly retract at the end, leaving them with a shorter-than-expected extrusion operation at the end. For completely manual operations, users often were observed attempting to start drawing a new form without first unretracting the filament, which meant repeating the entire operation and manually checking the filament position and left them frustrated and confused as to why nothing had happened. This happened despite the retraction display area in the top "dashboard" of the GUI.

Finally, semi-automatic operations that attempted to guess whether to unretract before a printing operations and retract at the end unless explicitly specified resulted in a number of logical errors in the code, and general confusion with users concerning edge cases. At first it seemed logical that the system should *always* unretract automatically before a printing operation to prime the filament, but that created *hidden dependencies* and exceptions to the general rule of retraction that confused some participants. It was much simpler and more in line with the transparent and user-empowered philosophy of the system to give users the control over automatic retraction, or provide a full manual mode so they could experiment as needed. There are certainly compelling cases where complex manual retraction is useful, including with "airprinting" as discussed in Section 8.5 (Airprinting) where some trials used different retraction speeds and lengths within the same chain of printer operations.

6.8.2 Automation and hidden dependencies

To give the user space to create unusual workflows, the system was designed *not* to use any form of prediction or automation, at least not initially, and to make the system transparent and its state as visible as possible so that the user has a clear picture of what is going on at all times. The system provided a basic code-based workflow for common 3D printing operations with some basic attempts at "automatic" materials handling, but at each state always allowing the user to override as many aspects of the 3D printing process as possible as illustrated in Figure 6.4.

For example, using a combination of functions and function parameters users can specify the exact x, y, z, e coordinates to move the print head and filament to (as discussed); on a per-operation basis, whether to enable retraction/unretraction, change the layer height (often called *thickness*), and set the speed of the current printing or travel operation and for all subsequent operations. These can either be specified as a chain of functions in pure JavaScript, or a cascaded "sentence" of operations in the LivePrinter minigrammar:

```
# drawspeed 12 | thick 0.2 | autoretract 0 | turnto 90 | dist 20
    | go 1 0

# drawspeed 12 | thick 0.2 | autoretract 0 | turnto 90 |
  unretract | draw 40 | retract
```

6.8.3 Push/pull user feedback

One possible way of mitigating problems associated with increasing visibility is to have users actively request feedback in some form, possibly as a textual output in an event log, as opposed to automatically displaying it on some kind of heads-up-display or dashboard. Interestingly, Christopher Hundhausen and Brown (2007) finds that users still make the same amount of errors (e.g. less) when they get it automatically versus when they request it, so the strategy for LivePrinter was to start with as few on-screen displays of information as possible at the start, coupled with mechanisms for quickly printing out feedback as per user request in the form of textual logging functions. Over time, more elements were added, but the optimal number and their onscreen configuration remains an open question as of this writing.

6.9 Conclusions

This chapter detailed the implementation of the LivePrinter system as a fully-functioning proof of concept for an I3DP system. Having reflected on this process, there are a number of opportunities for improvement and future research, including the difficulty of making the process of material handling visible and understandable to users during printing; describing printer movements in ways that are more consistent and clear to users; working with asynchronous operations, especially when there are many of them; and finding the right balance of providing on-screen information and allowing users to focus on the act of making without cognitively overloading them during a time-sensitive process.

It is highly likely that, in the future, this implementation will evolve further; this is the nature of experimental work, and this was an early experiment with a narrow focus on specifically using livecoding for 3D printing at the exclusion of other forms of interaction. Still, we hope that the current rationale and reflections on the inner workings of the system will have some future value, as long as 3D printers continue to employ the similar mechanisms, materials physics and modes of interaction. They may even catalyse some necessary changes in the design of printers themselves.

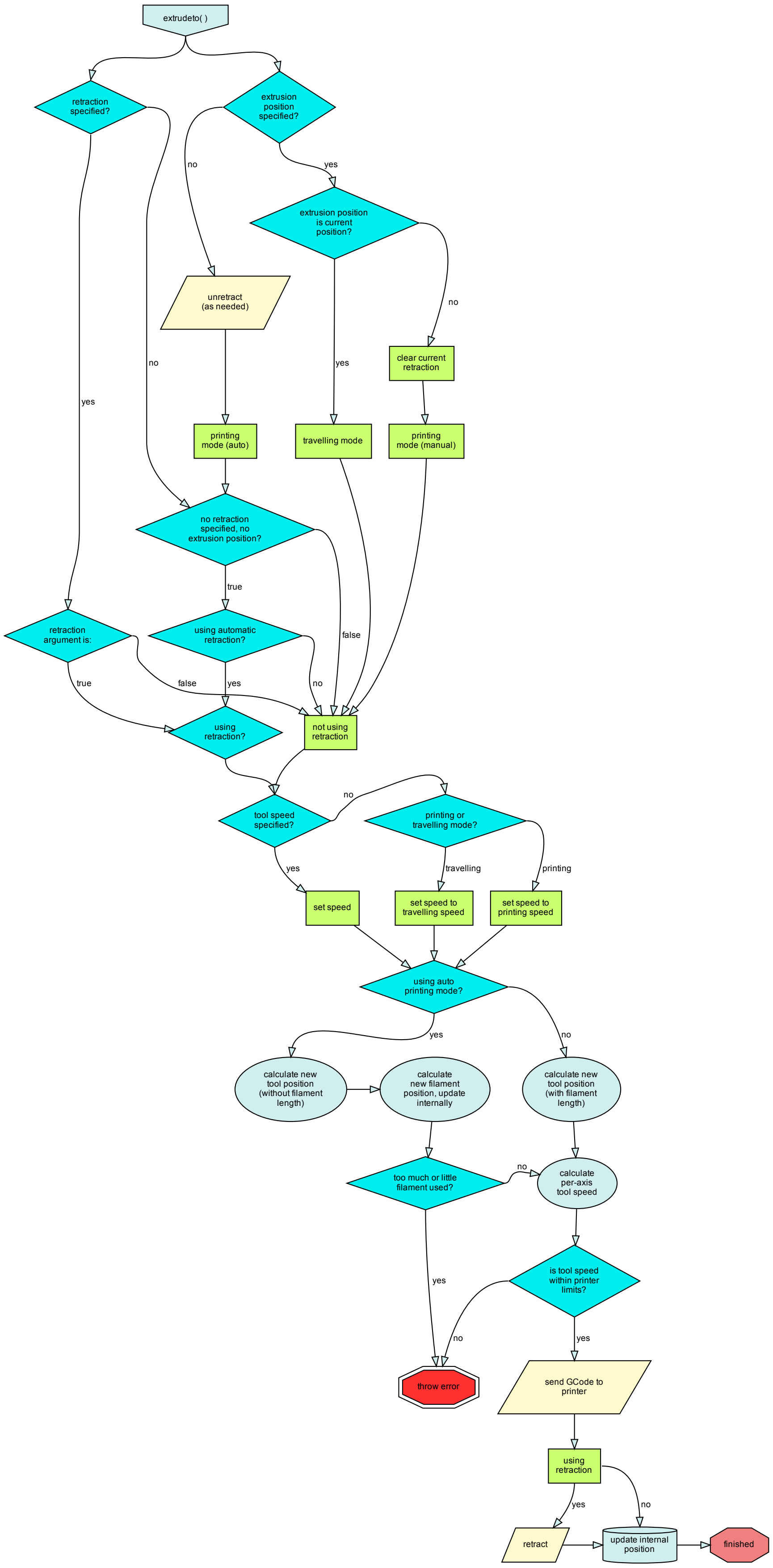

extrudeto( )
retraction specified?
extrusion position specified?
no
yes
extrusion position is current position?
unretract (as needed)
no
no
yes
clear current retraction
yes
printing mode (auto)
travelling mode
printing mode (manual)
no retraction specified, no extrusion position?
true
false
retraction argument is:
using automatic retraction?
true
false
yes
no
using retraction?
not using retraction
no
tool speed specified?
printing or travelling mode?
yes
travelling
printing
set speed
set speed to travelling speed
set speed to printing speed
using auto printing mode?
yes
no
calculate new tool position (without filament length)
calculate new filament position, update internally
calculate new tool position (with filament length)
too much or little filament used?
no
calculate per-axis tool speed
yes
is tool speed within printer limits?
no
yes
throw error
send GCode to printer
using retraction
yes
no
retract
update internal position
finished

Figure 6.9: All printing operations share a common function, `extrudeto()`, which is visualised here. This function decides which of the 3 main operations was intended (travel move; manual extrusion; automatic extrusion) and what parameters to use for that operation, how to handle retraction and unretraction, along with calculating the amount of filament to use and performing basic checks that the operation is within the printer's specified limits of print volume and speed. Lastly, it is responsible for sending the GCode to the printer and bubbling any errors up the chain of calling functions.

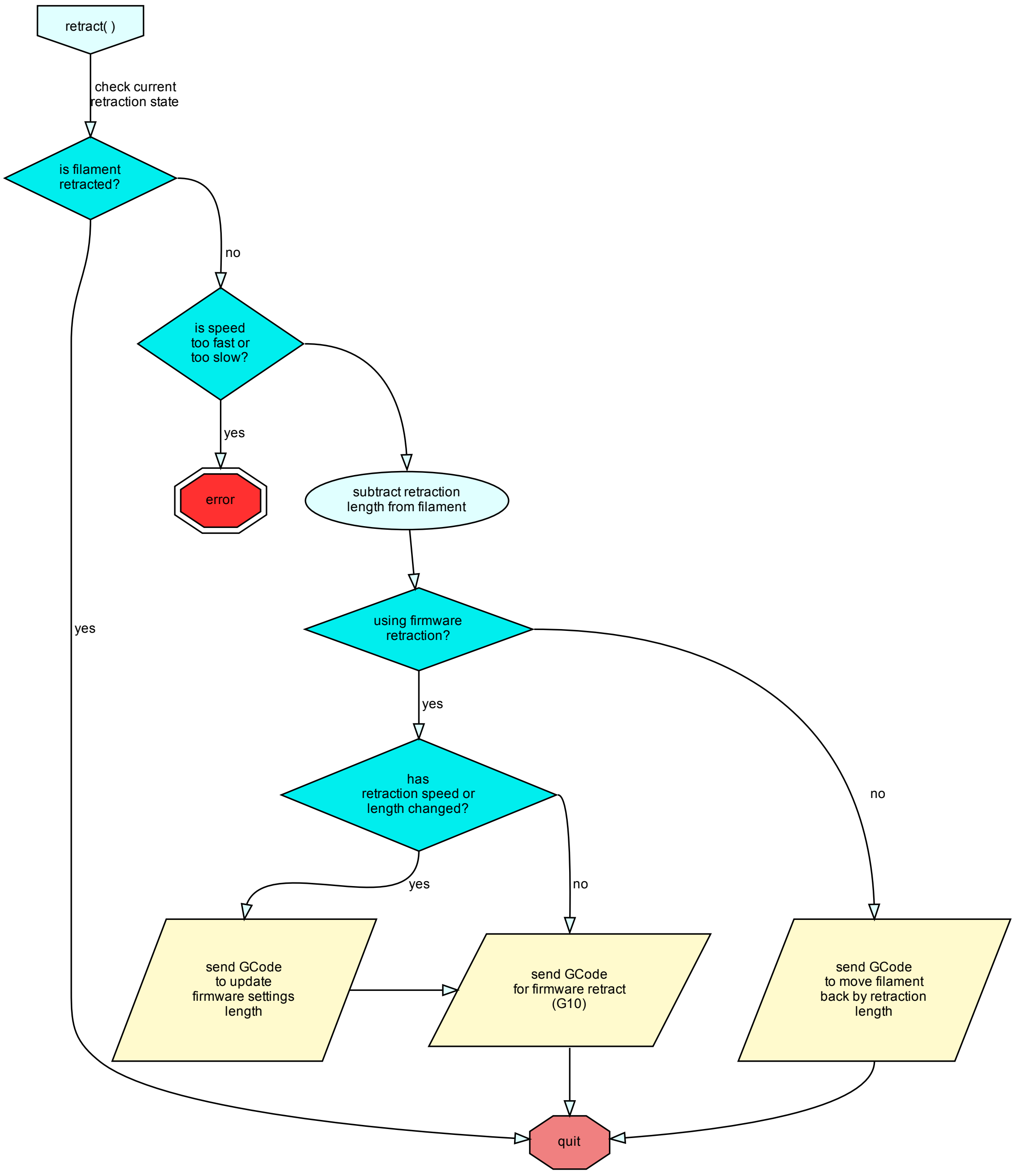

retract( )
check current retraction state
is filament retracted?
no
yes
is speed too fast or too slow?
yes
error
subtract retraction length from filament
using firmware retraction?
yes
no
has retraction speed or length changed?
yes
no
send GCode to update firmware settings length
send GCode for firmware retract (G10)
send GCode to move filament back by retraction length
quit

Figure 6.10: The "retraction" process of LivePrinter, as implemented in the API function retract().

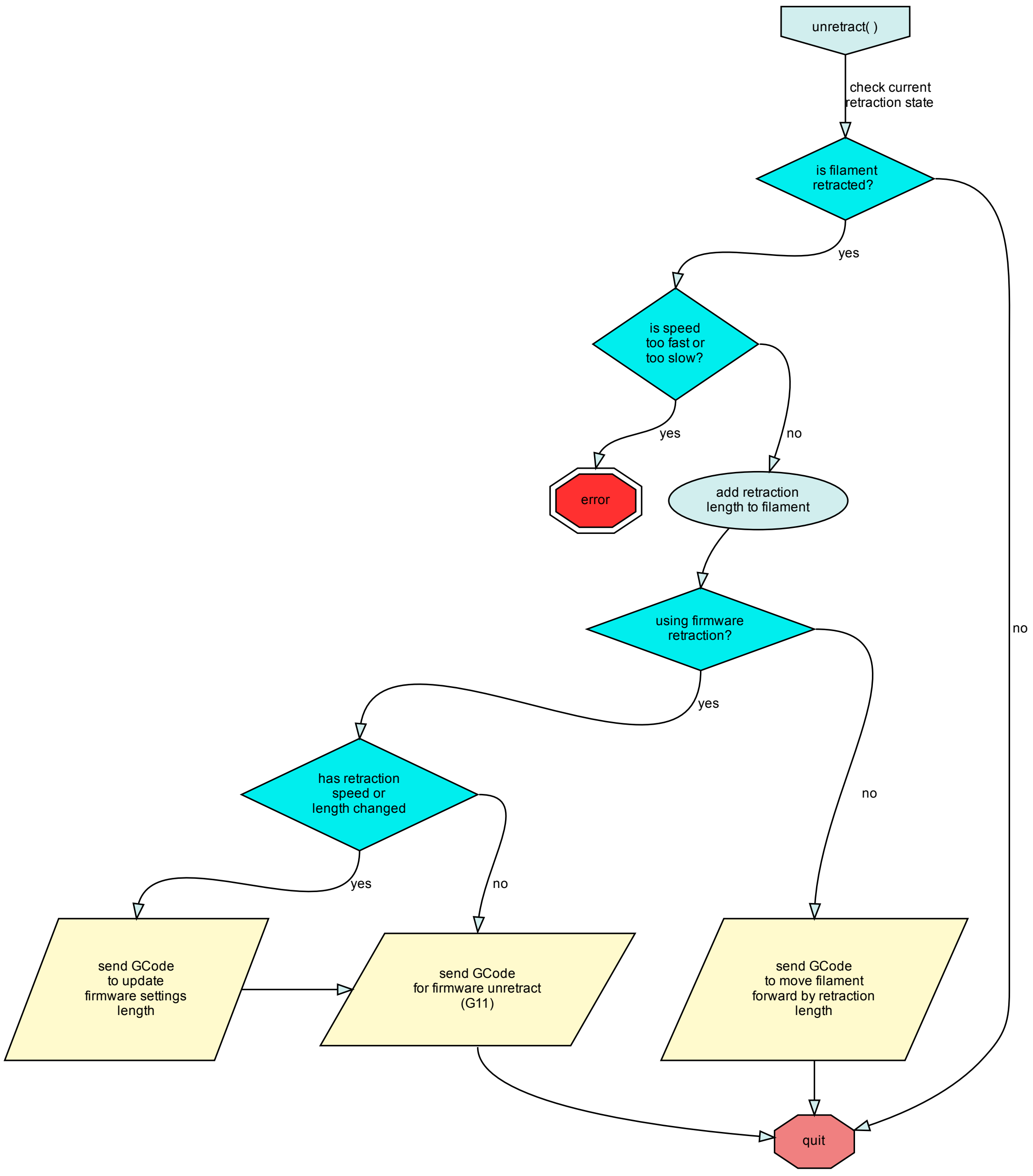

unretract( )
check current retraction state
is filament retracted?
yes
no
is speed too fast or too slow?
yes
no
error
add retraction length to filament
using firmware retraction?
yes
no
has retraction speed or length changed
yes
no
send GCode to update firmware settings length
send GCode for firmware unretract (G11)
send GCode to move filament forward by retraction length
quit

Figure 6.11: The “unretraction” process of LivePrinter, as implemented in the API function unretract().

Analysing the Design of LivePrinter through user studies 7

7.1 Introduction

This chapter views the insights gained from the process of testing, refining and reflecting on the LivePrinter system, starting with the initial interviews that helped shape the early development of the LivePrinter system. These interviews helped collect some other perspectives on 3D printing practice, use, and future applications, to help give the LivePrinter development some direction.

Then, a series of user study workshops examined the usability and usefulness of the system. The workshops were built around a series of tasks that mapped to project goals, and which are each discussed in detail in the following sections, along with feedback from tests, reflections from researchers, and evaluations of the results.

The full list of user study workshops covered in this chapter (and which are also listed in Subsection 4.2.1 (User studies undertaken)) are:

- **2017–8:** Initial exploratory interviews with 3D printing end users and practitioners
- **2019 Jan. 9–10, Goldsmiths:** 4 workshop sessions on LivePrinter (2 per day), approx. 24 participants of mixed ages, technical literacy and professional backgrounds
- **2019 Jan. 16, ICLC at MediaLab Prado, Madrid**: workshop session with about 8 livecoders attending the conference
- **2019 Feb. 19, Goldsmiths**: workshop session with 6 MA/MFA Computational Art students with experience of physical computing
- **2019 June 3, Brooklyn Research, Brooklyn**: workshop session with 7 participants with backgrounds in music production, art, or hardware/software development

Participants were self-selected by asking them to sign up for workshops based on the user descriptions from the LivePrinter GitHub page (at `https://github.com/pixelpusher/liveprinter`) and also the Eventbrite sign-up page in Section A.3 (User workshops and public demonstrations)):

> LivePrinter is an open source system for live, immediate drawing and fabrication with 3D printers. It's particularly useful for:
>
> - Textile artists who want to print onto fabrics and make new shapes and textures; for artists who want to use a printer like a 3D plotter and draw new forms
> - Product and industrial designers who want to understand more about how 3D printing works and fine-tune their materials and tool paths
> - Materials scientists who want to study 3D printing materials in more controlled, repeatable ways Computational and computer artists, either looking for new

tools or making generative works Educators who teach fabrication
- HackSpace and MakerSpace staff who need more tools to fine-tune their machines

LivePrinter combines the design and 3D printing of objects into a single, iterative process. Livecoding is used to control manufacturing precisely and procedurally with real-time flexibility. It brings improvisation and intuition to the act of making new computational forms. The system extends digital printing and CNC machining into the realm of performance and also has potential in design and science pedagogy.

The current software toolchains of 3D printing place the artist and designer at a difficult distance from the physical process of making. There is little space for live improvisation and experimentation, especially with key properties that directly affect printing materials, like temperature and print speeds.

In all, the results of the workshop were quite positive, although there were certainly a number of issues that came up, and a few unfortunate bugs identified during live sessions. The results of the feedback sheets demonstrated that a majority of the participants completed all the tasks and reported that they understood the intentions of the system and enjoyed the process of learning about it:

- 262 individual pieces of task-specific feedback
- 129 pieces of written feedback
- 195 (74.4%) pieces of task-specific feedback rating the task 4 or 5 stars (on a scale of 0 to 5 where 5 was "liked this part", 0 meant it as not filled in, and 1 was "disliked")
- 60 "positive" pieces of written feedback (for positive experiences such as compliments, remarks about it being easy, clear, helpful etc., excitement about using the system, and interest in using the system in the future)

Taken together, this data helped gauge the effectiveness of the LivePrinter System in blending the representational aspects of machine-manufacturing, e.g. the precision control of tool position and distance over time, with performative and reflective modes of working that rely more on exploratory, iterative processes that vary speed, direction and time over a number of experiments.

7.2 Notes from Initial interviews

The project was initially shaped by a series of loosely-structured interviews with selected digital manufacturing practitioners, taking place before the core of the software was written. The interviews were casual in nature, and provided some initial direction towards the goals of the user experience for a new livecoding/3D printing system, specifically what sort of form-making and exploration it should support and at what level of expertise.

Some interviews were recorded in their entirety, others were recorded in written notes from the researchers and reviewed with interviewees afterwards to make sure that they accurately reflected their thoughts. Interviews were later transcribed using what Mayring (2014, p. 45) calls the "selective" protocol to focus on the topics more relevant to any 3D printing that used code as a medium of creative exploration, DIY 3D printer construction, the act of 3D printing itself, and issues around users' perception of the capabilities and purposes of 3D printing.

Notable interviewees were found in the principal researcher's extended networks. From Hangar, an arts organisation based in Barcelona, Spain and other organisations nearby came: Belen Soto, Ali Yerdel, Patricio Rivera from collective TMTMTM (The Machine That Makes The Machine); technical staff at FabLab Barcelona; through the livecoding community came Ezra Teboul, a researcher and performer sometimes using 3D printers for musical compositions; through an alumni network for New York University's ITP programme came a developer at an open source CNC wire-bending firm (who wishes to remain anonymous) that I will call Beth, and David Lobser, digital artist; through colleagues at the Royal College of Art came James Lamb and Tim Rundle, two independent product designers teaching on the Royal College of Art's MA Design Products programme.

Mostly, these initial interviews helped illustrate the different levels of expectation for people using 3D printing. Some, like Belen, expected to be using 3D printing to quickly make complex objects, explore social or artistic themes like feminism through making more personal objects not for wider sale, repairing broken objects, or to generally gain independence from large manufacturers. They were less concerned with the process of making than the results.

The same could be said for Tim and James. They used 3D as part of a more established design process to test out the manufacturability of their prototypes in development. 3D printing could potentially save them from making expensive mistakes in final mass-production, as well as allowing them to experiment with the feasibility of form-making at an earlier stage and at a lower cost. 3D printing was a helpful addition to their usual development process and a step towards making finished, industrial products, but not an end in itself.

Finally, experimental artists like Patricio and David, and also software developer Beth, were more interested with the mechanics of the machine and experimenting with the underlying code. Beth did this as part of her job, writing software for a CNC wire-bending machine. Out of all interviewees, Beth had the most detailed grasp on usability issues because of her work designing user-facing interactive coding tools for creating shapes with the CNC bender and running workshops with participants of all ages.

Beth's experience provided some helpful early direction when she discussed the difficulty of a user's perspective when they attempted to take manual control of a semi-automated manufacturing process. Specifically, the way the wire-bender worked to twist wire back and forth across small wheels was hard to describe in the text of code. Beth likened the way of guiding the wire through the machine to "driving a car", a phrase she used with school-children in workshops. To her this was a potentially interesting avenue of further research, but, as an employee of a commercial company, she had to focus not only on the control of the machine but on making it create useful objects.

David, by comparison, had much more freedom to experiment with process. His work as an animator and working artist freed his work from having to be practical or useful in nature. His *Vessels* series was an exploration of tools that created free-form shapes from G-Code, the machine-level code that described shape-making for most CNC machines. Interestingly, as an artist coming from the digital world he was less concerned with shape and more concerned with the "texture" of his physical works.

Patricio was the most knowledgeable of the group, due to his hours of experience with electronics and building different types of 3D printers. For Patricio, form-making was arguably as important as the process of making. His experience as a software developer and generative artist gave him the technical knowledge necessary to understand and even extend the software that made 3D printers function. He built CNC machines like 3D printers and using 3D printer parts to test a variety of ideas, like using plastic syringes to "paint" on the canvas in a controlled way, and making "light paintings" using precision 3D printing motors running at very slow speeds to control lights and cameras with long-exposure times.

It was clear from speaking to him that he was at least as excited about the machines and how they worked than about the end results. Patricio spoke in great detail about his machines and kept them in prominent places in the open plan studio. Part of that could be the nature of artistic process, where it takes a great many intermediate works to produce one that the artist considers worthy.

Patricio was the only initial participant who was interested in using the first version of the experimental LivePrinter software. This version had numerous problems, but Patricio and the researcher were able to use it on one of his self-built machines to print a diagonal line into the air, free of support, as a proof-of-concept of what LivePrinter could do that other 3D printing systems could not.

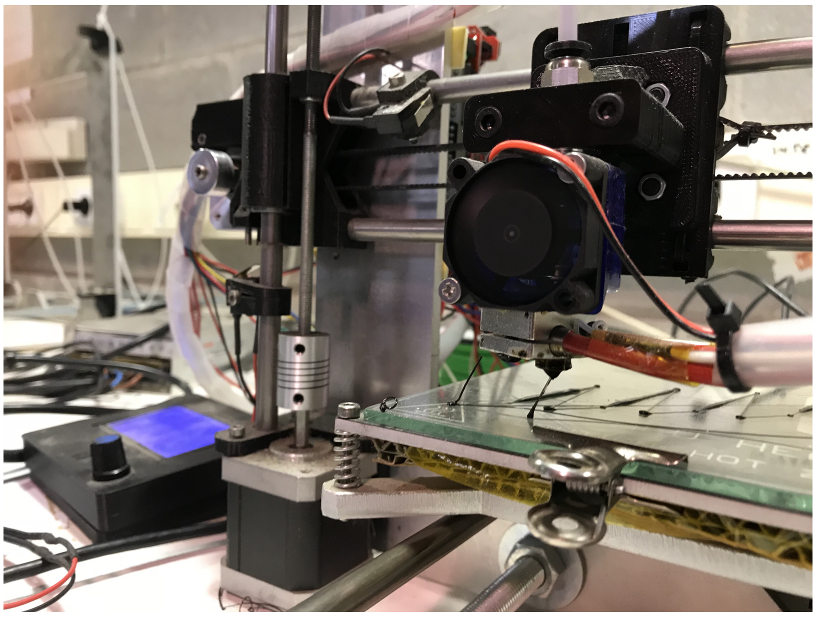


Figure 7.1: During the interview with Patricio of TMTMTM in 2018, an experiment introducing LivePrinter and testing if his 3D printer could print into the air with no support.

The main takeaway of these interviews was that experienced users (Tim and James; David; Beth; Patricio and Ali from TMTMTM) saw 3D printing as a difficult and time-consuming process that was best reserved for special applications such as free experimentation or when certain parts were needed that otherwise couldn't be manually made. The 3D printing process was a useful process to go through when a person needed to understand manufacturability and materiality. Otherwise, people were attracted to the potential to make and share collections of new objects and works of art (especially Belen) for intellectual and political purposes.

7.3 LivePrinter: Structured user studies

The LivePrinter technical system and documentation was being developed alongside these initial interviews, sometimes forming a point of discussion in the interview (especially with Patricio's). When it was functional and stable enough to be tested with outside users, a series of public user workshops was scheduled to test and evaluate the usability, learnability and future utility of the system. Participants were recruited from the populations of the creative industries, communities of practitioners of creative technology, and students at both graduate and undergraduate level. All workshops were open to registration from the public as well, and a few people not falling into any of those categories participated as well.

Workshops took place internationally, in London at Goldsmiths University's maker space; as part of the International Conference for Live Coding in MediaLab Prado in Madrid, Spain; and in New York City, USA, at the Brooklyn Research organisation's maker space. All sessions were organised and led by the main researcher. Some sessions employed volunteer helpers to give out materials and troubleshoot general computer issues or programming difficulties.

Their overall structure was based around providing potential users with conceptual models of 3D printing in general, and our I3DP system, LivePrinter, in particular, and then working through these models in context-appropriate tasks during a 2–3 hour long session[1]. These researcher-led, contextually-rich activities were interwoven with some short, topical lectures and discussions with prospective users.

1: Conceptual models and their application to user studies are introduced and discussed in Section 5.4 (Aligning user understanding and notational systems).

All the workshops followed the same basic plan: to give some context to 3D printing; introduce the LivePrinter system conceptually; install it to understand its different parts; test it out by teaching participants some basic operations; allow space for improvisation; reflect on the session at the end. They took place in "makerspaces" outfitted with comparable 3D printers that were (mostly) compatible with the software. Participant brought their own computers so they would leave with the software installed and hopefully continue experimenting in the future.

Workshop activities were designed to generate rich qualitative feedback, in the form of written feedback from participants, structured interviews, and recorded video and audio, as well as quantitative feedback recorded manually by participants on short, task-specific feedback forms during the sessions. These would be later analysed using standard usability metrics, and discursively using the Cognitive Dimensions of Notations (CDNs).

7.3.1 Data collection and evaluation

In designing the workshops and evaluation framework, the CDNs were used both as a design guide for workshop activities and also for structuring interview questions and developing the observation schedules for gathering usability-focused data during the workshop (as mentioned in Section 4.3 (The research process)). The collection and evaluation process was written into a *LivePrinter User Research Plan* which was printed out and provided to all researchers prior to them taking part in the workshops, and which can be found in Section B.1 (Workshop plan for the researchers).

In order to triangulate the researchers' first-hand experience of running through the sessions with other forms of qualitative and quantitative data from users, collected at regular points in the sessions, the *Research Plan* outlined relevant CDNs for each workshop task and offered question prompts to ask either participants or for the observer to reflect on themselves both during and after the research activity.

Before the workshop, participants were provided with details of the workshop and instructions to install necessary software. They also received a questionnaire to fill in about some personal details that would establish their *user type* based on their experience with software development, 3D printing, age, and relevant interests. This quantitative data was meant to be stored off-line in a paper format, but in the end it was decided that it wasn't useful enough for establishing user types[2] and was destroyed.

2: See the discussion in Subsection 7.6.1 (Problems with individual user types) below.

In each workshop, another printed questionnaire was given to participants to fill in as the workshop progressed[3]. This questionnaire was explicitly linked to individual sections of the workshop plan and referenced specific tasks. Each section and task could be rated with a simple 5-point Likert

3: See Section B.2 (Workshop tasks feedback sheet) scale to roughly measure sentiment (from “disliked” to “liked this part”). More importantly, space was given for comments on each task, which most participants filled in and sometimes accompanied by additional full pages of feedback.

At the end of the studies, in multiple passes, the section ratings, written feedback, notes and self-reflections were entered into a database by the main researcher and categorised by assigning multiple tags. Loosely, this process of tagging (“coding”) qualitative data followed Mayring (2014, pp. 62–64)’s general content-analytical step model of starting with a combination of theory (the CDNs in this case), the research question, and task-specific workshop goals as initial dimensions for naming tags and revising them in an iterative way as problematic examples appeared. Tag names were also assigned to help with the software development process by highlighting problems, opportunities and mapping feedback to specific parts of the LivePrinter system, as well as to associate feedback with specific areas of the user studies and highlight the usability and utility of the system.

At each revision point, tags were sorted into general themes of interest; printer-specific activities; and software development areas. Then, feedback was sorted into clusters of the same or similar tags and tags within each cluster were compared to make sure that the feedback within them seemed related enough. If not, they were re-tagged according to how well they matched other tag clusters. If some tags only existed alongside others then they were marked as possibly redundant, and, unless they had something unique to offer the analysis, they were removed.

An example of this was the tag “extrusion” which was felt to be too broad to have anything original to contribute beyond the “movement”, “retraction” and “difficulties” tags which covered more related issues in ways that more closely mapped to the workshop tasks.

Some tags were not used often and were consolidated, like “variable naming” meaning “Syntax but more specifically about names of variables”, which could be combined into the broader tag group for “syntax”. Similarly, the initial tag “complex shapes”, which related to the L-systems-like function and the observations about some participants making generative shapes, was attached to too few of the feedback entries and was felt to be too specific, overlapping with either the more general “syntax” tag when it came to details of usage or the “improvisation” and “play” tags when it came to participant intent.

CDN-specific tags were separately added in a later exercise following a similar pattern of tagging, clustering, comparing clusters, and re-tagging. Then, the original tags were revisited according to how they related to the CDN clusters. The end result was a database of individual pieces of qualitative data (observations, researcher reflections on sessions, and snippets of user feedback from workshop worksheets) mapped to multiple tags and also multiple CDNs. In the diagrams shown in Figure 7.19, Figure 7.20, Figure 7.21, and Figure 7.22, some key examples of data are mapped to CDNs. In Table 7.1 (general tags), Table 7.2 (software development tags), and Table 7.3 (3DP activity-centric tags), tags are divided into categories along with key examples of data associated with each of them.

As with the content-analytical model, feedback was reviewed at multiple points during the research process and new tags were created when interesting aspects of that feedback that didn't fit the current tagging system were recorded. When issues from multiple feedback sources appeared to be related, they were consolidated into the same general tag or given a new one if the issues seemed both specific and unique to them. Over time, tag names were consolidated as much as possible and categories further refined for simplicity in analysis.

Semi-structured interviews were conducted after each workshop. These were used to interrogate participants' views about the potential current and future usefulness of the system and gauge the desirability for such a system across different user groups. Interviews were guided by four CDNs that were pre-selected by the researchers. The results of these interviews, along with the evidence of and reflection on the physical artefacts created during my experiments, forms a collection of perceptions, methods, and artefacts that begin to describe this new field of live computational sculpting. As A. Blackwell (2005) caution, this "filtering" of the CDNs by the designers themselves might lead to blind spots in the user study where users weren't given the chance to respond to other potentially interesting CDNs, but there are a few responses to this worry.

Sessions were recorded in audio and video and stored offline for privacy concerns, and participants were assured that the data would be deleted at the end of the study. Participants also signed a data protection and usage agreement at the start of the workshop. After each test, the researcher's notes were recorded and reflected upon, and the software and documentation were improved according to user feedback.

7.3.2 Tasks and goals

Each workshop was divided into a series of nine numbered *Tasks*. These tasks were described in corresponding areas on the printed feedback sheet provided to each participant at the start of the workshop. Tasks followed the study's goals outlined in Section 6.3 (Design Goals) and, taken in order, formed a conceptual introduction to I3DP that would help the participant establish their own mental model of how I3DP worked. Thus, the workshop-specific goals for users were to get them to:

- Goal 1: understand the process of 3D printing and relation to self-practice
- Goal 2: use interactive programming to make 3D forms using physical properties like speed, temperature, material flow
- Goal 3: use interactive programming to make 3D forms using geometric strategies other than optimising for speed and strength
- Goal 4: use livecoding for 3D basic 3D printing operations

Workshop tasks were mapped to these goals as follows:

- Task 1: Introduction talk about 3D printing — Goal 1
- Task 2: Introducing LivePrinter (overview) — Goals 1,2,3,4
- Task 3: Installing LivePrinter — Goal 4
- Task 4: Using LivePrinter (basics) — Goals 1,4
- Task 5: Using LivePrinter (printing a square) — Goals 3,4

- Task 6: Using LivePrinter (retraction & material flow) — Goals 3,4
- Task 7: Using LivePrinter (height and layering) — Goals 3,4
- Task 8: Using LivePrinter (freestyle drawing) — Goals 3,4
- Task 9: Future Use — Goals 1,2,3,4

7.3.3 Supporting goals: installation

In order to successfully complete each task and achieve the study goals, the underlying software and hardware needed to work properly, meaning that they were both *usable* for the task and *useful* for completing it. To begin, the user needed to install the software successfully in the first place, so evaluating the success of the installation process was a key part of the study.

One of the sub-goals was to determine if the current installation process, where users simply copied all the code for the project and executed it on their computers, was both simple and allowed for future development. It was hypothesised that having the code in a single, editable place would make its functionality more transparent to inexperienced programmers who were unfamiliar with build tools and lacked a sophisticated development setup, although experienced programmers might not appreciate having to navigate a single file instead of being presented with a more conventional and organised file and folder structure. This was contrary to many software installation processes where the project code is distributed without support libraries and organised into in multiple files, requiring an extra "build" phase using command-line tools to make it usable.

7.3.4 Supporting goals: learning and conceptual models

One of the main goals of the workshop was to help users learn the system by providing an explicit *conceptual model* (Norman, 1983) of interactive 3D printing mapped to specific conceptually-oriented tasks. This was reflected in the section of the workshop detailed in Subsection 7.4.1 (Task 1: Context) and Subsection 7.4.2 (Task 2: Introducing LivePrinter), which explained to participants where LivePrinter came from, in terms of historical and contemporary 3D printing developments, and provided a conceptual model of how FDM 3D printing worked that was meant to map to LivePrinter's terminology.

Alongside learning helpful conceptual models, the workshop tasks were designed to help *defamiliarise* (Kilpinen, 2009) the "magical" process of 3D printing. As Brooks (2017) pointed out in his essay on the "sins" of predicting the future, "if something is magic it is hard to know the limitations it has. . . anything one says about it is no longer falsifiable, because it is magic." To help participants think critically about the object creation process they needed to learn the actual limitations (and less explored opportunities) of 3D printing.

Defamiliarisation helped estrange both participants and researchers from the often-familiar practices of programming and 3D printing, creating a conceptual distance needed to gain a more objective view of these practices. This was achieved through the strangeness of a project itself, e.g. forcing participants into the act of making physical forms with code instead of by hand or 3D modelling. It was also achieved by re-engaging with the process of 3D printing at different levels, such as introducing them to simpler, alternate methods for fabricating objects through historical examples. Later on, they actively experimented with the minutia of the process by following step-by-step guides to fabricating common 3D shapes, like cubes, as in Subsection 7.4.7 (Task 7: Using LivePrinter (height and layering)).

7.3.5 Supporting goals: helpful documentation

Documentation, at minimum, needs to be clear, usable, and easy to find when needed. As Gorski and Iacono (2016, p. 255) and Winter, Wagner, and Deissenboeck (2007) note, the documentation of systems (especially that of APIs) can be considered to be an inseparable part of that system. Users were expected to encounter the documentation at a few different levels: first, before using the system, when they were deciding whether to try it out initially; second, as they tried out the system in an explorative, playful, and learning mode to evaluate whether to continue using it; thirdly, when they were using the system for a specific purpose or project in a more structured, task-oriented and experimental way. That meant that documentation served a number of roles ranging from promoting the system to assisting users during use and providing prompts for reflection afterwards.

It was decided to somewhat forgo the methodology of *Minimalism* (Carroll, 1990), where documentation is short, task-specific, and focuses on activities, but does not include conceptual material. The documentation was still written to be brief as possible and focus on active learning activities loosely grouped into tasks. Where it differed was that much of the documentation was arranged according to a series of concepts that provided both scaffolding and background to this new form of 3D printing, so as to better situate unfamiliar concepts and promote a more reflective and strategic learning process that would hopefully avoid "inefficient bricolage" (in Ben-Ari and Yeshno (2006, p. 1341)'s words).

The effectiveness of the documentation was integral to success of the user testing workshops. The workshops were structured as journey through key parts of the system led by a "quick start" document (described below), and sign-posted by the integrated documentation (also described below), starting with the origins of additive manufacturing and leading participants through a series of activities meant to help them understand how 3D plastic forms are physically constructed and conceptually described in code.

7.3.5.1 The project website

The main GitHub website for the project, which contained the downloadable code necessary to run it, was the most verbose out of all the documentation. It was expected to be the first point of contact for potential users who were well versed in software development, and possibly the only point of contact for many of them. It described how to download the software and get started; the aims of the project, including some detailed diagrams; how to get in contact with the developer(s) and community; and referenced other related and relevant projects. Importantly, it described how to download and install the software.

This documentation was referenced during workshops, especially the installation tasks, and as a result was continually updated throughout the project. Videos were added to illustrate the installation process and general usage of the system, and a "Quickstart" document was later created as a shortened version of the workshop activities[4].

4: https://github.com/pixelpusher/liveprinter/blob/master/reference/Quickstart.md

7.3.5.2 Workshops documentation

For workshops, users were provided with a PDF document containing all the learning materials needed to get started (see Section C.1 (User Workshops presentation)), which was also available to download online. This document provided both a guided tour of the system and a reference for the system's terminology. Whenever possible, it pointed users towards the documentation embedded in the LivePrinter interface itself.

All the documentation for the workshops followed a task-based approach, linked to the data collection process. It was also meant to be accessible by users who didn't take part in the workshop but still wanted to try out the system, or past participants who needed to use it as a reference.

7.3.5.3 Integrated documentation

One of the most important goals of the documentation was to help get LivePrinter installed quickly, so people could start actively experimenting. Once the system was installed, users were expected to dive straight in and start actively learning by using it, as opposed to reading the documentation first. This meant focusing on short and conceptually-organised activities coupled with contextual explanations as with the *Quickstart* above. Many of these were integrated into the system itself, in the form of *Examples* (see section 5 of Fig. 6.2).

7.3.5.4 In-code documentation

Much time was spent during the project making sure that all code was thoroughly commented. *Comments*, in a coding sense, are special lines of text written for humans, not computer compilers, existing *in situ* inside the source code alongside executable code. These comments formed a major part of the integrated documentation, guiding users in midst of their *bricolage*-style livecoding activities where they would be running code line-by-line, surrounded by helpful code comments. Sometimes they outlined explicitly conceptual activities and tasks, as with this selection from the *default.js* example for getting started used in the workshop tasks:[5]

5: https://github.com/pixelpusher/liveprinter/blob/master/liveprinter/static/examples/default.js

```
    // ...
    // Step 5: click printer settings tab, select serial port. Wait for
    //   connection messages and green dots to move up top.
    //
    // Step 6: click code tab to load editor
    //
    // Single lines of liveprinter code start with ’#’
    // Blocks of liveprinter code start and end with ’##’ (more on this later)
    // Run code by clicking on the line you want to run and hitting CTRL and ENTER keys,
    // or SHIFT+ENTER.
    // Or, highlight some code and hit CTRL+ENTER or SHIFT+ENTER.
    //
    // If there’s a problem with your code, it should pop up at the top of this editor.
    // If it’s a problem with the system, you might have to open the JavaScript console
    // for this web browser.

    // Step 7: Home the axes so the printer knows where the print head is positioned.
    // (Do this every time it loses power). Set print head temperature to 210 (for PLA) // and turn on print head fan:
    # start 210

    // Step 7b.: turn on the print bed to 50C
    # bed 50

    // You can also set the print head temperature directly:
    # temp 220

    // same goes for fan (0-100%):
    # fan 100

    // Now check the display above - the numbers should have changed. Try:
    # sync   // get back current temperature and print head position from printer
    // It should update very quickly!

    // Step 8: click printer tab again and hit the button to start live temperature
    // polling, or keep running sync until the temperature is hot enough (190+ for PLA).
    // If the temperature is too cold, it will ignore your printing command and give
    // you an error in the side info panel.

    // Step 9: Get some feedback! The right panels show info and errors.
    // Print out some info yourself:
    loginfo("This is my info!");

    // Print out an error:
    logerror("ERROR! I caused this.");

    // If things go wrong internally, they show up there.

    // Click on the "history code editor" tab above - you should see all the code you ran, with the time you ran it. This is useful for recording a session! You can also run code directly from it, same as here.
    //
    // Next to it is a GCode editor that records all GCode from this session. You can At the bottom right of this window pane is a "download" button that downloads all code editors to your computer.

    // Now your printer is ready to print! (When the target temperature of 210C is reached)

    //
    // MOVING (traveling)
    //--------------------------------------------------------
    // Try moving - you can do this whilst it warms up:


```

Figure 7.2: Excerpt from the LivePrinter Quickstart provided in the built-in Examples (part 1/2)

```

        //absolute move to x=20mm, y=20mm at speed 80mm/s:
        # moveto x:20 y:20 speed:80


        // you can also use the shorthand:
        # mov2 x:20 y:20 speed:80

        // relative move at 1/2 the speed - we should be at x,y (60,20) now.
        # move x:40 speed:40

        // you can move up and down too: try moving the print head down 10mm
        // (meaning the bed moves up 10mm)
        # mov z:-10

        // lower the bed to 50mm less than the bottom:
        # moveto z:lp.maxz-50 speed:40

        // END

```

Figure 7.3: Excerpt from the LivePrinter Quickstart provided in the built-in Examples (part 2/2)

7.3.5.5 Generated documentation

There are automated systems for turning in-code documentation into web pages. Before the workshops, LivePrinter used an open source system called JSDoc[6] and a standard documentation template to make the function references available to potential developers and workshop users, but the results were not promising. Most users viewed it once and then ignored it. Others found the formatting difficult to follow. The main complaint was that this generated documentation was organised around file structure and not typical usage. It might have been helpful for future developers of LivePrinter looking to extend the system, but was less so for people using it for activity-oriented livecoding.

6: https://jsdoc.app/

7.3.6 Evaluating the usability of livecoding

One of the most important goals for the livecoding portion of the system was to be both a usable and useful vehicle for supporting people's live material explorations. The usability studies for this part focused on the GUI itself (as described in Fig. 6.2) and the combined learnability and usefulness of the software API and associated language syntax, which was developing throughout the project. The tags *usability*, *usefulness*, *programming*, and to a lesser extent *GUI* were helpful for establishing relevant feedback for this goal.

The GUI was evaluated in the workshops according to its role as both interactive "dashboard" for the connected printer and software internals, and also in its utility as a non-linear code editor supporting a range of livecoding styles. It was evaluated implicitly whether it enabled the user to successfully complete the workshop tasks, rather than explicitly through a specific, GUI-centric task (which might have taken the form of "try and find the location of the print head in the after this movement"). Effectiveness was determined from participant feedback, post-workshop interviews, and in-situ observations by researchers looking for problematic situations when users were unable to find necessary information or showed signs of confusion, and positive situations where they were engaged in the act of making. This holistic approach prioritised understanding the quality of the time users spent learning interactive programming and making objects and over testing the effectiveness of specific notational elements.

Starting with the first version, all versions of LivePrinter included an API that mostly described GCode functions, or combinations of functions, and ultimately compiled to a series of GCode commands that were sent to the printer. To evaluate the API, the workshop guided participants through a "scaffolded" learning process with general goals that built from simple to complex, such as moving at a specific speed through to drawing a square. They were provided with example code to use and documentation to reference during the study. The focus was on whether enough functionality was provided to the users to support their creative process, and whether it made sense to them. To further test the context-specific abstraction levels, recognisability and learnability of the notation, functionality was often implemented in different ways (e.g. different terminology for "absolute" vs "relative" movements or relational terms such as "up" vs. "down"). This is reviewed in Subsection 7.4.4 (Task 4: Using LivePrinter (basics)).

To evaluate the effectiveness of the code syntax, participants were given different forms for constructing "code sentences," roughly categorised as *compact* vs. *verbose* notation for describing operations; method chaining; and method cascading as described in Subsection 7.4.5 (Task 5: Using LivePrinter (printing a square)).

7.4 User workshop results

The results of the user workshops are presented here, divided into an explanation of each section (or task) of the workshop and followed by a discussion of the user feedback and research notes relevant to that section. The task explanations were taken from the LivePrinter Quickstart document presented to the users during the workshop, and the feedback and notes were taken from researcher notes, reflections, observations, and notes taken from the various video and audio recordings of each workshop, and the 31 filled-in, task-specific, user feedback sheets, that were collected from all the workshops.

Again, these can all be found in the Appendix at the end of this document.

7.4.1 Task 1: Context

Each workshop began by with a short lecture overview of 3D printing to provide context for LivePrinter. The lecture started with a brief history of semi-automated manufacturing, then, in the second part (described in the sext subsection) introduced a conceptual model of how 3D worked from both mechanically (identifying key electronic parts, and discussing plastic layering), and on a service level (the aforementioned *Process Planning*: modelling, slicing, rendering to G-Code, printing, finishing). The notes for this section and all future sessions were provided as a PDF file for participants to take away with them, as well as projected on the screen in the workshop space.

This task and next task on LivePrinter were a direct response to Baudisch and Mueller (2017)'s Challenge 4 of "machine-specific knowledge", or understanding better how 3D printers work and are controlled, and what sort of tooling options were available.

Participants mostly found the context section inspiring and helpful in creating their own basic conceptual model of 3D printing, at least "enough to get up and running" as one participant wrote. Without a basic understanding of the mechanisms and applications of 3D printing, the "role expressiveness" and "closeness of mapping" of the LivePrinter notation and would be basing their actions on their non-expert interpretation of technical terms like "retraction" and "extrusion" in the LivePrinter API. "Enjoyed going back to 'first principles' of manufacture. Very interesting contextualisation of what we're about to do".

Despite the relatively long length of this section at 30-odd minutes, participants found it useful: "Informative & inspiring with visual examples of how creative practitioners have approached 3D printing. Good pace to presentation." Some wanted to know even more about how 3D printers were actually used in industry, and more in-depth details about how the machines actually worked. Later sessions took this feedback into account and skipped some historical examples in order to focus on contemporary examples more relevant to current practice, and also to leave time to cover specifics of printing that were better illustrated in the later, more hands-on sections of the workshop.

7.4.2 Task 2: Introducing LivePrinter

Next, the LivePrinter system was introduced as a new way of making 3D printed objects incrementally, including a brief overview of the hands-on, experimental design philosophy it represented. This was necessary to set up the expectation that participants would be experimenting with the *process* of 3D printing and not as much with fully-formed *products* of it.

7.4.2.1 Feedback and Responses

There were numerous positive comments about how this section was "clear" and "informative" and "good general background info, enough to get up and running". As one put it, "Good explanation on why it exists and the goals for the project and how it differes [sic] from traditional 3D printing." Participants were more critical about two distinct areas: the workings of the specific 3D printers in the workshop, and how 3D printers are used in industry.

A description of the mechanics of desktop FDM 3D printing came later, so it would be concurrent with the activities that directly used those mechanics, which may have been late for some participants. Also, FDM printing in industry (large or small) wasn't discussed in detail, which may have been an oversight on the researcher's part.

One unforeseen issue with this introduction was that some users with no experience of either 3D printing or coding recorded that they didn't understand the difference between LivePrinter's I3DP method and how other 3D printers normally were used. They had never used software to slice a 3D model and print it out, so they had nothing to compare this new system to. Fortunately, as one participant put it, "when you introduce 3d printer at the beginning, [they were] not confident to understand the theory and the connection between coding and 3d printer you showed on the slide, [but felt better when we] started to practise on 3d printer because it's more obvious and visual."[7] To give new users a sense of the difference in these two approached, it might have been better to start this section by slicing a model to a file and printing it, and then approaching drawing a similar shape using LivePrinter.

7: This quote has been edited slightly to keep the anonymity of the participant.

7.4.3 Task 3: Installing LivePrinter

In this task LivePrinter was installed on the participants' personal machines using either USB keys provided by the research staff or personal internet downloads. During the sign-up process participants were directed to pre-install some necessary software before the session, which many did, and potentially to even install LivePrinter themselves, which very few did. This was the most difficult part of the workshops, despite the step-by-step instructions and provision of installer software for different computer setups. It required file copying and typing in installation commands, and sometimes there were complex conflicts in software versions (especially with Linux users and Python).

7.4.3.1 Feedback and Responses

This was the most problematic part of the workshop, as evidenced by the volume of negative feedback comments. Out of a total of 17 comments tagged "installation" a further 10 were also tagged "difficulties" and only 2 were tagged as "positive". In the feedback, some participants preferred textual instructions that they could cut and paste, others preferred video instructions showing intermediate actions like opening software and mouse movements and clicks. Video instructions were especially difficult to provide for this experimental system, because of the time needed to create them and update them: with every change to the system the entire video had to be re-recorded, unlike text where small updates would suffice.

After every workshop, the instructions were edited and expanded upon and a video was recorded in response to that feedback. The installation process seemed to either work easily for participants because they had experience developing software and understood the basic programs, or it was difficult because they had problems with "different software and language" or were "unfamiliar with the interface for the initial set up" (meaning the textual Terminal program).

Participants who understood the process (or had followed the instructions and done this before) either helped those who didn't or found it tedious when we waited for everyone to get their system working before moving on to the next section. There was a large difference in knowledge between participants with computational experience as opposed to more traditional designers, educator, artists, and others. This pointed towards the different user experiences of 3D printing, as a hybrid computational/physical process, depending on those users' backgrounds. As one technically-experiences participant put it, "Very hands on - great! A little boring when people became a bit tech challenged."

7.4.4 Task 4: Using LivePrinter (basics)

For this task, participants started running the LivePrinter system on their computer and executed some simple drawing operations. It introduced the LivePrinter Graphical User Interface (GUI), communicating with the 3D printer, some basic syntax and functionality, and the specific order in which the system should be started up. The starting order was very important, both due to the experimental state of the software and the actual physics of the 3D printer.

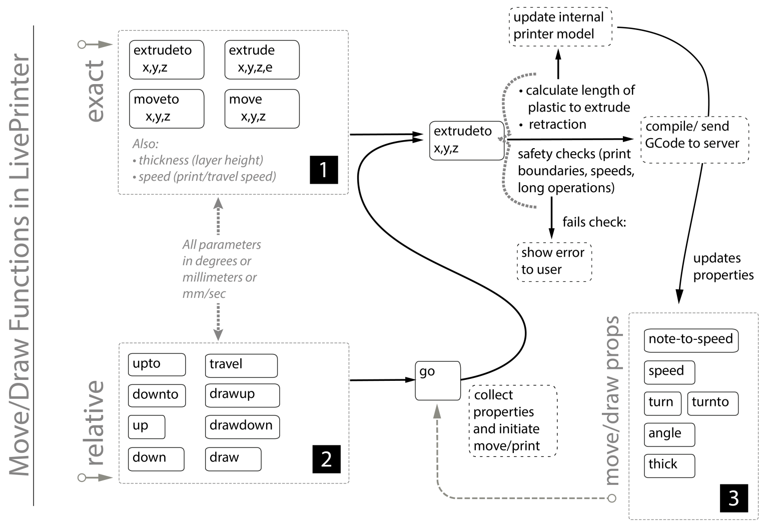

Move/Draw Functions in LivePrinter
exact
extrudeto x,y,z
extrude x,y,z,e
moveto x,y,z
move x,y,z
Also:
• thickness (layer height)
• speed (print/travel speed)
1
All parameters in degrees or millimeters or mm/sec
relative
upto
travel
downto
drawup
up
drawdown
down
draw
2
extrudeto x,y,z
update internal printer model
• calculate length of plastic to extrude
• retraction
safety checks (print boundaries, speeds, long operations)
fails check:
show error to user
compile/ send GCode to server
updates properties
go
collect properties and initiate move/print
move/draw props
note-to-speed
speed
turn
turnto
angle
thick
3

Figure 7.4: Movement and drawing (extrusion) functions in LivePrinter showing the internal workings of the (1) relative (specified vs. current position and heading) and (2) exact (specified using absolute position) movements. Function names are inside the rounded boxes. These functions result in immediate printer movements. In section (3) is a listing of the functions that affect the properties of movements and printing but do not result in immediate movement. Note that temperature and retraction functions are not listed here.

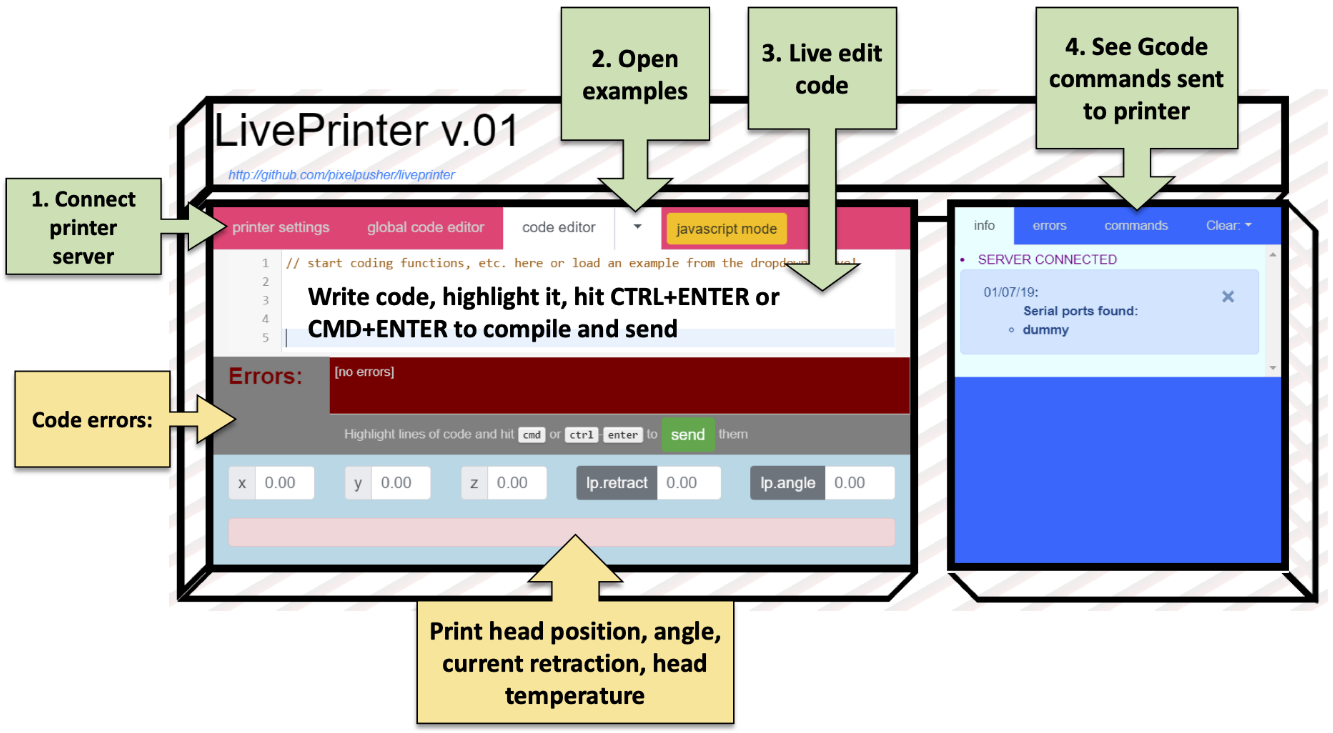
7.4.4.1 Connecting

First, the participants connected the LivePrinter system to the attached 3D printer via a serial connection. If this was not done, commands would have no effect. This was a problem in early software versions because it was unclear that the connection wasn't established, whereas in later versions a "connected" animation in the form of a line of scrolling dots appeared at the top of the screen and the software actively prevented commands from being executed when not connected to a printer.

Once connected, the printer needed to be started up in the correct sequence, beginning with "homing" the print head to the origin position in the top left of the printer cavity and moving the print bed to the printer floor, then heating up both the print head and print bed in preparation for printing. This sequence was consolidated into a single function, but forgetting to run it would result in a misaligned printer head and cold material that could not pass through the extruder, both fatal errors in the printing process.

In terms of the CDNs, this "error-prone" sequencing was a major case of "premature commitment" since it was up to the user to make sure that the bed was constantly aligned and the head was the correct temperature. Unfortunately, checking that the print bed was aligned would have required additional hardware that the printers didn't have installed. Another problem was that querying the printer properties had to be done in code, which was a high level of "viscosity" for retrieving information that was helpful at any point in the printing process.

The viscosity was lowered by increasing the "visibility" of the printer temperatures after the first workshop by adding elements to a "dashboard" displaying printer properties in number-display elements at the bottom of the interface. With this dashboard users could quickly scroll to see the current x,y,z position of the head and the current print head and bed temperature (as seen in the workshop presentation's "Step 1" in Section C.1 (User Workshops presentation)).


LivePrinter v.01
http://github.com/pixelpusher/liveprinter
1. Connect printer server
2. Open examples
3. Live edit code
4. See Gcode commands sent to printer
printer settings
global code editor
code editor
javascript mode
Write code, highlight it, hit CTRL+ENTER or CMD+ENTER to compile and send
Code errors:
Errors:
info
errors
commands
Clear:
x 0.00
y 0.00
z 0.00
lp.retract 0.00
lp.angle 0.00
Print head position, angle, current retraction, head temperature

```
// start coding functions, etc. here or load an example from the dropdown...
```

[no errors]

Highlight lines of code and hit cmd or ctrl + enter to send them

SERVER CONNECTED

01/07/19:
Serial ports found:
- dummy

Figure 7.5: The original GUI for LivePrinter at the start of the user workshops had a section at the bottom that displayed printer properties. This was moved to the top as seen in Figure 6.2, based on workshop feedback where participants often did not scroll downwards and thus did not see it.

Very quickly, it became apparent in the workshop that putting the dashboard at the bottom was a mistake, because scrolling down in mid-operation was also adding to the viscosity. Additionally, a user noted that we should “signify current vs. target temps”, meaning we should show the temperature of the head and bed in relation to what they were set at in the firmware, which made sense given that the temperature changes took significant amounts of time which was not visible to the users. This led to the last GUI design in this study, seen in Figure 6.1.

7.4.4.2 Moving

Lines drawn in 3D space form the basis for most, if not all, 3D printed shapes. In terms of printing movements, the standard 3D printer GCode syntax provides two modes of movement, called *absolute* and *relative*. Absolute movements moved the print head (or bed) to specified coordinates, whereas relative movements moved the print head (or bed) by a certain offset. Both specified movements in millimetres across one or more axes: the x-axis (left–right); y-axis (forward–back, facing the user); z-axis (up–down); and e axis (the filament forward–reverse feed) and at a specified tool speed from the current position to the end. Internally, in the Marlin 3D printer firmware this speed is split into individual motor speeds across each of the x, y, z, and e axes to achieve the desired overall tool speed over the entire movement. An illustration of this process is included in Fig. 7.4.

In this task, participants experimented with the two alternative ways of "drawing" plastic lines. To first familiarise participants with printer motions without the additional complexity of managing material flow they were instructed to move the printer head without any plastic extrusion in what is called a *travel move*.

They began by using *absolute* mode, e.g. moving the print head to and from explicitly specified positions in x, y, z in the printer cavity using a function with a verbose JavaScript Object-style argument containing one or more of the movement properties as an argument as in `{x, y, z, e, speed}`.

They were instructed to use the `lp.moveto()` function as follows: `lp.moveto({x:20, y:30, z:10, speed:20})`

In this example, looking at the printer from the front, the head moves 20 mm to the right, 30 mm towards the back, the space increases between the head and bed by 10 mm, and this complete movement takes place at a fixed speed of 20 mm/s across all axes.

Then, they moved the print head to and from positions *relative* to the current position in x,y,z in the printer cavity using the `lp.move()` function: `lp.move({x:0, y:20, z:10, speed:20})`

This verbose style of coding was meant to be more transparent than simply listing parameters in an arbitrary order. For example, some APIs like p5js[8] use bare lists of numbers to specify arguments, as in this example for drawing parts of an image to the screen: `image(imgName, 0,0, 20,40,50,60,70,80)`. The trade-off in verbosity was meant to help with learning movement properties through repetition, minimise the need to keep looking up function in the documentation, and also minimise error due to misplaced arguments.

8: http://p5js.org

Participants continued experimenting with moving the print head varying distances and changing speeds, using different function arguments, to get an intuitive sense of both the size of the printer cavity and the range of speeds at which the printer head could move.

7.4.4.3 Extruding

Once they had some experience with movement, participants moved on to drawing with plastic, or *extruding*. In this task, participants worked on moving the printer head until it was positioned slightly above the printer bed and experimented with drawing plastic lines.

For print head movements *with* plastic extrusion (sometimes called "extrusion moves"), the API provided a function called `lp.extrude()` for relative moves and `lp.extrudeto()` for absolute moves, with the same style of JavaScript object passed in as the movement functions[9]. The aforementioned **e** parameter in the function arguments `{x, y, z, e, speed}` controlled the length of filament to extrude, which was useful for manual retractions and priming the filament in the nozzle.

9: These functions are defined in the printer API at https://github.com/pixelpusher/liveprinter/blob/master/js/liveprinter.printer.js

7.4.4.4 The GUI

In the first two workshops, the GUI was located on bottom of the screen, but was moved in subsequent workshops because it was hard to scroll down to find it when the code editor was at a larger size, and people were not inclined to scroll down to find it. Also, the buttons were made smaller and streamlined to accommodate more screen space, and the colours were adjusted to be higher-contrast to accommodate colour-sensitive users.

Users also offered a few feature requests for the interface that were later integrated, such as a "new file" option. Another issue was the top breakout display of important printer properties, such as temperature and head position. Users said that they were uncertain whether this was a display or a way to quickly set properties, so in later workshops the ability to quickly set properties through the top display was added. This new "dashboard" was not explicitly tested with users, so as not to further complicate the workshop schedule, but the researchers found it helpful to be able to do things like set the target head and bed temperatures quickly and immediately, without code, lowering the viscosity of these simple actions at the expense of increased visibility of the printer parameters over the interactive code editor.

7.4.4.5 Feedback and responses

This task was meant to be an introduction to the user activity of *Experimental I3DP* (see Subsection 5.6.1 (Activity Cluster: Experimental I3DP)), a slower and more deliberative mode of working with I3DP since users were new to the system and weren't expected to have much if any intuition about how it might work. They needed to first commit the basic syntax and modes of working to memory before they could quickly recall and combine them together "on-the-fly". It relied heavily on *transcription* and *modification* activities to get them working quickly and provide them with future building blocks for more intuitive cut-and-paste coding.

Despite the initial complexity of this section, user feedback was very positive. It was described as playful, "easy, simple", "straightforward", and "very satisfying to use", and "Delivered well, diagrams were easy to follow, friendly interface. Built confidence around coding for someone with very little experience in coding." 21 out of 30 responses gave it 5/5 and 6 gave it 4/5.

The interactive code editor was quickly adopted by many participants, but interestingly became a source of confusion for moderately experienced coders. Some users found it jarring that they could execute individual lines of code in a code editor when they were used to running an entire file or project at a time, as with the creative coding environment Processing.

The verbosity (i.e. "diffuseness") of the function arguments seemed effective at preventing confusion for novices and advanced participants alike, meaning that the "role expressiveness" of the notation was helpful enough and the specificity of the parameters (e.g. `{x:XX, y:YY, z:ZZ, speed:SPEED}`) helped with the "error-proneness" of having to specify so many of them. The justification in the workshop documentation appeared to satisfy some participants as to why this high level of diffuseness could be helpful, although it wasn't without some grumbling: "quite like the API, not biggest fan of having to input JSON for each `move()` command but understand why used."

Participants understood the movement functions conceptually, but were unsure about units of distance: "I get the principle but not really sure between number and position." Some asked for visual references on the printer itself. One wrote "Say where 0,0,0 is on printer" and during discussion participants liked the idea of a printed grid on the printer bed or inside the build cavity for reference. They we also unsure about the starting point, and asked for clarification: "specify a universal origin | zeroed out (bed down, head back left)". Others seemed to develop an intuition of the size of the space after some experimentation: "...not initially straightforward but after some playing it was better understood."

In response to this feedback, since the printers themselves couldn't be modified later workshops provided each participant with calipers, so they could measure for themselves. This pointed towards a need for "secondary notation" for deliberately planning shapes and paths without using just the code provided.

While the code examples provided were generally a useful start for most participants, others wanted some more contextual information about available movement commands and their arguments: "if not already, add 'movement commands' slide info onto the webpage in a 'quickguide' kind of page". In response, a list of movement commands was added in future workshops as an appendix in the back on the documentation PDF and participants readily referenced it during the more advanced tasks.

Some other feedback was entered in different task sections because participants used the questionnaire sheet as a general repository for feedback across the entire workshop. Under Task 2 were some comments that appeared to be from Task 4, mostly from novice programmers who seemed to have some difficulty with coding in general as asked the researcher to "...introduce basic function before starting coding (for task 4)". Other participants noted how they worked alongside more experienced participants who took the lead in coding, again highlighting the different experience of computationally-literate participants.

7.4.5 Task 5: Using LivePrinter (printing a square)

The next concept introduced was how to connect plastic lines to form continuous shapes. Participants were tasked with drawing a square on the print bed by moving the print head using absolute coordinates (i.e. a more deliberative *Experimental I3DP*), and then by using relative coordinates (a more intuitive *Exploratory I3DP*). JavaScript's *for loops* were introduced as ways of automating shape construction, somewhat.

The task was deceptively simple, because a number of things could (and did) go wrong at this point which highlighted the problems of "premature commitment" and "hidden dependencies" and generally brittle "error-proneness" of the whole process. Firstly, the head temperature needed to be set high enough or the printer firmware would refuse to advance the material and return a "cold extrusion prevented" error. To rectify

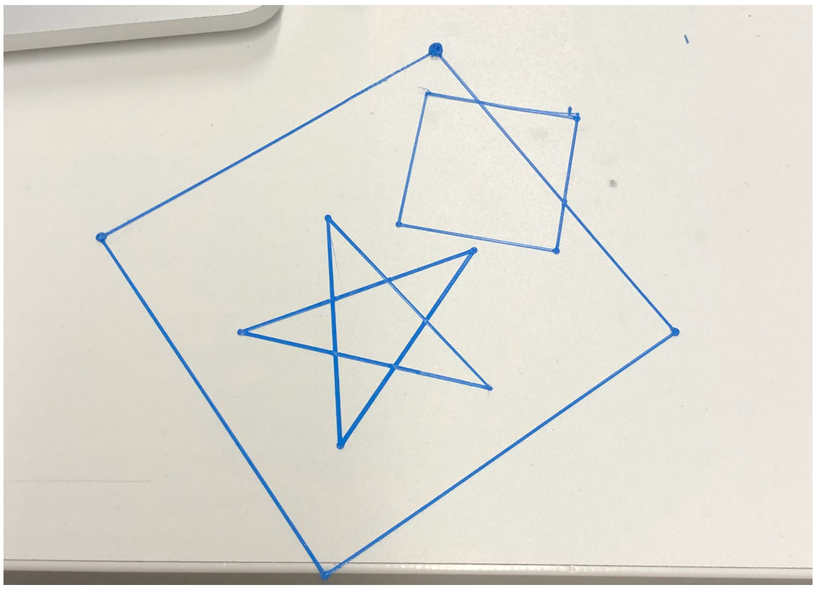


Figure 7.6: Examples of single-layer rectangular and star shapes of different sizes drawn by participants

this, participants needed to turn the printer off an on, and re-run the LivePrinter server, which was not ideal.

Also, the printer bed needed to be perfectly flat at all points, within a tolerance of around +/- 0.05 mm. If any of the 4 independent springs controlling the bed height had loosened during regular operation, or if someone had miscalibrated the printer previously, the extruded plastic would not adhere properly to the bed. This was time-consuming to fix properly because it required a researcher or lab technician to re-run the calibration routine on the printer.

Thirdly, to make the shape cleanly participants needed to account for material flow, before, during, and after the extrusion. That is, the material needed to be extruded until it completely filled the print head (i.e. *unretracted*), extruded a specific amount during the operation, and then either pulled backwards by the filament motor (i.e. *retracted*) or left in the head where it could be used for additional extrusions, or potentially drip out onto the bed. This was expected, and led them into the next section, Subsection 7.4.6 (Task 6: Using LivePrinter (retraction and material flow)) which focused more on material handling. The invisibility of the retraction process again was a problem here, despite the increased visibility of the retraction amount in the top dashboard.

Also introduced during the session were the two different syntactic methods for specifying multiple operations: using traditional loops and discrete methods, or using method chaining. These were designed to better highlight the more intuitive and improvisational modes of making, *Exploratory I3DP* and *Livecoding I3DP*, versus the more deliberative *Experimental I3DP*.

It was apparent from the start that writing lines of code with low abstraction levels, like *move* and *extrude*, would quickly take up most of the screen space in the interactive code editor. Unlike desktop Integrated Development Environments (IDEs), livecoding performances often use single files or buffers of code and favour less verbose syntax. This arguably reduces cognitive load on both audience and performer. In LivePrinter, a complex object described in single lines of code would be highly

Task 5: code for printing a square absolutely

```
// Heat up the printer to 200C
lp.temp(200);

// Prime the filament (extrude a bit to start)
lp.moveto({x:20, y:20, z:80, speed:80});
lp.extrude({e:10,speed:8});

// (repeat last step until we see filament! Then wipe it
 off)
// Move the print head to the print surface

lp.moveto({x:20, y:20, z:0.2, speed:80});

//Draw lines:
lp.extrudeto({x:120, y:20, z:0.2, speed:30});
lp.extrudeto({x:120, y:120, z:0.2, speed:30});
lp.extrudeto({x:20, y:120, z:0.2, speed:30});
lp.extrudeto({x:20, y:20, z:0.2, speed:30});

//Finally, move the head up:
lp.up(60);

```

Figure 7.7: Task 5: code for printing a square absolutely

Task 5: code for printing a square relatively

```
    lp.moveto({z:0.2}); // move to build height
    for (let i=0; i<4; i++) {
        lp.dist(40).go(1); // draw side
        lp.turn(90).go(); //turn for next side
    }
    lp.up(40); // move up

```

Figure 7.8: Task 5: code for printing a square relatively

operationally visible as to what was happening to an audience, and yet would take up most of the screen space and quickly become unintelligible as to its overall intentions.

To increase legibility and typing speed and decrease diffuseness by limiting the number of redundant characters, punctuation and lines of code that needed to be typed and reviewed, LivePrinter tested out three different grammatical strategies for constructing highly abstract "sentences" with code:

- An abbreviated, L-systems-like syntax (without recursion)
- Method chaining[10] — implemented in standard JavaScript
- Method cascading[11] — implemented in JavaScript as a syntactic sugar using the special LivePrinter minigrammar syntax

10: https://en.wikipedia.org/wiki/Method_chaining

11: https://en.wikipedia.org/wiki/Method_cascading

Originally, LivePrinter used a method chaining design pattern inspired by Turtle graphics (Papert, 1980). With method chaining in LivePrinter, function return an instance to an object representing the 3D printer state that can be modified using successive operations. For example, to move to a certain position and draw a 30 mm horizontal line one could write: `moveto({x:20, y:30, z:0.15, speed:30}).extrude({x:30})`.

This approach opened up the possibility of atomising movement properties and chaining them together, instead of specifying them in one JavaScript object eat time. Participants could set properties such as speed and direction, similarly to Turtle's first-person perspective syntax for *telling* a drawing cursor called *turtle* to take on certain properties: `TELL TURTLE SETVELOCITY 20`. For example, to set the speed outside a `move()` function, one could write `lp.speed(40).move({x:30mm})`.

Issuing first-person commands let participants assume the viewpoint of the print head, as if they were *driving* it directly rather than operating it from the perspective of looking down at the machine from above. After describing what the head should do, they then initiated the drawing operation with the method `go()`. This design pattern resulted in functions that were more noun-like. For example, `lp.angle(90).speed(50).dist (100).go();` set the movement angle to 90 degrees, the desired movement speed to 50 mm/s, the desired distance to 100 mm, and then ordered the print head to move as specified.

The L-system-inspired syntax was used in a special function called `lp. run()` which allowed participants to quickly specify a number printing operations in a single line like `lp.run('E10M20R30E10L30E10U60')` for an extrusion of 10 mm, followed by a movement of 20 mm, followed by a right-hand turn of 30°, followed by an extrusion of 10 mm, followed by a left turn of 30°, followed by an extrusion of 10 mm, followed by moving the print head up by 60 mm.

7.4.5.1 Feedback and responses

User feedback was overwhelmingly positive about this approach. They found the chains of commends more legible: "there's a whole set of commands together, so it just goes this, this, this, this, all in one line." At the same time, they recognised there were limitations to what a person could understand in a single line.

In particular, some beginners and other more experienced programmers who were interested in generative graphics found the print head-relative syntax intuitive and also useful. Participants who self-identified as generative artists and designers remarked during the sessions and following discussion that it opened up possibilities for experimenting with 3D printing that had not been previously available to them. Participants also appreciated how quickly the abbreviated syntax could be typed (e.g. it had a "low repetition viscosity"), remarking that it would be useful during livecoding, where time (and low viscosity) is of the essence.

These chained function actually used more characters and thus took longer to type than other methods, mostly because of the additional punctuation and brackets. In CDN terms, they were very "diffuse" and had a "high repetition viscosity" because of all the typing. Despite the diffuseness and verbosity, or perhaps because of the way they formed a "narrative" sentence describing the operations necessary to make an object, participants, especially from a design background, preferred them to the more concise line-by-line versions that used JavaScript object arguments, or as one put it, "[I] preferred the ones chained up", and according to another "there's a whole set of commands together so it just goes this, this, this, this, all in one line."

They did recognise that both approaches have benefits and drawbacks, however: “you need both, anyway. for some stuff, you need it separately to make it clearer”.

They likened the chained methods to viewing the printing process “from a person’s frame of view. . .looking at the printer, not relative to the [print] head.” Another reasoned that “If it is a metaphor for having a pen in your hand, like an extension of your self, that’s why that sounds more intuitive” and “I feel this is the 3D version of the pen I draw.” Some responded positively to the analogy introduced during the workshop that live 3D printing is akin to oil painting: “It is like sculpting. . .the analogy of an oil painting is quite good. . .it felt like that, line-by-line.”

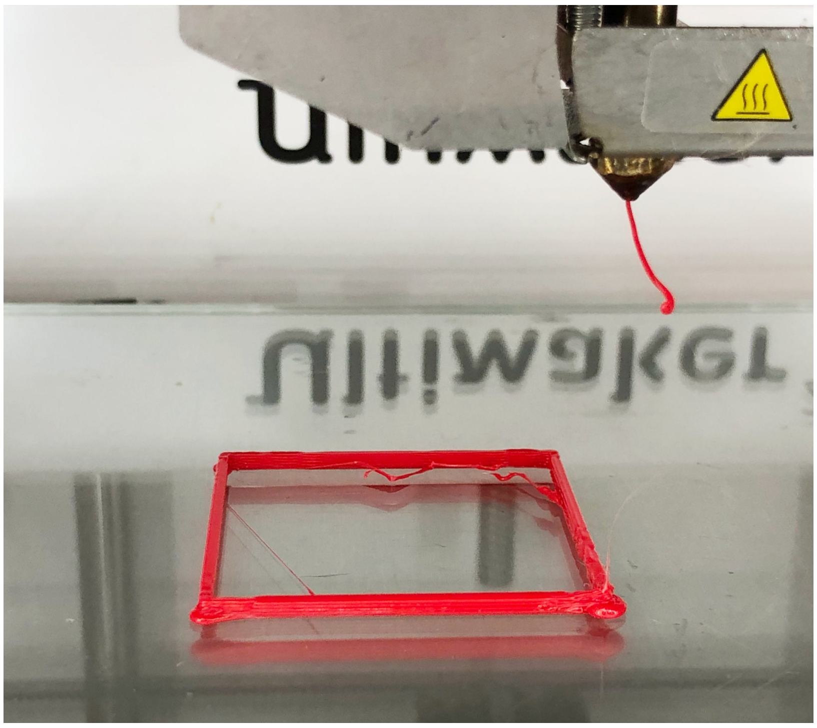


Figure 7.9: A stack of squares drawn by a participant in Subsection 7.4.7 (Task 7: Using LivePrinter (height and layering)). Note the material pooling at the corners due to pauses between printing operations.

7.4.6 Task 6: Using LivePrinter (retraction and material flow)

This task was a modified version of the last square-drawing task, this time drawing a “clean” square by handling material flow properly during live printing using code. The importance of material handling was illustrated in the last task by instructing participants to print shapes slowly using the `extrude()` function, where it quickly became apparent that whenever the print head paused, material would leak out and form plastic blobs on the build surface. To solve this problem, users needed to perform what is called a *retraction*. The term *retraction* is a mechanically-descriptive term describing the act of reversing the filament feeder motor, so it pulls back the filament and prevents leakage of the molten material from the print head. It covers a complex action that has no visible cause because it occurs hidden from the user’s view inside the print head, and a slow effect because the melting of the material takes time to drip out of the nozzle.

The LivePrinter system helped users handle the potential complexity of the material handling process in a few ways. By default, it automatically retracted the filament at the end of movements and unretracting at the beginning. It also provided an option to manually retract and unretract using a *retract: true/false* argument to `lp.extrude()`, or to advance the material explicitly using the `e` parameter. In early workshops, the complexity of predicting when to use automatic retraction and the limitations of the initial use cases quickly became clear and its inability to handle common use cases was noted in the feedback. Retraction and material handling is explained in full detail in the next chapter.

7.4.6.1 Feedback and responses

Participants were observed learning quite quickly about the mechanics of retraction when the print head stopped moving and the "blobs" of plastic formed from leaking material. An unexpected result was that participants enjoyed "playing" with the blobs and creating lasso-like loops of filament. This process is usually an undesired side effect of suboptimal retraction settings in the 3D printing slicing software, and usually users do not get the chance to see blobs being created in such an obvious way because the printing action is so quick and often runs unattended.

There were a number of cases where manual control mixed with automatic retractions put the system into an unstable state. For example, advancing the filament on its own using only the `e` parameter was useful to manually move the filament backwards and so it could be easily changed without stopping; unclogging the print head; or creating blob-like shapes.

Unfortunately, after a manual extrusion it was hard to predict whether or not to automatically retract. Later designs built on the use cases identified in the workshops to heuristically handle retraction, with the option of turning it on and then off again globally as opposed to as an argument in each function call.

Another major conceptual and metaphorical difficulty was, as one participant put it, "lack of some knowledge about the machine (don't know the relationship among temperature/materials/shapes/coding)". Material handling, an invisible process that occurs inside the print head as molten material is pushed out of the head and pulled backwards slightly on retraction, was a difficult concept for most participants to understand. Pulling the material back during a retraction operation *might* prevent leakage, but *how much* and for *how long* based on *how much retraction* and at *what speed*? The overly-descriptive mechanical metaphor fails to cover the basic material handling use case, here.

This was compounded by the difficulty of designing a system that effectively handled retraction automatically and also, manually. Additionally, there was a software bug in early workshops and inherent flaws in the retraction logic for certain uncommon use cases that were finally fixed in May 2020, after most of the user workshops.

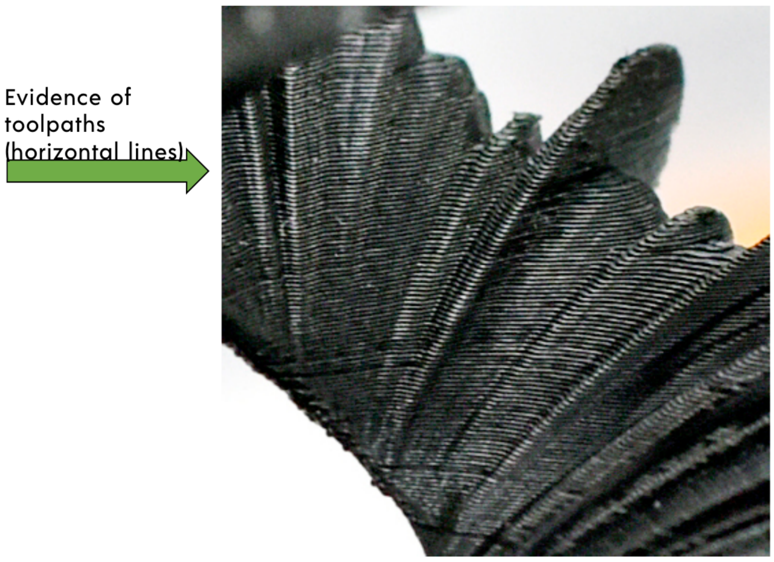


Figure 7.10: An example of how layering is used to create 3D shapes

7.4.7 Task 7: Using LivePrinter (height and layering)

After participants developed some proficiency with mostly 2D drawing operations, this task introduced them to the layering technique necessary to make 3D shapes illustrated in Figure 7.10. Users repeated the square-drawing process from the last task, this time slowly incrementing the height (z-axis) to draw a "stack" of squares, as illustrated in the code in Figure 7.11 and Figure 7.12.

Task 7: code for printing a "stack" of squares

```
lp.lh(0.25); // set layer height

lp.moveto({z: lp.layerHeight}); // move to build height
// draw 5 layers of a square
for (let layers=0; layers < 5; layers++) {
    lp.unretract();
    for (let sides=0; sides<4; sides++) {
        lp.dist(40).turn(90).go(1,false); // draw side
    }
    lp.retract();
    lp.up(lp.layerHeight); // move up to next layer and
repeat!
}
lp.up(40); // move up

```

Figure 7.11: Task 7a: code for printing a "stack" of squares

7.4.7.1 Feedback and responses

Participants continued to have problems with retraction in this task, as with previous tasks but the larger problem was the potential for errors in the two loops of code that printed the 2D square shape and then incremented the height. If there were any errors in the 2D drawing code, or, as was often the case, the participants added extra retractions, the printing process would run slowly with no real way to interrupt it. Often this required a complete restart of the system and the 3D printer itself,

Task 7: code for printing a "twisty stack" of squares

```
        lp.moveto({z:0.2}); // move to build height
        // draw 5 layers of a square
        for (let layers=0; layers < 5; layers++) {
            lp.unretract();
            for (let sides=0; sides<4; sides++) {
                lp.dist(40).go(1); // draw side
                lp.turn(90).go(); //turn for next side
            }
            lp.retract();
            lp.up(lp.layerHeight); // move up to next layer and repeat!
            lp.turn(1); // rotate slowly for next layer
        }
        lp.up(40); // move up
```

Figure 7.12: Task 7b: code for printing a "twisty stack" of squares

another problem of "premature commitment", "progressive evaluation", and certainly "error-proneness".

This was a culprit of the simplicity of the LivePrinter code evaluator and scheduler which ran all code immediately, as fast as possible. The feedback from self-testing and these user workshops was unequivocal that users needed a way to stop operations in progress, but designing such a feature was difficult to implement and required a major redesign of the entire system architecture. This was not completed until May 2020, well after the user workshops, for technical reasons that are outlined in the next chapter.

Participants did grasp one of the fundamental issues in 3D printing, which is how to get the material to stick to the bed during printing. As one participant put it, "The first layer is very important!" This was affected by the print bed calibration, layer height, printing speed, and material temperature in complex ways which are difficult to determine using traditional 3D printing software. This could be seen as successfully rising to Baudisch and Mueller (2017)'s Challenges 2 and 4, around getting people to better understand physical knowledge of materials, mechanics and machine tooling.

Some experimented with different 3D shapes, like stars, and hexagons, and attempts at cylinders. The limitations of the approach of stacking "2D" shapes to create "3D" ones quickly became apparent. Partially, this was due to the limited vocabulary of shape abstractions (of a sort) provided by LivePrinter or supported by the participants prior knowledge of geometry. For many, the geometry of shapes besides rectangles and triangles was outside their ability, leading them to ask: "Can we have more default shape [sp]? Can we stack this as well?"

LivePrinter provided experimental functions that could fill rectangular areas with plastic (see Figure 7.13), as opposed to drawing single lines at the layer thickness (i.e. the *layerheight*). These were not well documented at the time, and the retraction bugs in the software prevented users from exploring them properly during the sessions.

Then there was also the question of the thickness of the lines that formed the outlines, or *walls* of the shapes. Some participants were unclear about

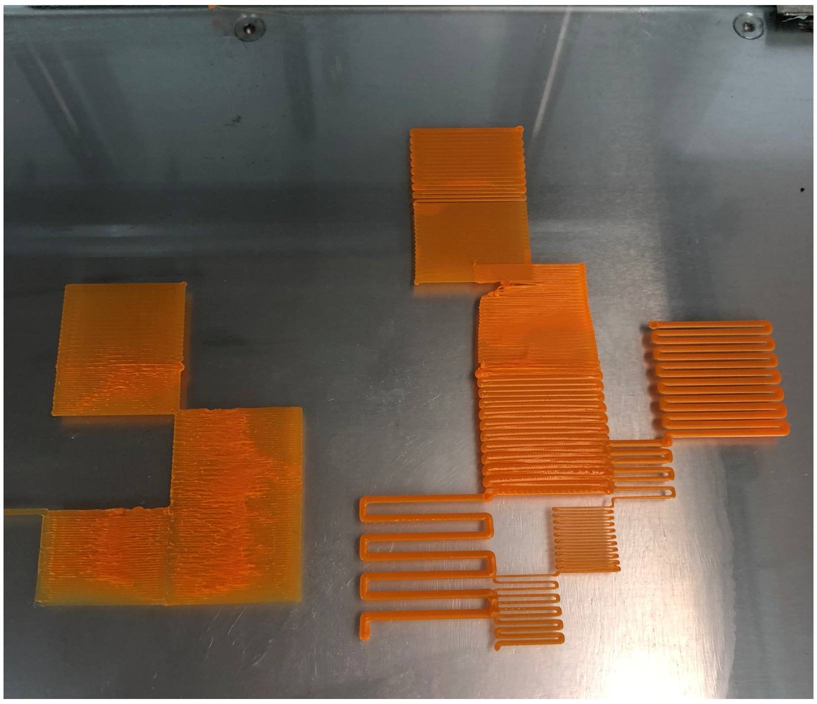


Figure 7.13: Different experimental fills in LivePrinter

how *layer height* was related to the size (diameter) of the printer nozzle, which was much smaller (0.2 mm layer height vs. 0.46 mm nozzle size, respectively). This was not discussed in detail in the workshop because it is not a straightforward concept because it depends on the nozzle diameter and the distance between the nozzle and print surface (either the bed or the previous layer). It can also potentially be affected by the printing speed and temperature of the material, and does not apply to printing when the nozzle is in the air (Gary Hodgson and Moe, 2021; Zuza, 2018). It could be a helpful topic for a follow-up workshop.

7.4.8 Task 8: Using LivePrinter (freestyle drawing)

This task was deliberately left open for individual or guided exploration. Users were encouraged to do what they liked, and many continued trying to create complex shapes like the aforementioned stars and polygons and other types of ropes and blobs. It was hoped that users would by now have enough of an intuitive understanding of the system to get started with more the intuitive *Exploratory I3DP* activity here. Often, they used pen and paper to sketch out shapes alongside their experimental code, again showing the importance of having access to more tactile and free-form "secondary notation". One new feature introduced during this task, in an optional lesson, was the ability to take Scalable Vector Graphics (SVG) shapes and render them into LivePrinter code for use in the live editor.

This was a slightly involved process that only worked on certain SVG files, a few of which were included with the LivePrinter distribution. First, a user loaded the webpage from the LivePrinter server, then selected an SVG file and entered some information about its desired position on the print bed and physical dimensions. The vector graphic was then rendered as both GCode, which could be pasted into any 3D printing

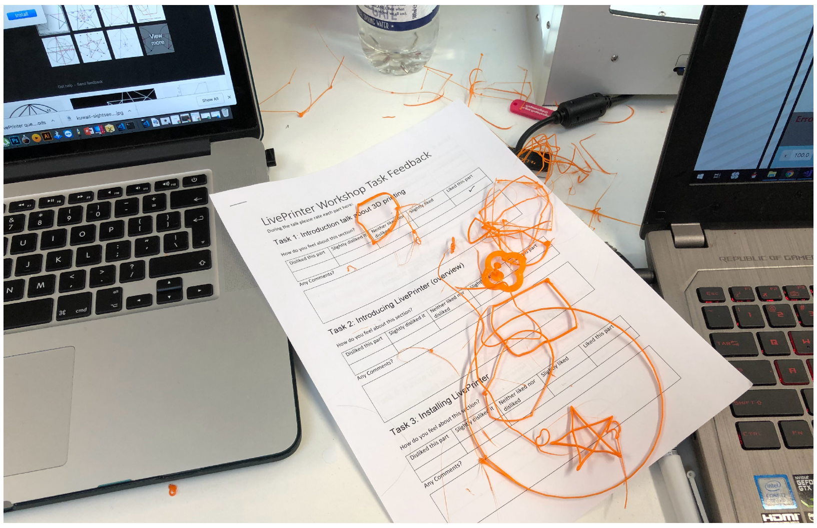


Figure 7.14: Examples of *freestyle* shape experiments printed by participants

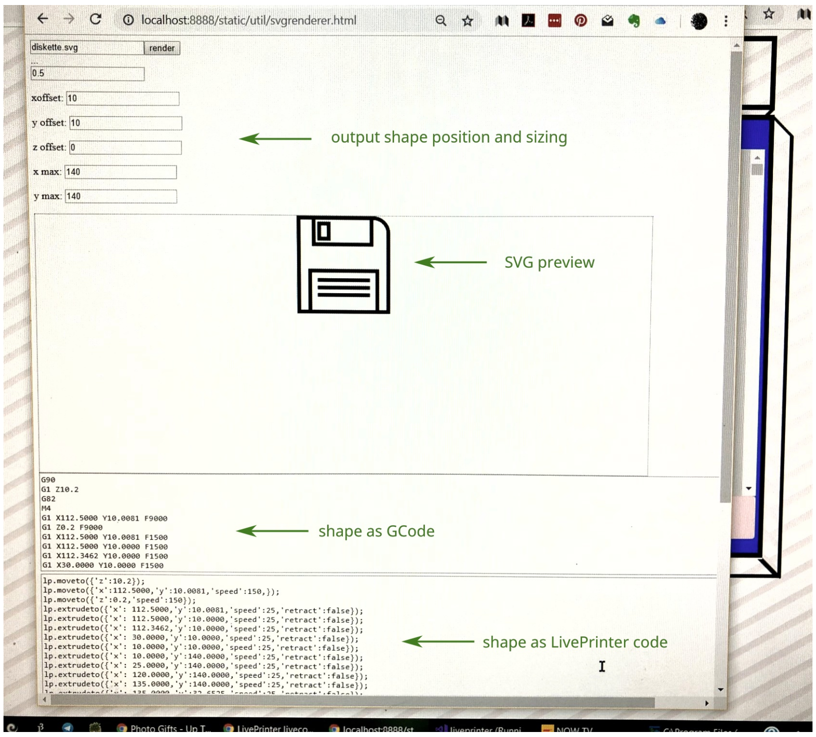
software, and LivePrinter code, which could be pasted into the web interface, loaded properly into the system, and used to print copies of the graphic as demonstrated in Figure 7.15. An example of an SVG designed by the researcher to have non-overlapping shape paths appropriate for 3D printing, rendered into LivePrinter, and then printed on paper is shown in Figure 7.16.

7.4.8.1 Feedback and responses

Participants reported that they enjoyed the SVG rendering feature, despite the complexity of getting started: “Didn’t get round to freestyling - the penguin was just too good :)”. They specifically asked for more “animals” in the database to experiment with. This feature was aimed at beginners, who self-reported as not knowing much about coding, to give them a “quick win” enabling them to successfully print some more visually-interesting objects at different scales and positions. It came at the end of the workshop to make sure that everyone had a chance to achieve something substantial and potentially take away a 3D printed memento, beyond the simple line-drawing and knowledge about 3D printing mechanisms.

Participants did appreciate that this part of the workshop gave them quick results: “The example of the printed SVG was very exciting to see – it gives a quick & clear idea of what LivePrinter” and “Exciting! Lots of possibilities – both by directly coding and also nice to have the .svg option to make it easy to draw complex shapes – smaller learning curve.”

Participants also experimented with printing on a variety of different surfaces: “I can play this whole day! I would try to print on different materials. My phone case, card, fabric, paper”. An example of such a shape, an SVG penguin, was also provided in the workshop as shown in Figure 7.17. This was encouraging, because one of the growing uses of 3D printing is for directly printing on textiles to achieve different textures and material properties, which is not specifically supported by current software.


localhost:8888/static/util/svgrenderer.html
render
output shape position and sizing
SVG preview
shape as GCode
shape as LivePrinter code

diskette.svg

0.5

xoffset: 10

y offset: 10

z offset: 0

x max: 140

y max: 140

```
G90
G1 Z10.2
G82
M4
G1 X112.5000 Y10.0081 F9000
G1 Z0.2 F9000
G1 X112.5000 Y10.0081 F1500
G1 X112.5000 Y10.0000 F1500
G1 X112.3462 Y10.0000 F1500
G1 X30.0000 Y10.0000 F1500
```

```
lp.moveto({'z':10.2});
lp.moveto({'x':112.5000,'y':10.0081,'speed':150,});
lp.moveto({'z':0.2,'speed':150});
lp.extrudeto({'x': 112.5000,'y':10.0081,'speed':25,'retract':false});
lp.extrudeto({'x': 112.5000,'y':10.0000,'speed':25,'retract':false});
lp.extrudeto({'x': 112.3462,'y':10.0000,'speed':25,'retract':false});
lp.extrudeto({'x': 30.0000,'y':10.0000,'speed':25,'retract':false});
lp.extrudeto({'x': 10.0000,'y':10.0000,'speed':25,'retract':false});
lp.extrudeto({'x': 10.0000,'y':140.0000,'speed':25,'retract':false});
lp.extrudeto({'x': 25.0000,'y':140.0000,'speed':25,'retract':false});
lp.extrudeto({'x': 120.0000,'y':140.0000,'speed':25,'retract':false});
lp.extrudeto({'x': 135.0000,'y':140.0000,'speed':25,'retract':false});
```

Figure 7.15: Demonstrating the SVG-to-LivePrinter renderer in LivePrinter

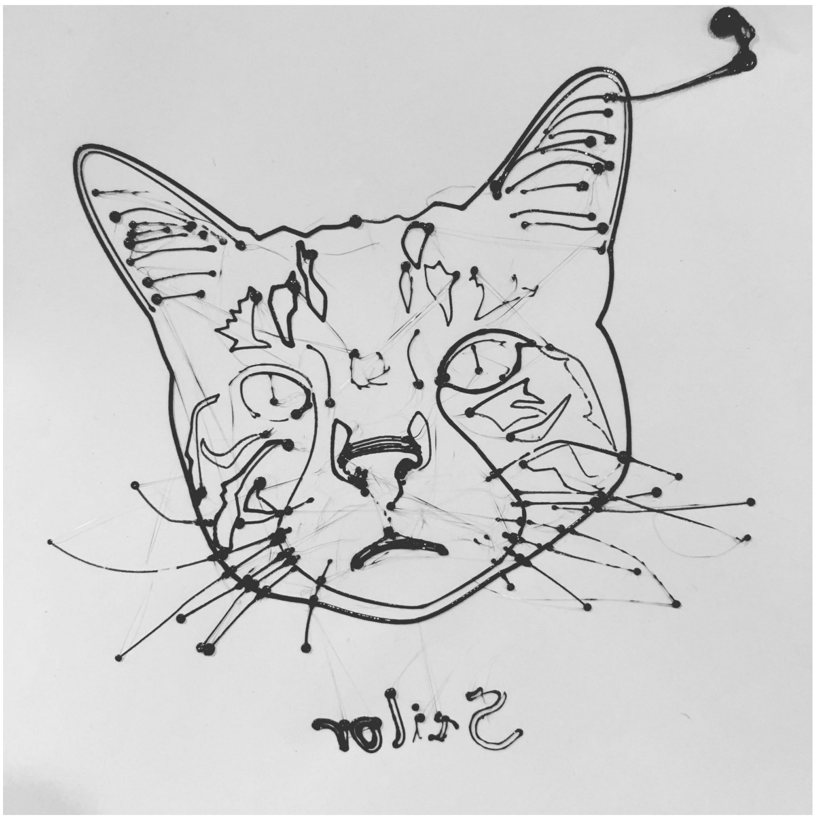


Figure 7.16: SVG created by Evan Raskob and rendered to LivePrinter code, 3D printed using the LivePrinter system.

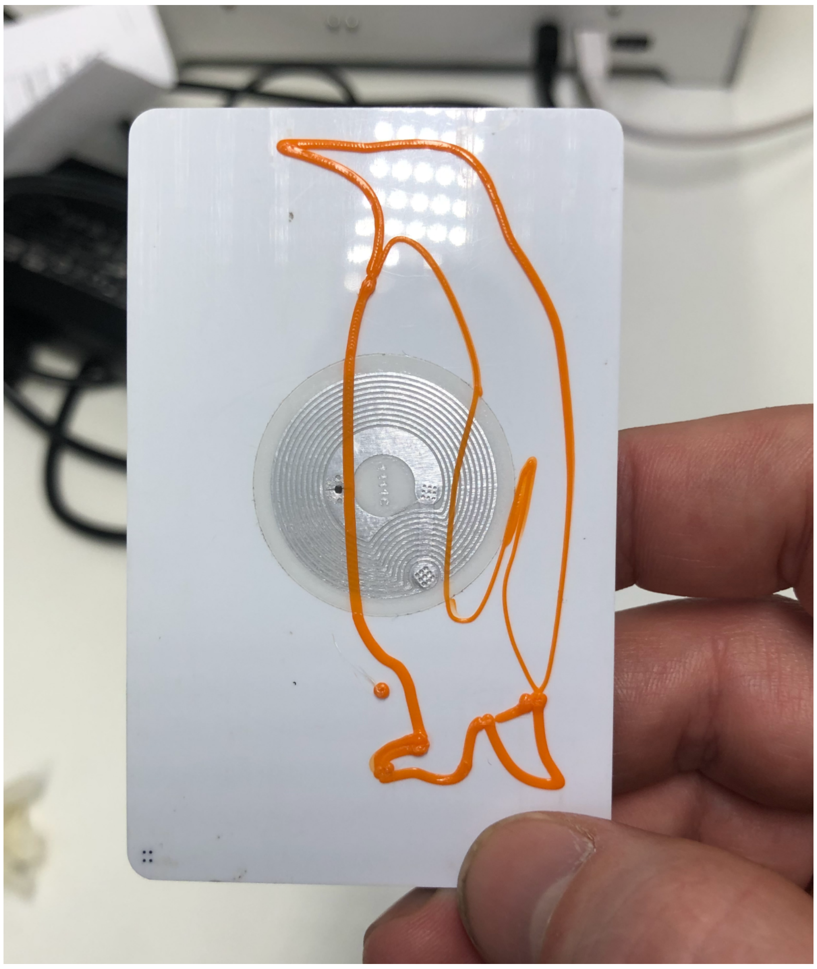


Figure 7.17: Participant's print of an SVG penguin on an access card

7.4.9 Task 9: Future Use

The last task was a discussion led by the main researcher about potential future uses of LivePrinter. Perceived usefulness varied by participant background, with self-identified generative artists appreciating potential new ways to physically create generative shapes; textile artists recognising new ways of experimenting with hybrid forms of 3D printing on fabric; and product designers interested in new methods for combining 3D printing and different surfaces.

Some participants were excited by the system, but asked for more specific direction on how it could be used:

"want to experiment more. got me excited about 3D printing would love a step-by-step instruction for the command line stuff."

"Project has a lot of potential and future use. Maybe some more slides about where this could be used."

Participants who were versed in computational design appreciated the possibilities of coding, but also of combining pre-designed tool paths (e.g. the SVGs provided in the workshop):

"Exciting! Lots of possibilities - both by directly coding and also nice to have the .svg option to make it easy to draw complex shapes - smaller learning curve."

For others, the potential future use was less clear or even non-existent. This was especially true of the self-identified beginners in the study: "No clear direction for any future use yet". One such participant, who started the workshop thinking about buying a 3D printer for their home office, was convinced that 3D printing was actually not something they wanted to do because of its complexity and the limitations about what types of objects it could make. Discouraging someone from buying an expensive and environmentally costly object that they wouldn't use or need could be seen as a net positive. Not all people need to own 3D printers in the future, or present.

7.4.10 User's perspective vs. method naming

Throughout the tasks, the methods "up" and "down" were sometimes a source of confusion for users of LivePrinter, often leading to movement errors. They were originally added as shortcuts to moving the print head more explicitly in the Z (upwards) direction because it was often the case that users would print something and then need to move the head out of the way, so they could see it.

In OpenGL, a graphics programming system, "up" is a function and a vector that represents the upwards orientation of the camera's view. In LivePrinter, it is a descriptive verb with the subject of the 3D printer's print head: **# up 20** literally means "move the print head upwards (away from the print bed) by 20 mm."

The problem of "up/down" shows a difficulty with 3D drawing tools that have multiple moving parts. When controlling the printer live, on many printer models it is the printer bed that moves, not the print head, so up/down may appear to be controlling the bed. When the bed moves down, the gap increases and so the "height" of the print head above the bed increases.

This is also contrary to GCode and model files which use the Z axis to refer to the distance between the print head and bed, from the reference of the print head, because that is what is building the model. The head prints the model, the bed accepts the model, in this metaphor. The height of the model is 0 mm at the bed level (it's "base"), and rises upwards from there. This also refers to drawing "higher up" or increasing the height of an object relative to its base. With object-relative CAD modelling there are no tools involved, so this is not an issue – the object's height is simply the standard measurement of the object in the Z direction. This illustrates the conceptual disconnect between the making process and the modelling process with 3D fabrication.

Unfortunately, the 3D printing print head has no visible indication of which direction it is "facing" according to LivePrinter, so the software must inform the user as part of the livecoding editor. In practice, after the first workshop this led to the moving of the LivePrinter GUI from the bottom to the top of the screen for prominent viewing. Also added was a readout of the current angle, position, retraction and speed.

These were helpful, but still users wanted more visual cues such as a clearly marked "universal origin | zeroed out (bed down, head back left)." Some ran into difficulties when the desired printer movements ran outside the printable area, resulting in truncated forms. One participant suggested printing a ruler onto the print bed itself, which could be achieved by placing a heat-resistant grid underneath the glass print bed, on top of the heating element. Or, as others suggested, using a graphical indicator in the GUI to represent orientation and movement.

Since the tool determines the frame of reference of the printer movements, and not the object being printed, this also has implications when using relative terms like "turn" and directions such as "left" and "right." Specifically, left and right relative to *whom or what?* This came up in background interviews, where one interviewee likened their approach to teaching interactive CNC commands to teaching someone how to drive a car. The CNC tool head becomes a "car" that the user imagines "driving" from the *car's* frame of reference, not the *user's*.

The benefit of the emphasis on "visibility" in LivePrinter's interactive system was clear, in this case. Users could immediately see the results of the commands and quickly develop an intuitive feel for their meaning (i.e. "role expressiveness"). As one user put it: "Down/up can be confusing, as the bed is moving it feels that down/up should be relative to it, but it makes more sense when you actually begin printing things with some depth/height."

7.5 Summary findings: issues for further exploration

Table 7.1: This table lists the major general thematic tags for classifying user feedback. These tags were either related to the themes built into the goals of the workshop tasks or were added later during reviews of the recorded feedback sessions. Most quotations given as examples here are taken from written feedback sheets, with others transcribed from workshop recordings. Observations were recorded in both session notes and after viewing video recordings of workshops.

tag	description	examples	occurences
3d shapes	About creating taller, more 3D shapes from 2D techniques introduced during tasks	"Unclear about how to change shapes from static 2Ds to 3D volumes. [Drawing of straight sides box and curved sided box]? Can we move from a freestyle print to a 3D layering with change"	4
background knowledge	General feedback around the need to understand the context and workings of 3D printing	"Would be good to talk about 3D printers in industry. Is it useful there? What for?"	27
difficulties	A very general tag encompassing all feedback where participants were confused or experienced errors	"This part was a little technical for my level. My partner helped me so it worked fine"	49
documentation	Feedback about the accompanying workshop guide and LivePrinter API documentation online	"Possible having a 'manual' of how to draw our own drawings such as circles, stars, arcs, curves etc."	10

Continued on next page

Table 7.1 – continued from previous page

tag	description	examples	occurences
editing	Having to do with code editing or the interactive code editor	Observation: There was confusion with the interactive editor about running code line-by-line (e.g. highlighting and executing blocks of code) versus running all the code at once (the usual mental model of the code editor for the participant).	2
future	Suggestions or observations about future use of the system	Participants wanted to try printing on different materials, such as on cardboard, metal, student ID badge	11
improvisation	Participants using the system to create new forms not specified in the workshops or experimenting with new techniques	"adjusting layer height during 3D printing is quite fun"; making a bird's nest by accident but enjoying making odd forms "it should be in a museum"; "Great fun. played with blobs"	7
learning	The learnability of the system: problems or examples of good practice	"Run into problems but help was there and it was well explained. Trial didn't work as expected."	8
mental model	Feedback about participants' understanding of the subject	Observation: a discussion about layer height property versus the print head nozzle size. Participant was trying to understand why height was set to 0.2mm, and how that relates to the nozzle. They were unclear about how the filament stacks up to that size.	9
metaphor	About the metaphors used in describing the operation of the system	Observation: The term "chaining" wasn't understood for linking together successive function calls, so "combining" was suggested; Q: Did it feel like you were driving the head or being the head? A: "More like driving a car."	2
mistakes	Interesting accidents or mistakes by participants that lead to insights or interesting findings	Observations: During the manual and automatic retraction task, participants extrude a stray blob of material because they didn't retract after extruding, which they recognise and exclaim "oh I got it"; Participants make a bird's nest by accident and keep on making similar forms: "it should be in a museum"	2
perspective	About understanding the perspective of the movements, such as whether the direction of travel is specified from the perspective of the print head moving or the bed moving	Participant: "So up makes it go down?" Observed confusion over whether the up and down functions pertain to the perspective of the bed or print head	3
play/fun	Examples of participants having fun or playing with the system, e.g. using it for non-traditional or non-productive outcomes	Observation: participants print a large plastic square, tweezer it off and play with it in turns over the next minute, then they take a silly picture using it as a picture frame	8
positive	Feedback where the participant expressed excitement or joy about using the system or was complimentary to researchers and/or the system	"Look forward to playing more with this! I enjoyed hacking the printer make sound etc. Thanks!"	55

Continued on next page

Table 7.1 – continued from previous page

tag	description	examples	occurences
tools	The use of extra tools to support participants	Researchers handed out callipers to help make distances more visible and tweezers to grasp small strings of printed filament, and observed them using them often during workshops	1
units	About the units of measure used (distance, time, etc.)	Observation: participants were unsure about whether movement units were in mm/s (e.g. a velocity) or absolute mm (e.g. a distance)	2
visualisations	Specific requests for making properties or operations more visible	"Maybe have a closed feedback loop with camera so a livecoder can understand better where the head is."	4
syntax	About the syntax and terms used in the API	"quite like the API, not biggest fan of having to input JSON for each move() command but understand why used"	8

Table 7.2: This table lists the major tags used for classifying user feedback that related to developing the LivePrinter software. These tags were used to fix bugs, identify areas for improvement, and keep track of changes to the software that resulted from participant feedback or researcher observations. Most quotations given as examples here are taken from written feedback sheets, with others transcribed from workshop recordings. Observations were recorded in both session notes and after viewing video recordings of workshops.

tag	description	examples	occurences
bugs	Specific problems with the software	Observation: a participant was trying to control retraction to seeing what happens when they retract but don't unretract afterwards. A close look at the print head shows the material isn't coming out because of a logical problem with the LivePrinter software.	14
changes	Suggestions or observations that led to software improvements	From the 1st workshop: "Would like to control it [retraction] manually" so added manual retraction options; "New file option" led to clearing the editor screen	9
connection issues	Issues with communication between the server and printer or GUI	Observation: One participant's backend server loses connection with the GUI and needs refreshing/restarting	2
feature requests	Requests for extra features, operations, or documentation	"Can we have more default shapes?"	12
gui	Having to do with graphical controls in the interface	Observations about participants missing seeing the dashboard at the bottom to set properties like speed, position	6
installation	About the installation process	"Installation instructions were clear and easy to follow, process was fast."	20
vector shapes	Specifically about the use of the SVG shape renderer	"add zigzag line to SVG shapes"	6

Table 7.3: This table lists the tags used for classifying user feedback into common printing activities. These tags highlighted essential 3D printing activites that could be better supported by the software and/or documentation. Most quotations given as examples here are taken from written feedback sheets, with others transcribed from workshop recordings. Observations were recorded in both session notes and after viewing video recordings of workshops.

tag	description	examples	occurences
calibration	Issues arising from the need to calibrate the printer bed and axes first, such as levelling the bed	"Only issue was calibration of the printer [at the start] which made things trickier."	3
cleaning	About maintenance of the printer bed during operation	Observation: participants discussed the need to clean the bed in between operations and keep it clean so the plastic would stick properly: "Needing to clean the bed!"	2
movement	General feedback about printer movements	Usually combined with other feedback such as "perspective" as in the observation about the up/down functions, participants discussed whether it moves "the world" (the bed) or the head	17
retraction	Specifically about the retraction process	"Problem with plastic flow"; "More in depth explanation of when to use not use retraction?"	19
sound	Using the 3D printer to make sound in some way	Observation: "I'll play it!" After the researcher demostrates livecoding music, one participant gets very excited and tries and we work out Mary had a little lamb and other tunes over next 15 minutes	3

Drawing from the difficulties identified in the discussion above and from other workshop observations, there are a few interesting areas of I3DP systems design that could benefit from further research and development.

7.5.1 Context, understanding and use

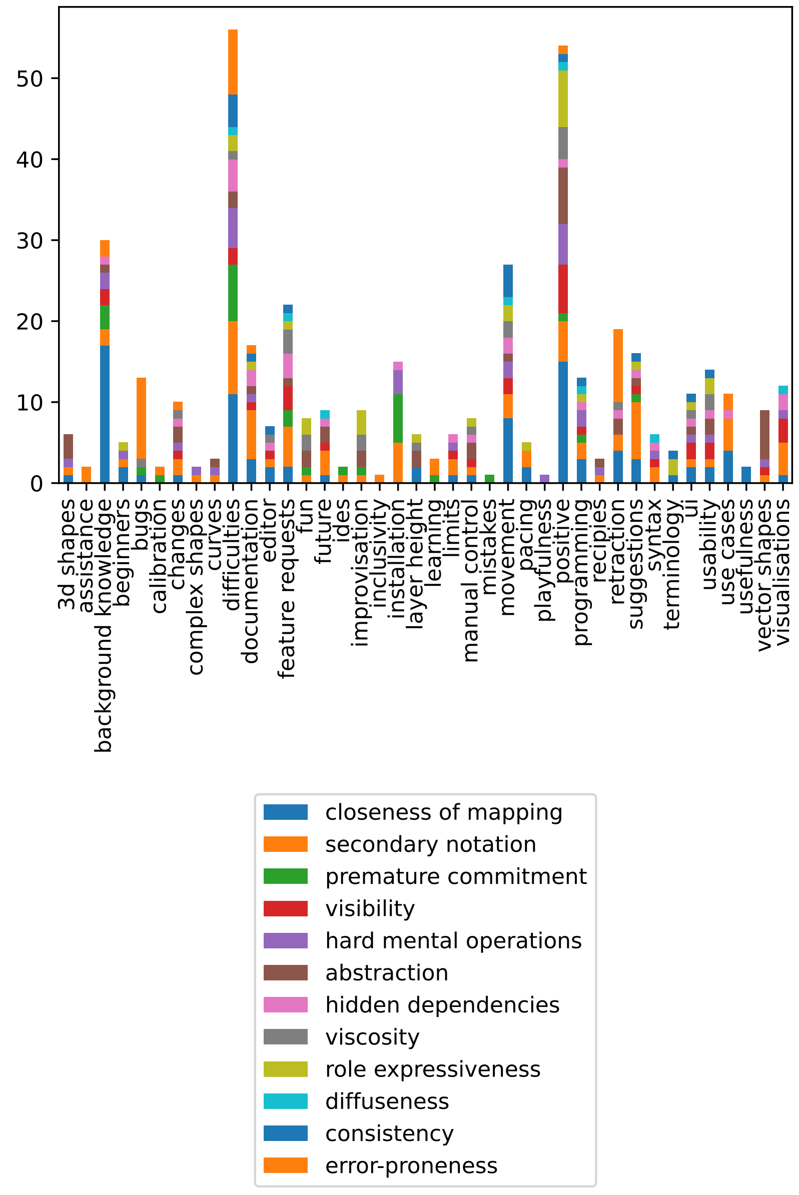


Figure 7.18: This stacked bar chart shows all the occurences of non-CDN tags that were associated with each CDN tag in the research entries to give a broad picture of how they related to themes, software development, and areas of interest. For example, entries tagged with the CDN "closeness of mapping" were also often tagged with "background knowledge", "positive", and "difficulties", underscoring the importance of participants' familiarity with 3D with their understanding of what the LivePrinter system was doing. Similarly, the prevalence of "secondary notation" across the tags hints that participants often relied on the documentation, sketching, and other methods beyond the LivePrinter interface.

Even more so than with screen-based graphics, mechanical systems are hard to abstract because of their complexity, especially with handling molten material flows during time-sensitive operations. This was a key point of Baudisch and Mueller (2017)'s Challenge 4 about "Machine-specific knowledge". Users of LivePrinter needed some background knowledge of retraction, movement, and printer mechanics before starting. During the workshop they asked for more contextual help and ready access to background knowledge in the form of documentation. Once given that basic knowledge, or at least some terminology, they were able to use the system to explore and come up enough of a mental model of the workings of 3D printing to usefully experiment.

Also, the user workshops demonstrated that users wanted to be provided with a broader context in which to situate this new practice of 3D printing. They found that both historical and contemporary knowledge of digital manufacturing was important to understand the difference between I3DP and contemporary 3D printing. The workshops presented some of this historical context, which participants reported as inspiring them, and which we interpreted as being a useful conceptual scaffolding for their work, but there were also requests for more specific contemporary context that we couldn't accommodate at the time.

Part of the problem was that this was the first series of workshops in interactive 3D printing, ever. There were some possible directions proposed in the call for participation of the workshops, but these were mainly provocations. How does one provide examples of a practice that doesn't yet exist? The workshops could have focused more on future applications of this new form of printing, as some initial interviews did. Yet, given the difficulty of getting users to understand the mechanics and physics of printing, they would lively have followed the same unhelpful path of treating the printer as a "magic black box". The excitement generated by the more detailed understanding of 3D printing as a process may be more helpful towards pushing the technology and practice forward than reinforcing inaccurate and overly optimistic notions of the possibilities of what 3D printing is able to print, and how quickly.

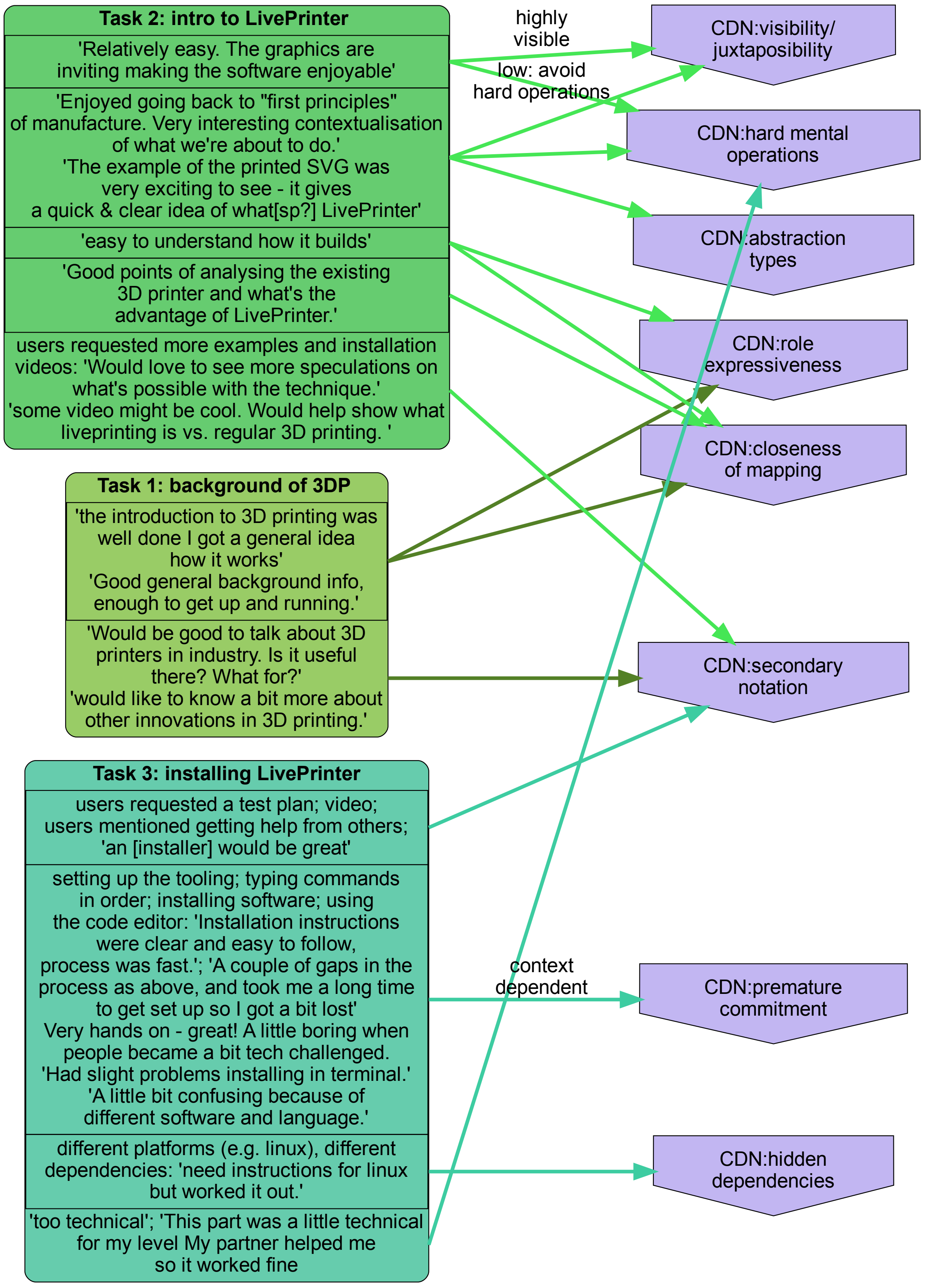
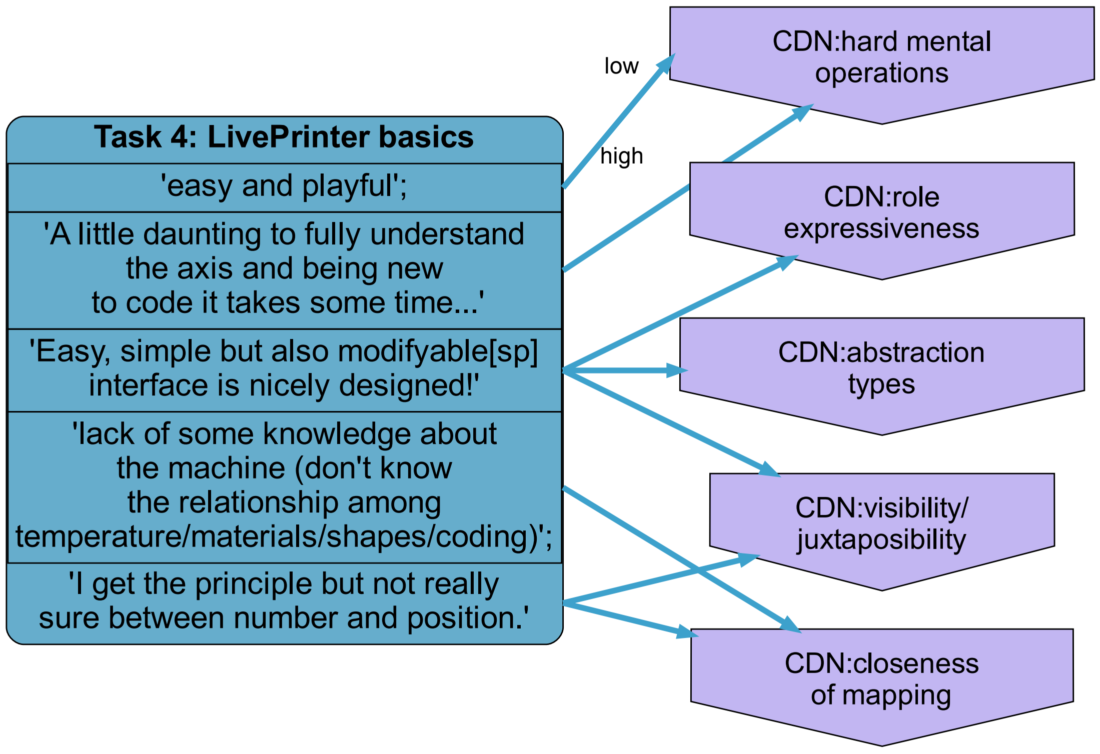

Task 2: intro to LivePrinter
'Relatively easy. The graphics are inviting making the software enjoyable'
'Enjoyed going back to "first principles" of manufacture. Very interesting contextualisation of what we're about to do.' 'The example of the printed SVG was very exciting to see - it gives a quick & clear idea of what[sp?] LivePrinter'
'easy to understand how it builds'
'Good points of analysing the existing 3D printer and what's the advantage of LivePrinter.'
users requested more examples and installation videos: 'Would love to see more speculations on what's possible with the technique.' 'some video might be cool. Would help show what liveprinting is vs. regular 3D printing. '
Task 1: background of 3DP
'the introduction to 3D printing was well done I got a general idea how it works' 'Good general background info, enough to get up and running.'
'Would be good to talk about 3D printers in industry. Is it useful there? What for?' 'would like to know a bit more about other innovations in 3D printing.'
Task 3: installing LivePrinter
users requested a test plan; video; users mentioned getting help from others; 'an [installer] would be great'
setting up the tooling; typing commands in order; installing software; using the code editor: 'Installation instructions were clear and easy to follow, process was fast.'; 'A couple of gaps in the process as above, and took me a long time to get set up so I got a bit lost' Very hands on - great! A little boring when people became a bit tech challenged. 'Had slight problems installing in terminal.' 'A little bit confusing because of different software and language.'
different platforms (e.g. linux), different dependencies: 'need instructions for linux but worked it out.'
'too technical'; 'This part was a little technical for my level My partner helped me so it worked fine
highly visible
low: avoid hard operations
context dependent
CDN:visibility/ juxtaposibility
CDN:hard mental operations
CDN:abstraction types
CDN:role expressiveness
CDN:closeness of mapping
CDN:secondary notation
CDN:premature commitment
CDN:hidden dependencies

Figure 7.19: This diagram maps the workshops tasks 1, 2 and 3 where participants were learning about 3D printing and installing the system, to the CDNs identified in the feedback. A selection of typical or insightful feedback summaries and quotes are used as specific evidence of that CDN. These activities focused on pre-usage fundamentals: (1) building the users' "mental model" of how 3D printing worked which was useful in discussing how close the system mapped to that model through the CDN "closeness of mapping"; (2) introducing terms that users would see in the interface's notation like "retraction", related to the CDNs "consistency" (consistency in naming conventions leading to high levels of guessiblity) and "role expressiveness" (users understand what notion is for through intuitive labels and contextual cues); (3) familiarising users with the documentation and support (which could be considered "secondary notation"). Since these were initial tasks, the aim was to render basic I3DP activities visible to users, and to avoid "hard mental operations" or difficult concepts that would slow them down.



Task 4: LivePrinter basics
'easy and playful';
'A little daunting to fully understand the axis and being new to code it takes some time...'
'Easy, simple but also modifyable[sp] interface is nicely designed!'
'lack of some knowledge about the machine (don't know the relationship among temperature/materials/shapes/coding)';
'I get the principle but not really sure between number and position.'
low
high
CDN:hard mental operations
CDN:role expressiveness
CDN:abstraction types
CDN:visibility/ juxtaposibility
CDN:closeness of mapping

Figure 7.20: This diagram maps the 4th task where participants were using the system (excluding the background and setup tasks) to the CDNs identified in the feedback. A selection of typical or insightful feedback summaries and quotes are used as specific evidence of that CDN.

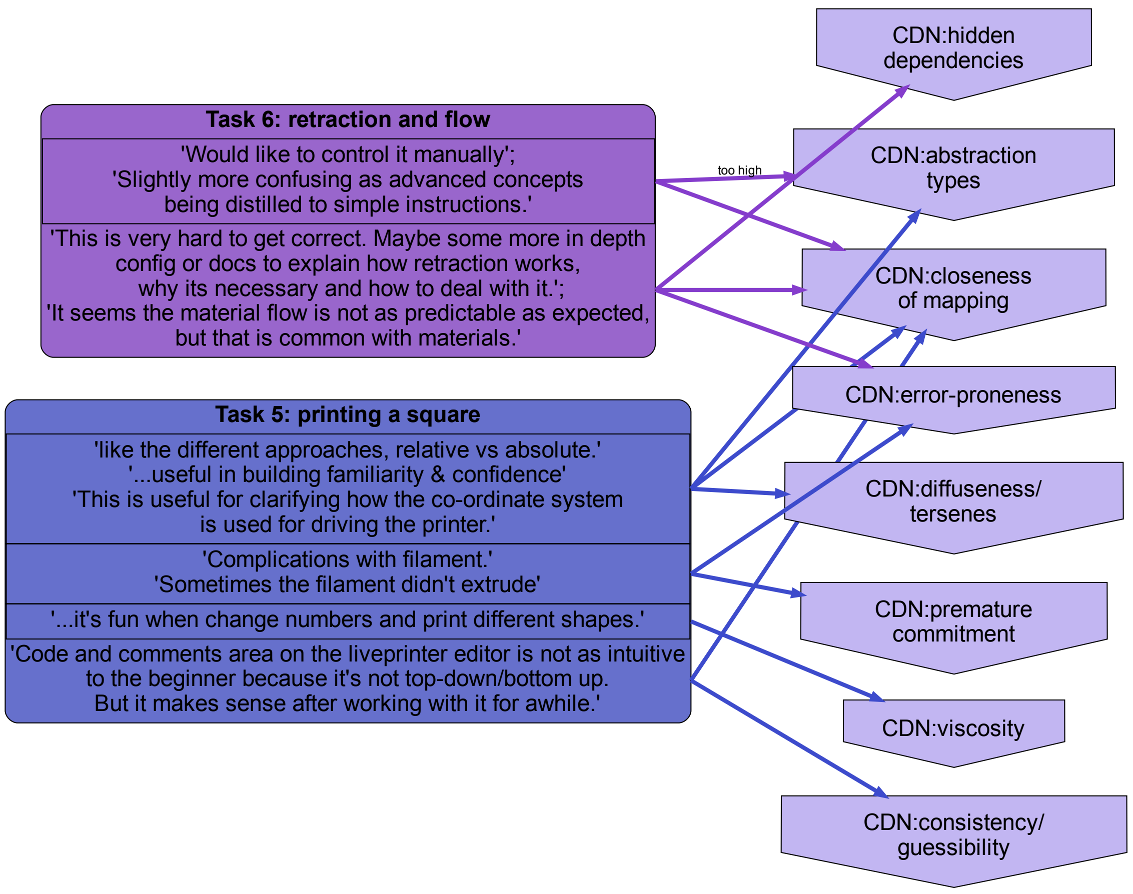

Task 6: retraction and flow
'Would like to control it manually';
'Slightly more confusing as advanced concepts being distilled to simple instructions.'
'This is very hard to get correct. Maybe some more in depth config or docs to explain how retraction works, why its necessary and how to deal with it.';
'It seems the material flow is not as predictable as expected, but that is common with materials.'
too high
Task 5: printing a square
'like the different approaches, relative vs absolute.'
'...useful in building familiarity & confidence'
'This is useful for clarifying how the co-ordinate system is used for driving the printer.'
'Complications with filament.'
'Sometimes the filament didn't extrude'
'...it's fun when change numbers and print different shapes.'
'Code and comments area on the liveprinter editor is not as intuitive to the beginner because it's not top-down/bottom up. But it makes sense after working with it for awhile.'
CDN:hidden dependencies
CDN:abstraction types
CDN:closeness of mapping
CDN:error-proneness
CDN:diffuseness/ tersenes
CDN:premature commitment
CDN:viscosity
CDN:consistency/ guessibility

Figure 7.21: This diagram maps the workshop tasks 5 and 6 where participants were using the system (excluding the background and setup tasks) to the CDNs identified in the feedback. A selection of typical or insightful feedback summaries and quotes are used as specific evidence of that CDN.

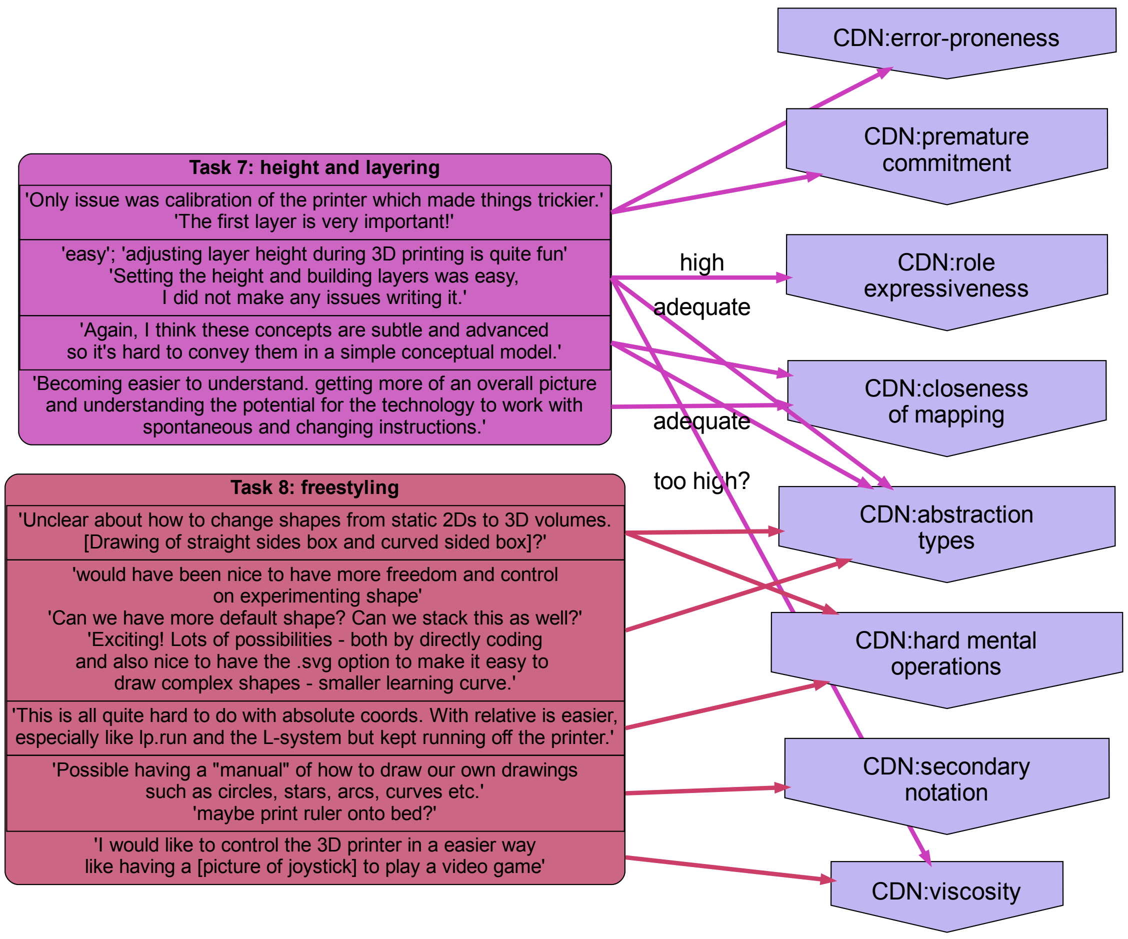

Task 7: height and layering
'Only issue was calibration of the printer which made things trickier.' 'The first layer is very important!'
'easy'; 'adjusting layer height during 3D printing is quite fun' 'Setting the height and building layers was easy, I did not make any issues writing it.'
'Again, I think these concepts are subtle and advanced so it's hard to convey them in a simple conceptual model.'
'Becoming easier to understand. getting more of an overall picture and understanding the potential for the technology to work with spontaneous and changing instructions.'
Task 8: freestyling
'Unclear about how to change shapes from static 2Ds to 3D volumes. [Drawing of straight sides box and curved sided box]?'
'would have been nice to have more freedom and control on experimenting shape' 'Can we have more default shape? Can we stack this as well?' 'Exciting! Lots of possibilities - both by directly coding and also nice to have the .svg option to make it easy to draw complex shapes - smaller learning curve.'
'This is all quite hard to do with absolute coords. With relative is easier, especially like lp.run and the L-system but kept running off the printer.'
'Possible having a "manual" of how to draw our own drawings such as circles, stars, arcs, curves etc.' 'maybe print ruler onto bed?'
'I would like to control the 3D printer in a easier way like having a [picture of joystick] to play a video game'
high
adequate
adequate
too high?
CDN:error-proneness
CDN:premature commitment
CDN:role expressiveness
CDN:closeness of mapping
CDN:abstraction types
CDN:hard mental operations
CDN:secondary notation
CDN:viscosity

Figure 7.22: This diagram maps the workshop task 7 where participants were using the system (excluding the background and setup tasks) to the CDNs identified in the feedback. A selection of typical or insightful feedback summaries and quotes are used as specific evidence of that CDN.

7.5.2 Appropriate visibility and abstraction levels for physical processes

When asked if they understood what was going on inside the 3D printer one participant responded: "I understand the layers of abstraction but you do lose the idea of how it is working, especially with retraction which is a very in-depth 3D printing thing. Things like that should be more in-depth. It did. . .magic it away a little bit."

There were outstanding questions about the abstraction level of the names of some API methods used to direct the making actions of the printer. API commands were designed to have an starting abstraction level similar to the GCode which they would be compiled into, but with more descriptive names. The low-level functions like "extrude" and "draw" (synonyms mapped to GCode command "G1 X Y Z E SPEED") were clearly helpful in understanding how 3D printers work. They were also too low-level for making more complex forms, as evidenced by the users' preferences for ready-made shapes like SVGs that they could manipulate and combine. What these higher-level abstractions should be like is an open question for future research, and will depend on the context in which they are used. One could see 2D forms being useful for printing on textiles, whereas 3D building blocks would be very useful for sculpting and designing 3D objects on-the-fly.

In most 3D printers the print head lacks a visible indicator of its direction of travel. They also often lack any indicator of position in the 3D printer cavity. This was a recurring theme in the feedback, and during observations, as discussed above. Participants wanted a more visual indicator of print head movement and position. They were given a textual read out of the print head position at the top of their screens, and, with practice, could learn generally where the print head was currently located to and where they would like it to move to, but the lack of a more precise indicator was observed to have slowed them down during the making process and led to printing errors.

Where to put such a visualiser in the interface was a difficult question, adding to the trade-offs in designing a user interface that fit comfortably on the screen whilst rendering visible enough of the key printer properties exposed by the LivePrinter API. The competition between the "dashboard" element at the top of the screen, with the print head and temperature properties made visible and modifiable, with the interactive code editor and the event and information log on the right side of the interface led to a few adjustments during the workshops. Participants' attention was divided between looking at the code editor and physical printer next to them, which is why many of them suggested combining the two by adding projected or other dynamic visual overlays on the printer.

7.5.3 Asynchronous dependencies

These issues of transparency and visibility were coupled with issues of immediacy, which was apparent when users had trouble connecting to the printer during the workshops and couldn't determine whether their actions were having any effect because of lag and lost messages.

This problem stemmed from the "hidden dependencies" introduced by the asynchronous nature of interactively programming a machine, which involves writing code, queuing that code to be run in the correct sequence, compiling it, sending it to a remote machine, receiving a response, and displaying that response to the user. The proper implementation and debugging of asynchronous interactive programming systems like this one is a non-trivial exercise, and took up a large amount of development time. It was an issue that extended beyond the user workshops, into later live performances before being successfully reconciled towards the end of this thesis.

7.5.4 The importance of easy installation

Despite having step-by-step instructions, many participants who were inexperienced with software development had difficulty installing LivePrinter and its dependencies (Python; optionally Visual Studio Code or Visual Studio). The process was expected to be fairly trivial, asking them to run an installer, copy some files, and then open a terminal prompt and type a command, but many were observed to be hesitant and uncertain due to the unfamiliarity of the terminal and terminology used, which is reflected in the mapping of the first 3 installation and setup tasks to associated CDNs in Figure 7.19. This experience, as a whole, speaks to a larger issue where the interdependent tool chains and specific terminology of software development acts as a bar to entry for people who want to use experimental software.

More simplified installation tools are available, as users commented, but the time costs of repeatedly packaging up new versions of software and reworking the installation documentation in multiple formats is not trivial for small development teams. This is an issue that will be familiar to any maintainer of a small, community-focused software project. Also, users asked for multi-modal installation documentation such as video instructions, as they were less comfortable with the mixed text-and-screenshot approach and wanted to step through the installation in a more visual, self-guided way. Again, video is time-consuming to record and edit, especialyl with experimental software that changes week-by-week.

7.5.5 Sustainability, or not everyone needs a printer

One of the secondary goals of this project was to reduce the materials usage and energy costs of 3D printing. What better way to do this than to convince people that they don't need 3D printing as part of their practice? At least one of the participants in this study reached just such a conclusion, which we count as a success. 3D printing has its benefits and drawbacks, and practitioners should get a chance to understand the trade-offs between them before incurring the personal and environmental costs of buying a pre-manufactured printer, expending energy experimenting with it, and then abandoning it when they find it too complex or inappropriate for their work.

7.5.6 Interactive programming is not always intuitive

It bears repeating here that not all participants understood the practical technique of interactively programming by highlighting lines of code on the screen and then hitting a key combination (or pressing a GUI button) to compile and run them. This was surprising at first, but should have been especially obvious to those who taught coding that the majority of participants' experiences with IDEs were with the $write \rightarrow compile \rightarrow run \rightarrow$ debug archetype where the entire program is run at once, not in pieces. Whilst they were able to learn this technique fairly quickly, this still shows that the "role expressiveness" of interfaces depends on experience.

This presents opportunities for revisiting the "line-by-line" execution model of interactive programming, possibly by taking advantage of the underlying system where blocks of code are sent to a queue for asynchronous processing. Users seemed to want more control over how that code is executed, specifically by making judgements about which code to re-execute, or stop executing.

7.5.7 The missed opportunity of modification?

The case of the CDN activity called *modification* was an interesting one, because it implied the possibility of changes to the notation and notational editor itself. Self-modification of a sort is natively provided by most web browsers, which run their own self-development tools that allow users to reprogram most of the web interfaces in JavaScript or to extend them via "extensions" or "plugins". It was also supported in the design of the LivePrinter API, which was itself written natively in JavaScript.

In the user testing workshops, users were made aware of this possibility, but few used it to create their own forms of notation or to alter the web-based notational environment. Since this study focused on the basic use of LivePrinter and 3D printing in general, most of the workshops focused on basic 3D printing terminology and techniques, but some more experienced users were observed creating their own functions for drawing different 2D polygons.

One user, with a background in livecoding, programming functional languages, and creating livecoding tools, was interested in creating their own extensions for the LivePrinter system itself and was pleasantly surprised when it worked as they expected. These results were encouraging and confirmed that some users would like to extend the system's notation, but since they followed well-established techniques from writing JavaScript they weren't explored in more depth during the study aside from assessing whether users might find them useful or not.

Interestingly, participants had the opportunity to propose changes to the documentation via the "GitHub issues" system, or through a "push request" to the open source code repository, or even via emailing the researcher with proposed changes. Yet, in workshops with LivePrinter, users preferred to discuss changes to the documentation directly with the researchers and to have the researchers implement the changes, rather than propose them using the online tools and methods.

The reasons behind users' hesitancy to modify the documentation and notational system itself are unclear. Contributing modifications does require a certain level of skill, knowledge of process, and time commitment. There also might be issues of ownership and authority. More studies are needed here.

7.6 Summary findings: issues with user study design

Beyond issues with the design of the system, we identified some potential issues with the design and evaluation of user study itself.

7.6.1 Problems with individual user types

As Cooper et al. (2014) points out, *user types* or *Media Usage Types* (MUTs) are often important for understanding the results of any user study. The Mozilla Organisation has an excellent typology of their users that we have used in our teaching, and that helped to inform our expectations of potential participants in this study (Selman, 2013). Yet, we found it difficult to build our own coherent user types that might be generalisable to wider populations. Part of this can be attributed to the small samples sizes, diverse group of participants, collaborative working, and limited time in which we had to run the study (as opposed to the larger team and timescale of Mozilla's study).

As Brandtzaeg, Heim, and Karahasanović (2011, p. 952) explained, such typologies cannot ever be truly comprehensive:

> "...no real world–classification systems meets this requirement because mutual exclusivity may be impossible in practice...there probably will exist hybrid user types that are combinations of the initial ideal types defined, because the same users could be defined as different user types in terms of various media platforms: A 'sporadic SNS user' might, for example, also be an 'instrumental user' in 'general media'. In other words; the same users can have different user profile types depending on the platform."

Before each workshop, surveys were used to collect some user data about participants' level of skill with programming, their experience of 3D printing, some background information about their age, occupation, and experience with physical design. This data was not used in the final analysis because discussions with participants highlighted that each had unique combinations of professional and life experience, being of different ages, and coming from the public, interaction design, product design, art, computational art, computer science, and other unrelated disciplines.

In workshops, we observed some of the more experienced programmers creating algorithmic structures (like stars and stacks of 2D shapes), but other participants of all skill levels were content to experiment with the capabilities of the system, and still others only played with simple forms or spent their time helping others. Participants in general often collaborated with one another and helped each another, partly due to the limited number of 3D printing machines available. These arrangements speak to theories of knowledge as more of a social construction, like socio-constructivism (Amineh and Asl, 2015; Glasersfeld, 1995), and raise questions about the value of individual user types, especially in smaller studies like this one.

7.6.2 Difficulties with user-facing CDNs

It was difficult to find a good solution to the problem of providing fully-detailed CDN questionnaires for users to feed back on and discuss during and after workshops. Having to provide a questionnaire or discussion outline in a longer form, in multiple formats (as in Clarke (2010, pp. 545–565) and Bernardo et al. (2020)), would take a non-trivial amount of time for the researchers and users and would not be practical in most workshops, which were already quite full of activities. In the example CDNs questionnaire provided by Blackwell and Green[12], the list of CDNs is relatively long and verbose, which could bias it against types of neurodiverse participants and more verbal and visual learners who have trouble with the general terminology and long paragraphs. Also, in our experience, at the end of a long user workshop users had limited energy left for an in-depth discussion. We had difficulty getting through even the more limited list of CDNs with shorter, more relevant descriptions and specific discussion prompts.

12: At https://www.cl.cam.ac.uk/~afb21/CognitiveDimensions/CDquestionnaire.pdf

The alternative of delaying the questionnaires by having a guided discussion at a later session, or giving it to users to fill in after the workshop, would create some distance from the actual tests that would change the nature of the results. Adding a delay would rely on the selectiveness of longer-term memory rather than prompting participants to reflect on recent experience. Scaling the size of the research team and the apportioning the CDNs across larger groups of users could be another solution, but these are not often possible for small and lightly-funded research teams, and contrary to the spirit of a universal and easily-applied discussion tool.

In practice, during the guided reflection at the end of the workshops only the CDN *visibility* was a useful discussion prompt. Participants found the other CDNs hard to reflect on, likely because of the meta-cognitive nature of questioning their own activities, whereas they could comment directly on their personal experience of the visibility of elements, from their own point of view. This is similar to Blandford and T. Green (1997)'s findings with user workshops based around *Ontological Sketch Models*, where participants' findings on meta-cognitive constructs (relationships, applied models) was highly variable, often inappropriate, and more based on direct experience than applying a model to their experience (Blandford and T. Green, 1997, p. 9).

Participants also had difficulty identifying bottlenecks and sources of friction because they were still learning the system and lacked a frame of reference for how quickly they could operate it in ideal conditions. It is not even clear that users and system designers will agree on what are actually bottlenecks in the system – after identifying the time-consuming but necessary process of organising visual patching notation on the screen ("tidying up the layout of the code") A. Blackwell (2005) relate an anecdote from their research where a participant has gone so far as to hire a member of staff just to that task, saving them time and giving their customers a better end product. Clearly, A. Blackwell (2005) consider this a bottleneck inherent to visual programming systems, but it is not clear that the participant sees it as anything but part of the cost of doing business and making his product better. Without another frame of reference for the same task, who is to say what is a bottleneck versus the typical way the system works?

Regardless, our research results showed that whilst users might lack the specific terminology of the CDNs, they could articulate the concepts well enough to be picked up on by the research term during more free-form discussion, in response to open-ended prompts. This is self-evident in the details of the analysis presented in the last chapter, and in the following sessions. Even though the research team was instructed to focus on four initial CDNs, which were also presented to users — *Secondary notation*, *Premature commitment*, *Viscosity*, *Visibility*[13] — our final analysis touched on almost all of them.

13: See the interview prompts in Subsection B.1.5 (Interview & Questionnaire Plans)

7.7 Conclusion

As demonstrated in the user research workshops, a successful programming tool consists of consistent metaphors, a functional editor to give context to that code along with visual feedback on the effects of the code and the state of the system, but also importantly some background knowledge of the system and its usage as well as use cases for future exploration. The results of the workshops demonstrated that LivePrinter provided all of these, at a level that was appropriate enough for participants to make real experiments with interactively-programmed 3D forms.

This study further highlighted the hidden knowledge embedded in the "usual" 3D printing process that was not readily explored nor normally exposed to users. LivePrinter gave users direct experience of how material flow and printing speed affects printing operations, and how layering works. It also underscored the complexity for making shapes live, step-by-step, and the need to provide new practitioners with more building-blocks and frameworks to support that exploration, especially in 3D.

Most importantly, it showed that 3D printers are not magical devices nor black-boxes that fabricate full-objects but logical extensions of previous technologies and techniques that use materials and layering, like painting and sculpting. This gives people the conceptual tools for understanding how 3D printing can be used in more radical and unexpected ways, even with a mixture of coding and manual material manipulation, as opposed to the incremental improvements to the machines that we have witnessed since the birth of the RepRap almost 20 years ago.

Filling space, filling time: experiments with I3DP 8

In a musical performance, sounds fill up time, divided into notes.
In a 3D printing session, the sound of a printer's movements fills up time and space, divided into objects emerging from layers of fused material.
Motors in action vibrate, making sounds; people watch: the act of making is an act of performance.

8.1 Note

This chapter is written in the first-person form because it documents the reflections of myself, Evan Raskob A.K.A. BITPRINT A.K.A. BITLIP, the author of this thesis and simultaneously a practitioner of computational art and design, educator, and software developer. Whilst I am part of the livecoding and academic communities, whose influence in both my identity and activities cannot be overstated, these works are part of my individual creative practice and are examined mainly from a self-reflective point of view.

8.2 About

*This chapter reviews experiments with interactive 3D printing combining visual aesthetics, physical structures, musical concepts, and live performances. Reflecting on the experience of developing the supporting software and planning and performing these experiments led to some new insights about new computational techniques for interactive 3D printing and livecoding, as well as more general thoughts on how to improve the shared aesthetic experience between performer and audience of long, interactive digital manufacturing workflows.

In particular, I was concerned with the question of what happens when we borrow from livecoding's philosophy of radical transparency and take into account not only the *products* of an interactive fabrication process, but also the aesthetic experience of the *process itself,* as it is co-experienced simultaneously by performer and by audience alike? Furthermore, what are some promising aesthetic possibilities for these types of performances? And what computational techniques support them, that we can use for simultaneously making both sound and form in a live setting where time is of the essence and concentration is a limited resource?

To fully explore these questions, I found it useful to re-think how computationally-fabricated objects are experienced. They are not only spatial objects, products of a computational process that can be measured in millimetres, but also exist temporally in terms of the duration of the

* A full listing of all experiments and research activities can be found in Section 4.2 (Introduction).

movements that made them. These movements possess their own inherent aesthetics of shape, sound, and feel, and represent new opportunities for computational composition.

Movement is an integral part of the experience of live performances, for performers and audiences alike. There is an undeniable beauty in watching an orchestra of string instruments finishing a performance on a long note, extended bows initially pointed anxiously like rows of porcupine quills, then sinking cautiously as the tension of the note resolves into a peaceful quiet. Similarly, in other performing art practices such as dance, the emotional experience produced by the influence of physical movements on an audience (and performers) is well recognised and has been studied extensively (see (Camurri, Lagerlöf, and Volpe, 2003) for such an example).

In the Algorave and other live performances, the focus was on making musical sounds exclusively using printer movements, whilst projecting a live visual feed of these movements to the audience. The technical challenges to creating a new system supporting this mode of working were considerable, including the performance setup of camera and microphones. Besides the core I3DP functionality in LivePrinter, the live performances necessitated the further development of notation that supported musical expression – scales, notes, and silent pauses.

The addition of musical performance also raised the question of how to handle melodies consisting of continuous runs of musical notes whilst also making continuous physical forms, without inadvertently destroying those forms during the performance by stray printer movements. This led to the development of algorithms for continuous curves, based on Hilbert curves, and other generative, 2D, space-filling algorithms roughly inspired by cellular automata such as Langton's ant and the "light cycles" game from the Disney film, Tron (1982). Then, experiments were extended into 3D shapes to explore how LivePrinter and other I3DP systems could streamline the experimentation process for developing new, complex forms.

In the following sections I reflect on this re-thinking of the process and products of I3DP as it was realised and evolved as I completed a series of key outcomes – performances, exhibitions, and experiments. In the process, I revisit my experience of participating in them, viewed through the different lenses of the I3DP activity clusters established in Section 5.6 (Determining user activities): *Exploratory I3DP*, *Experimental I3DP*, and *Livecoding I3DP*. I also discuss the computational tools and techniques that were developed to support each one, which can be thought of as "building blocks" for 2D and 3D form-making: functions for describing objects in terms of sound and duration; repeating geometric fill patterns; space-filling curves; and simple generative algorithms with low computational complexity that can be interacted with in real-time.

Activity clusters for I3DP can be found in: Subsection 5.6.2 (Activity Cluster: Exploratory I3DP) – quickly sketching ideas, mainly intuitive; Subsection 5.6.1 (Activity Cluster: Experimental I3DP) – relatively quick but more reflective and deliberative, focused on specific experiments; Subsection 5.6.3 (Activity Cluster: LiveCoding 3DP) – improvised performance focused on intuitive improvisation, combined with rote repetition of learned phrases and precomposed pieces.

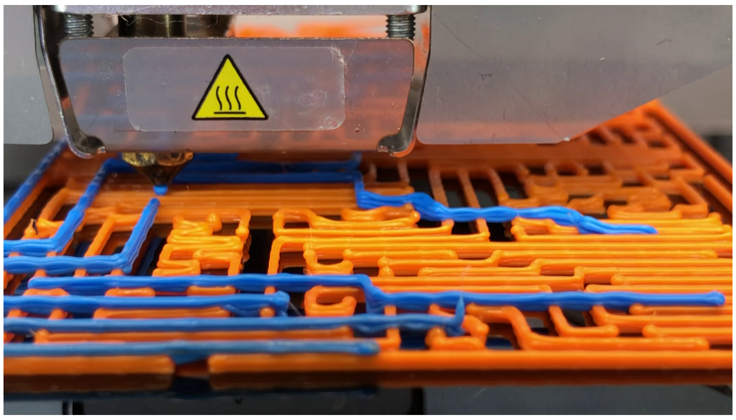


Figure 8.1: Making a sculpture for LDF 2019

8.3 Background

Before describing these experiments and their outcomes, it is important to review some fundamental concepts and prior art of 3d-printing-as-musical-score-and-performance. To understand how a 3D printer's form-making process can be intentionally used to make sounds and compose music, we first need to understand how exactly 3D printers "fill up space" to create freestanding, solid structures from molten plastic. It is also helpful to explain in technical detail how exactly the LivePrinter system supports this form of structural and musical expression, to define some common operations and terms that will later on appear in reflections on experiments. Finally, there are some mathematical concepts to introduce around continuous curves that show promise for solving some challenges of I3DP.

8.3.1 Note To Speed, Speed To Distance

To review, the 3D printer has 4 digitally-controlled motors. They move the print head side-to-side (x), forward-backward (y), the print bed up-down (z) and feed and retract filament (e). Often, they are the same model of motor and have identical properties. When the motors spin, they emit sound that can be mapped to notes in the equal temperament scale used by MIDI synthesizers using some simple linear scaling in the following JavaScript-like pseudocode:

```
// calculate the frequency of the note from
// MIDI note number:

frequency = Math.pow(2.0, (note - 69) / 12.0) frequency =
     frequency * 440.0

// convert to motor speed in millimetres
// per second for GCode (see Table 1)
```

X axis	Y axis	Z axis
47.069852	47.069852	160.0

Table 8.1: Typical speed scale for x, y, z axis values for the motors used in the Ultimaker 2 printers to convert their speed into musical notes. From Westcott's MIDI-TO-CNC library (Westcott, 2015). Note that no values were given for the filament feeding (e-axis) motor.

```

speed = frequency / speed_scale_for_axis
```

In this example, `speed_scale_for_axis` is a simple fraction determined experimentally.

There is a caveat to this simple distance function – 3D printer motors do not have unlimited torque, and so they take a brief but perceptible time to accelerate to full speed. That means that, in practice, movement durations are lengthened as the movement speed increases. In our experiments, this was perceptible around MIDI notes 81 and above and caused synchronisation issues when we tried to pair the printer with other musical equipment. The acceleration curve for movements depends heavily on the mechanics of the printer, the type of motor, the motor driver, and any firmware-level acceleration settings, so the exact amount of lag would need to take all that into account.

Knowing the travel speed of each motor that produces a desired musical note, along with the desired duration of that note, one can calculate the distance of travel across each axis by using a simple movement equation:

```
d = st
```

where d is the distance in mm to be calculated, s is a scalar representing the speed of the print head in mm/s in the current direction of travel, and t is the desired movement time in seconds. The first, called `_midi2speed(NOTE)` or shortened to `m2s`, converts a MIDI note `NOTE` into a corresponding motor speed in mm/s, for one axis. Then, a function called `_time2dist(TIME)` or the shorter `t2d` can be used to convert that speed and a desired duration of movement_ `TIME` into a movement distance. The following pseudocode uses these two functions to move the print head making a pitch of MIDI note C5 with a duration of 1 second (1000ms):

```
lp.m2s(72).t2d(1000).go()
```

In the minigrammar, this is more tersely written as:

```
# m2s 72 | t2d 1000 | go
```

These two functions evolved over time from a single function that combined both operations:

```
lp.note(note=40, time=200, axes="x")
```

In the minigrammar:

```
# note 40 1000 | go // defaults to x axis
```

The flexibility of the 2-function version is a trade-off in verbosity and increased viscosity that makes it more visible to the user that once the printer speed is set it remains set for future operations, whereas with the single function it is implied that the speed stays the same. For example, drawing a square using two notes:

```
## m2s 72
  | t2d 1000 | turn 90 | go 1
  | t2d 1000 | turn 90 | go 1
  | m2s 74
  | t2d 1000 | turn 90 | go 1
  | t2d 1000 | turn 90 | go 1
##
```

8.3.2 Space Filling Curves

One solution to the problem of crossing tool head movements is to use algorithmic tool movement paths that can completely fill up spaces in predictable ways without crossing paths. This concept of non-crossing *space-filling curves* originated in around 1890 with G. Peano (Peano, 1890), but it was the mathematician David Hilbert who laid most of the

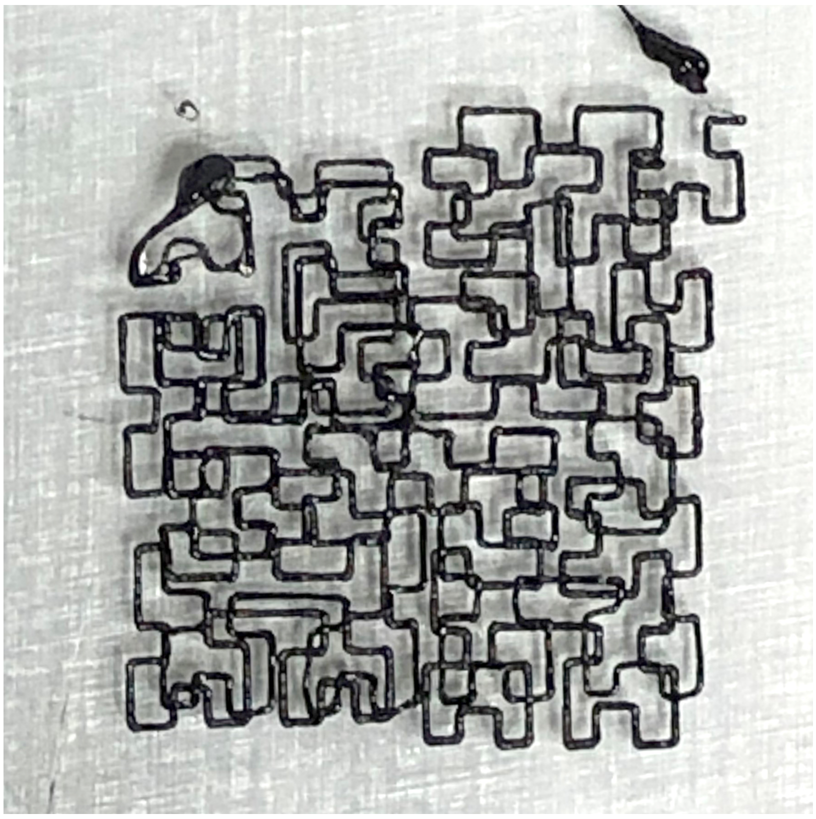


Figure 8.2: The result of a 3D printing performance using a Hilbert curve as a form for musical structure

mathematical foundations for a class of curves that we call the Hilbert Curves (Hilbert, 1891; Sagan, 1994).

Hilbert curves have the property that they map a one-dimensional space to a multidimensional space by passing through every point in that space once and only once, in a single, continuous curve. We focus on it here because of this non-crossing property, but also for its inherent visual aesthetic and the purely 2-directional movements needed to draw it, making it a useful pattern to begin experimenting with for live performances (as illustrated in Figure Figure 8.3 on page 149).

1: `https://github.com/pixelpusher/liveprinter/blob/test/liveprinter/static/examples/hilbert.js`.

Hilbert curves can be constructed using a recursive, infinitely-repeating geometric process. Each iteration of this process produces a set of connected points that fills up more of an n̲-dimensional space than the previous iteration. This is easily be seen in a diagram of the first three iterations of the process in a 2D space, as demonstrated by Sagan (1994).

Creating the curve in this manner is primarily a geometric operation of recursive substitution at each stage and can be represented by a Lindenmeyer System (L-system). This LivePrinter example by Raskob (2020) uses a starting axiom of L̲ and replacement rules of:

```
L: +RF-LFL-FR+

R: -LF+RFR+FL-
```

Note that **L** and **R** symbols are ignored in the rendering of the curve.

Finally, symbols are iterated in order and mapped to drawing functions (here demonstrated in ECMAScript 6 (ES6)):

```
'F': () => { await lp.dist(2).go(1, false) }

'+': () => { lp.turn(-90) }

'-': () => { lp.turn(90) }
```

There are also ways to directly calculate a mapped point in $\underline{n}$-dimensional space given an initial Hilbert index and a desired resolution. These are less useful for describing tool paths because the information about how to move from point to point, i.e. the rotations and directions of movements encoded into the L-System string, are lost and must be re-calculated. The fact that an L-System encodes a literal set of movement instructions for a 3D printer, without need for matrix multiplications or any other form of interpolation, makes it attractive as a means for generating forms.

A major downside of this L-system implementation is that it is recursive in nature and its representation grows exponentially with each iteration. Another one that is specific to livecoding performance is that not every iteration of the L-System produces a physical movement, only ones that have an 'F'. The iterations that result in only turn will not be picked up on by the audience or performer because they don't result in any printer movements, unless they are otherwise visualised, so care must be taken to either make them visible to the audience or to quickly skip through them in favour of physical movements.

During this thesis, Hilbert curves were explored for their repetitive properties. They were used in the Subsection 8.4.5 (London Design Festival 2019) and for a poster submission at ICLC 2020. Some video of "Hilbert curve techno" can be seen in Section A.2 (Experiments). It was hoped that they would be useful in live performances, but at the conclusion of this thesis such a system was still under development due to the multistep complexity of coding them and the memory-intensiveness of computing them in a web browser.

8.4 Key performances and early experiments

In this thesis, a number of key performances and early experiments shaped the development of the LivePrinter system in particular, ultimately leading to the concept of an I3DP system in general. Each of these outcomes can be considered artefacts in their own right, but are also clusters of practice-led activities that have led to new understanding about this practice. They are interrelated in that, for example, performances contain phases of composition and practicing that overlap with the exploratory design phases of form-finding that led to the creation of new artefacts for exhibition. In this spirit, these outcomes are presented in a more-or-less chronological order, interspersed with links to theory, technical details and findings, and previous user research, so the reader can get a sense of the overall shape of the process of creative development over the life of this project.[2]

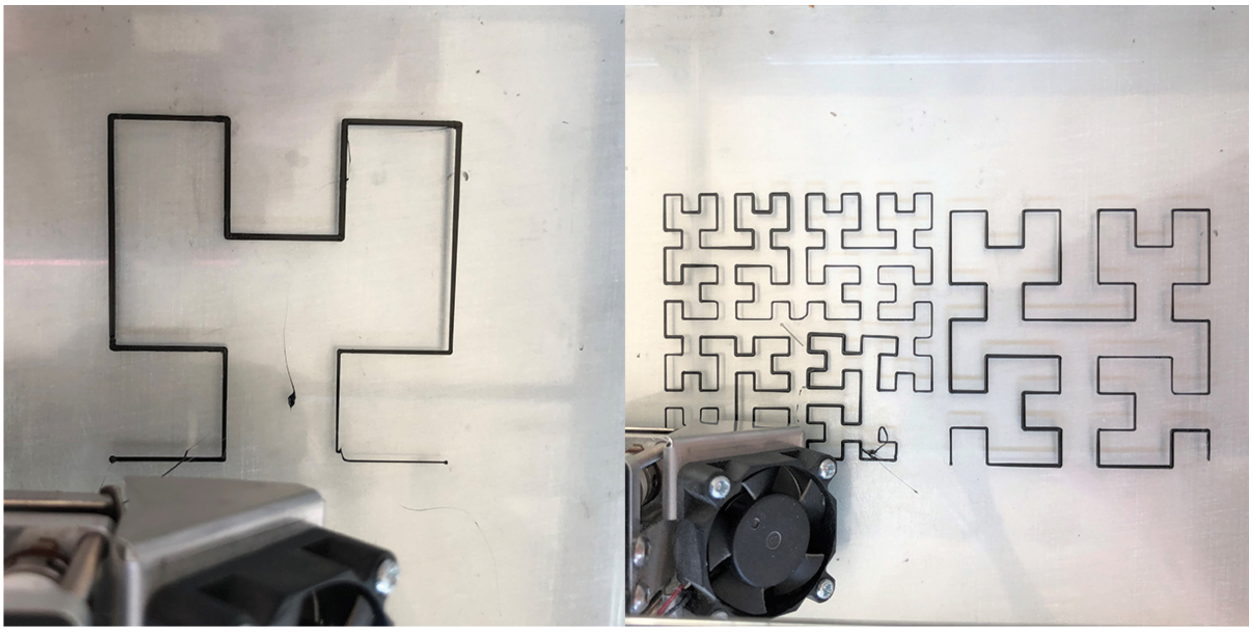


Figure 8.3: Three examples of the Hilbert curve printed on a 3D printer. On the left is one iteration, on the far right is two iterations and on the middle right is 3 iterations. The code for generating these can be found in the LivePrinter repository[1]

8.4.1 The first public performance

The first ever performance of livecoding 3D printing (LC3DP) took place in the early evening on 1 Sep 2018, as part of the TOPLAP Moot pre-Algorave performances a venue called DINA, in the city of Sheffield in the north of the UK[3]. Based on previous informal observations, people seemed to enjoy listening to the unintentional sounds of printers in action, and so this performance was designed mostly to give the audience space to listen to more considered 3D printer movements and to try out basic, musical printer movements in a live setting.

2: A full list of performances and outcomes can be found in Subsection 4.2.2 (Exhibitions and presentations participated in).

This performance was meant to follow the established tradition of machine art, and was influenced by artists like Jean Tinguely who created machines that set machinic processes in motion with the understanding that something interesting would inevitably emerge from them over time. They also were meant to demystifying the process of 3D printing for the audience by slowing it down and focusing mainly on the movements of the motors, linking this work to the Six Challenges of Baudisch and Mueller (2017). Finally, the focus on "rationality" and minimalist experience, under the influence of Marcel Duchamp and Tinguely, led me to focus on the machine-made quality of the sound and movement and not try to duplicate anything "human" and "haphazard". It was much more of a work of sound art than a rave-culture-adjacent piece of musical performance, but it took place at the Algorave, nonetheless.

Sonically, the performance was partially influenced by composer Lamonte Young's experiments with long held notes and drones, which in turn came from classical Indian music, and also by composers Terry Riley and Steve Reich's who often focused on the repetition of single notes and phrases. The long, sustained notes made by unusually lengthy printer movements would draw attention to the quality of the sound and the novelty of the performance. The "pure" notes generated by motors moving in single-axis directions (e.g. in the x or y direction but not both together) alternated with moving at slight angles, across multiple axes, to produce discordant two-note combinations. The movement speeds of the motors were tuned to major scales using the `note()` function, as in `lp.note(note=40, time=200, axes="x")`. Even when the software crashed (but continued moving the print head) the crowd was quiet and attentive, but unfortunately a fire alarm ended the performance earlier than intended.

3: http://livecode.toplap.org/2018/events/algorave/

The composition process for this performance was difficult and minimal, because the LivePrinter software (discussed in Chapter 6 (Implementing an I3DP system: LivePrinter)) was still so experimental and rapidly changing. Crashes were frequent and a real problem for composing reproducible work. Issues of asynchronous communication with the printer would not be solved for over another year, and the software could miss some erroneous responses from the sparsely-documented Marlin printer firmware that would cause it to get stuck in a fatal loop.[4] This highlighted the difficulty of quickly sketching out concepts in an *Exploratory I3DP* mode of working on an experimental system. A large amount of work needed to be done to support a mode of working that was intuitive and not broken with pauses due to system crashes, questions about the consistency of syntax, or uncertain levels of abstraction, for starters.

4: Source code for the main Printer driver is at https://github.com/pixelpusher/liveprinter/blob/master/js/liveprinter.printer.js

8.4.2 2D fills and Algospirals

In January 2019, after some experimentation with fill patterns that began early on in the development of LivePrinter, I worked on creating printed shapes that would be more complex than simple boxes and triangles but also be easy to recognise by audiences. These forms would give audiences some familiar point of reference in an otherwise obscure and unusual performance and create a kind of abstract narrative journey as they gradually emerged on the printer bed out of a nest of extruded lines. For this reason, I chose the unofficial logo of Algoraves, sometimes called "algospiral" or "triangle-spiral" or even"strangle" according to Wikipedia[5]. This shape is often used on Algorave promotional media, such as flyers and online graphics.

A performance image from the Goldsmiths Algorave follows in Figure 8.9.

The first spirals I produced were composed of very thin lines. For my upcoming performance, the audience would mainly see the printer in action via a live camera feed because they would be standing in a medium-sized room with mostly obscured views of the printer itself. The spiral needed to be bold and visible, but it was difficult to get its lines thick enough to be clearly seen by a camera pointed at the printer. Varying the thickness (also called "layer height") parameter could only increase the width of the printed lines by 0.1-0.2 mm at best.

To create wider forms, I experimented with two simple types of fill patterns: a zigzag across the desired *width* of the fill with a set gap in between strokes, as seen in Figure 8.8, and a similar zigzag across the *length* of the fill, as see in Figure 8.7. Both forms used a new function called `drawFill(width, height, lineGap)`.

This method had some issues of overlapping with successive drawings at the ends, but otherwise worked well enough at creating recognisable

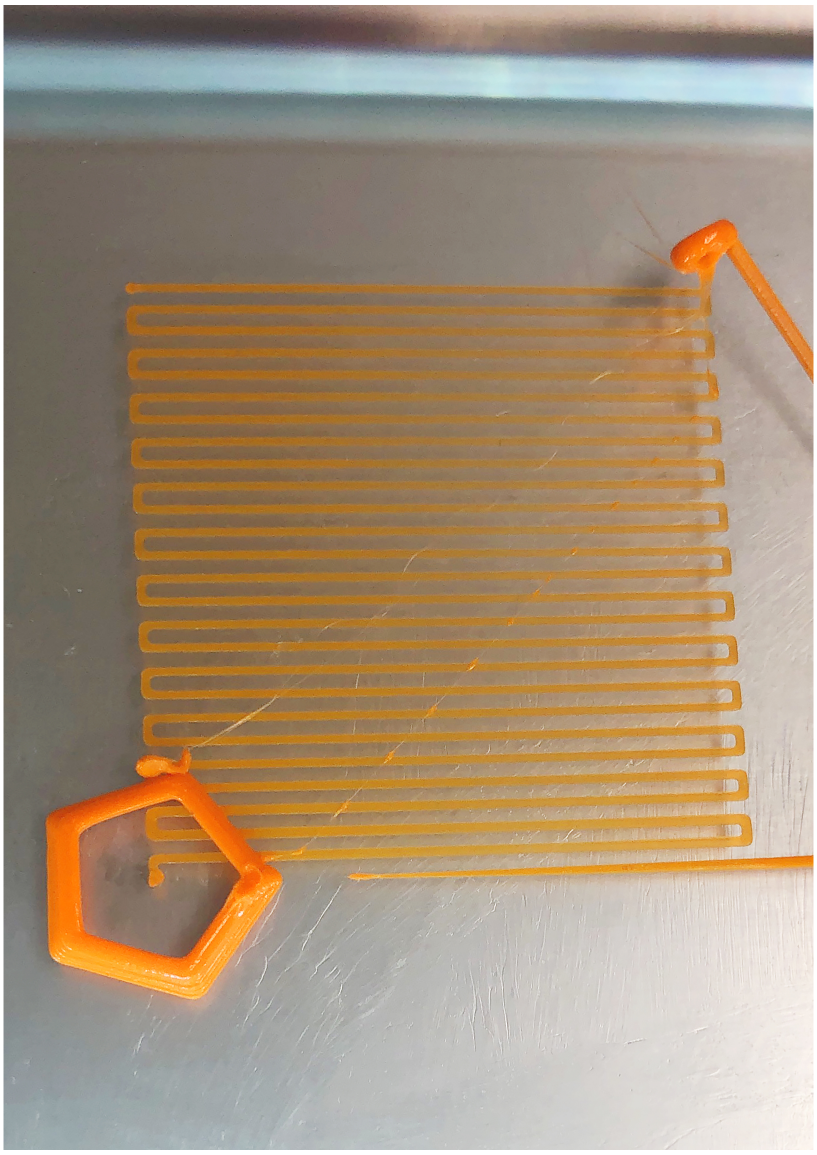


Figure 8.4: Early experiment with rectangular 2D fills from 2018, with a stray pentagon experiment.

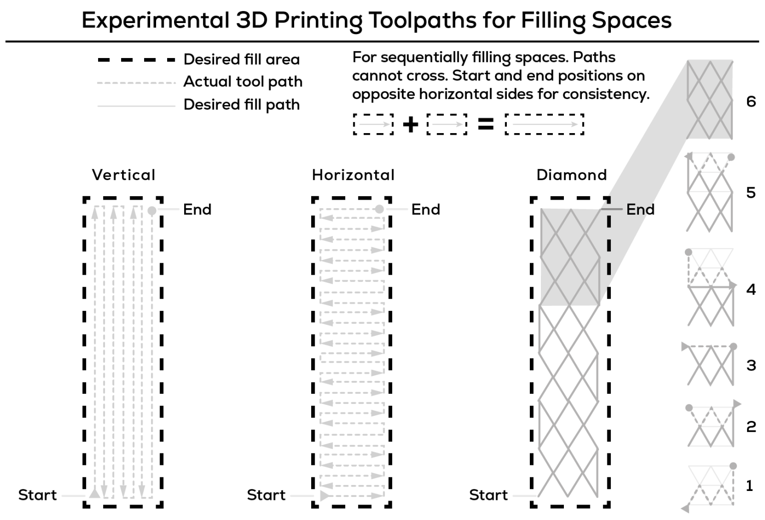
shapes for the purposes of an experimental performance. The shorter "zigs" across the width were useful for short, rhythmic sections, and the longer strokes were more useful for series of durational drones.

5: `https://en.wikipedia.org/wiki/Algorave`

8.4.2.1 Printing on different surfaces

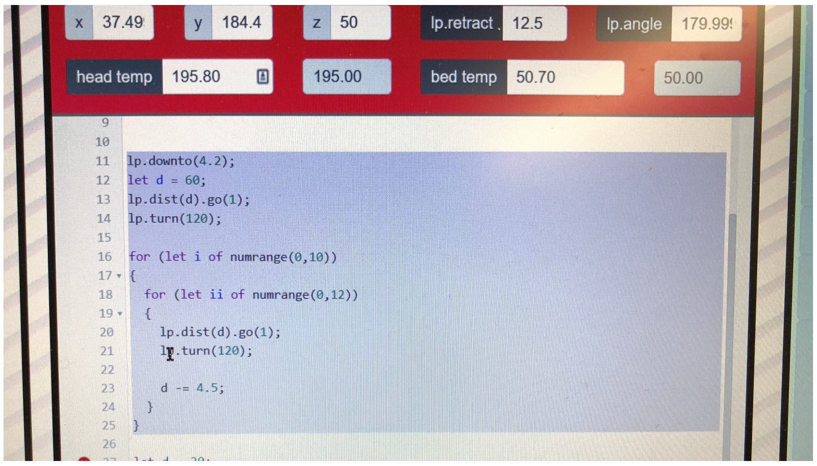
The results were aesthetically pleasing, and became the basis for the poster for the event, that I also designed. As a way of capturing the image, I opted to print onto a piece of paper secured onto the printer bed by strong earth magnets. The print was then digitally scanned in a flat-bad scanner. This technique was similar to pen plotter art, but the slightly molten and embossed plastic lines on the paper had their own character.

This technique of printing on paper had originated in November 2018, and led to further experiments printing directly onto paper and, eventually, acrylic, in layers, which was used in further experiments for exhibitions. It had also led to developing an SVG-format rendered for LivePrinter



Experimental 3D Printing Toolpaths for Filling Spaces
Desired fill area
Actual tool path
Desired fill path
For sequentially filling spaces. Paths cannot cross. Start and end positions on opposite horizontal sides for consistency.
Vertical
Horizontal
Diamond
End
End
End
Start
Start
Start
6
5
4
3
2
1

Figure 8.5: A selection of 2D toolpaths used by LivePrinter for incrementally filling in flat spaces with plastic material. The printed traces must end in predictable ways and cannot overlap, or else they may disrupt of break previously printed shapes. These computational methods are needed to build up volumetric forms, because the material coming from the extruder is quite thin (tenths of millimetres). These 2D forms can be used in intermittent (stop/start) drawing operations to create combinations of shapes, or just thicker 2D line fills.



x 37.49 | y 184.4 | z 50 | lp.retract 12.5 | lp.angle 179.99

head temp 195.80 | 195.00 | bed temp 50.70 | 50.00

```
lp.downto(4.2);
let d = 60;
lp.dist(d).go(1);
lp.turn(120);

for (let i of numrange(0,10))
{
  for (let ii of numrange(0,12))
  {
    lp.dist(d).go(1);
    lp.turn(120);

    d -= 4.5;
  }
}
```

Figure 8.6: The code for an Algospiral, in LivePrinter.

in order to streamline the experimentation process somewhat, which in January 2019 became a well-received part of the user research workshops (described in Subsection 7.4.8 (Task 8: Using LivePrinter (freestyle drawing))).

8.4.3 Second performance: the Goldsmiths Algorave

The second performance was a more polished affair, taking place in the Sonics Immersive Media Labs (SIML) at Goldsmiths. The SIML is a special space, a smaller variation on Recombinant Media Labs' "Cinechamber" with 360 degrees of floor to ceiling projection (6x 1080p projectors with special short throw lenses) and a 12.2 sound system. This concert was part of TOPLAP's 15th world-wide birthday event, with about 168 live performances streamed over a period of 84 hours[6].

6: TOPLAP 15: https://toplap.org/wearefifteen/, Goldsmiths's event: https://gold.ac.uk/calendar/?id=12264

This performance was meant to be more musically-rich than the previous one, in that it would contain more musical structures and rely less on long, continuous drone notes. Mainly, this was an excuse to experiment with LivePrinter's growing ability to work with more common musical structures like melodies, chords, and rhythms. One thing that became clear quite quickly was the cognitive overload of having to constantly keep

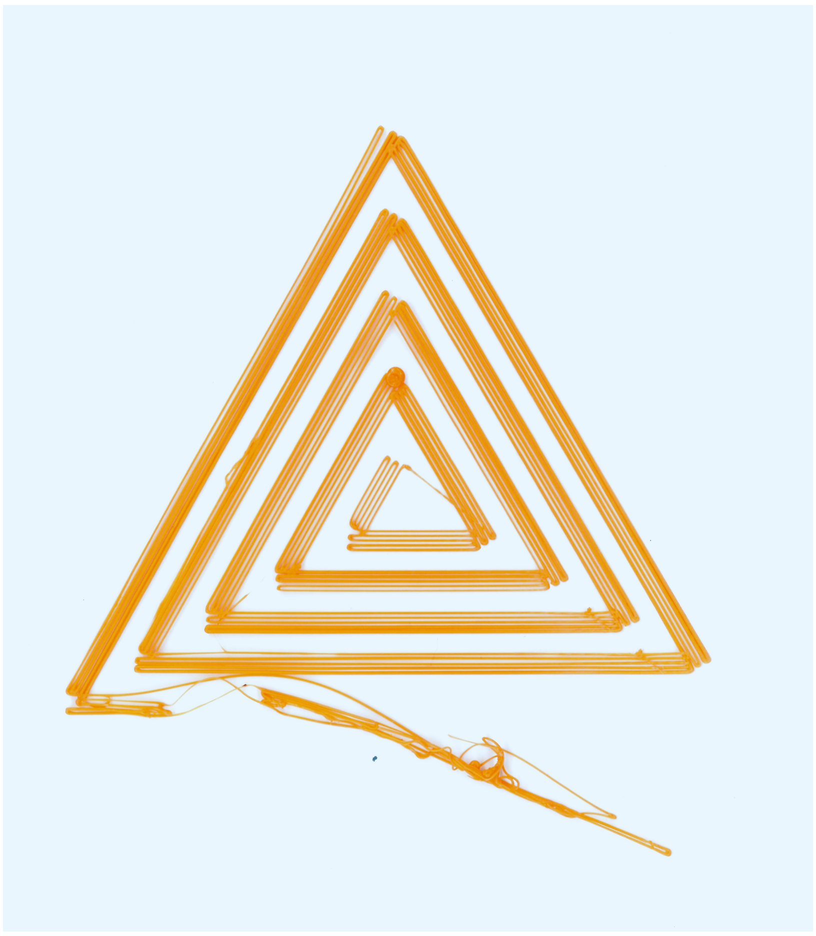


Figure 8.7: Fill pattern for the Algospiral used in the event poster, from January 2019.

the system in motion making sounds whilst simultaneously worrying about the physical state of the printer and printed material – the heat of the print head and print bed, the movement of the print head and whether it would hit an object or whether it was near the edge of the bed or hitting the little metal clips holding the bed together.

In a musical livecoding performance, performers are often writing musical phrases that repeat over time. They write code, execute it, and have some time to sketch out new ideas before committing to new code. With live 3D printing, the print bed is relatively small, so when a performer is using higher-pitched notes the print head moves across it quickly and there is little time to think and reflect and plan ahead to the next movements.

Whereas looping constructs like *for* and *do. . .while* could have been useful, in practice their low abstraction level and verbosity added to the cognitive overload of livecoding. This difficulty was mainly due to the verbose syntax of JavaScript, especially when dealing with asynchronous functions. Typing these out was physically involved, and required a performer to keep track of many matching brackets types that, if missed or misplaced in a transcription or modification action, could cause fatal errors. Whilst the minigrammar helped somewhat, its relationship to JavaScript as a part of it or an alternative to it was still unclear at this stage of development.

Also, ideally, there would be no lag between printer operations so that commands sent to the printer would be queued and run immediately,

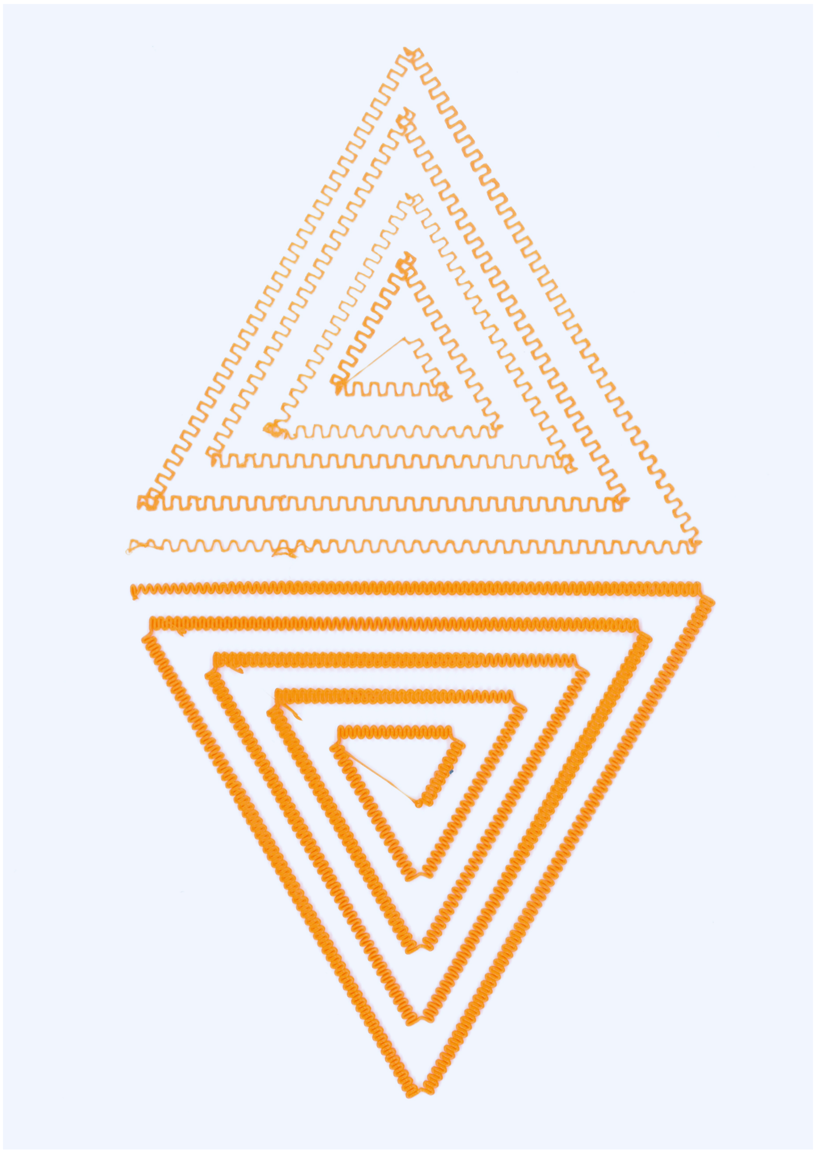


Figure 8.8: Fill pattern for tests of the Algospiral, from January 2019.

using the printer movements themselves as the timing mechanism for the piece. At this point in development, this was impossible due to communications lags and error-handing issues with the system.

8.4.3.1 Tasks and scheduling

One interim solution to this was to add a task scheduler mechanism, so a performer could write smaller print routines and execute them at specific timed intervals, giving them time to plan their next move and the audience something to experience whilst they did it. This was far from ideal, because it introduced a lot of unpredictability into the system when multiple tasks were running simultaneously, but it also led to some interesting (if chaotic) sounds and shapes.

A useful innovation was a "Task dashboard" section in the main LivePrinter interface where tasks appeared as pop-up notifications when they were run, and could be easily cancelled by clicking an "x" in the top of their pop-up container. This increased the visual clutter of the interface somewhat, but also reduced the hard mental operation of having to imagine what processes were running at any point in time, and also the constraint on the order of operations where it could be difficult for a user/performer to remember how to stop processes running.

In later experiments and performances, once timing issues were solved and the system's timing was demonstrably accurate when using only the printer movements as a clock source, and the minigrammar and other helpful abstractions were added to reduce cognitive overload and verbosity, the composition and performance process became mostly a process of using loops and repeated actions rather than tasks and scheduling.

8.4.4 Expressive art 2019

I was asked to submit a piece for the 8th ACM/EG Expressive Symposium, May 5-6, 2019, in Genoa, Italy. The symposium "explores the capacity of computer graphics, animation, and computational media to be used in artistic, aesthetic, and creative way"[7]. It included an exhibition of

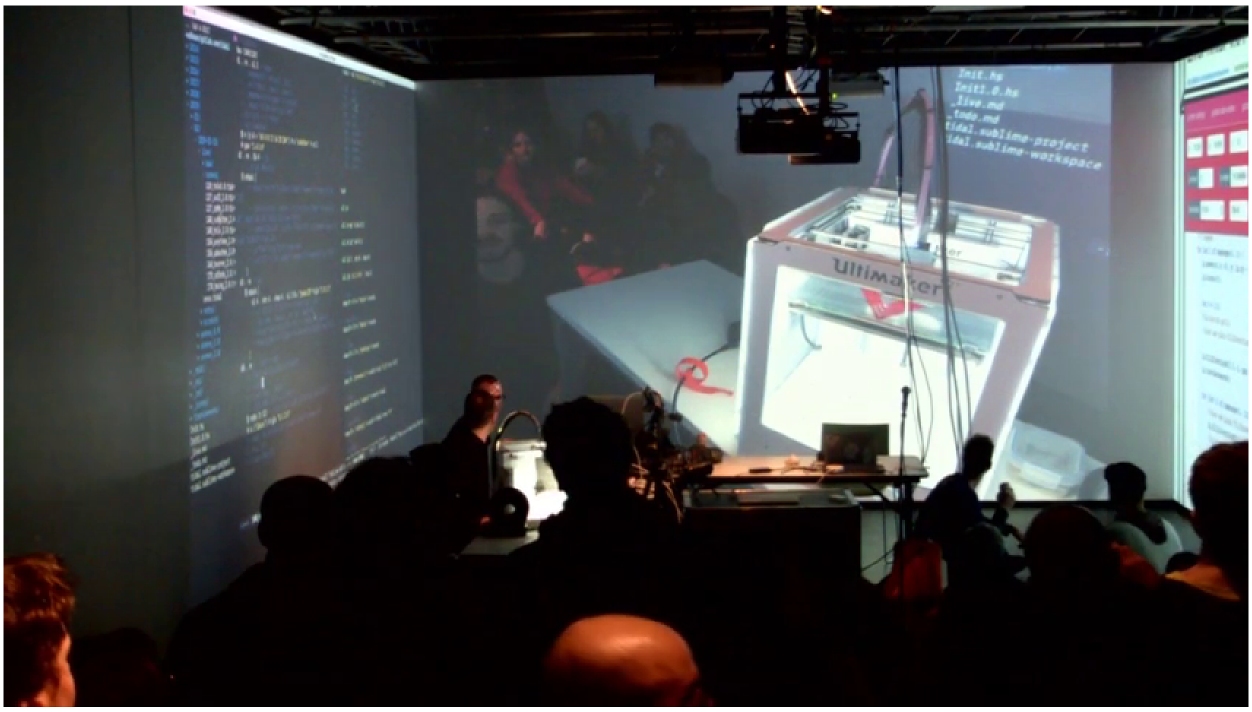


Figure 8.9: A view of the Algorave at Goldsmiths with LivePrinter visual projections.

works to which I contributed some sculptures and the printed code from livecoding sessions where I sketched them out.

This collection of my work, titled The CyberAnthill, brought together a series of generative sculptures using a particular type of adapted cellular automata. As the title alluded to, the cellular automata process was somewhat inspired by Langton's Ant and also the "Light Cycle" racers in the cult 1980s science-fiction film *Tron*. Instead of the normal process of printing exacting, predetermined 3D models, the 3D printer generated its plastic forms by running unpredictable computer code that generated layered grids. This was a reference to automated manufacturing and artificial intelligence, especially given its inclusion with other artworks in the symposium and more general research, as part of Eurographics, using AI and computer-generated imagery.

The idea was to physically embody Wolfram's theory of "Computational Equivalence," which posits that the only way to understand the future of complex, emergent systems like the universe itself is to run through each computational stage of their existence, in sequential order, from their beginning conditions. In other words, the only way to simulate a complex system like the universe is on a computer with at least the equivalent computational power of the universe. There are no shortcuts to predicting the future (Wolfram, 2002; Andrew Pickering, 2011).

The work was also meant as a step towards a future where human-assisted robots might create materials from layered mixes of different microstructures, perhaps forming new *metamaterials* as discussed in Chapter 3 (Literature Review). As I wrote, "for example, airy grids of 3D printed 'bubbles' ranging from large to tiny could be used to create chairs using the same material. This continuous form could alternate between hard but light, both flexible and firm at different places as needed. The humans would set the design requirements, and the computer would manage the complexity of creating the finished product" (Raskob, 2020).

7: According to their website: http://expressive.graphics/2019/

8.4.5 London Design Festival 2019

A further iteration on the Expressive '19 artefacts and code was accepted for exhibition as part of the London Design Festival's "Design for Change" (DR4C) showcase in 2019 shown at the London Design Festival (LDF), Old Truman Brewery, London from Thursday 19 to Sunday 22, September 2019. The exhibition was curated and organised by Prof. Paul Rogers and Dr. Francesco Mazzarella at Lancaster University, UK (Rodgers, 2020).

There were 11 artefacts displayed in total, along with the code used to generate three of the automata-inspired fill patterns, as seen in Figure **??**. The largest artefacts were 80 mm square or 120 mm wide for the longer rectangles, the rest were of varied dimensions ranging from 20-40 mm.

The curated projects in the exhibition "illustrate[d] wide-ranging social, cultural and economic impacts and highlight the significant roles that UK-based Design researchers play in some of the most complex and challenging issues we face both in the UK and globally."

This excerpt was taken from the text displayed on the wall in the centre of the exhibition.

My description of the LivePrinter project focused on its potential to engage people more actively and directly in the act of making with 3D printers, and ultimately with digital manufacturing in general:

> This project is about developing an open, interactively programmed 3D printing system for live computational making. It explores the role of improvisation and intuition in design and making new forms for automated manufacturing. Current processes for 3D printing place the artist and designer at a difficult distance from the physical process of making. There is no space for live improvisation and experimentation,

> especially with key properties that directly affect printing materials like temperature and print speeds. This new system extends digital printing and CNC machining into the realm of performance and also has potential in design and science pedagogy and materials science. It also might spur us to consider how we as humans might have a more active part to play with automated manufacturing. LivePrinter has already been used to create live musical performances with 3D printers and generative works of art. You can find out more at `https://github.com/liveprinter`.

Many of the artefacts exhibited here were similar to or identical to the ones from Expressive '19, except for the more complex and multilayered Hilbert-curve variations that can be seen in the top right of Figure **??**, printed in bright yellow plastic. The accompanying exhibition text highlighted the use of automata, generative techniques and Hilbert curves as part of a live fabrication process:

> LivePrinter is the start of an open, interactively programmed 3D printing system for live computational making. Automated manufacturing can be a collaborative process rather than replace human craftspeople. These experiments were all done with LivePrinter to show that new forms of improvisation and intuition are possible with digital manufacturing. One uses digital "agents" that are guided by the coder/maker to create overlaying paths of plastic in grids of different sizes. The other explores more manual ways of filling space, here using livecoded Hilbert curves with different properties. Each experiment was a combined automated and manual process where the maker manipulated the material in the 3D printer and interactively ran and edited the code over the making session. The code for each session in recorded and displayed here along with the artefacts.

On reflection, what interested me the most about the work was the sense of depth conveyed by the multilayered pieces, especially those that used different colours at different grid resolutions, as in the piece mounted to black acrylic seen in Figure 8.1. This comes from a tension between the positive space of the plastic and the literal negative space of the gaps between them.

When I printed multiple layers in the same colour, at similar grid resolutions, the effect was lost as the layers converged towards a regular rectilinear grid. This was mainly because each layer was so thin that it was hard to see the semi-random gaps in between them that gave each piece an individual texture. This layer thickness was severely limited, determined by the physical width of the printer's extruder nozzle to a range of only about 0.1-0.3 mm, with the latter achievable only at slower speeds and when printing directly on top of other layers.

The result was that I was more drawn to the smaller, finer details of the Hilbert pieces and smaller red plastic automata grids more than the larger and more sparse pieces. Yet, when I experimented with a number of long, grid-like forms inspired by Agnes Martin's regular compositions, I felt they were converging towards something like her blend of near-Minimalism in form and Abstract Expressionism in intent and execution.

Each printed piece was fragile. Their regularity was easily perturbed by temperature variations and the inherent randomness of fluid dynamics; plastic lines didn't stick properly or shifted slightly during printing for unknown reasons. This nicely complemented the bounded anarchy of the automata-inspired algorithms, coupled with my aesthetic judgments of when to stop and start them and modify them during operations, including how many layers to use and at what resolution.

I am not sure if this was the right venue for such a meditative piece of work, as they were meant to be small pieces of a larger future for people to contemplate, small shards of computationally-fabricated objects that would be commonplace in everyday life. Given more time, I would have liked to experiment more with different colours and different ways of mounting the artefacts to bring more attention to their negative space. I would also have appreciated giving them more space, as exhibitions of Agnes's work usually do, to let people appreciate the minimal forms on their own without the noise of adjacent pieces. Whether these early works were subjectively good enough to merit such a spacious and luxurious exhibition is another question.

8.4.6 ICLC 2020

For 3D printing and music-making, the durations of movements and silences are of paramount importance to the musical aesthetics of the piece. As with any movement that creates music, a performer must control it precisely. Any gaps in movement or extraneous movements to position the printing head become part of the performance, for better or worse. A performer needs predictable tool paths at their disposal to improvise with, much as a jazz musician riffs on different musical scales and motifs. Continuous curves such as the Hilbert can be useful in that respect. As Papacharalampopoulos, Bikas, and Stavropoulos (2018) observed, Hilbert curves keep the print head moving throughout their length and thus minimize or altogether remove any extra waiting between operations and travel times needed to reposition the head after movements.

A printer following a Hilbert curve follows a predictable, regular path and doesn't need to stop extruding until the curve is finished. It is guaranteed not to hit any other point in the curve as it moves, removing a potential source of error. They fill up a rectangular space completely without crossing themselves and can be run backwards once finished. With dense Hilbert curves of higher orders, the number of movements can be quite large which is helpful to a livecoding performer trying to compose a dense melody live. They are especially useful when the printer movements are quick, giving them more time to think as they type new lines of code.

For example, a performer might wish to play a sequence of MIDI C5 notes (MIDI number 72) every beat, at a tempo of 120 beats-minute (or 0.5 seconds-per-beat) for a 12-beat segment. That means each beat the print head would be moving at a speed of 11.1165 millimetres-per-second for a distance of 5.5582 millimetres. A second-order Hilbert curve could completely contain that movement because the curve is made of 15 segments packed into a space that is 3 segments by 3 segments square, or 16.6747 mm by 16.6747 mm which will take a total of 6 seconds to play (12 notes at a duration of 0.5s per note). This whole curve fills up a little over 13% of the total printer bed space at that level, leaving room for about 8 repetitions of that motif before starting a new print layer.

Also, when the curve is oriented to the x/y axes of the print bed the tool head will move at right angles when drawing the curve. This uses only one motor at a time, playing distinct notes rather than chords. A performer could take advantage of this fact by alternating motor speeds with every segment of the curve, thus playing arpeggios. It is also possible to make more complex chords using rotated Hilbert curves. Rotating 45 degrees with respect to the x/y axes of the bed engages two motors simultaneously when moving, playing a two note chord consisting of the same two notes. A movement at any other angle produces a more complex tone that in practice can be difficult to control. In practice, it can be hard to manage Hilbert curves because the L-Systems representation of them grows exponentially with each iteration. This makes storing them as strings a non-trivial task, especially in a live, real-time performance setting.

Stepping through an entire curve in a performance is a dangerous task because it encompasses hundreds or even thousands of steps. Some language-specific techniques are needed, such as using ES6 generators to write iterative Hilbert functions whose execution is not continuous (Mozilla, 2019). Additionally, more research is needed into the properties of other space-filling curves that have a number of segments that divide up evenly into common musical time signatures, like 4 and 8 beat segments. There are many other types of space filling curves that could have beneficial musical properties, including diagonal segments that represent chords.

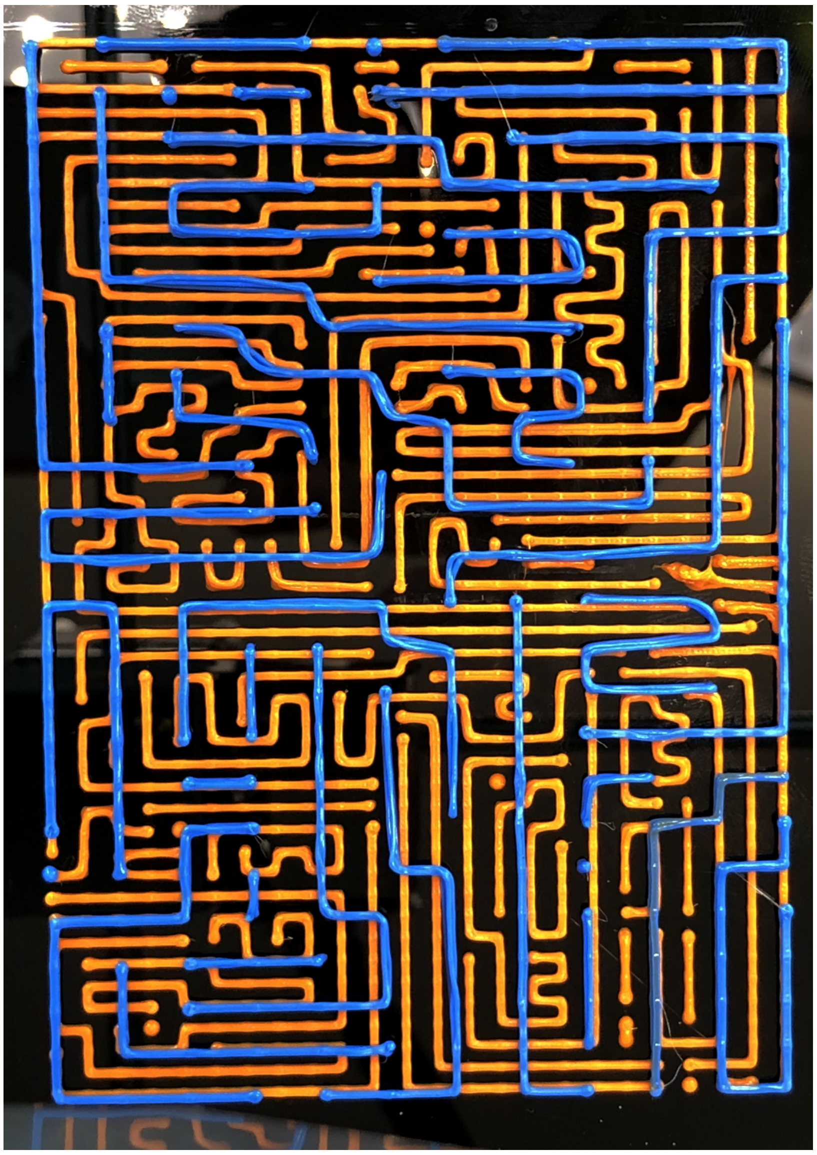


Figure 8.10: Printing a variation of *the CyberAnthill* using two different generative algorithms realised in two different-coloured materials, printed directly onto a sculpted piece of glossy black acrylic.

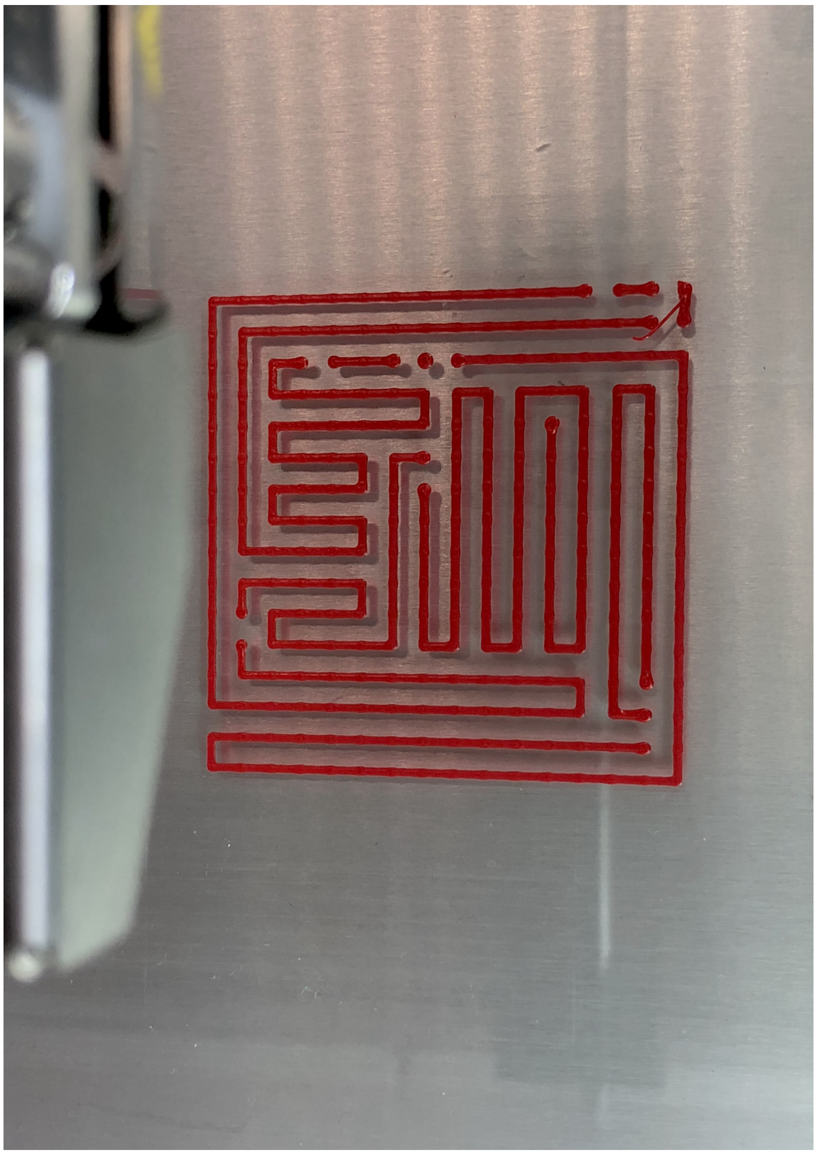


Figure 8.11: Another, smaller variation of *the CyberAnthill* using different generative algorithms in different layers, on top of one another, realised in a single colour.

8.5 A case study in "airprinting"

Both the opportunity and conceptual confusion inherent in these types of operations as was hinted at in the discussion on travel movements in Subsection 6.6.2 (travelling upwards and downwards).

3D printers usually build layers out of 2D forms, but they are also capable of directly fabricating 3D objects. . This functionality was particularly difficult to implement because the use cases were not as clearly defined as with the usual travel and extrusion operations. There simply aren't many examples of 3D printers extruding plastic into the air, unsupported, in the current literature. Mainly, printers use stacks of 2D layers for building up 3D shapes[8], although there are commercial devices taking advantage of "airprinting" that have been around for a few years, like the 3Doodler pen[9] which let users draw free-form in space with melted plastic.

8: See the Literature Review chapter's discussion referencing (Livesu et al., 2017)

9: `https://intl.the3doodler.com/`

Two more key examples are *WirePrint* by Mueller, Im, et al. (2014) and "augmented reality" interactive fabrication by Peng et al. (2018). Both projects look at how we can reduce the time spent 3D printing by drawing sparse structures into the air, similar to the 3D wireframes that are commonly used for fast previews in 3D design software. *WirePrint* in particular was an inspiration for a series of experiments looking at whether their results could be approximated using the LivePrinter system, on a non-modified 3D printer.

The *WirePrint* software was a standalone renderer that took in a 3D model and output GCode for 3D printers. The main gist of the algorithm, as described in Mueller, Im, et al. (2014), was to first *slice* the model into horizontal slices, detect their contours, and then link together successive vertical slices with a vertical zigzag pattern of alternating triangles with their points spaced evenly between the layers. To make these zigzags, the printer needed to "zig" or move upwards quickly into the air whilst extruding plastic, then pause and retract the filament slightly to tighten it and let it cool, before moving downwards to finish the "zag".

The basic concept was straightforward, but the details of the implementation relied on the control of a number of independent variables dictating printer speed, temperature, and optimal line segment lengths, for example, leading to a very large number of experimental possibilities. In addition, the study didn't look at optimal variables for each of these operations, or at least they weren't specified in the text. This left a number of questions unanswered, such as the optimal printing speed for each segment of the zigzag shape (i.e. how fast should the initial lead-in segment be, as opposed to the vertical and downwards ones?) What was the optimal retraction speed and length? How long should the printer pause at the top? What was the minimum possible vertical angle that this form of printing could support?

Using standard 3D printing software that worked with a traditional *process planning* workflow, e.g. from 3D model to 3D file to 2D layer slicing, it would be difficult or even impossible to experiment directly with these variables. A research would need to generate GCode directly in a controlled, iterative, multistep process. With LivePrinter, each of these variables could be controlled for explicitly in rendered GCode to find optimal parameters, and with the results combined into a further series of repeated experiments to find some candidates for optimal combinations.

The initial reverse-engineering this part of the *WirePrint* algorithm, including finding and patching bugs in the LivePrinter code and extending its capabilities to properly run coding loops took about 2 hours before effective parameters and an algorithm was discovered that could print a series of unsupported triangles reliably on the printer bed. In the later user test of the LivePrinter minigrammar, finding the code to recreate this basic shape took about 21 tries over a period of 25 minutes. This could have been automated somewhat to use coded loops to cycle through the possible parameter values, but manual control and intuition proved fast enough for our purposes (see Figure 8.14 and Figure 8.15).

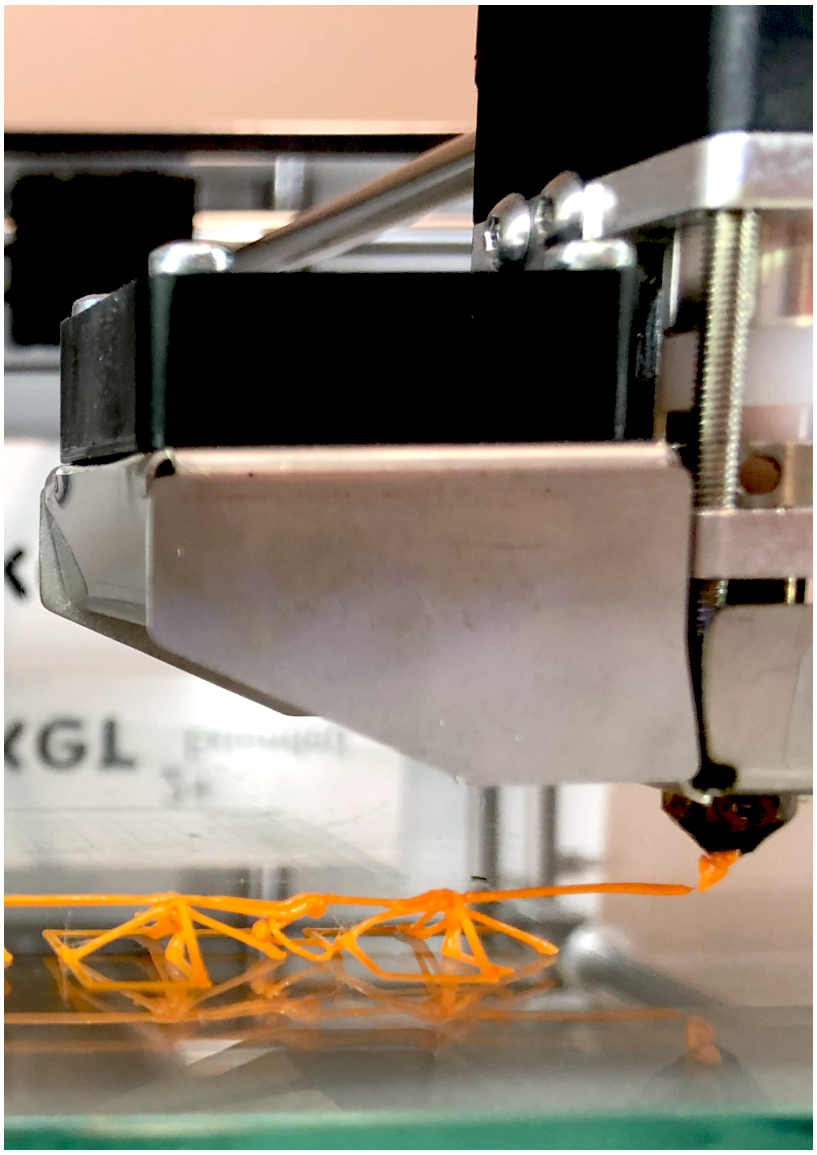


Figure 8.13: Experimenting with printing sparse forms in 3D space by making pyramidal shapes.

```
##
autoretract 0 | fan 0
unretract | turnto 0 | printspeed 15
draw 40 | fan 100 | elev 35 | printspeed 20 | drawup 3 | retract
   12 | wait 1500
unretract | elev -35 | printspeed 18 | drawdown 3
printspeed 20 | draw 30 | retract 6 | fan 0
##
```

Figure 8.14: Airprinting with the mini-grammar

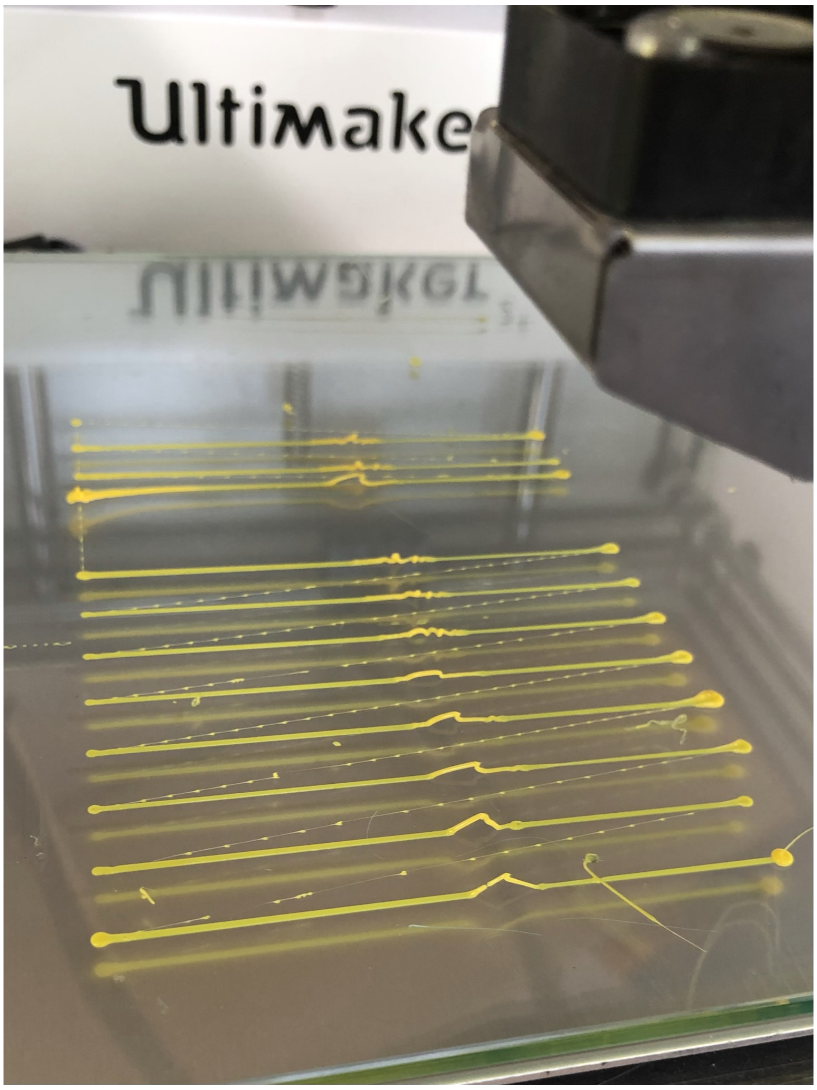


Figure 8.15: The results of reverse-engineering the airprinting algorithm using the LivePrinter minigrammar.

8.6 Conclusions

Normally, the speed at which the printer prints is determined by the desired quality of the print balanced by the total time of manufacture. Slower printing times generally lead to higher quality prints because the print head can follow a more precise path and the layers have time to cool properly before the next layer is applied on top (Matter, 2015).

When making music with printers, the relationship between form and making time is subverted. Performance time and musical quality can have as much weight as the quality of the finished object, whatever that final output might be in such a live setting. Slower speeds that might produce higher quality prints may be either inaudible or in the wrong key or frequency for the piece being performed.

This leads a maker/performer to reframe manufacturing as a mainly durational activity. Instead of describing objects in the usual way using technical drawings or digital models specifying physical dimensions in millimetres, one can consider objects using their durational dimensions and specify them in terms of the speed, angle and duration of movement used to manufacture them. This tightly integrates the making of the object with the description of the object itself. It stands in opposition to the process planning approach that separates out a design concept from its fabrication processes.

In 3D printing livecoding performances, the performer choreographs (or composes) the movements of the printer (speed, direction, duration) and the properties of the printer itself (temperature, fan speed, filament flow rate) by manipulating and writing code which is projected into the performance space. Both the act of making, with its resulting physical forms, and the sound of making are intrinsic to the performance. This leads to a dual mode of composition when making music for printers, where one can prioritise the aesthetics of the form by composing movements in millimetres or the aesthetics of the sound by composing in milliseconds of movement at specific speeds that correlate to musical pitches (e.g. musical notes and scales).

This reframing of object-making as a mode of composition stands in sharp contrast to the usual visual or shape-based software modelling techniques, where object manipulation is primary and textually descriptive methods of form-finding like code are secondary, if allowed at all. Another insight into the difference between these two modes of creation comes from T. Magnusson and Mendieta (2008)'s 2009 survey of "musical instrument phenomenology". The survey results were concerned with acoustic physical instruments versus digital ones, but we can similarly compare the act of physical making with the process of digital fabrication using software. In the first case, the quality of the output relies mainly on the physical dexterity of a craftsperson, the second on the usability of the software.

To paraphrase T. Magnusson and Mendieta (2008), the music that one creates using acoustic instruments (e.g. violin, piano) is immediately limited by the skill of the performer and thus a matter of increasing that physical skill (through practice) to expand the performer's repertoire. With digital instruments (e.g. MaxMSP, SuperCollider, tidal) the computer is already "skilful" at making sound and comes with many examples, modules and materials to choose from, so the performer builds their repertoire through a process of selecting and refining, rather than expanding. Similarly, physical makers start out producing simple forms that gain complexity as their skills increase, whereas digital makers learn to sort through complexity, choosing which elements to use and which to discard or ignore, often resulting in a skilful simplicity of code and form.

The practical result of using text (livecoding) for the act of making is a trade-off where users lose direct physical control over the making process but gain the ability to replicate objects with more complex forms, to experiment with different visual (and acoustic) aesthetics and also to embed intellectual ideas in the construction and form of shapes, such as musical concepts. For example, LivePrinter allows makers to embed musicality into the making process by providing a suite of such functions for running the printer motors at musical frequencies for specific durations, and repeating them. Alongside these functions are others for converting distance travelled (at a certain musical speed) to duration so that making operations can be specified in terms of both precise dimensions and musical effect.

Some special types of space-filling algorithms that are composed of a single, continuous line, called space-filling curves, can be used for this particular type of performative 3D printing. These curves are drawn in predictable ways that fill up space to build printed shapes without crossovers that might damage the shape being built. These curves could be a useful tool for expressing physical objects in mixed musical/sculptural performances because of the ways in which they can completely fill up spaces in repeating and predictable ways.

This potentially has applications outside sculptural performance, in the realm of industrial 3D printing, where 2D and 3D space filling patterns and techniques are current areas of research. Unfortunately for the present, these curves can be complex to design leading to difficult mental operations in live settings, and computing-resource-intensive to run. Further study is needed to come up with methods for integrating different fill patterns over time, across unpredictable and improvised forms that arise during performance. These methods might also find application in areas like metamaterials research, and other methods of 2D fill patterns that save time, materials, and thus energy, as was discussed in Subsection 3.9.4 (Controlling 3D printing with code).

End reflection 9

9.1 About

This chapter reflects on the current state of an ongoing Research-Through-Design (RTD) process as experienced by the researcher, Evan Raskob, through his practices as software developer, educator, and artist; a process that was simultaneously philosophical, functional, social, and aesthetic*. In the spirit of Gaver (2012), these recollections are loosely centred around a "theory nexus" weaving together "pure" theory (such as human cognition) by way of a grounded "technical activity" of developing a functioning Interactive 3D Printing (I3DP) system that was used to create learning experiences, experiments and artefacts – performances, sculptures, techniques – embedded within the livecoding and also the generative art and design communities.

In this spirit, these reflections are presented as a loose collection, ranging from abstract questions about the metaphoric language of 3D printing to the specific implementation details of the LivePrinter I3DP system that formed the main research outcome of this thesis.

9.2 Why use code for 3D printing?

This thesis, through a reflective and iterative research-through-design-led process, showed that interactive 3D printing can change how people think about the process of 3D printing and how this process has led to the development of new aesthetics, grounded in generative art, and new understandings of how interactive programming systems can be designed to support this new mode of working. The thesis began with the inductive notion that it might be interesting to combine interactive programming and 3D printing, so it makes sense that at its end we revisit and reflect on *why* it was so interesting. Given what we now know, why would anyone want to interactively use code for 3D printing, when there are other means available?

Language has an established place in the design process. Schön (1991) observed that experienced designers tend to use evocative words that describe certain archetypes as part of their designerly process. For example, the architect Richard MacCormac used the word "vessel" when referring to his designs for a round inner worship space for the chapel at Fitzwilliam College, Cambridge. "Vessel" as a metaphor for a physical space evokes feelings of floating and disconnectedness, as in a ship floating alone at sea.

These designerly metaphors can be much more than placeholders for types of designs when discussing them with clients or describing them

* These multifaceted aspects of the research process, essential components of works of both art and design, are mentioned with nods towards Bill Gaver and especially his predecessor Josef Albers, as discussed in the Introduction chapter.

to audiences. In Schön (1991)'s thinking, the act of designing can be thought of as a conversation involving a designer, their idea, potential stakeholders, and physical making activities along with the materials used in that process. Metaphors become intrinsic to that process.

In a more literal examination of "design as conversation" undertaken in Lawson and Loke (1997)'s study of "conversational programming", these conversations swung between a detailed specificity and metaphoric vagueness. Metaphors of space, like "vessel," alternated with frames of reference for specific drawing operations and movements, and the details of how they were meant to contribute to the construction of 3D volumetric forms. Computer programmers might find this jump between archetypal language and implementation language familiar, as they shift between metaphors for describing software archetypes to one another, such as Gamma et al. (1995)'s classic "design patterns" of *Factories* and *Observers*, and the more machine- and process-specific language of actual computer programmes[1].

1: See also the earlier section on livecoding systems design patterns in Subsection 3.7.4 (Design patterns for livecoding software systems).

As similar as programmers and designers might find these jumps in language, their application of the language can be quite different in context. Lawson and Loke (1997), coming from a design background, saw the future potential for computer-aided-design as one of a conversation resembling a structured version of the traditional art studio critique session, where design concepts are presented and then discussed. A designer would propose, and a software-agent would critique, and vice-versa. This system described in Lawson and Loke (1997) would parse through highly structured, textual statements from a user describing a design and then provide appropriate "critical" responses by drawing implicit links between that input and the design-related entries stored in its database. The conversation was meant to unfold over time, in a performative way, and could involve other designers and even autonomous software agents as well.

With its structured syntax, REPL-like input, performative mode of operation and immediate responses, Lawson and Loke (1997)'s system resembles an interactive programming environment for design ideas, but with a difference in approach. Rather than proposing software programs for critique, computer programmers are more used to writing a program that fairly-specifically tells a computer what to do, and then analysing the results. New AI-infused systems such as GitHub's Copilot[†] are beginning to offer more conversational approaches as they offer up possible code implementations as a form of "autocompletion" in response to the user's typing, but they are a far way off from having any kind of critical conversation with their users.

AutoDesk, creator of most of the leading 3D CAD software, is also well on their way towards combining concepts of machine intelligence and generative design into a more conversational software package. They have a major investment in machine learning and augmented intelligence (AI) groups (AutoDesk, 2019) and an ongoing project called DreamCatcher that would turn user's "design constraints" into production-ready models. The combination of cloud computing, big data and advances in machine learning and AI like GPT-3 might soon move design software from point-and-click model-building to a more live, conversational, curatorial process

[†] https://copilot.github.com/

where the software makes suggestions based on physics, best-practice, legal issues and commonly used aesthetics.

It is exciting to think of future performances based on new kinds of semi-automated systems that enter the programmer into critical dialogues between past code (via software agents trained on past data) and other designers, programmers, and even audiences, but we need to be careful of certain risks. Our act of future-gazing risks an incursion into the territory of science fiction, where we suspend disbelief and assume everything is possible, however unlikely. In order to stay in a state of critical, grounded research, we must be clear about what sort of possibilities are likely afforded by computational conversational design, even lesser forms such as I3PD in general and our LivePrinter I3DP implementation in specific.

9.3 Reflecting on the constraints of I3DP systems

One way of staying within the realm of actual possibility with science-fiction-like technology is by framing discussions about the *affordances* offered to users by such systems, and likewise the *constraints* embedded in them that prevent certain operations or possibilities. According to Thor Magnusson (2010, p.63)'s interpretation, one way of looking at affordances is to see them as "potential applications derived from the agent's [user's] embodied relationship with the object in the enactive sense".

As Magnusson notes, his definition of affordances is open to interpretation and carries with it a variety of competing definitions.

With I3DP systems that use rich language as a means for interaction, the number of affordances can be difficult to determine since the system opens up all sorts of possibilities to the user. An examination of the creative constraints that both livecoding and 3D printer mechanics imposed on this particular method of 3D form-making was, in practice, more useful when designing LivePrinter. This view follows on from Margaret A. Boden, who looked at constraints as a way to map out and explore the structural possibilities of a creative space. By using constraints as boundaries for a conceptual space to be explored, we limit the creative possibilities but also focus the inquiry and create paths (patterns) for users to follow (Boden 1990, p. 95).

Looking back, the design of LivePrinter coalesced around three main constraints. These constraints were quite straightforward, but with major implications:

1. Forms must be described using code
2. The user is responsible for specifying tool paths and machine properties (movement, speed, temperature)[‡]
3. The machine does all the making[§]

[‡] Note that an advanced user could still write functions to handle these automatically for certain forms, as is the case with more complex space filling operations like LivePrinter's rectangular space *fill()* command.

[§] But still with the possibility of human manipulation during that process

These were the only three absolute constraints that users needed adhere to. Users could run code, start or stop the machine and choose different materials to use in it. They could create physical forms computationally, using different functions that take into account physical properties like speed and temperature.

The constraint of forcing the user to think about tool paths and other low-level implementation details was at odds with the mostly higher-level abstractions of Lawson and Loke (1997)'s conversational system of design. By bypassing the more familiar language of design in favour of textual metaphors of manufacturing we in effect forced people to think about them from strange and different points of view. Instead of discussing fully-realised forms that could be manufactured, they had to focus on describing the molten plastic lines that incrementally build up such shapes under the pull of gravity and the influence of complex fluid dynamics. Interactive programming became a method of defamiliarisation for breaking down the "magic" idea of objects that appear fully-realised from a printer, and reframing it as intentional, detail-oriented, incremental making.

Using code in such a way had the effect of preventing users from directly manipulating forms in favour of opening up a more intellectual and thoughtful distance from the act of making and designing that was both enabling and sometimes frustrating, as we saw. It was enabling because using an I3DP system was seen to help these participants better understand the basic mechanics of 3D printing. The workshops sparked some productive conversations, and helped people reframe 3D printing from a semi-magical, "black box" machine, to an understandable physical and computational process consisting of some straightforward steps using the simplified commands from the LivePrinter system.

When not feeling enabled, people signalled their frustration with the complex process of making even simple shapes. Squares and simple polygons took only a few lines of code, but when shapes were combined and built on top of one another to create truly 3D forms, the large amount of code required to make them began to approach an upper conceptual limit of what most people could understand. In the workshops, one solution was to include intermediate software to create more complex forms and then export them as self-contained, "plug and play" objects that users could directly integrate into their code. We introduced a workflow using a popular visual vector graphics shape editor to design forms that were converted directly into LivePrinter code (standard JavaScript) which was mostly well-received. These 2D shape paths could be scaled, rotated, stacked on top of one another, and printed on different materials like fabric and paper (as described in Subsection 7.4.8 (Task 8: Using LivePrinter (freestyle drawing)) and in the user feedback section of Subsection 7.4.7 (Task 7: Using LivePrinter (height and layering))). As one participant put it, these pre-rendered shapes provided "more recipes" for users to quickly to make more complex forms and learn from them.

This points towards future avenues of research into how to describe more complex forms by developing new syntactical abstractions, but without sacrificing the clarity and precision of tool path programming that the current syntax supports. These "recipes" for form-making could take inspiration from the study of metamaterials, where repeated typologies of micro-forms are used in layered combinations that result in different material properties, such as variable stiffness, floatation, and even mechanical operations.

9.4 Searching for a grounding metaphor of 3D "printing"

The search for a workable level of abstract, metaphoric language to conceptually anchor the descriptions of forms and simple operations formed an important part of the programming-language design process. For example, take the name "3D printing," a particular form of CNC manufacturing referring to a family of layer-by-layer, semi-automated fabrication processes. We call them "printing" because, ostensibly, it extends the familiar metaphor of desktop paper printers that have often been attached to personal computers.

Interestingly, in our study, a possibly more effective metaphor for the process was found to be "painting," as in "painting with plastic." This was recorded in early sketches of the system by the main researcher, arose in discussions with participants in Subsection 7.4.5.1 (Feedback and responses), and similarly came up earlier in initial interviews with Patricio Rivera from collective TMTMTM (The Machine That Makes The Machine) who more explicitly used 3D printing mechanics to paint with more traditional materials like acrylic paint, or with light using cameras and long exposure photography.

The fundamental importance of establishing this grounding metaphor is hard to understate – it forms the root of **all** the naming conventions in an I3DP system's syntax and creates consistency across different workflows. For example, the metaphor of *3D printing* leads to function names such as in *print()* and *printspeed()*, and likewise *3D drawing* (inspired by the *3D Doodler* pen) led to *draw()* and *drawspeed()*. *3D Painting* was left unexplored, but given the response in the user workshops, could be a strong future contender.

As it stands now, for better or worse, the LivePrinter system simply supports **all** of these workflows by making these functions synonyms of one another. This creates a number of competing abstractions and also introduces the possibility of mixed metaphors in programs. Future studies would do well to look at the effectiveness of each of these metaphoric systems, and in what context. They are likely to vary according to the background of the user, especially their previous experience with painting or making and what physical or metaphorical systems of language they are used to using to describe their process.

9.5 Problems and opportunities when describing 3D movements

Finding a grounding metaphor for print head movements in 3D space was also challenging, and could be a useful avenue for further research. With the specific printer models we used, how could users visualise the

3D movements of the printer, whilst looking at it? In 2D, we found that using code to describe how to move the print head was straightforward enough, but became confusing to participants when 3D movements such as "up" and"down" and moving at angles were introduced (see Subsection 7.4.10 (User's perspective vs. method naming)).

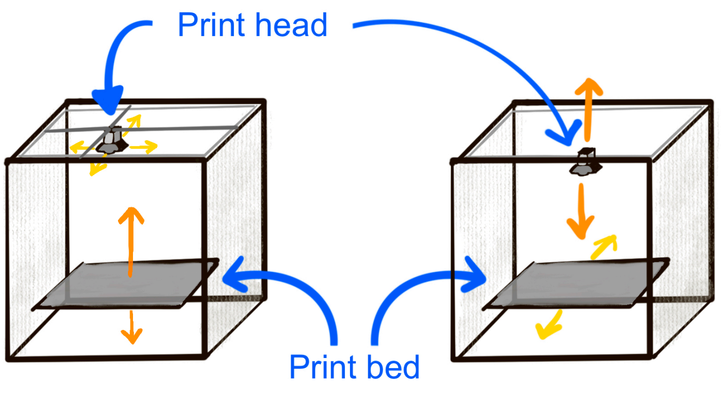


Figure 9.1: Two types of printer movements, showing one with a bed fixed in the horizontal position that moves only vertically with a vertically-fixed print head that moves horizontally (left) and another with a print head fixed in the horizontal position that moves only vertically with a vertically-fixed bed that moves horizontally.

Some reasons for this difficulty were due to differing printer designs, such as with printers that moved a print head horizontally using two motors and utilised a third motor to raise and lower the bed, versus printers that moved the bed horizontally whilst the print head only moved up and down.

Other difficulties were caused by looking at movements from the perspective of the bed or the model on the print bed. With printers that used fixed-position beds, users sometimes viewed movements commands as relative to the model on the bed, and at other times as relative to the print head itself. This was further confused by GCode convention, which is that a greater Z value means more distance between the print head and print bed, but with no set notion of which is fixed and which is moving.

These difficulties became particularly pronounced when users tried to direct 3D print head movements at arbitrary angles, as opposed to simply up and down (i.e. purely vertically). LivePrinter initially avoided this problem by taking inspiration from most slicer programs which divide 3D shapes up into 2D vector shapes oriented in the x–y (horizontal) plane (illustrated in Figure 9.2). The main vertical movement functions provided were up and down, although it was entirely possible to use the more explicit modes of movement and extrusion, e.g. `moveto({x, y, z})` and `extrudeto({x, y, z})`.

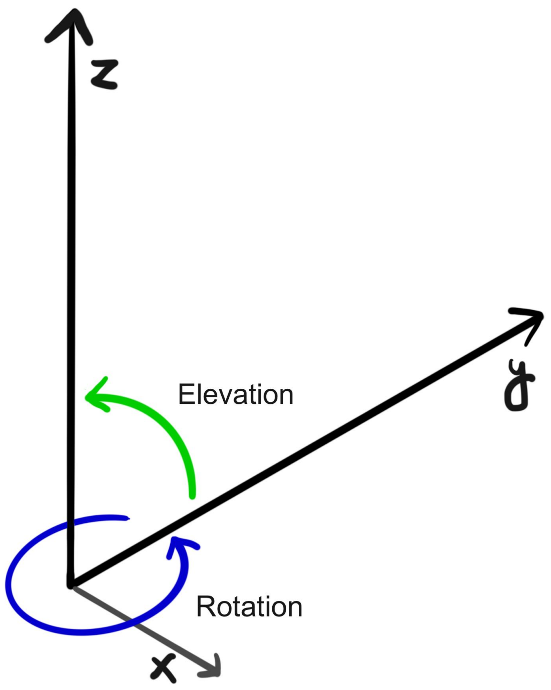


Figure 9.2: LivePrinter uses two types of angles to describe movements, vertical "elevation" and horizontal rotations around the z-axis, or "turns."

Complications arose later on when the drawing functions evolved to be chained =together to form "code sentences," based on user feedback and researcher self-reflection. For example, the `draw` function in the code sentence `# travel 15 | turn 20 | draw 45 | up 15` is explicitly 2D. Compare this to the later experiments using "air printing," discussed in Section 8.5 (Airprinting) and illustrated in Figure 8.15. Looking in detail at a subsection of commands, simplified here to focus on the drawing movements:

```
##
draw 40 | elev 35 | drawup 3
elev -35 | drawdown 3
##
```

In the first part of this code, the print head is directed to move horizontally for 40 mm (`draw 40`), then tilt the direction of travel upwards to a 35° angle (`elev 35`), then extrude material upwards at that angle for a distance of 3 mm ((`drawup 3`)). During the upwards movement of 35° from z=0 to z=3 mm whilst extruding, the print head will have extruded a total length of 5.23 mm ($\frac{3}{\sin(35°)}$ = 5.23) of material.

The `elev(angle)` function virtually "tilts" the direction of printing upwards by a number of degrees (i.e. the `angle` parameter) so that an elevation of 0 is fully horizontal and an elevation of 90 is fully vertical (upwards), with a corresponding elevation of -90 in the downwards direction.

The syntax was helpful for airprinting experiments where researchers were primarily interested in determining useful angles of elevation and moving exact vertical distances so that shapes sat properly on the printer bed, with less concern about horizontal dimensions for printed forms. This approach has its drawbacks for more general usage, but adding new functions for 3D vectors and rotations would have added a lot of complexity to the system at a late stage of development. In the already-completed user studies, movements in an upwards direction were uncommon, when more often users drew 2D layers and moved vertically only when beginning a composition or when a layer was completed.

It also exacerbated the "hard mental problem" of visualising the direction and distance of movements in 3D space, especially without other forms of visualisation such as 3D renders, screens on the printer, or even projections onto the printer space. Where and when to add pre- or post-visualisations to printer movements could be an interesting area for future research.

Additionally, it was not clear what frameworks to use when integrating more advanced concepts of 3D spatial movement and orientation. Aeronautics provides some grounding metaphors that could be useful, such as pitch and yaw, but these may not be as applicable to such a mixed 2D/3D frame-of-reference as 3D printing where 2D layers are prioritised over full 3D manoeuvrability. This points to future opportunities to develop conceptual systems for working more explicitly with 3D coordinate systems and 3D printing movements, whilst still retaining a layer-by-layer approach.

Such systems might also involve more sophisticated sensing and imaging technologies than those that currently exist, such as heat-resistant micro-cameras attached to moving print heads and highly-accurate measuring sensors, mixed with other computer visualisation approaches that give users the perspective of the printer and detailed knowledge of the properties of already-constructed physical layers. Perhaps augmented or virtual reality could be combined with such sensing to let future makers feel that they are a part of the printer itself, building on the AR-enhanced 3D fabrication experiments of Peng et al. (2018).

9.6 Reflecting on different approaches to composition

In addition to user activities of *exploratory design* and that are supported by LivePrinter, it can be helpful to reframe the livecoding process in terms of the users' approach to "composition" (of objects, or music, or any multimedia) to better understand the modes of working that LivePrinter supports. One compositional approach has been described as "planner," referring to users who try to work out compositional details before a livecoding or interactive programming session. The other approach refers to an iterative process of trying out small changes to understand their effect, and is often called either "bricolage" (McLean and Wiggins (2010) following Turkle and Papert (1990) and Turkle and Papert (1991)) or "tinkerer" (Ben-Ari and Yeshno, 2006, pp.1337–8).

We would expect more experienced users to gravitate towards a "planner" or "whole composition" approach since it requires a deeper understanding of how the system works and its patterns of use (e.g. an "internal model"). As we discussed in Section 5.4 (Aligning user understanding and notational systems), this approach is less "trial-and-error" and more strategic and conceptual. By comparison, bricoleur composers and tinkerers incrementally build up a composition through just such a process of trial-and-error. They often fashion little pieces of code by tweaking parameters and syntax in a separate file and stitch them together during the composition process (McLean and Wiggins, 2010).

The process of bricolage has been observed to be more common to beginners and users with little previous understanding of the system at hand. The original French meaning of word *bricolage* reinforces this contrast between advanced and beginner or professional and amateur: Catherine Letondal in Ben-Ari and Yeshno (2006, p.1337) translates it as referring to "work by a competent amateur." Similarly, calling someone a "tinkerer" is often an insult referring to their unprofessional or uneducated approach that belies their lack of systematic knowledge.

This does not mean that an advanced user *won't* also tinker with their code. Bricolage, like "auteurism," can be an explicitly anti-strategic approach to working in order to capture a spirit of amateurism or "beginner's mind," but it does mean that a beginner or casual user is far less likely to have enough of an inner model of the system, such as the domain-specific vocabulary and syntax committed to memory, in order to compose away from the system itself.

9.7 On designing for extensibility and tweakability

The emphasis of this project was on experimentation, meaning that both the system its syntax had to support forms of trial-and-error development. In (Trevino, 2013, p.30) this is called "extensibility" and "tweakability". The term *Extensibility* relates to the system itself, representing the assumption that users gaining an understanding of the low-level construction of the system can then create extensions of it. These extensions might be new applications of all or part of the system, or develop a more flexible alignment between thought and individual programming style. *Tweakability* relates especially to the degree of user-controlled flexibility in the system's output, allowing the programmer use the higher-level means of development provided by the system to customise and control the low-level manipulations of outputs.

The LivePrinter front-end is both highly extensible and tweakable because it is both written in JavaScript and also uses JavaScript as a means of livecoding output. Even the minigrammar is ultimately compiled to JavaScript, meaning that it can itself be extended and tweaked during a live development or performance session. To support this, LivePrinter provides an extensible code module for a CodeMirror-based[2] text editor **liveprinter.editor.js**, an extensible and tweakable abstraction of the 3D printer **liveprinter.printer.js**, an extensible communications layer **liveprinter.comms.js** and also an extensible representation of the Graphical User Interface (GUI) **liveprinter.ui.js**. Additionally, there is a means for extending and tweaking the minigrammar via the Nearley[3] grammar file **language/lpgrammar.ne** and language extensions to the text editor.

2: `http://codemirror.org`

3: `http://nearley.org`

Tweaks to the running system are fairly straightforward, usually comprising operations where parameter values such as speed, direction, shifts or alterations in musical scales or even serial port connection speeds are modified using minimal code. In practice, it is more difficult to *extend* the system whilst it is in operation because it can only be done through the LivePrinter text editor embedded in the web browser. These sorts of extensions are ephemeral because they aren't saved anywhere and disappear once the system is shut down, unless users capture them separately.

Extensions of this sort might be adding new 2D geometries to the library of fill pattern functions, or new musical patterning functionality such as functions for creating bespoke arpeggios. They are likely to be quick to code and specific to the current task at hand. Often they are aimed at solving a particular compositional or performance-specific problem, such as Sam Aaron describes in (Alan Blackwell and Aaron, 2015). Sometimes the line between "extension" and "tweak" becomes blurred in these quicker sessions, once the "tweaks" have been saved and incorporated into the project's history.

Other extensions to the system, for example adding support for other 3D printer models or other fabrication machines, are expected to be more thought-through, strategic and longer-term undertakings. They may represent new ways of thinking through syntax, alternate ways of handling repetition, iteration and loops, and other avenues for more longer-term exploration. These modifications are better attempted in code editors geared towards longer editing sessions and recording session histories and alternative states (such as with git).

That doesn't mean that this system shouldn't support this type of experimentation. Even if many of the users of LivePrinter aren't "technical" (meaning, versed in coding) and are more comfortable in the role of product or fashion designer interested in exploring new ways of making, that does not mean that they aren't capable of some level of software engineering. Alan F. Blackwell and Morrison (2010, p. 8) observes that "end-user programmers, if working in a professional environment" (like a fabrication lab) "are very likely to need software engineering facilities" and failure to provide them could be taken as "a lack of respect" for their professionalism.

Also, the interests of these users likely differ from the main developer's:

> contrast between the priorities of technical and non-technical users has been a common theme in past work that has emphasised the importance of providing tools for end-users that offer visible progress toward achieving the user's own goals, rather than teaching abstract principles with no clear relationship to user priorities (Alan F. Blackwell and Morrison, 2010, p. 6). (This)

These distinctions are important because they show how a livecoding system supports a range of development strategies, from easy, quick and live tweaks in the moment of performance to non-live, longer-term strategic development of deeper ideas that extend the system into new conceptual territories. It is important to look deeper at what sort of coding behaviour the system expects of its range of intended users over the entire lifecycle of use. Different design strategies bring trade-offs between legibility (in typical usage of code) and portability (i.e. installation) that bubble up quite quickly to the end users who work mainly with in-the-moment tweaks and also to potential project collaborators or contributors who delve into the depths of the code in search of possible ways to extend it.

9.8 On ease of installation versus development

Until quite late in development, the emphasis on LivePrinter was on ease of installation and portability over extending the system. Users "installed" the software simply by downloading it and then running the main file in a Python interpreter, meaning that only Python had to be installed. This meant that most of the code for the system was in two text files, written in JavaScript: one for the communications, GUI and text editor, and the other a more encapsulated version of the 3D printer abstraction code.

It was hoped that by leaving the bulk of the code in two searchable files that could simply be included in other projects they would be more easily extensible as well as easier for novices to download and start using straight away, without any sort of external file manipulation or concatenation, interpretation or other build tooling installation. Whilst it was true that the project was easier to download and run quickly, the feedback from some other professional developers was that the large files were intimidating to try and understand.

There was clearly a trade-off between dividing up the project code into conceptual "chunks" (i.e. files and project directories) and the dependencies and prior experience with build tools that this would require of future developers with the legibility of the project as a whole. In May 2020, after conversations with Guy John of Livecodelab[4], the project was reorganised into such conceptual chunks using a standard JavaScript *npm* (Node Package Manager) build system called Browserify[5] and pre-built versions and instructions for building them from source code were provided on the GitHub page. As of this writing, the line count of the code and the amount of lines of code per file fell dramatically (in the 1000s) but any other potential benefits of that transition were still unclear.

4: `https://livecodelab.net/`

5: `http://browserify.org/`

9.9 Revisiting the start of the journey

> "...a Black Box ontology is a performative image of the world" where "a Black Box is something that does something, that one does something to, and that does something back — a partner in a dance of agency." A. Pickering (1995)

The original research questions evolved out of the initial direction of this research, which was focused on exploring the possibilities for visualising data as computationally-generated, 3D printed forms. Very quickly it became apparent that computational form-finding, a practice that has produced some interesting results on screen, was a much less well-defined enterprise when applied to a physical process like 3D printing. It was even more difficult using desktop 3D printers, which are relatively inexpensive compared to industrial methods but also produce less detailed objects that generally require more labour-intensive manual post-processing and finishing.

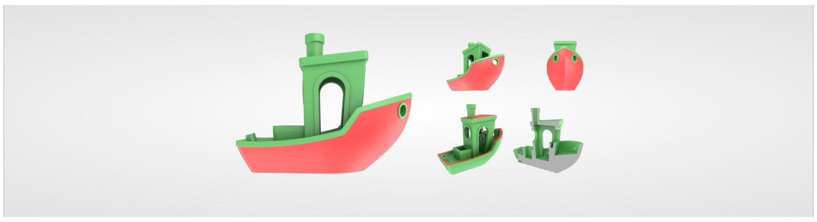


Figure 9.3: 3D Benchy, an object designed to stress-test 3D printers by requiring difficult printing operations. Image from Wikimedia Commons, licensed under the Creative Commons Attribution-Share Alike 4.0 International licence.

At that time, there were few, if any, peer-reviewed or trustworthy sources for essential printing techniques and parameters needed to create complex forms on desktop 3D printers. Academic research into optimal printing settings certainly existed for more industrial methods like Selective Laser Sintering (SLS) and ABS, (Kranz, Herzog, and Emmelmann (2015) and Sung-Hoon et al. (2002) are two examples), but such studies were often missing for the less expensive and more widely available consumer PLA and FDM printers. It was hard to find good data on minimum features sizes, maximum and minimum vertical slopes for curved surfaces, and on how these were affected by different printing speeds and material temperatures.

Perhaps this was due to the diversity of printer brands and rapid iteration of models, but a larger part of the problem had to do with the difficulty of sharing test results in standard ways. A major stumbling block was that standard 3D printing file formats are mainly geometry-based and thus lack the machine-specific knowledge needed to fabricate them. This is an issue that Baudisch and Mueller (2017, p. 251) discussed in detail in their book. In the absence of a standard format for sharing that knowledge, most of it exists anecdotally on message boards and Internet forums.

6: `https://www.thingiverse.com/thing:763622`

Some special 3D-printable models do exist, like the “3D Benchy” boat[6] seen in Figure Fig. 9.3, but due to these file format limitations they contain only basic geometry, lacking any helpful metadata for specific printer settings, and are mainly designed to help operators “stress test” and tune their 3D printer's performance rather than allowing them to discover different techniques for fabricating complex forms.

On reflection, it was entirely possible to use code as a means for creating and specifying the design of printable objects for a range of people with mixed technical backgrounds. LivePrinter workshops demonstrated that coding objects directly can create new forms that are not limited by CAD tool metaphors and take full advantage of 3D printers, like printing sparse forms in the air.

They also raise questions about current object-centric metaphors for describing how objects are fabricated, opening up the possibility of hybrid, tool path- *and* object-related descriptions that blend the manufacturing process with the desired manufactured object. New expressive languages like the LivePrinter “minigrammar” will need to be further developed to support ongoing practice and research, alongside 3D printer hardware design to support these new metaphors.

Conclusions 10

In the beginning, Interactive 3D Printing (I3DP) was seen as a means of transcending the current limitations of software for controlling how objects are fabricated on 3D printers, after a series of difficult experiments trying to 3D print new computational forms. It was designed to fulfil a need for higher-level, tool path-specific control of 3D printers, in addition to the nearly machine-instruction level possible using only GCode.

By the end of the thesis, the research focus around I3DP had settled into four main questions, influenced by Baudisch and Mueller (2017)'s articulation of six challenges for *Personal fabrication*: Domain knowledge; Visual feedback and interactivity; and Machine-specific knowledge. They were also heavily influenced by my own art practice of livecoding and my experience as a computational art teacher:

1. **Can livecoding 3D printers help participants understand how the process of manufacturing using 3D printers relates to their discipline**, so they can start to experiment usefully with it (or not)?
2. **Can livecoding 3D printers allow users to create physical forms using novel functions that take into account physical properties like speed and temperature,** instead of the usual method of beginning with 3D modelling and automating fabrication?
3. **How can the combination of visual aesthetics and musical concepts lead to knowledge about new digital manufacturing toolpaths, and vice versa?** (e.g. exploring how 3D printing toolpaths can be influenced by concerns other than optimising for speed and strength)
4. **How can a livecoding environment be useful for 3D printing?**

The search for answers to these four questions took me down a number of avenues, leading to a number of outcomes that were performative, sculptural, designerly, intellectual, and technical:

As a reminder, a full listing of outcomes and activities was introduced in Chapter 4 (Methods).

- ▶ an analysis of seven user interviews of experts across different forms of 3D printing, art and design practice
- ▶ a framework for a system of interactive 3D printing (I3DP) (in Chapter 5 (Designing for Interactive 3D Printing))
- ▶ a functioning software system implementing that framework, called *LivePrinter*
- ▶ documentation for LivePrinter
- ▶ thirteen user workshops, with at least forty-one participants in total
- ▶ five public performances of livecoding 3D printing
- ▶ two exhibitions of sculptural work — The Design for Change pavilion at the London Design Festival '19; Expressive '19
- ▶ novel experimental 3D printing techniques (see Chapter 8 (Filling space, filling time)) such as using space-filling Hilbert curves for performing sculptural techno; "airprinting" sparse 3D shapes on unmodified desktop PLA 3D printers

The development of the prototypical I3DP software system that forms the core of this thesis, *LivePrinter*, began in earnest in June 2018, a few years after the initial work had begun. During the initial direction of this research, which was focused on exploring the possibilities for visualising data as computationally-generated, 3D printed forms, it very quickly became apparent that computational form-finding, a practice that has produced some interesting results on screen, was a much less well-defined enterprise when applied to a physical process like 3D printing. It was even more difficult using desktop 3D printers, which are relatively inexpensive compared to industrial methods but also produce less detailed objects that generally require more labour-intensive manual post-processing and finishing.

Developing an interactive programming environment for controlling 3D printers was a relatively large technical undertaking that took over two years to get to a functional state, and is still in progress as of this writing in 2022. The design, testing, and development of this software helped establish the idea of I3DP and showed that it was possible, potentially useful, and could be used to create works of art and design.

During LivePrinter's development, the research process was heavily influenced by the interviews with other 3D printing and livecoding practitioners. Then, when the LivePrinter technical system and documentation was at a fairly complete stage, the user workshops were developed, planned and run with potential users in the creative industries, other practitioners of creative technology, and students at both graduate and undergraduate level, as discussed in Chapter 7 (User Studies and Analysis).

This technical system (essentially, the livecoding environment and supporting software interface for 3D printing hardware) established a practice-led research vehicle for testing out ideas, integrating results, starting conversations with users and other software developers and to reflect on at key points in the process. The project's documentation captured fundamental concepts of digital manufacturing and served as a dialogue between the designer and users, especially during research sessions. It also grounded the project in the history of augmented manufacturing in general and 3D printing in specific. Finally, both the documentation and experiments illustrated the possibilities of augmentation, both creative and mundane.

The user studies showed that the LivePrinter I3DP system was successful in showing participants how 3D printing could be integrated into their practice, or not. Their experiences in successfully completing all the workshop tasks showed that a livecoding environment was indeed a useful archetype for interactive 3D printing, even if it was a new and sometimes uncertain experience for some. This success of these workshops, and future issues to explore based on our experience of running them, were summarised in Section 7.5 (Summary findings: issues for further exploration).

Both the production of artefacts and planning and delivery of live performances were mainly self-reflective processes, although the artefacts and impressions created by these processes opened up opportunities for further dialogue with the community and beyond. Inversely, the user studies were mainly an opportunity to collect data from others' experiences, but also involved a measure of self-reflection in their design and in the analysis of results between sessions.

These artefacts, performances and their associated reflections document the process of working within a new discipline of interactive augmented manufacturing, and point towards some of the new techniques that will need to be established to further support it. This thesis introduced the concept of "filling space, filling time" in Chapter 8 (Filling space, filling time) as fairly literal framing device for these challenges. During the immediacy of a live performance it can be difficult to think in terms of both time and 3D space when directing a machine along a complex path. This presents opportunities for computational augmentation and more standardised techniques that could correspond with the development of metamaterials, 2D/3D building blocks like space-filling curves, simple triangles, or sparse pyramidal forms.

The documentation from of all thesis activities is collected in hours of video of observations of workshops, recordings of performances, audio and video recordings of interviews, questionnaires from workshop participants, reflections from self-practice journals before and after performances, photographs of exhibitions and associated exhibition catalogues. There are also tens of thousands of lines of code and pages of documentation for the LivePrinter system, as well as a public, open source repository where it can be downloaded and modified[1], correspondence from participants and collaborators, and reflective journaling.

1: https://github.com/pixelpusher/liveprinter

Finally, reflections on the outcomes help us understand how this new work should be interpreted in terms of what came before. The results lay out a rationale for I3DP and livecoding performances with 3D printers, as well as a blueprint and some helpful scaffolding in the form of working tools for practitioners looking to get started in these practices.

Appendix A

A

A

This Appendix collections audiovisual records of some performances and events from throughout the thesis, organised by event type and name.

A.1 Selected Performances

A.1.1 Goldsmiths Algorave 2019

Event poster and listing: https://gold.ac.uk/calendar/?id=12264

Video of the process of livecoding the Algospiral for the flyer for the Goldsmiths Algorave for TOPLAP 15: http://content.flkr.com/liveprinter/printing%20algorave%20logo%20jan%202019.mp4

Performance video of the Goldsmiths Algorave for TOPLAP 15: https://www.youtube.com/watch?v=_IRMDliYkgs&list=PLuA35183Y-6-GB69A7t3pcTRx6nrI7QxO&index=2

A.2 Experiments

A.2.1 Pathfinding automata

Running the path-finding automata algorithm use in the Expressive '19 exhibition artefacts:

https://www.youtube.com/watch?v=0if6rfPExQU&list=PLuA35183Y-6-GB69A7t3pcTRx6nrI7QxO&index=6

A.2.2 Hilbert curves techno experiment

Short demonstration of Hilbert curve techno livecoding for the ICLC 2020 conference. This video uses unprocessed audio from contact microphones on the printer: http://content.flkr.com/liveprinter/hilbert_experiments/2020-01-04-11.mp4

A.2.3 Various demonstrations

A "Demonstration video" of a range of LivePrinter experiments for public engagement, including airprinting:

https://www.youtube.com/watch?v=POwENjC6qO4&list=PLuA35183Y-6-GB69A7t3pcTRx6nrI7QxO

A.3 User workshops and public demonstrations

V&A Museum (London) flyer for the April 2016 demonstration of LivePrinter at the Digital Weekend: https://vanda-production-assets.s3.amazonaws.com/2018/08/14/09/09/32/a9a2e652-bff4-4240-a66a-cb010f3961b9/V&A%20MUSEUM%20BOOKLETvfinalnobleed.pdf

Eventbrite listing for 3D printing user study workshops at Goldsmiths (2019): https://www.eventbrite.co.uk/e/liveprinter-3d-printing-research-workshop-tickets-53682186866?ref=estw#

Brooklyn Research event link: https://brooklynresearch.org/eventbrite-event/liveprinter-instant-3d-printing-a

A.4 Other materials

A.4.1 Quickstart demo video

A video aimed at getting users up and running quickly, now slightly out of date:

https://youtu.be/jdqrpgFGCgc

Appendix B

B.1 Workshop plan for the researchers

The following workshop plan was solely used by the researchers as a guide to running the workshop. It outlined the workshop structure, along with preparation and setup tasks, included the structure for interviews and observations and some basic theories for context. Note that personal emails of researchers have been removed, otherwise it is included here in its (rough) state.

User Workshop Plan (for researchers)

B.1.1 Before the Workshop

- Download and install Visual Studio code and Python support (or python)
- Make coffee/tea and bring snacks
- Email everyone about workshop & prep
- Produce example materials
- Consent forms
- Questionnaire and extra paper & pens
- LOGGING FOR LIVEPRINTER -- G-CODE, results, JavaScript (save code when compiled with timestamps) -- session ID
- Self-recording sheet for each task/feature (1-5 likert scale bad/good time + comment)
- Cog dims at end questionnaire
- Get occupation/background

B.1.2 Materials Needed

1. Metric ruler (small) for measuring moves and distances
2. Small red pointer lasers for triangulating and highlighting head position
3. Paper (to print on)
4. Blue tape (to print on)
5. Glue stick (to help prints stick)
6. Metal tweezers (for filament issues)
7. Exacto knives (to cut stray bits of filament)
8. Cameras & tripods for pictures/video
9. Microphones for sound recording, plus audio device
10. Notebooks & pens for sketching

B.1.3 Overview Blurb

I'm running some research workshops in early January for my PhD/research project with live 3D printing called LivePrinter, and need some participants. LivePrinter is an open source system for live, immediate drawing and fabrication with 3D printers. It's particularly useful for:

- Textile artists who want to print onto fabrics and make new shapes and textures; for artists who want to use a printer like a 3D plotter and draw new forms
- Product and industrial designers who want to understand more about how 3D printing works and fine-tune their materials and tool paths
- Materials scientists who want to study 3D printing materials in more controlled, repeatable ways
- Computational and computer artists, either looking for new tools or making generative works
- Educators who teach fabrication
- HackSpace and MakerSpace staff who need more tools to fine-tune their machines

This workshop is a free introduction to the basics of 3D printing. It also introduces the new LivePrinter system for directly drawing shapes, lines, and objects using code:

- How 3D printing uses digital motors and hot melted plastic to make precise and intricate shapes, layer by layer
- How your 3D programs communicate with 3D printers and other CNC fabrication devices using G-Code
- Hands-on introduction to using LivePrinter: setting it up
- Using LivePrinter to drawing shapes with plastic in different ways
- Turning vector drawings and art directly into 3D prints
- Printing on paper, fabric, and other materials

Tea/coffee/snacks will be included.

There will be 4 sessions, limited to 8 people each:

Wednesday 9th January, 10am-1pm

Wednesday 9th January, 2:30pm-5:30pm

Thursday 10th January, 10am-1pm

Thursday 10th January, 2:30pm-5:30pm

Sessions will be held in the new state-of-the-art Hatch Lab, the workshop at Hatcham House (the old church), Goldsmiths University: `https://www.gold.ac.uk/find-us/places/hatcham-house-hh/`

Contact: e.raskob@gold.ac.uk

B.1.4 Hands-on workshop outline

1. What is 3D printing? Mechanisms, software, printers: 4 stepper motors (x,y,z,e) and a microcontroller (Arduino or PI or another)
2. How are objects built from layers? Quick diagrams of plastic flow, calculations of filament extrusion lengths and layer heights (to be picked up in the hands-on section)

Installing and setting up LivePrinter:

1. Download from github
2. Installing Visual Studio / VS Code and python
3. Connecting a 3D printer (or using the 'dummy' printer)
4. Running the LivePrinter server

Using LivePrinter (note: these steps are built-in to one of the examples so they can livecode these directly):

1. Turn on printer
2. level bed (on printer) — (note: can also probe via gcode)
3. start server
4. open `http://localhost:8888`
5. click printer settings tab
6. select serial port
7. click code tab
8. run lp.start(210) to set temperature and ready bed
9. click Printer tab and click button for start temp polling

10. (wait for printer to warm up - watch temperature readout)
11. Try moves with no extrusion - absolute (centre of bed) and relative
12. when head is hot, move bed to low (z = 50 or so) and start extruding until material comes out (using relative e values like 15 and low speeds like 5-10)
13. Remove excess filament
14. move head to z = layer height
15. extrude a line, absolute and then relative
16. Extrude a line and turn
17. These are the things you can do *now:*
    a. Make a shape (square, etc.) ---- play with shapes (circle, square)
    b. ----- stack them up (z)
    c. Make a wall? Bracelet?
    d. L-System -- snowflakes
    e. Stamp / SVG paths
    f. Printing on paper, fabric
18. Look at G-code readout to understand what's happening
19. experiment drawing lines: thickness & speed
    g. Change lp.layerHeight or use "thickness" parameter
    h. Change speeds
20. Retraction settings for slow and fast drawing
    i. Look at filament pooling - enable/disable retraction, change retraction length
21. Advanced: making your own functions (global and part of lp namespace)
22. *make a base for future shapes*
23. try extruding a line into space: x:10, z:20, speed:3, thickness:0.05 (close to nozzle width)
24. note the difference in filament between extruding in space and extruding a line on top of another (one has to do with nozzle radius, other is a flattened smear based on nozzle height)
25. experiment with external fans for cooling
26. Try loops to make complex shapes
27. Discussion

B.1.4.1 Factors to test in interview and observationally during the workshops

(using the cognitive dimensions framework from Thomas R.G. Green and Marian Petre):

1. Abstraction - types and availability of abstraction mechanisms
    a. Names of functions - do they map to known concepts and physical reality?
    b. Code constructs (syntax - as much as it ban be changed to make more "natural" sense)
2. Hidden dependencies - test if important links between entities are not visible
    c. Is there anything unclear about the system startup and device selection? (probably) and what is it?
    d. Are there functions/menus/submenu items that aren't readily visible or easy to find? For example, selecting a serial port.
3. Premature commitment - constraints on the order of doing things
    e. Does the user feel that the workflow makes logical sense to them? Can they follow it step-by-step successfully or do steps get confused?
    f. How do people feel about the livecoding concepts of chained functions vs. declarative ones? Is one more intuitive to them, or more efficient?
4. Secondary notation - extra information in means other than formal syntax
    g. Are there hacky function calls or other examples of the system rules being violated?
5. Viscosity - resistance to change
    h. Can the user make what they want, without frustration? E.g. is the system flexible enough to accomodate them without making major changes? Must other functions be added to help enable that?
    i. When properties are set, how hard are they to change? Or are they too easy? For example, retraction settings and boundary modes.
6. Visibility - ability to view components easily
    j. Is there any missing information that the user needs to complete their task? Anything left unknown?
    k. Do they feel like they understand what's going on in the printer? Can they articulate that?

B.1.5 Interview & Questionnaire Plans

B.1.5.1 Goals

Need to understand: is this system useful, desirable, usable (Sanders 2006)

Users: what people's strategy is towards 3D printing: how do they approach it, what do they do (or not do) with it. What is their understanding of it? (self-identity, beliefs, attitudes, motivations, behaviours)? These are potentially influenced by factors like their background, working environment, education level, income, etc. This helps create an MUT (media user typology)-informed list of "user types" to design and validate against (for usefulness, for one). Need to keep this as simple as possible, but still be able to make conclusions.

Usability: how well does the software/hardware system currently support the user's needs? (cognitive dimensions framework below)

B.1.5.2 Notes

Things to avoid:

- Having people guesstimate numbers without getting direct evidence, like "How many people do you usually see in a week?"
- The “novelty effect” where you ask someone if they would like something new and shiny and they of course say yes
- The social desire to please the observer (`https://en.wikipedia.org/wiki/Social_desirability_bias`) neutrally or in 3rd person.

B.1.5.3 Questionnaire

Purposes: to establish a background level of ability for participants to test against usability; to collect personal user data that would be tedious in interviews like (adapted from Brandtzæg 2011 as per notes below):

(1) Country: collect home country and country of residence for analysis

(2) Access: ask if they own a 3D printer, or get access through school, makerspace, etc.

(3) Gender: Numerous studies have found a large gender divide in the use of Internet (e.g., Hargittai, 2010, Dholakia, 2006, Bimber, 2000) and in technology adoption contexts in general (e.g., Venkatesh et al., 2003).

(4) Age: A generational divide is identified between older and younger Internet users - see how 3D printing users fit this

(5) Household: will presence of children correlate with 3D printing access? Maybe not, but worth a check.

Adding, based on their research:

(6) Employment status (student, self, fully, none, other)

(7) Disability

(8) Occupation - how do they self-identify?

ADAPT these to 3D printing

From Carillo 2017’s review of common user classification criteria:

1. Frequency of use: the rate at which the use of a system occurred over a particular period of time in the past.
2. Computer Knowledge: the skill level or capability a user has regarding the use of technology in general, or a specific computer system in particular.
3. Interface Knowledge: the user’s familiarity or acquaintance with the system’s interface and analogous systems.
4. Motivation: the reason that triggers the use of the system.
5. Other: such as task domain knowledge, programming experience, technical knowledge, ambition of mastering the system, or range of operations (i.e., task structures).

This will need to be anonymous and secure (collect emails separately - via Eventbrite workshop list)

B.1.5.4 Semi-structured interview: their practice

1. How do they self-identify? Their role(s): (self-articulated, like "designer" or "3D artist"). What do you call what you do?
2. Ways, meaning, experience - where did you learn your role?
3. what is most important in your work: singularity, reproducibility, quality, provocation, time/speed of making?
4. Skills: how do you do things for your work? Sketches, by hand, writing, coding, by machine...
5. Community: where do you do your work?

B.1.5.5 Structured Interview: Experience Coding

1. What kinds of coding have you done? When? What is your favourite language and why?
2. Is coding part of your usual practice? If yes, what do you use it for? Why?
3. What do you like about coding?
4. What bothers you about coding?

B.1.5.6 Semi-structured Interview: 3D printing Experience

1. What kinds of 3D printers have you worked with? When?
2. [prompt: optional] What is your favourite printer and why?
3. Is printing part of your usual practice? If yes, what do you use it for? (fabric printing, sculpting, algorithmic imaging, product design, DIY)? Why?
4. [prompt: if not answered] What do you like about 3D printing?
5. [prompt: if not answered] What bothers /frustrates you about 3D printing? What is limiting about it?
6. If part of their practice: What is your workflow, e.g. describe how you come up with a concept (or acquire one) and then get to a finished object. *Guided storytelling:* (get them talk about a specific situation where they 3D printed and what they did - perhaps diagram on paper)
7. [prompt: if above isn't clear] How do you start printing? Is there special setup that you do?
8. What would you like to print but can't? Why not? (drawing exercise below)

B.1.6 Other [Future] Research Workshop Ideas

B.1.6.1 Research probe - day in the life

Purpose: identifying missing workflow steps; identifying other unforeseen styles of making that could possibly be integrated; understand users and behaviour; identify new/key collaborators

Method: Get people to document their process. Day in the life: take one day and have them take photos of their setup, photos of prints and work in progress, photos of finished work. Textual descriptions too if possible. Schedule follow-up interview (over phone, etc.) to have them describe what's in the photos.

Analysis:

- Environment: Where are they (categorise) and what are the key characteristics (obstacles, lighting, space constraints, smells, etc.)
- Actors: whom is present?
- Actions/activities: what are the actors doing?
- Objects: what are the interesting objects in the space (printers, tech, tools, filament, workspaces, storage racks, informational posters on the wall, etc.)

B.1.6.2 Workshop - free sketching shapes

Purpose: understanding the abstraction needed for coding: function names, methods; identifying missing workflow steps; identifying other unforeseen styles of making that could possibly be integrated

Method: Tell them it can draw lines in space and ask them to describe them in words (like code). Have people sketch algorithms based on possible printer movements (not hitting shapes?) Save sketches.

Generative (algorithmic) vs. specific (heuristic? concrete?) forms:

1. [if time allows:] Show pictures of generative/coded art like #plottertwitter and discuss

B.1.7 References

Sanders, E. (2006), "Design research in 2006", Design Research Quarterly, Vol. I No. 1.

Dennett, D.C. (1995), Darwin's Dangerous Idea. Evolution and the Meanings of Life, Penguin Books, London.

Jonas, W. (2007) Research through DESIGN through research: A cybernetic model of designing design foundations.

B.2 Workshop tasks feedback sheet

This user-facing sheet was given to all workshop participants to record their feedback during the workshop. Each section corresponds to a workshop task, with an associated Likert scale for recording quick, context-specific feedback.

During the talk please rate each part here:

B.2.1 Task 1: Introduction talk about 3D printing

How do you feel about this section?

Disliked this part	Slightly disliked it	Neither liked nor disliked	Slightly liked	Liked this part

Any Comments?

B.2.2 Task 2: Introducing LivePrinter (overview)

How do you feel about this section?

Disliked this part	Slightly disliked it	Neither liked nor disliked	Slightly liked	Liked this part

Any Comments?

B.2.3 Task 3: Installing LivePrinter

How do you feel about this section?

Disliked this part	Slightly disliked it	Neither liked nor disliked	Slightly liked	Liked this part

Any Comments?

B.2.4 Task 4: Using LivePrinter (basics)

How do you feel about this section?

Disliked this part	Slightly disliked it	Neither liked nor disliked	Slightly liked	Liked this part

Any Comments?

B.2.5 Task 5: Using LivePrinter (printing a square)

How do you feel about this section?

Disliked this part	Slightly disliked it	Neither liked nor disliked	Slightly liked	Liked this part

Any Comments?

B.2.6 Task 6: Using LivePrinter (retraction & material flow)

How do you feel about this section?

Disliked this part	Slightly disliked it	Neither liked nor disliked	Slightly liked	Liked this part

Any Comments?

B.2.7 Task 7: Using LivePrinter (height and layering)

How do you feel about this section?

Disliked this part	Slightly disliked it	Neither liked nor disliked	Slightly liked	Liked this part

Any Comments?

B.2.8 Task 8: Using LivePrinter (freestyle drawing)

How do you feel about this section?

Disliked this part	Slightly disliked it	Neither liked nor disliked	Slightly liked	Liked this part

Any Comments?

B.2.9 Task 9: Future Use

How do you feel about this section?

Disliked this part	Slightly disliked it	Neither liked nor disliked	Slightly liked	Liked this part

Any Comments?

Appendix C

C.1 User Workshops presentation

The following section contains the slides used in the user workshops that introduced LivePrinter and gave it some historical and contemporary context. This version is the second version, edited after the first day of user workshops to include more context-specific API documentation and to reduce the amount of background material at the start.

LivePrinter Workshop

By Evan Raskob
Goldsmiths, University of London
e.raskob@gold.ac.uk
v.01

Project page: https://github.com/pixelpusher/liveprinter
Email list: http://pixelist.info/liveprinter-email-list/
Forum: https://talk.lurk.org/channel/liveprinter

Me:

- Lecturer at Goldsmiths (Physical & Creative Computing, MA, MFA, BSc)
- PhD student Goldsmiths (part-time)
- 12+ years teaching art, design, computing
- Practicing artist (mixed media; interactive)
- LiveCoding performer since 2007 (as BITLIP & BITPRINT and pixelpusher)
- http://pixelist.info // @evanraskob (twitter) // @evanraskob_art (Instragram)

Today:

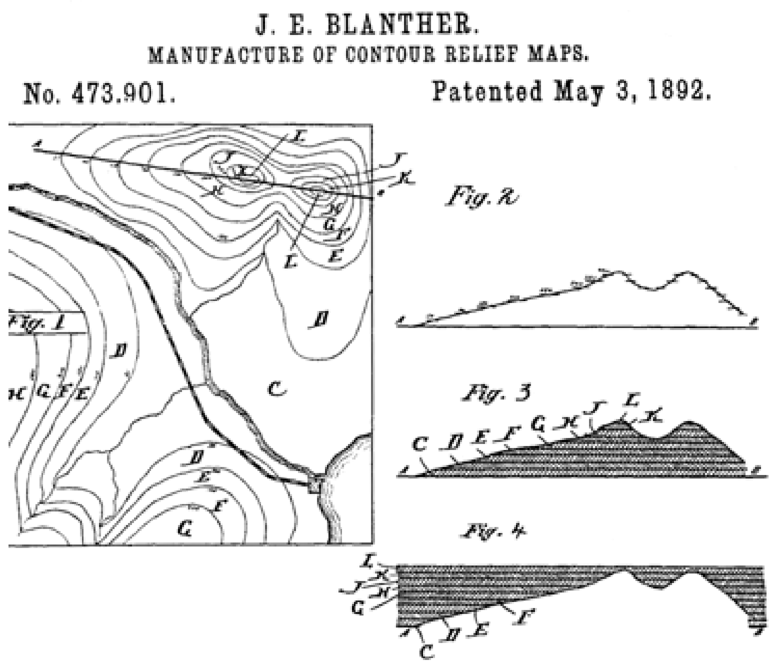
- Brief overview of 3D printing (additive manufacturing)
- Introducing the LivePrinter project
- Hands-on demonstration and workshop of LivePrinter
- [break]
- Final questions & interviews for research

Types of Manufacturing

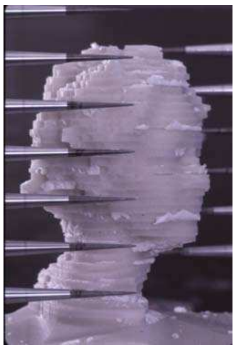
- Weaving
- Subtractive – carve or remove material
  - CNC router
  - Laser cutter
  - Hand tools
- Moulding / casting
- Thermoforming
- Welding
- Additive manufacturing
  - 3D printing
  - Deposit material in layers
  - Solidify material (light, lasers, powder)

Maps from layers

- Blanther's countour map moulds made of layers (from http://www.wtec.org/loyola/rp/03_01.htm), Patent no. US000473901
- Stacked wax plates cut out following the contours of topographic maps

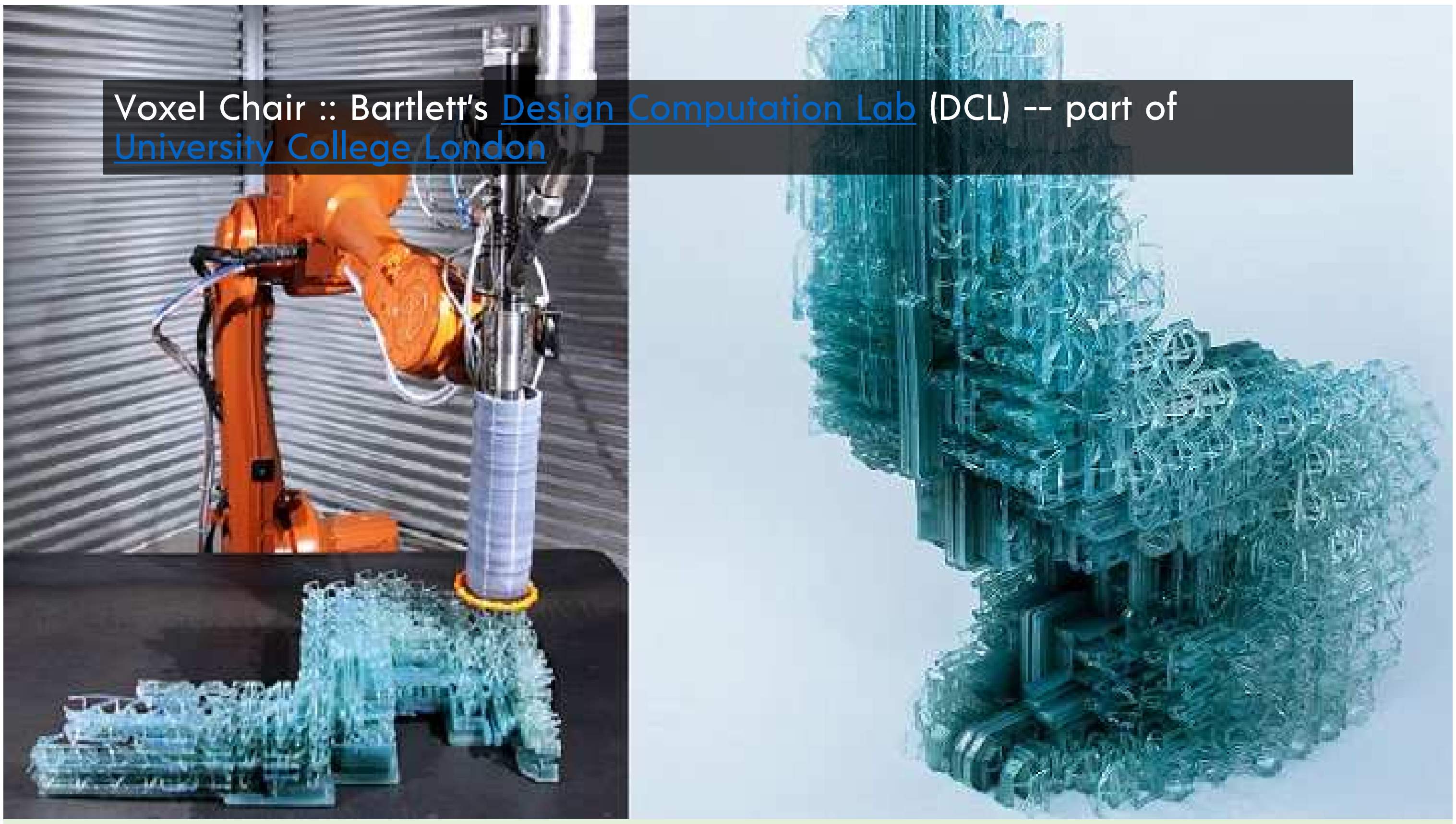


- Sculpture by Solid Photography process (No longer in business) (Photos courtesy of SOHO Image Works) from Solid Freeform Fabrication: An Historical Perspective by Joseph J. Beaman http://sffsymposium.engr.utexas.edu/Manuscripts/2001/2001-66-Beaman.pdf

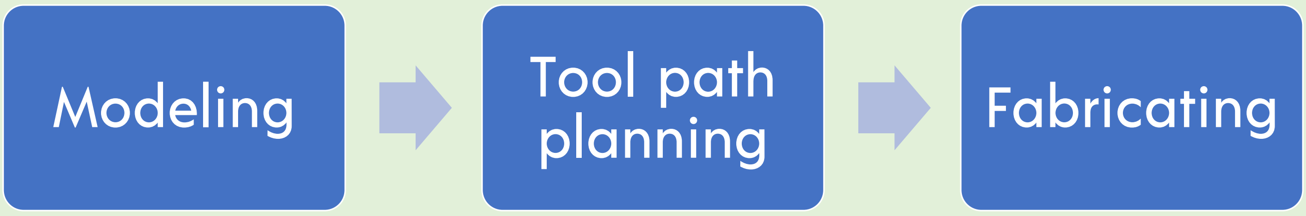


GCODE.clay by Ronald Rael, Virginia San Fratello, Kent Wilson, Alex Schofield

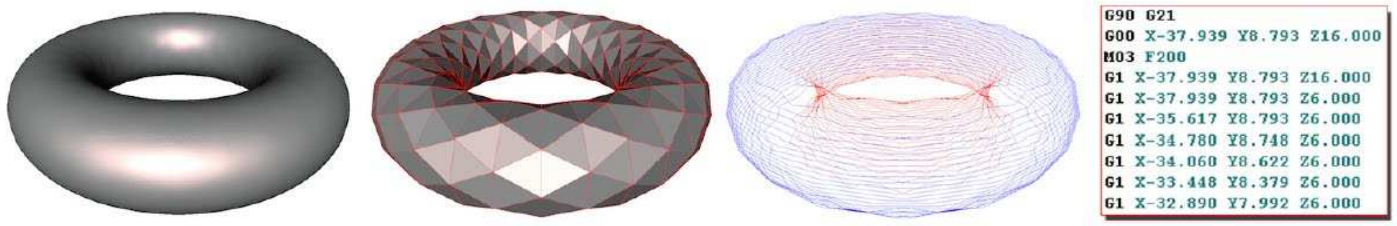
http://www.slate.com/features/drivingforces/ceramics/index.html


Voxel Chair :: Bartlett's Design Computation Lab (DCL) -- part of University College London

Basic 3D Printing process:



M. Livesu, S. Ellero, J. Martínez, S. Lefebvre & M. Attene / From 3D Models to 3D Prints : an Overview of the Processing Pipeline



```
G90 G21
G00 X-37.939 Y8.793 Z16.000
M03 F200
G1 X-37.939 Y8.793 Z16.000
G1 X-37.939 Y8.793 Z6.000
G1 X-35.617 Y8.793 Z6.000
G1 X-34.780 Y8.748 Z6.000
G1 X-34.060 Y8.622 Z6.000
G1 X-33.448 Y8.379 Z6.000
G1 X-32.890 Y7.992 Z6.000
```

Figure 1: Typical process planning phases in additive manufacturing: a design model (left) is tessellated to enter the Process Planning phase (centre-left). Such a tessellation is sliced (centre-right), and each slice is converted to a sequence of machine instructions (right). Other operations are typically required depending on the specific technology at hand.

Geometry

- Adapt geometry to fabrication requirements. The (possibly tessellated) geometry must enclose a solid that the target printing technology can actually fabricate.

Orientation

- The model must be correctly oriented to fit the printing chamber, minimize surface roughness and printing time, reduce the need for support structures…

Support structures & infill

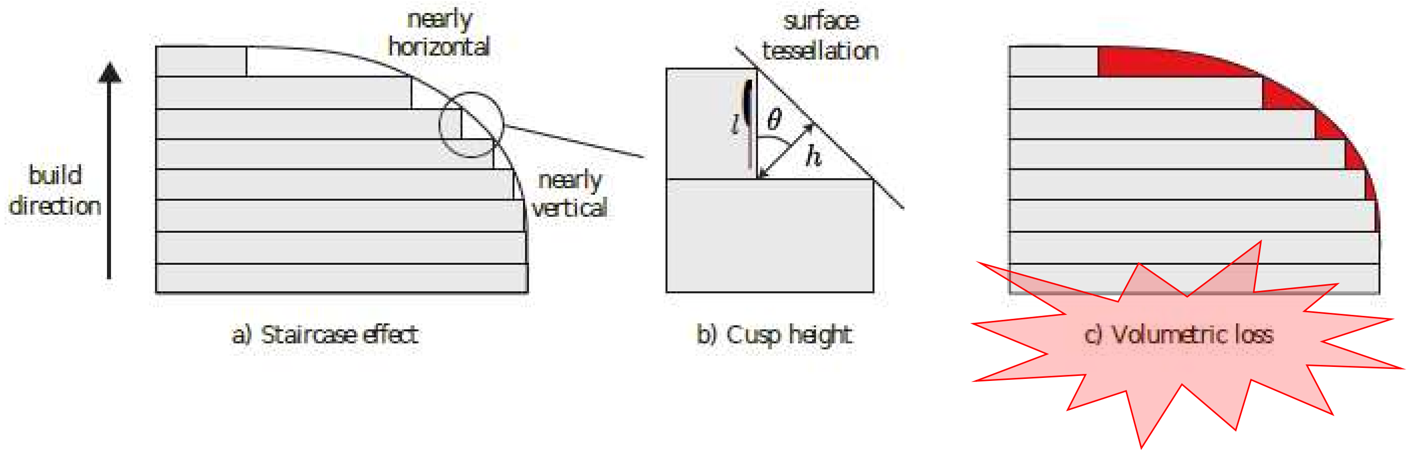
- Depending on both the fabrication tool and the shape, additional geometry may be necessary to support overhanging parts and to keep the part from moving during printing

Slicing

- The model must be converted to a set of planar slices whose distance might be either constant or adaptive

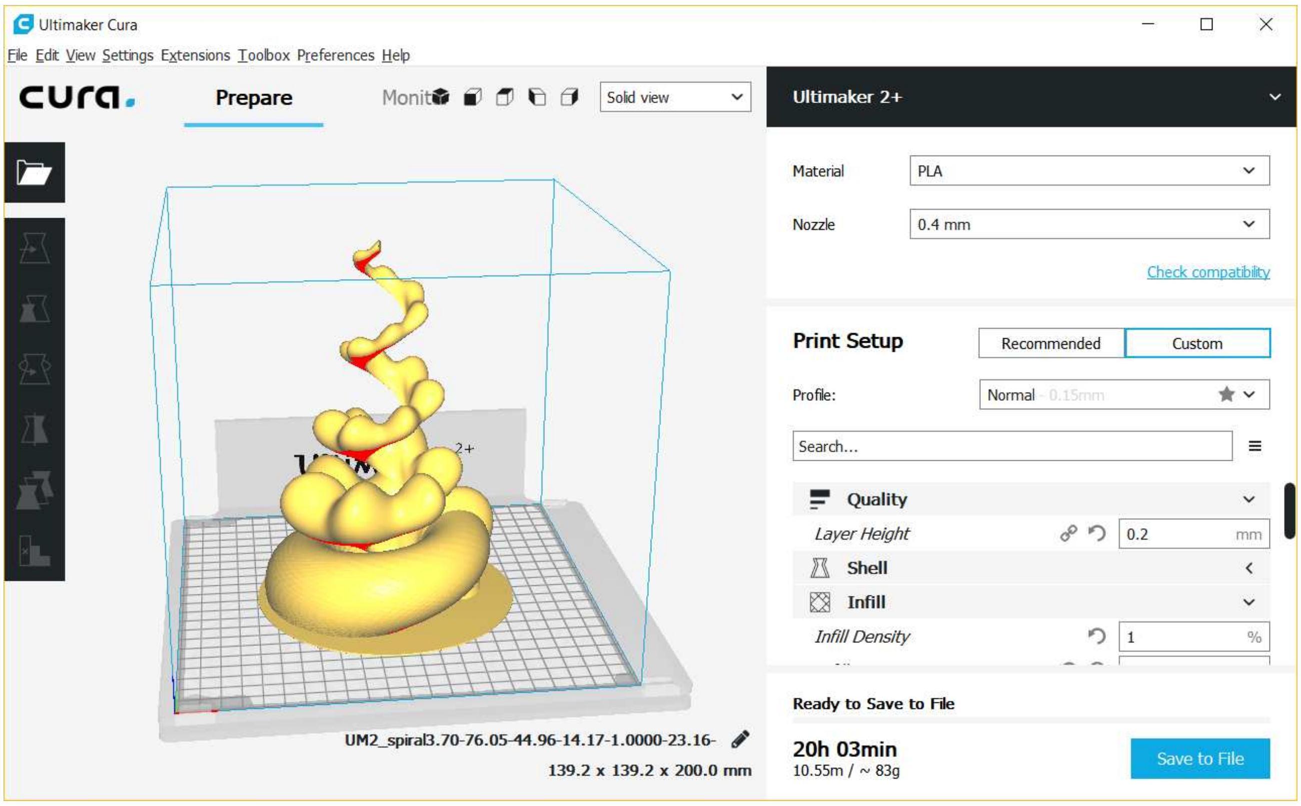
Machine instructions

- Each slice must be converted to either a sequence of movements of the fabrication tool (vector-based) or a grid of pixels that define the solid part of the slice (raster-based)



- Approximating a curved surface by a stack of layers
- Staircase effect reduces fidelity (a).
- Nearly horizontal surfaces introduce more error than nearly vertical ones
- Cusp height (b) and volumetric difference (c) are among the most widely used proxies to estimate fidelity

(from *From 3D Models to 3D Prints: an Overview of the Processing Pipeline*, Livesu et al, EUROGRAPHICS 2017)



Ultimaker 2+

Material: PLA

Nozzle: 0.4 mm

Check compatibility

Print Setup: Recommended | Custom

Profile: Normal - 0.15mm

Search...

Quality

Layer Height: 0.2 mm

Shell

Infill

Infill Density: 1 %

UM2_spiral3.70-76.05-44.96-14.17-1.0000-23.16-

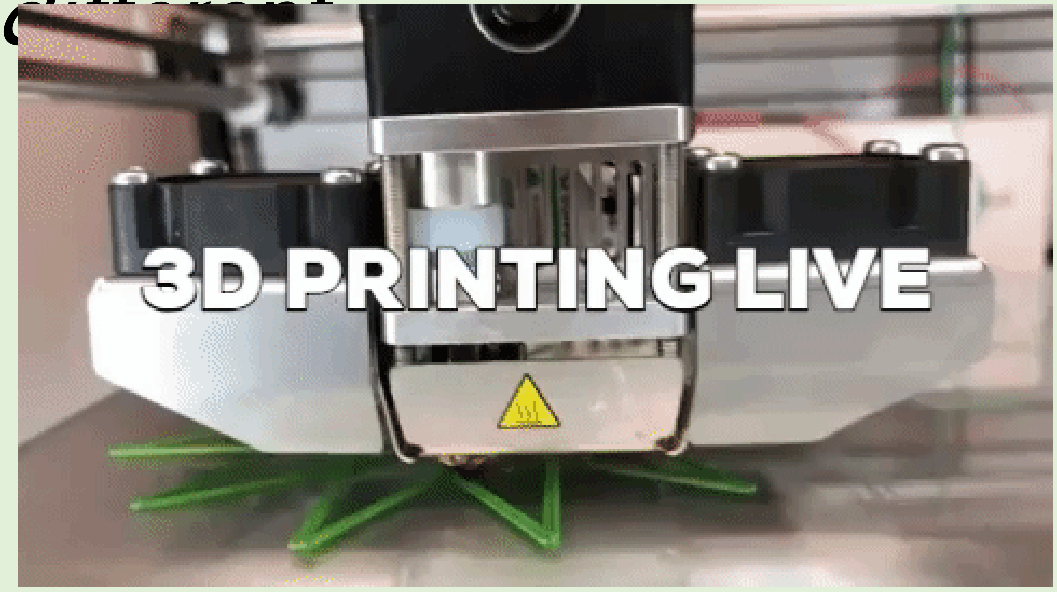
139.2 x 139.2 x 200.0 mm

Ready to Save to File

20h 03min

10.55m / ~ 83g

Task 2: LivePrinter is different



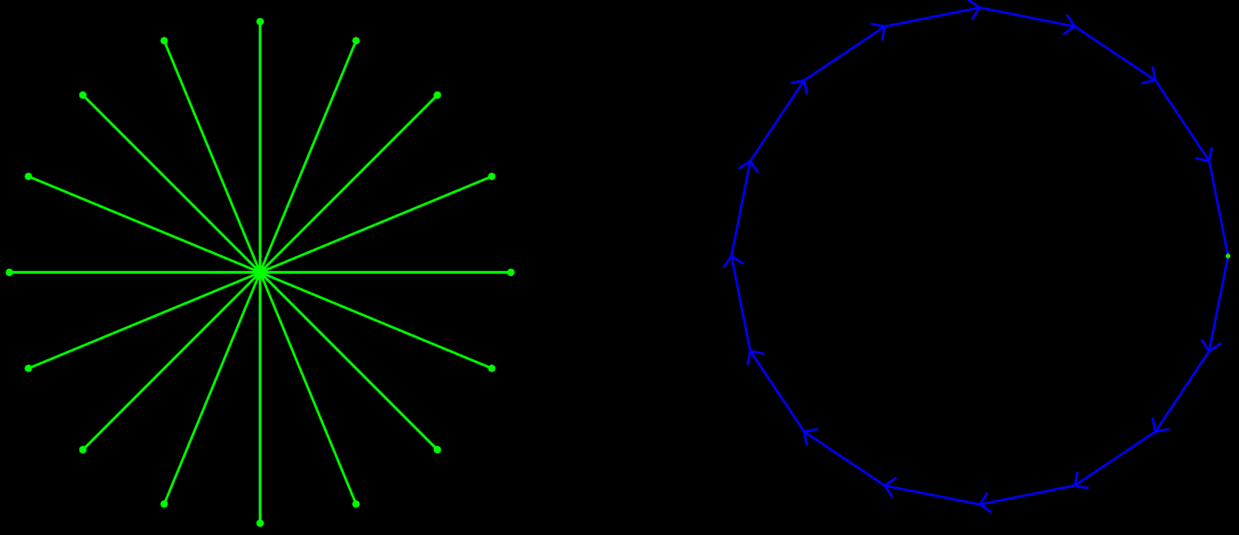
Philosophy of design:

- A 'hypercyclic' design process model
- Shift in 'machine' design from *adaptation of a system to its environment* towards *co-evolution of autonomous systems* – Morgan (1986, p. 245)

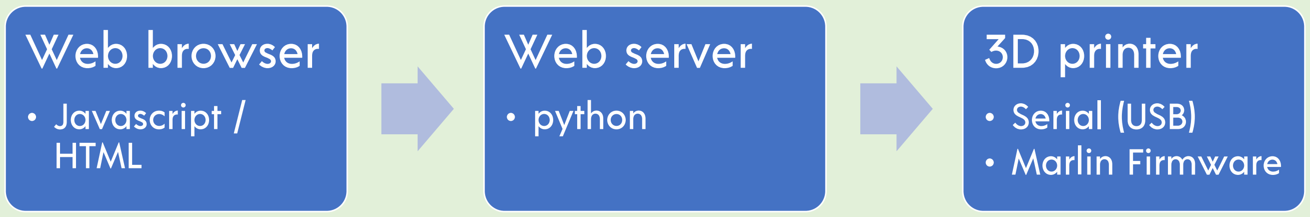
Modes of making: Today vs The Future

Planned: make EXACTLY this thing from initial instructions	Live: BUILD UP an object based on a series of consecutive actions
• Inductive (hypothesis) • Deliberate • Representative • Pre-planned • Imperative • ERROR: throw it out and start again	• Abductive (intuition) • Explorative • Improvisational • Generative • Functional • ERROR: recover/integrate

Two Circles



Task 3: Setting up to run LivePrinter



Setup overview:

- Download code from github: https://github.com/pixelpusher/liveprinter
- Run! Use VS Code, Visual Studio, or any python environment (command line or other)
- Open web browser to http://localhost:8888 and start coding
- Errors will appear in the command window, and in the JavaScript console (and the page itself)

Note!

- LivePrinter only supports Marlin-based printers, for now
- Marlin firmware: https://github.com/Ultimaker/Ultimaker2Marlin

Python setup – bare minimum

- Requires python 3.6+ (tested on 3.6 and 3.7)
- Install libraries
  - pyserial (serial communication to printer)
  - json-rpc (javascript remote procedure
  - tornado (web server)
- In command window, type:
  - pip3 install pyserial json-rpc tornado
  - Or, if you downloaded them: pip3 install <local library file path>

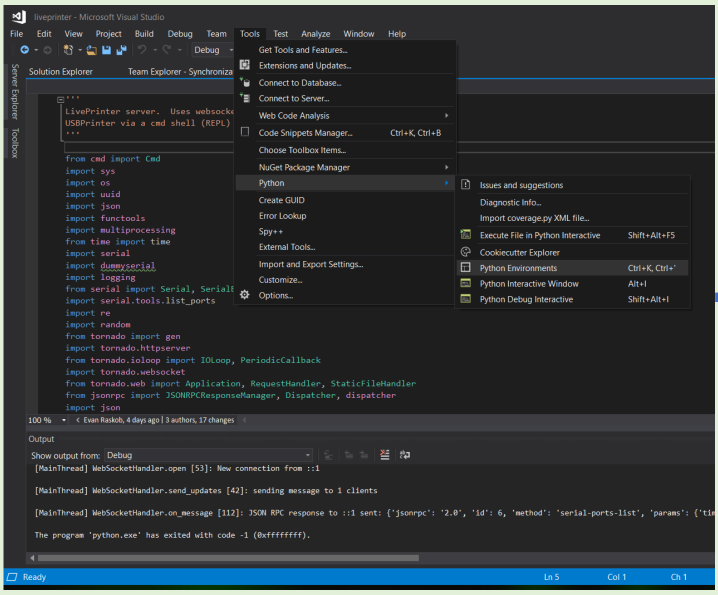
Running the server (bare python)

- liveprinter/LivePrinterServer.py is the main web server
- Open terminal to livePrinter-master download folder (or just liveprinter if you cloned it directly)
- cd liveprinter
- Type python3 LivePrinterServer.py
- Open web browser, type in address: localhost:8888

Visual Studio 2017

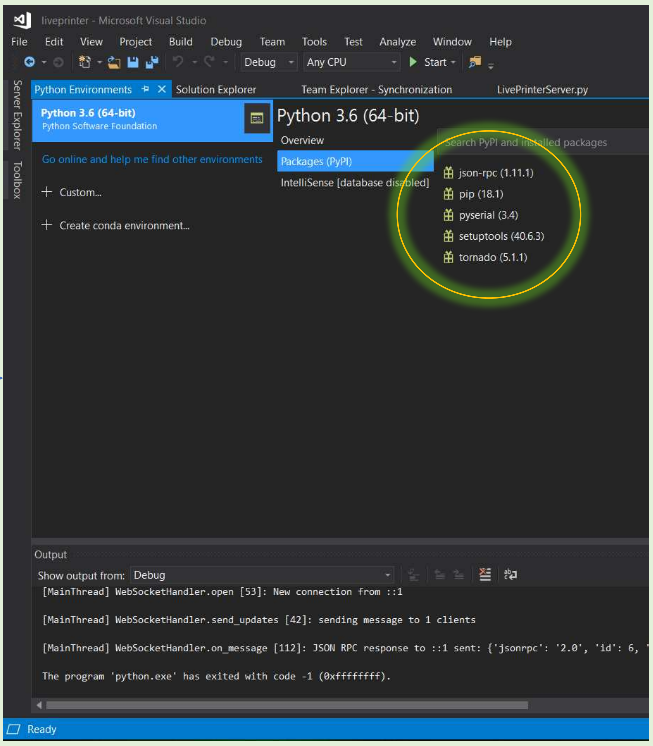
- Only on Windows: open liveprinter.sln in the liveprinter folder you downloaded (or create your own project using the Python template)
- Make sure the python environment matches yours (3.6, but you may have 3.7). → See next slide

Visual Studio 2017:
Install python dependencies
(make sure version matches!).

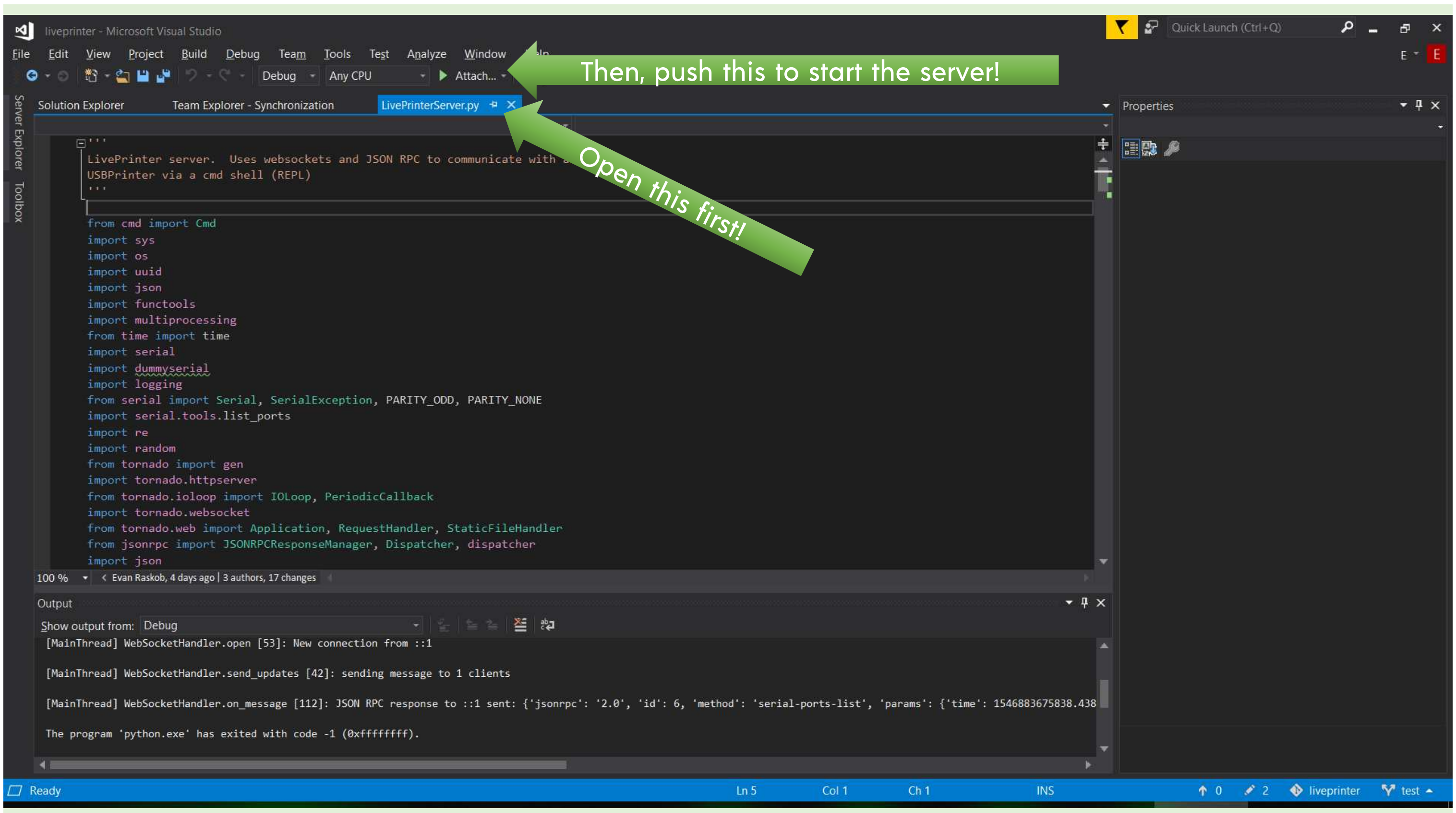

liveprinter - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug
Solution Explorer
Team Explorer - Synchronizat
Get Tools and Features...
Extensions and Updates...
Connect to Database...
Connect to Server...
Web Code Analysis
Code Snippets Manager... Ctrl+K, Ctrl+B
Choose Toolbox Items...
NuGet Package Manager
Python
Create GUID
Error Lookup
Spy++
External Tools...
Import and Export Settings...
Customize...
Options...
Issues and suggestions
Diagnostic Info...
Import coverage.py XML file...
Execute File in Python Interactive Shift+Alt+F5
Cookiecutter Explorer
Python Environments Ctrl+K, Ctrl+`
Python Interactive Window Alt+I
Python Debug Interactive Shift+Alt+I
Server Explorer
Toolbox
100 %
Evan Raskob, 4 days ago | 3 authors, 17 changes
Output
Show output from: Debug
Ready
Ln 5
Col 1
Ch 1

```
'''
LivePrinter server.  Uses websocke
USBPrinter via a cmd shell (REPL)
'''
from cmd import Cmd
import sys
import os
import uuid
import json
import functools
import multiprocessing
from time import time
import serial
import dummyserial
import logging
from serial import Serial, SerialE
import serial.tools.list_ports
import re
import random
from tornado import gen
import tornado.httpserver
from tornado.ioloop import IOLoop, PeriodicCallback
import tornado.websocket
from tornado.web import Application, RequestHandler, StaticFileHandler
from jsonrpc import JSONRPCResponseManager, Dispatcher, dispatcher
import json
```

```
[MainThread] WebSocketHandler.open [53]: New connection from ::1

[MainThread] WebSocketHandler.send_updates [42]: sending message to 1 clients

[MainThread] WebSocketHandler.on_message [112]: JSON RPC response to ::1 sent: {'jsonrpc': '2.0', 'id': 6, 'method': 'serial-ports-list', 'params': {'tim

The program 'python.exe' has exited with code -1 (0xffffffff).
```


liveprinter - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug
Any CPU
Start
Python Environments
Solution Explorer
Team Explorer - Synchronization
LivePrinterServer.py
Server Explorer
Toolbox
Python 3.6 (64-bit)
Python Software Foundation
Go online and help me find other environments
Custom...
Create conda environment...
Python 3.6 (64-bit)
Overview
Packages (PyPI)
IntelliSense [database disabled]
Search PyPI and installed packages
json-rpc (1.11.1)
pip (18.1)
pyserial (3.4)
setuptools (40.6.3)
tornado (5.1.1)
Output
Show output from: Debug
Ready

```
[MainThread] WebSocketHandler.open [53]: New connection from ::1

[MainThread] WebSocketHandler.send_updates [42]: sending message to 1 clients

[MainThread] WebSocketHandler.on_message [112]: JSON RPC response to ::1 sent: {'jsonrpc': '2.0', 'id': 6, '

The program 'python.exe' has exited with code -1 (0xffffffff).
```

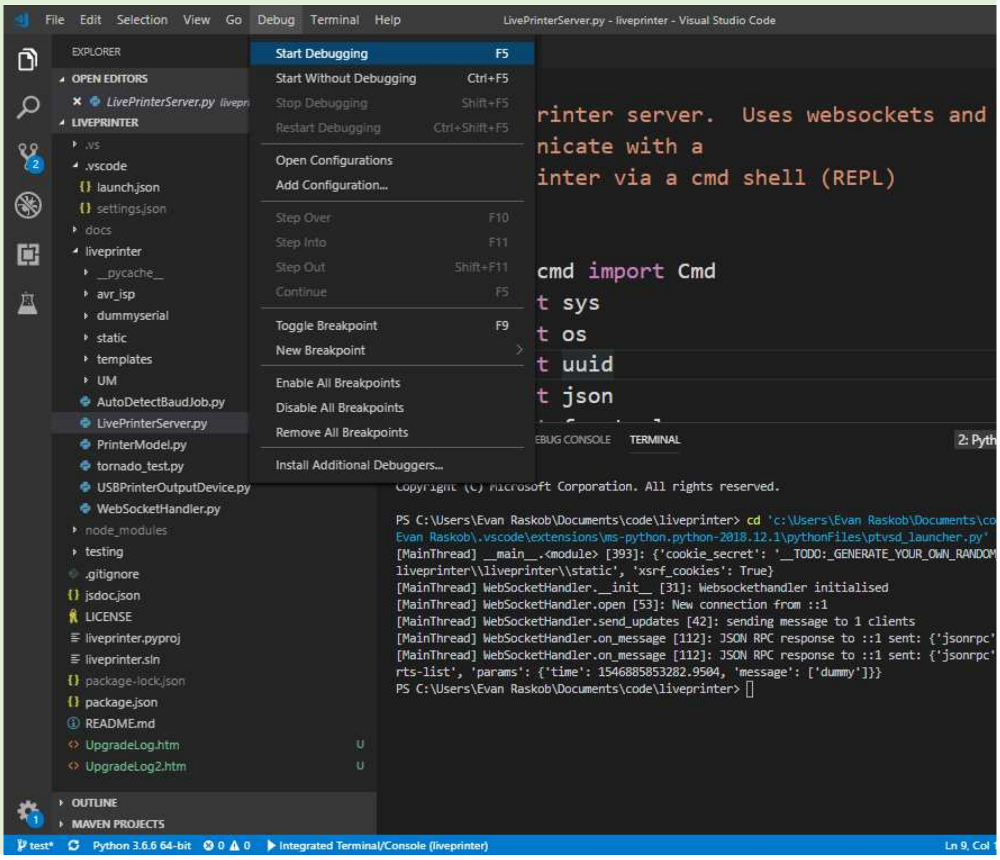

liveprinter - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug
Any CPU
Attach...
Then, push this to start the server!
Solution Explorer
Team Explorer - Synchronization
LivePrinterServer.py
Open this first!
Properties
Server Explorer
Toolbox
100 %
Evan Raskob, 4 days ago | 3 authors, 17 changes
Output
Show output from: Debug
Ready
Ln 5
Col 1
Ch 1
INS
liveprinter
test

```
'''
LivePrinter server.  Uses websockets and JSON RPC to communicate with a
USBPrinter via a cmd shell (REPL)
'''

from cmd import Cmd
import sys
import os
import uuid
import json
import functools
import multiprocessing
from time import time
import serial
import dummyserial
import logging
from serial import Serial, SerialException, PARITY_ODD, PARITY_NONE
import serial.tools.list_ports
import re
import random
from tornado import gen
import tornado.httpserver
from tornado.ioloop import IOLoop, PeriodicCallback
import tornado.websocket
from tornado.web import Application, RequestHandler, StaticFileHandler
from jsonrpc import JSONRPCResponseManager, Dispatcher, dispatcher
import json
```

```
[MainThread] WebSocketHandler.open [53]: New connection from ::1

[MainThread] WebSocketHandler.send_updates [42]: sending message to 1 clients

[MainThread] WebSocketHandler.on_message [112]: JSON RPC response to ::1 sent: {'jsonrpc': '2.0', 'id': 6, 'method': 'serial-ports-list', 'params': {'time': 1546883675838.438

The program 'python.exe' has exited with code -1 (0xffffffff).
```

In VS Code:

1. File → Open Folder → browse to liveprinter folder
2. Click Extensions button on left panel, install python form Microsoft
3. Install linter, look for log to add to your path in environment variables (Windows) or path (OS X or Linux): C:\Users\YOUR NAME\AppData\Roaming\Python\Python36\Scripts
4. Open LivePrinterServer.py and click "debug" from top menu!
5. In Terminal type "pip3 install tornado json-rpc pyserial"



```
rinter server.  Uses websockets and
nicate with a
inter via a cmd shell (REPL)


cmd import Cmd
t sys
t os
t uuid
t json
```

```
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\Evan Raskob\Documents\code\liveprinter> cd 'c:\Users\Evan Raskob\Documents\co
Evan Raskob\.vscode\extensions\ms-python.python-2018.12.1\pythonFiles\ptvsd_launcher.py'
[MainThread] __main__.<module> [393]: {'cookie_secret': '__TODO:_GENERATE_YOUR_OWN_RANDOM
liveprinter\\liveprinter\\static', 'xsrf_cookies': True}
[MainThread] WebSocketHandler.__init__ [31]: Websockethandler initialised
[MainThread] WebSocketHandler.open [53]: New connection from ::1
[MainThread] WebSocketHandler.send_updates [42]: sending message to 1 clients
[MainThread] WebSocketHandler.on_message [112]: JSON RPC response to ::1 sent: {'jsonrpc'
[MainThread] WebSocketHandler.on_message [112]: JSON RPC response to ::1 sent: {'jsonrpc'
rts-list', 'params': {'time': 1546885853282.9504, 'message': ['dummy']}}
PS C:\Users\Evan Raskob\Documents\code\liveprinter> []
```

Finally, open a web browser to http://localhost:8888

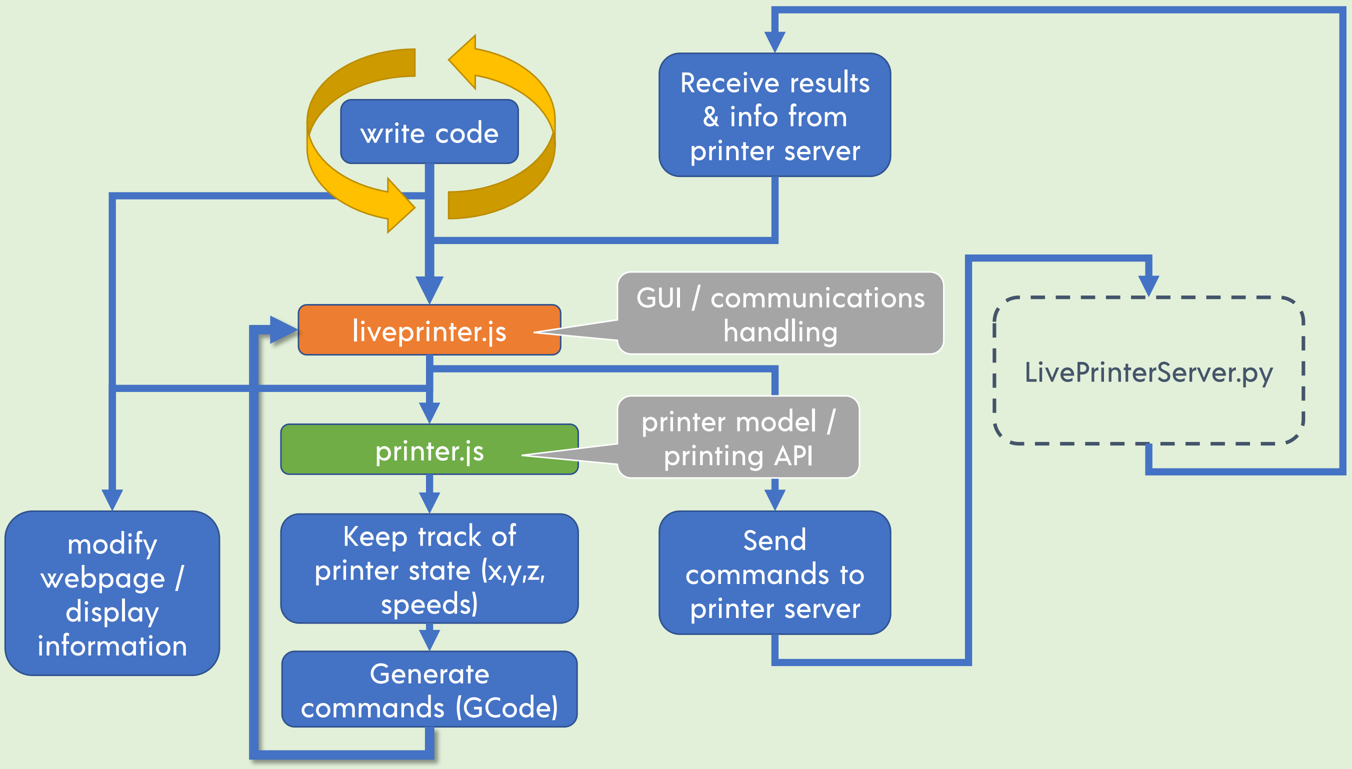
You should see the web client!

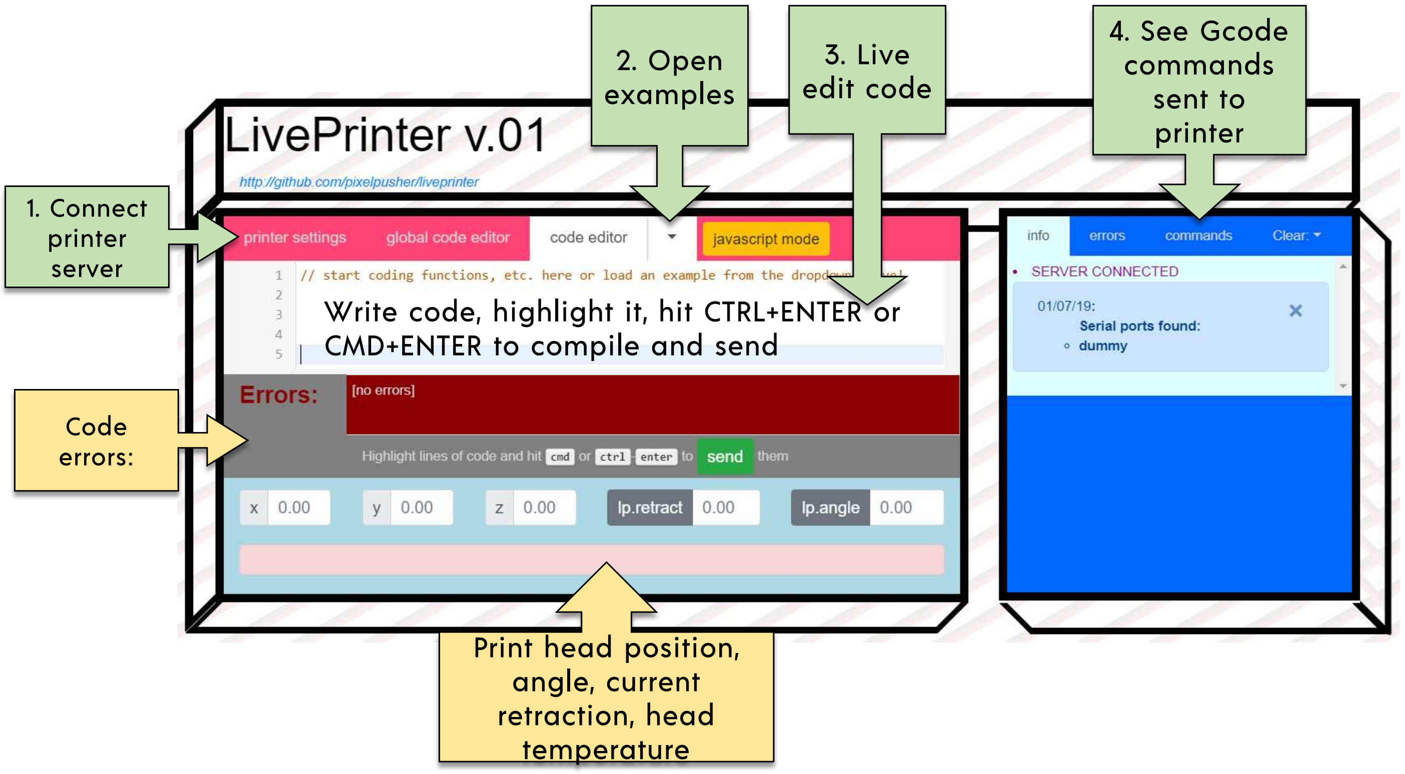
Let's move!

Task 4: Using LivePrinter (basic interface & movement)

Starting up:

- Get familiar with GUI
- Home axes
- Set temperature
- Practice moving


write code
Receive results & info from printer server
liveprinter.js
GUI / communications handling
LivePrinterServer.py
printer.js
printer model / printing API
modify webpage / display information
Keep track of printer state (x,y,z, speeds)
Send commands to printer server
Generate commands (GCode)



```
// start coding functions, etc. here or load an example from the dropdown above!
```

Write code, highlight it, hit CTRL+ENTER or CMD+ENTER to compile and send

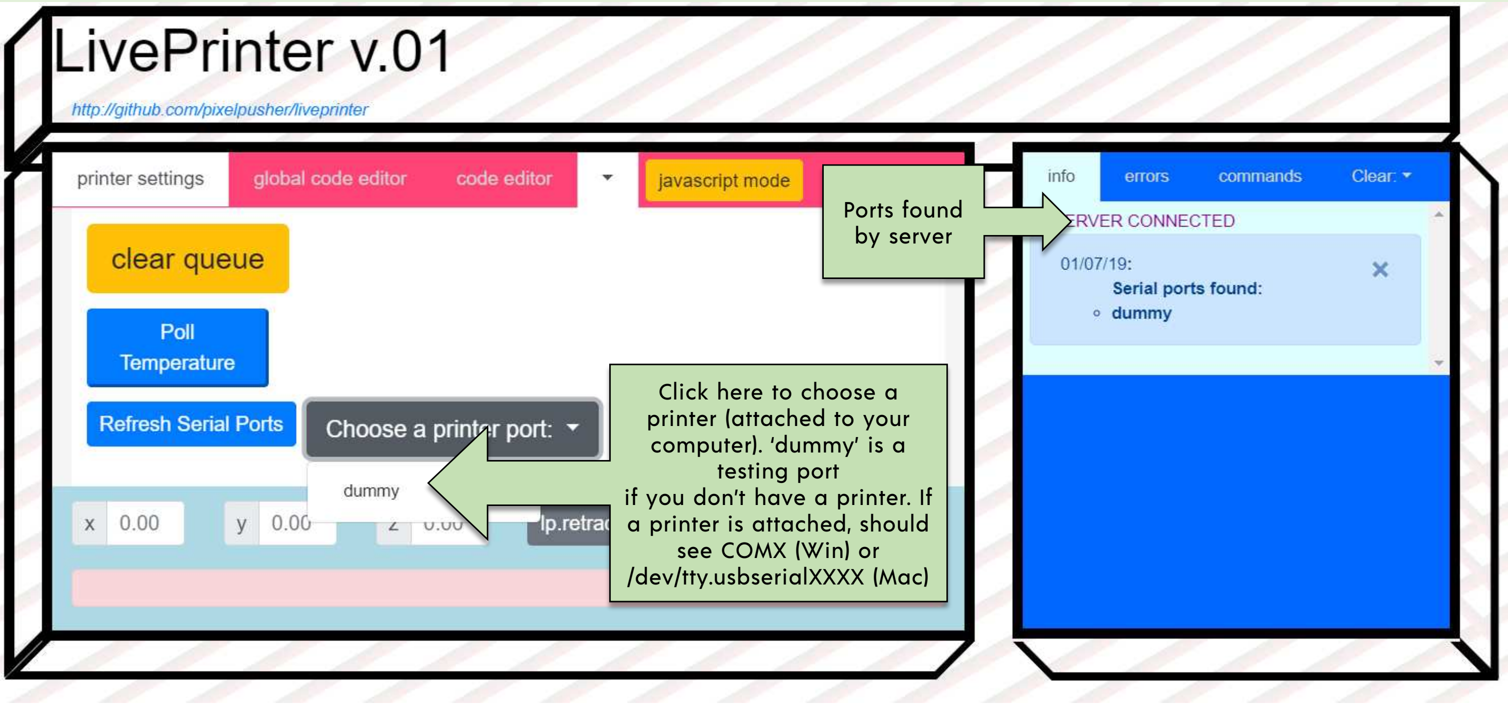
[no errors]

Highlight lines of code and hit cmd or ctrl enter to send them

SERVER CONNECTED

01/07/19:
Serial ports found:
- dummy

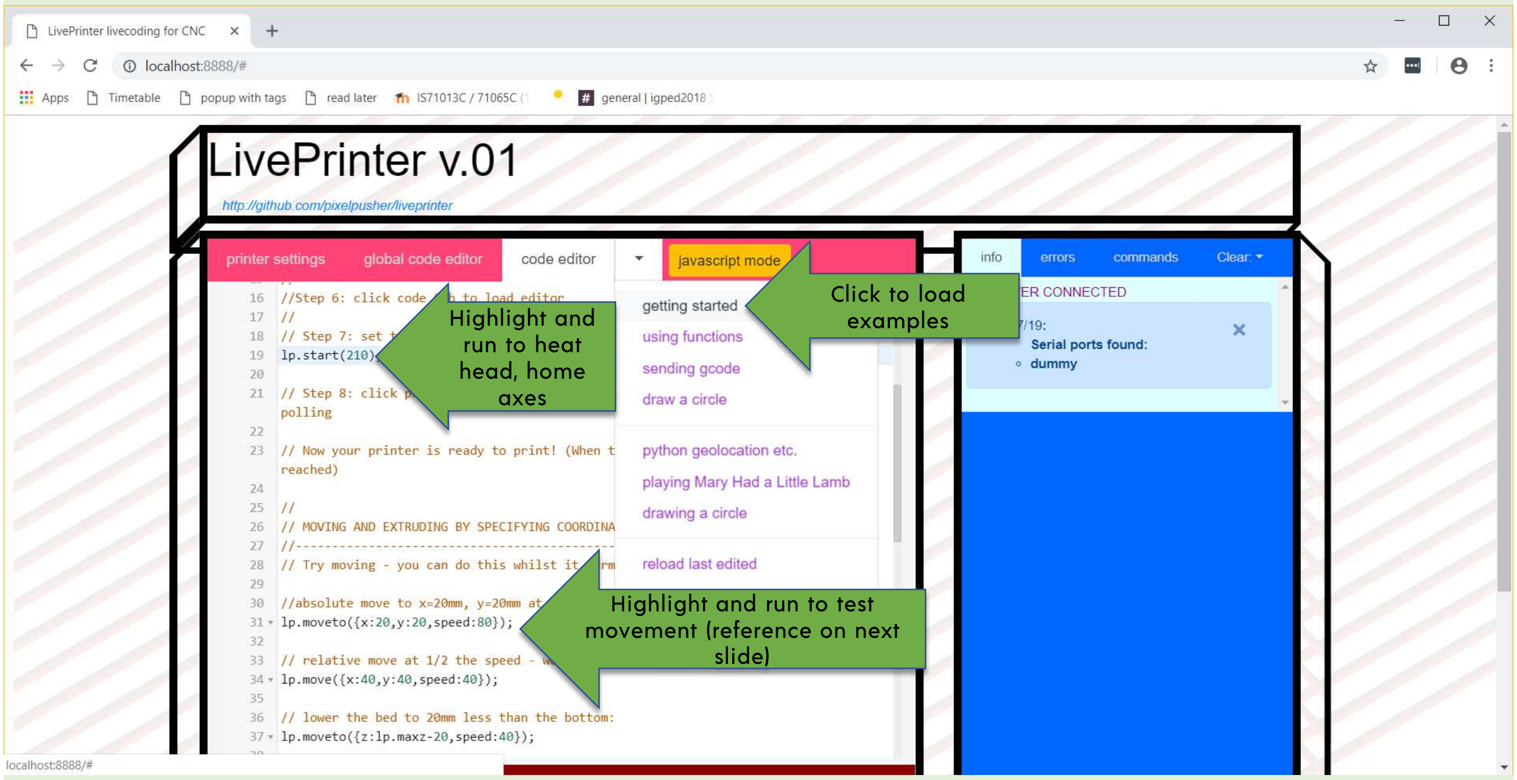
Step 1: Connect to server and printer



Click here to choose a printer (attached to your computer). 'dummy' is a testing port if you don't have a printer. If a printer is attached, should see COMX (Win) or /dev/tty.usbserialXXXX (Mac)

SERVER CONNECTED

01/07/19:
Serial ports found:
- dummy



```
//Step 6: click code   b to load editor
//
// Step 7: set t
lp.start(210)

// Step 8: click p
polling

// Now your printer is ready to print! (When t
reached)

//
// MOVING AND EXTRUDING BY SPECIFYING COORDINA
//--------------------------------------------
// Try moving - you can do this whilst it   rm

//absolute move to x=20mm, y=20mm at
lp.moveto({x:20,y:20,speed:80});

// relative move at 1/2 the speed - w
lp.move({x:40,y:40,speed:40});

// lower the bed to 20mm less than the bottom:
lp.moveto({z:lp.maxz-20,speed:40});
```

Click to load examples

Highlight and run to heat head, home axes

Highlight and run to test movement (reference on next slide)

Movement commands

command	description
lp.move({ x: , y: , z: });	Relative movement: "move *by this amount*"
lp.moveto({ x: , y: , z: });	Absolute movement: "move *to this exact position*"
lp.up(AMOUNT)	Move up immediately by AMOUNT as fast as possible (travel speed)
lp.down(AMOUNT)	Move down immediately by AMOUNT as fast as possible (travel speed)
lp.upto(HEIGHT)	Move bed up to this exact height immediately as fast as possible (travel speed)
lp.downto(HEIGHT)	Move bed down to this exact height immediately as fast as possible (travel speed)

Note: upto() and downto() have exactly the same result, but help you think about it differently

Movement Notation (all in mm)

- x: x position (horizontal L/R) from 0 (left) to 220 (right)
- y: y position (horizontal F/B) from 0 (front) to 220 (back)
- z: z position (vertical) from 0 (floor) to 205 (top) → Ultimaker 2+ is larger
- e: filament position (starts at 0 when axes are reset, can be negative to retract)

Task 5: Using LivePrinter (printing a square)

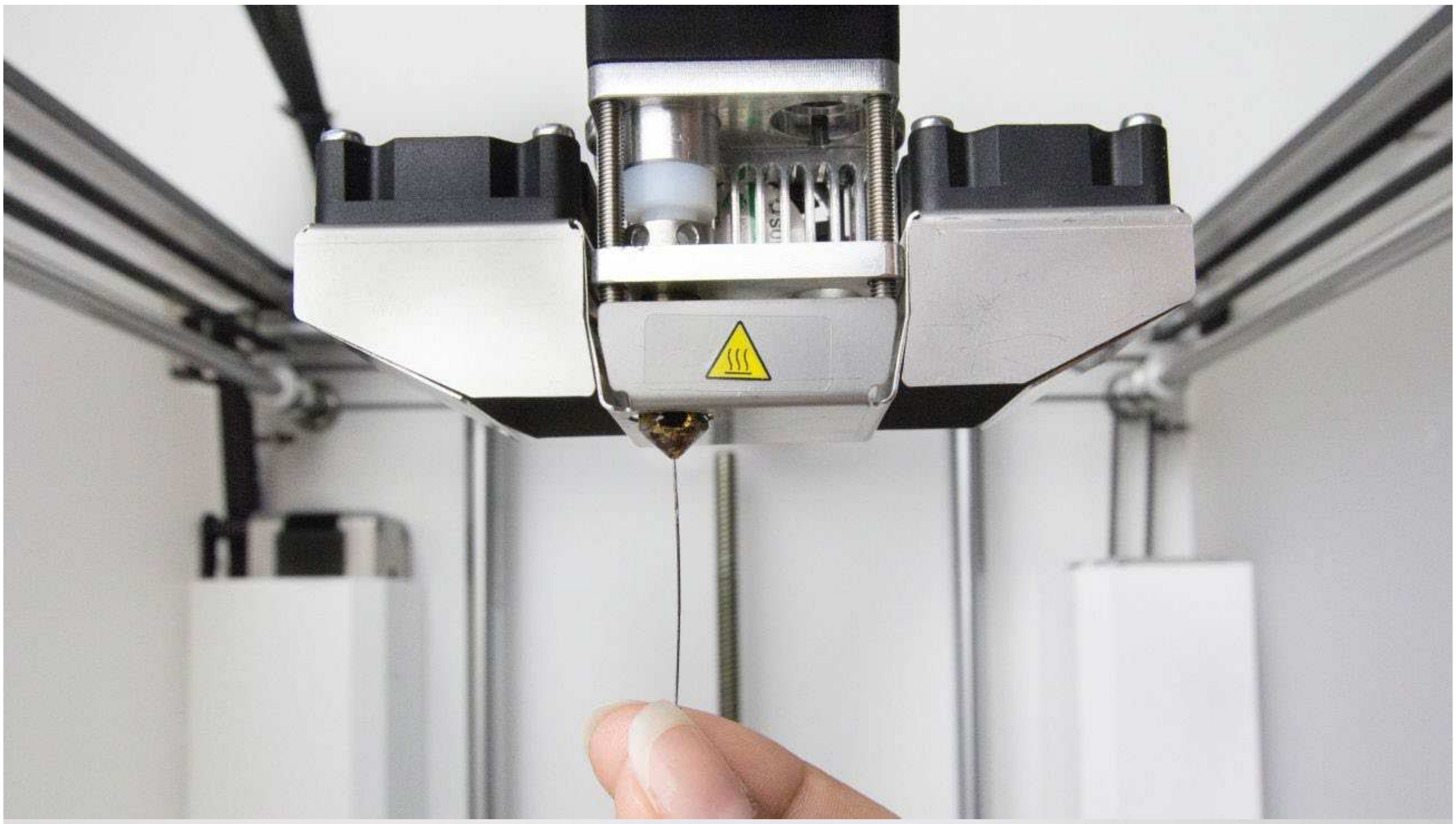
4 Things to know at all times:

Tool temperature

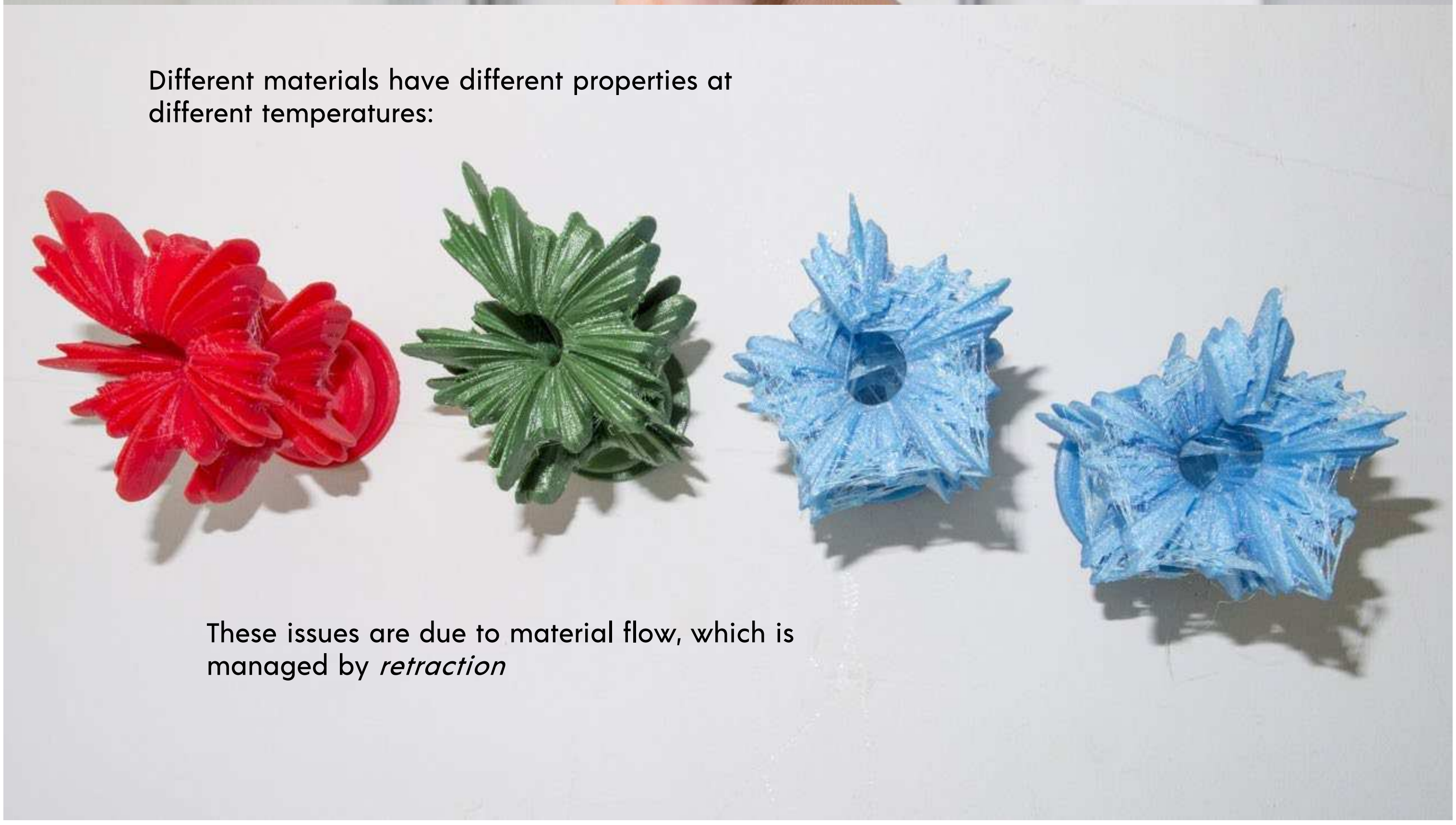
Tool speed

Filament position

Tool position



Different materials have different properties at different temperatures:



These issues are due to material flow, which is managed by *retraction*

Extrusion (printing) commands

command	description
lp.extrude({ x: , y: , z: , speed: , thickness: , retract});	Extrude plastic in a line using relative movement: "extrude while moving *by this amount*"
lp.extrudeto({ x: , y: , z: , speed: , thickness: , retract});	Absolute movement: "extrude while moving from current position *to this exact position*"

Let's draw a square! (using absolute coords)

- Heat up the printer to 200C → lp.temp(200);
- Prime the filament (extrude a bit to start)
  - lp.moveto({x:20, y:20, z:80, speed:80});
  - lp.extrude({e:10,speed:8});
  - (repeat last step until we see filament! Then wipe it off)
- Move the print head to the print surface
  - lp.moveto({x:20, y:20, z:0.2, speed:80});
- Draw lines:
  - lp.extrudeto({x:120, y:20, z:0.2, speed:30});
  - lp.extrudeto({x:120, y:120, z:0.2, speed:30});
  - lp.extrudeto({x:20, y:120, z:0.2, speed:30});
  - lp.extrudeto({x:20, y:20, z:0.2, speed:30});
- Finally, move the head up: lp.up(60);

Using relative movements to draw (like turtle graphics):

function	What it does
lp.dist(80)	Move the print head moves 80mm in the current direction (default is 0 degrees, from left to right)
lp.turnto(90)	Set the direction of movement in degrees (anti-clockwise, to the left)
lp.turn(90)	Queue a turn: turn 90 degrees anti-clockwise (to the left)
lp.speed(15)	Set the print speed, in mm/s. 15-30mm is a good quality speed.
go(0 or 1)	Actually execute the series of moves. If 1, extrude while moving, or if 0 (default) just move
lh(0.25) or thick(0.25)	Set the height of each extruded line (layer height or thickness - same thing)

ex: draw a "V"

```
lp.speed(15).lh(0.25).turnto(-45).dist(40).go(1).turn(90).dist(40).go(1);
```

Full function reference: https://pixelpusher.github.io/liveprinter/docs/Printer.html

Draw a square relatively:

```
lp.moveto({z:0.2}); // move to build height

for (let i=0; i<4; i++) {
        lp.dist(40).go(1); // draw side
        lp.turn(90).go(); //turn for next side
}

lp.up(40); // move up
```

Task 6: Using LivePrinter (retraction & flow)

Draw a square relatively handling material:

```
lp.moveto({z:0.2}); // move to build height
lp.unretract(); // explicitly move material into print head
for (let i=0; i<4; i++) {
        lp.dist(40).turn(90).go(1,false); // draw side w/no retraction
}
lp.retract(); // move material back up head to avoid dripping
lp.up(40); // move up
```

Task 8: Using LivePrinter (height & layering)

Draw a stack of squares:

```
lp.lh(0.25);
lp.moveto({z: lp.layerHeight}); // move to build height

// draw 5 layers of a square
for (let layers=0; layers < 5; layers++) {
	lp.unretract();
	for (let sides=0; sides<4; sides++) {
		lp.dist(40).turn(90).go(1,false); // draw side
	}
	lp.retract();
	lp.up(lp.layerHeight); // move up to next layer and repeat!
}

lp.up(40); // move up
```

Draw a twisty stack of squares:

```
lp.moveto({z: lp.layerHeight}); // move to build height

// draw 5 layers of a square
for (let layers=0; layers < 5; layers++) {
    for (let sides=0; sides<4; sides++) {
        lp.dist(40).go(1); // draw side
        lp.turn(90).go(); //turn for next side
    }
    lp.up(lp.layerHeight); // move up to next layer and repeat!
    lp.turn(1); // rotate slowly for next layer
}


lp.up(40); // move up
```

Task 9: Using LivePrinter (drawing freestyle)

Try drawing your own shapes:

Multiple squares

Stars

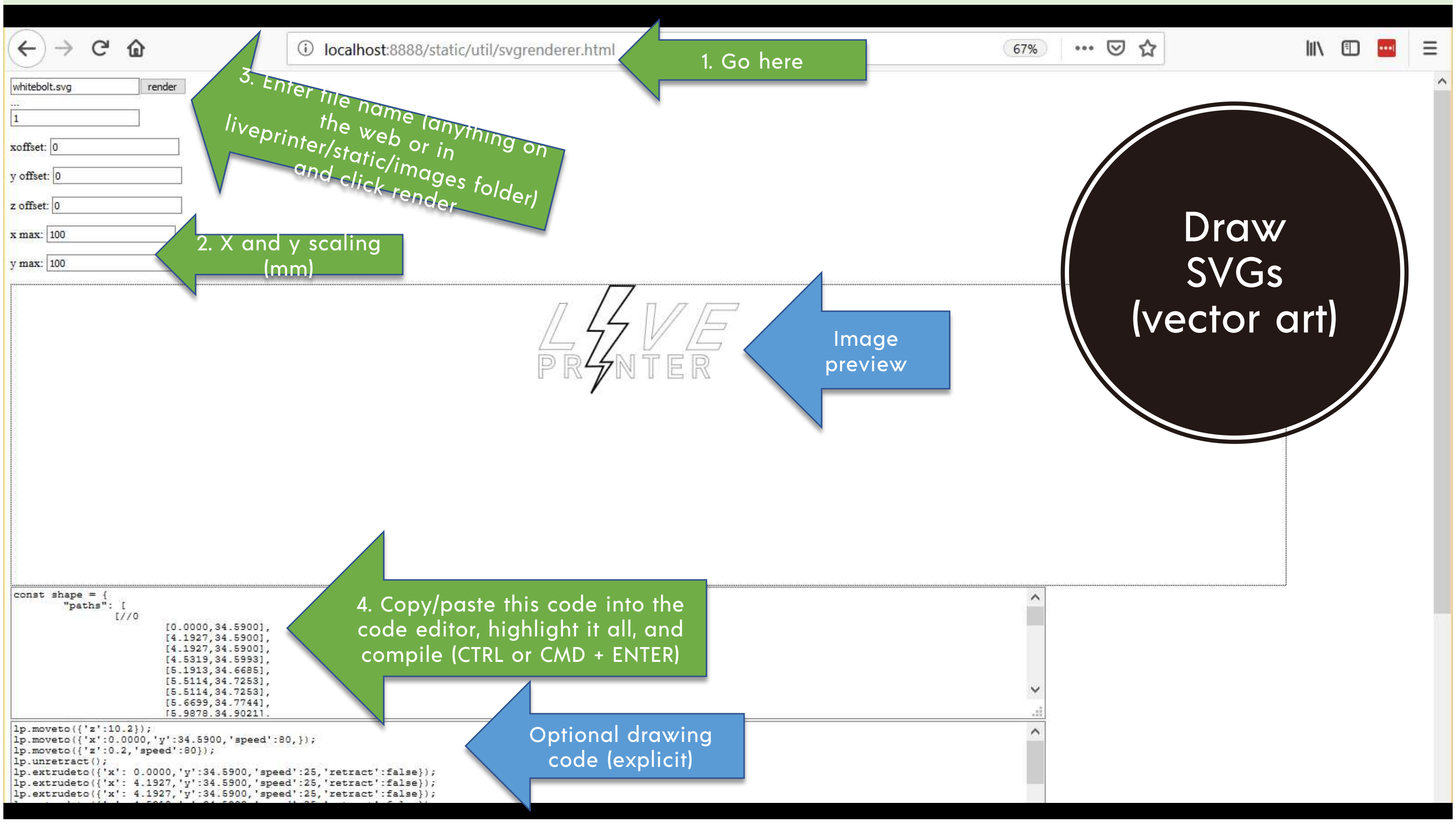
triangles

Draw repeating shapes using lp.run()

```
lp.moveto({z: lp.layerHeight}); // move to build height
// lp.run(`COMMANDS`):
// M(move),E(extrude),L(left turn),R(right turn)
// ex: lp.run(`E20R90`);

    // Koch snowflake with side 10 and angle 60 degrees: F→ELERRELE
    //http://interactivepython.org/runestone/static/CS152f17/Strings/TurtlesandStringsandLSystems.html

    lp.run(`E10L60E10R60R60E10L60E10`);
    lp.up(40); // move up
```

Draw repeating shapes using lp.run()

```
lp.moveto({z: lp.layerHeight}); // move to build height
// lp.run(`COMMANDS`):
// M(move),E(extrude),L(left turn),R(right turn)
// ex: lp.run(`E20R90`);

    let d = 10;
    let angle = 60;
    lp.turnto(0); // horizontal
    // Koch snowflake with side 10 and angle 60 degrees: F→ELERRELE
    //http://interactivepython.org/runestone/static/CS152f17/Strings/TurtlesandStringsandLSystems.html

    lp.run(`E${d}L${angle}E${d}R${angle}R${angle}E${d}L${angle}E${d} `);

    lp.up(40); // move up
```



localhost:8888/static/util/svgrenderer.html
67%
1. Go here
whitebolt.svg
render
3. Enter file name (anything on the web or in liveprinter/static/images folder) and click render
2. X and y scaling (mm)
Image preview
Draw SVGs (vector art)
4. Copy/paste this code into the code editor, highlight it all, and compile (CTRL or CMD + ENTER)
Optional drawing code (explicit)

```
...
1
xoffset: 0
y offset: 0
z offset: 0
x max: 100
y max: 100
```

```
const shape = {
        "paths": [
                [//0
                        [0.0000,34.5900],
                        [4.1927,34.5900],
                        [4.1927,34.5900],
                        [4.5319,34.5993],
                        [5.1913,34.6685],
                        [5.5114,34.7253],
                        [5.5114,34.7253],
                        [5.6699,34.7744],
                        [5.9878,34.9021],
```

```
lp.moveto({'z':10.2});
lp.moveto({'x':0.0000,'y':34.5900,'speed':80,});
lp.moveto({'z':0.2,'speed':80});
lp.unretract();
lp.extrudeto({'x': 0.0000,'y':34.5900,'speed':25,'retract':false});
lp.extrudeto({'x': 4.1927,'y':34.5900,'speed':25,'retract':false});
lp.extrudeto({'x': 4.1927,'y':34.5900,'speed':25,'retract':false});
```

Draw repeating shapes using lp.printPaths:

```
//Using lp.printPaths you can render any paths (a list of lines)

lp.moveto({z: lp.layerHeight}); // move to build height

// only need width OR height if you want it to scale in
proportion
// note: "shape" is whatever you named the shape!
lp.printPaths({paths:shape.paths, xMin:20, yMin:20, width:80});
lp.up(40); // move up
```

Task 9: Future Use

End evaluation (cognitive dimensions) pt 1

- Abstraction:
  - Names of functions - do they map to known concepts and physical reality?
  - Which way of drawing did people prefer? (chained, relative functions vs. explicit ones)
- Secondary notation - extra information in means other than formal syntax
  - Is there any information (about printer, state) that was missing?

End evaluation (cognitive dimensions) pt 2

- Premature commitment - constraints on the order of doing things
  - Does the user feel that the workflow makes logical sense to them? Can they follow it step-by-step successfully or do steps get confused?
- Viscosity - resistance to change
  - What was frustrating? Why?
  - When properties are set, how hard are they to change? Or are they too easy? For example, retraction settings and boundary modes.

End evaluation (cognitive dimensions) pt 3

- Hidden dependencies - test if important links between entities are not visible
  - Is there anything unclear about the system startup and device selection? (probably) and what is it?
  - Are there functions/menus/submenu items that aren't readily visible or easy to find? For example, selecting a serial port.
- Visibility - ability to view components easily
  - Is there any missing information? Anything left unknown?
  - Do people feel like they understand what's going on with the printer? Discuss.

3D printing: "designed things"

"A designed thing, then, is either a living thing or a part of a living thing, or the artifact of a living thing, organized in any case in aid of this battle against disorder (Dennett, 1995, p. 69)."

How might we use this? How might it be useful for you?

- Textile artists who want to print onto fabrics and make new shapes and textures; for artists who want to use a printer like a 3D plotter and draw new forms
- Product and industrial designers who want to understand more about how 3D printing works and fine-tune their materials and tool paths
- Materials scientists who want to study 3D printing materials in more controlled, repeatable ways
- Computational and computer artists, either looking for new tools or making generative works
- Educators who teach fabrication
- HackSpace and MakerSpace staff who need more tools to fine-tune their machines

Reference

Printer Environment

command	description
lp.temp(200)	Set temperature of 1st printing head to 200 C
lp.bed(50)	Set temperature of printer bed to 50C

Printer properties

command	description
lp.x, lp.y, lp.z, lp.e	Current position of head (x,y), height of bed (z), length of filament that has been extruded since last homing command (e)
lp.cx, lp.cy, lp.cz	Centre coordinates in x,y,z
lp.minx, lp.maxx, lp.miny, lp.maxy, lp.minz, lp.maxz	Min/max coordinates (e.g. printer physical dimensions)
lp.angle	Current angle of movement, also set by turn() or turnto()

Exact Movement & Extrusion commands

command	description
lp.move({ x: , y: , z: });	Relative movement: "move *by this amount*"
lp.moveto({ x: , y: , z: });	Absolute movement: "move *to this exact position*"
lp.up(AMOUNT)	Move up immediately by AMOUNT as fast as possible (travel speed)
lp.down(AMOUNT)	Move down immediately by AMOUNT as fast as possible (travel speed)
lp.upto(HEIGHT)	Move bed up to this exact height immediately as fast as possible (travel speed)
lp.downto(HEIGHT)	Move bed down to this exact height immediately as fast as possible (travel speed)

Note: upto() and downto() have exactly the same result, but help you think about it differently

Using relative movements to draw (like turtle graphics):

function	What it does
lp.dist(80)	Move the print head moves 80mm in the current direction (default is 0 degrees, from left to right)
lp.turnto(90)	Set the direction of movement in degrees (anti-clockwise, to the left)
lp.turn(90)	Queue a turn: turn 90 degrees anti-clockwise (to the left)
lp.speed(15)	Set the print speed, in mm/s. 15-30mm is a good quality speed.
go(0 or 1)	Actually execute the series of moves. If 1, extrude while moving, or if 0 (default) just move
lh(0.25) or thick(0.25)	Set the height of each extruded line (layer height or thickness - same thing)

ex: draw a "V"

```
lp.speed(15).lh(0.25).turnto(-45).dist(40).go(1).turn(90).dist(40).go(1);
```

Full function reference: https://pixelpusher.github.io/liveprinter/docs/Printer.html

Other printing functions

command	description
lp.fill(w, h, gap)	Draw a filled rectangle, in the current direction, of width w (x), height h (y), and with a gap between fill lines of gap
lp.rect(w, h)	Draw a rectangle, in the current direction, of width w and height h
lp.wait(time)	Pause the printer for an amount of time (in ms)
lp.run(COMMANDS STRING)	Execute a series of commands in a string: M(move), E(extrude), L(left turn), R(right turn)
lp.fan(speed)	Speed of cooling fan (on the printer head) from 0-100 %
lp.retract(length)	Reverse the filament by length (and set as the default retraction length for future operations)
lp.unretract(length)	Advance the filament by length (and set as the default retraction length for future operations)

For fun

command	description

lp.printPaths({ paths = [[ ]], minY = 0, minX = 0, minZ = 0, width = 0, height = 0, useaspect = true, passes = 1, safeZ = 0 })	Used with SVG rendering. Print a list of paths, at the x,y and with dimensions specified. Will auto-scale if given only a width or height (and useaspect is true, which is by default). With "passes" you can print them multiple times to make higher lines. Paths are like:
	p = [
	   [20,20],
	   [30,30],
	   [50,30]
	];

lp.note(note, duration, axes)	Play MIDI note for a duration (ms) on 1 or more axes specified by "x" or "xy" or "xyz" etc. Example: Play MIDI note 41 for 400ms on the x & y axes: lp.note(41, 400, "xy").go();

Bibliography

Here are the references in alphabetical order.

Aaron, Sam and Alan Blackwell (2013). 'From Sonic Pi to Overtone: Creative Musical Experiences with Domain-Specific and Functional Languages'. In: *Proceedings of the ACM SIGPLAN Workshop on Functional Art, Music, Modeling and Design (FARM) 2013*. FARM '13. Boston, Massachusetts, USA: ACM, pp. 35–46. DOI: 10.1145/2505341.2505346 (cited on pages 14, 21, 52, 53).

Abraham, Katharine G. and Melissa S. Kearney (Feb. 2018). *Explaining the Decline in the U.S. Employment-to-Population Ratio: a Review of the Evidence*. Working Paper 24333. National Bureau of Economic Research. DOI: 10.3386/w24333 (cited on page 13).

Acemoglu, Daron, Andrea Manera, and Pascual Restrepo (Apr. 2020). *Does the US Tax Code Favor Automation?* Working Paper 27052. National Bureau of Economic Research. DOI: 10.3386/w27052 (cited on page 13).

Acemoglu, Daron and Pascual Restrepo (June 6, 2018). 'The race between man and machine: Implications of technology for growth, factor shares, and employment'. In: *American Economic Review* 108.6, pp. 1488–1542 (cited on pages 11, 13).

Agre, Philip E. (1997). 'Lessons learned in trying to reform AI'. In: *Social science, technical systems, and cooperative work: Beyond the Great Divide (1997)* 131 (cited on page 27).

Alexander, Christopher (1966). 'Notes on the Synthesis of Form'. eng. In: *Ekistics* 21.127, pp. 395–396 (cited on page 17).

Amineh, R. and H. Asl (Apr. 30, 2015). 'Review of Constructivism and Social Constructivism'. In: *Journal of Social Sciences, Literature and Languages* 1.1, pp. 9–16. (Visited on 08/08/2021) (cited on page 141).

Amorim, Davide Jose Nogueira, Troy Nachtigall, and Miguel Bruns Alonso (June 1, 2019). 'Exploring Mechanical Meta-Material Structures through Personalised Shoe Sole Design'. In: *Proceedings of the ACM Symposium on Computational Fabrication*. SCF '19. Pittsburgh, Pennsylvania: Association for Computing Machinery. DOI: 10.1145/3328939.3329001 (cited on page 43).

Ashby, W. R. (1956). *An Introduction to Cybernetics*. New York: Wiley (cited on pages 16, 17).

AutoDesk (July 2019). *AutoDesk Machine Intelligence Group*. (Accessed July 10, 2019). URL: https://autodeskresearch.com/groups/machine-intelligence (cited on page 170).

Bak, Georg (Aug. 20, 2019). *Generative Photography - An Interview With Gottfried Jäger*. Ed. by Artnome. URL: https://www.artnome.com/news/2019/8/18/generative-photography-an-interview-with-gottfried-jager (cited on page 6).

Baudisch, Patrick and Stefanie Mueller (2017). *Personal Fabrication*. now (cited on pages 3, 5, 31, 33, 36–39, 61, 71, 107, 121, 131, 149, 180, 181).

– (2016). 'Personal Fabrication: State of the Art and Future Research'. In: *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. CHI EA '16. San Jose, California, USA: ACM, pp. 936–939. DOI: 10.1145/2851581.2856664 (cited on pages 34, 61, 68).

Bellingham, Matt, Simon Holland, and Paul Mulholland (June 2014). 'A cognitive dimensions analysis of interaction design for algorithmic composition software'. In: *Proceedings of Psychology of Programming Interest Group Annual Conference 2014*. Ed. by Benedict du Boulay and Judith Good. University of Sussex, pp. 135–140 (cited on page 23).

Ben-Ari, Mordechai and Tzippora Yeshno (2006). 'Conceptual models of software artifacts'. In: *Interacting with Computers* 18.6. Special Issue: Symbiotic Performance between Humans and Intelligent Systems, pp. 1336–1350. DOI: https://doi.org/10.1016/j.intcom.2006.03.005 (cited on pages 59, 102, 176, 177).

Berio, D., S. Calinon, and F. Fol Leymarie (Oct. 2016). 'Learning dynamic graffiti strokes with a compliant robot'. In: *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*. Daejeon, Korea (cited on page 7).

Berio, Daniel (Oct. 5, 2020). *AutoGraff research notes*. URL: http://www.doc.gold.ac.uk/autograff/ (cited on page 7).

Bernardo, Francisco et al. (2020). 'Designing and Evaluating the Usability of a Machine Learning API for Rapid Prototyping Music Technology'. In: *Frontiers in Artificial Intelligence* 3, p. 13. doi: 10.3389/frai.2020.00013 (cited on pages 23, 24, 141).

Bertoldi, M. et al. (1998). 'Domain decomposition and space filling curves in toolpath planning and generation'. In: *Proceedings of the 1998 Solid Freeform Fabrication Symposium*. The University of Texas at Austin. Austin, Texas, pp. 267–274 (cited on page 44).

Bhavnani, Suresh K., Frederick Reif, and Bonnie E. John (2001). 'Beyond command knowledge: identifying and teaching strategic knowledge for using complex computer applications'. In: *Proceedings of the 19th ACM SIGCHI Conference on Human Factors in Computing Systems*. ACM, pp. 229–236. doi: 10.1145/365024.365107 (cited on page 59).

Blackwell, A. (2018). 'A Craft Practice of Programming Language Research'. In: *Proceedings of the Psychology of Programming Interest Group (PPIG) Conference* (cited on pages 27, 46).

– (2005). 'Cognitive Dimensions of Notations'. eng. In: *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*. IEEE, pp. 3–3 (cited on pages 23–25, 61, 100, 142).

Blackwell, A. F. et al. (2001). 'Cognitive Dimensions of Notations: Design Tools for Cognitive Technology'. In: *Cognitive Technology: Instruments of Mind*. Ed. by Meurig Beynon, Chrystopher L. Nehaniv, and Kerstin Dautenhahn. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 325–341 (cited on pages 21, 23, 48).

Blackwell, Alan (July 2015). 'Patterns of User Experience in Performance Programming'. In: *Proceedings of the First International Conference on Live Coding* (Leeds, United Kingdom). Leeds, United Kingdom: ICSRiM, University of Leeds, pp. 12–22. doi: 10.5281/zenodo.19315 (cited on pages 23, 62).

Blackwell, Alan and Sam Aaron (2015). 'Craft Practices of Live Coding Language Design'. In: *Proceedings of the First International Conference on Live Coding (ICLC) 2015*. Zenodo. Leeds, UK: ICSRiM, University of Leeds, pp. 41–52. doi: 10.5281/zenodo.19318 (cited on pages 20, 26, 27, 51, 178).

Blackwell, Alan and Sally Fincher (2010). 'PUX: patterns of user experience'. eng. In: *interactions* 17.2, pp. 27–31 (cited on pages 39, 66–68).

Blackwell, Alan and Thomas Green (2003). 'Notational Systems—The Cognitive Dimensions of Notations Framework'. In: *HCI Models, Theories, and Frameworks: Toward a Multidisciplinary Science*. Ed. by John M. Carroll. sAN fRANCISCO: Morgan Kaufmann, pp. 103–133. doi: 10.1016/b978-155860808-5/50005-8 (cited on pages 21, 23, 25, 48, 56).

Blackwell, Alan, Alex McLean, et al. (2014). 'Collaboration and learning through live coding (Dagstuhl Seminar 13382)'. In: *Dagstuhl Reports* 3.9. Ed. by Alan Blackwell et al., pp. 130–168. doi: 10.4230/DagRep.3.9.130 (cited on pages 15, 16, 26).

Blackwell, Alan F. and Cecily Morrison (2010). 'A Logical Mind, not a Programming Mind: Psychology of a Professional End-User.' In: *PPIG*, p. 16 (cited on page 178).

Blandford, A. and T. Green (1997). 'OSM: An Ontology-based Approach to Usability Evaluation'. In: *Proceedings of Workshop on Representations*. Ed. by E. O'Neill. Queen Mary and Westfield College, pp. 82–91 (cited on page 141).

Bourell, David et al. (2017). 'Materials for additive manufacturing'. In: *CIRP Annals* 66.2, pp. 659–681. doi: https://doi.org/10.1016/j.cirp.2017.05.009 (cited on page 33).

Brandtzaeg, Petter Bae, Jan Heim, and Amela Karahasanović (2011). 'Understanding the new digital divide – A typology of Internet users in Europe'. In: *International Journal of Human-Computer Studies* 69.3, pp. 123–138. doi: https://doi.org/10.1016/j.ijhcs.2010.11.004 (cited on page 140).

Britton, C et al. (2002). 'An empirical study of user preference and performance with UML diagrams'. eng. In: *Proceedings IEEE 2002 Symposia on Human Centric Computing Languages and Environments*. IEEE, pp. 31–33 (cited on page 23).

Brooks, Rodney (Sept. 2017). *The Seven Deadly Sins of Predicting the Future of AI*. https://rodneybrooks.com/the-seven-deadly-sins-of-predicting-the-future-of-ai/ (cited on page 101).

Bruun, K. (Nov. 18, 2013). *Skal vi danse til koden? Kunsten.nu*. url: http://www.kunsten.nu/artikler/artikel.php?slub+livekodning+performance+kunsthal+aarhus+dave+griffiths+alex+mclean+algorave (cited on page 66).

Bulloch, Angela (1998). *Satellite*. Zurich: Museum für Gegenwartskunst (cited on page 8).

Camurri, Antonio, Ingrid Lagerlöf, and Gualtiero Volpe (Jan. 7, 2003). 'Recognizing Emotion from Dance Movement: Comparison of Spectator Recognition and Automated Techniques'. In: *Int. J. Hum.-Comput. Stud.* 59.1–2, pp. 213–225. DOI: 10.1016/S1071-5819(03)00050-8 (cited on page 144).

Carroll, J. (1990). *The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill*. Cambridge, MA: MIT Press (cited on page 102).

Chen, Hou-Tong et al. (2006). 'Active terahertz metamaterial devices'. In: *Nature* 444.7119, pp. 597–600 (cited on page 43).

Clark, Andy (2016). *Surfing uncertainty: prediction, action, and the embodied mind*. eng. Oxford: Oxford University Press (cited on page 60).

Clarke, S (2010). 'How Usable Are Your APIs?' In: *Making Software: What Really Works, and Why We Believe It*. Ed. by A. Oram and G. Wilson. Sebastopol, CA: O'Reilly Media, pp. 545–565 (cited on pages 24, 141).

Clarke, S. and C. Becker (2003). 'Using the cognitive dimensions framework to evaluate the usability of a class library'. In: *Proceedings of the First Joint Conference of EASE and PPIG*. Keele, pp. 359–366 (cited on page 23).

Collins, Nick et al. (2003). 'Live Coding in Laptop Performance'. In: *Organised Sound* 8.3, pp. 321–330. DOI: 10.1017/S135577180300030X (cited on pages 15, 21, 72).

Collins, R. (June 1, 1994). 'Why the Social Sciences Won't Become High-Consensus, Rapid-Discovery Science'. In: *Sociological Forum* 9 (2), pp. 155–177. DOI: 10.1007/BF01476360 (cited on page 27).

Cooper, Alan et al. (Sept. 1, 2014). *About Face: The Essentials of Interaction Design, 4th Edition*. Wiley John + Sons (cited on pages 58, 140).

Craik, Kenneth James Williams (1952). *The nature of explanation*. Vol. 445. CUP Archive (cited on page 60).

Dagit, J. et al. (Jan. 1, 2006). 'Using cognitive dimensions: Advice from the trenches'. In: *Journal of Visual Languages & Computing* 17.4, pp. 302–327.

DeStefano, Diana and Jo-Anne LeFevre (2004). 'The role of working memory in mental arithmetic'. In: *European Journal of Cognitive Psychology* 16.3, pp. 353–386 (cited on page 60).

Ding, Donghong et al. (2016). 'Advanced Design for Additive Manufacturing: 3D Slicing and 2D Path Planning.' In: *New Trends in 3D Printing*. Ed. by Igor V. Shishkovsky. DOI: 10.5772/63042 (cited on pages 41, 44).

Dohm, Katharina and Justin Hoffmann (Mar. 17, 2008). *Kunstmaschinen Maschinenkunst / Art Machines Machine Art*. Museum Tinguely, Basel, Switzerland (cited on pages 6, 8).

Dunne, Danielle D. and Deborah Dougherty (Oct. 2016). 'Abductive Reasoning: How Innovators Navigate in the Labyrinth of Complex Product Innovation'. In: *Organization Studies* 37.2, pp. 131–159. DOI: 10.1177/0170840615604501 (cited on pages 26, 53).

Edmonds, Ernest (2018). 'Algorithmic Art Machines'. eng. In: *Arts* 7.1, p. 3 (cited on page 12).

Edwards, Jonathan, Jodie Chen, and Alessandro Warth (2016). *Live End-User Programming: A Demo/Manifesto*. Proceedings of the Second Workshop on Live Programming (LIVE) 2016 (cited on page 40).

Fallman, Daniel (2003). 'Design-Oriented Human-Computer Interaction'. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '03. Ft. Lauderdale, Florida, USA: Association for Computing Machinery, pp. 225–232. DOI: 10.1145/642611.642652 (cited on page 26).

Fann, K. T. (1970). *Peirce's Theory of Abduction*. 3Island Press (cited on page 26).

Florijn, Bastiaan, Corentin Coulais, and Martin van Hecke (Oct. 1, 2014). 'Programmable Mechanical Metamaterials'. In: *Physical Review Letters* 113.17. DOI: 10.1103/physrevlett.113.175503 (cited on page 43).

Frankjær, Raune and Peter Dalsgaard (2018). 'Understanding Craft-Based Inquiry in HCI'. In: *Proceedings of the 2018 Designing Interactive Systems Conference*. DIS '18. Hong Kong, China: Association for Computing Machinery, pp. 473–484. DOI: 10.1145/3196709.3196750.

Gamma, Erich et al. (1995). *Elements of reusable object-oriented software*. Vol. 99. Addison-Wesley Reading, Massachusetts (cited on page 170).

Gao, Wei et al. (2015). 'The status, challenges, and future of additive manufacturing in engineering'. In: *Computer-Aided Design* 69, pp. 65–89. DOI: https://doi.org/10.1016/j.cad.2015.04.001 (cited on pages 31, 33, 35).

Gary Hodgson, Alessandro Ranellucci and Jeff Moe (Mar. 23, 2021). *Slic3r Manual: Flow Math*. English. Slic3r. URL: https://manual.slic3r.org/advanced/flow-math (visited on 03/23/2021) (cited on page 122).

Gaver, William (2012). 'What Should We Expect from Research Through Design?' In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '12. Austin, Texas, USA: ACM, pp. 937–946. DOI: 10.1145/2207676.2208538 (cited on pages 26, 27, 51, 169).

Glasersfeld, Ernst von (1995). 'A Constructivist Appoach to Teaching'. In: *Constructivism in Education*. Ed. by L. P. Steffe and J. Gale. Lawrence Erlbaum Associates, pp. 3–16 (cited on page 141).

Goodman, Jonathan (Oct. 2014). 'Dendroids, Replicants, and Sculpture Machines: Roxy Paine'. English. In: *Sculpture* 33.8. Copyright International Sculpture Center Oct 2014; Last updated - 2014-09-16, pp. 24–31 (cited on page 9).

Gorski, Peter Leo and Luigi Lo Iacono (2016). 'Towards the Usability Evaluation of Security APIs.' In: *Proceedings of the Tenth International Symposium onHuman Aspects of Information Security & Assurance (HAISA 2016)*, pp. 252–265 (cited on page 102).

Green, T. R. G. and M. Petre (1996). 'Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework'. eng. In: *Journal of Visual Languages and Computing* 7.2, pp. 131–174 (cited on pages 4, 21, 23, 25, 48).

Grierson, Mick (May 16, 2018). 'The Routledge Research Companion to Electronic Music'. In: *The Routledge Research Companion to Electronic Music*. Ed. by Simon Emmerson. Routledge. Chap. Creative coding for audiovisual art, pp. 12–25. DOI: 10.4324/9781315612911 (cited on page 7).

Griffiths, Dave (2019). *fluxus*. (Accessed July 10, 2019). URL: https://www.pawfal.org/fluxus/ (cited on page 17).

Gropius, Walter (1965). *Die Neue Architektur und das Bauhaus (The New Architecture and the Bauhaus)*. English. Trans. German by P. Morton Shand. First published 1925. Cambridge, MA: MIT Press, p. 75 (cited on page 29).

Gupta, Prashant, Bala Krishnamoorthy, and Gregory Dreifus (2020). 'Continuous toolpath planning in a graphical framework for sparse infill additive manufacturing'. In: *Computer-Aided Design* 127, p. 102880. DOI: https://doi.org/10.1016/j.cad.2020.102880 (cited on page 44).

Hancock, Christopher (2002). 'Toward a Unified Paradigm for Constructing and Understanding Robot Processes'. In: *Proceedings of the IEEE Symposia on Human Centric Computing Languages and Environments (HCC) 2002*. HCC '02. Washington, DC, USA: IEEE Computer Society, pp. 107–109. DOI: 10.1109/HCC.2002.1046362 (cited on page 18).

Hergel, Jean et al. (Nov. 2019). 'Extrusion-Based Ceramics Printing with Strictly-Continuous Deposition'. In: *ACM Trans. Graph.* 38.6. DOI: 10.1145/3355089.3356509.

Hilbert, David (1891). 'Uber die stetige Abbidung einer linie auf einFlaechenstueck'. In: *Math. Ann.* 38, pp. 459–460 (cited on pages 5, 147).

Hippel, Eric von (Nov. 2016). *Free Innovation*. Cambridge, MA: MIT Press (cited on page 32).

Hoskins, S. (2014). *3D Printing for Artists, Designers and Makers: Technology Crossing Art and Industry*. Bloomsbury Academic (cited on pages 31, 33).

Hughes, Gordon (Oct. 2006). 'Power's Script: or, Jenny Holzer's Art after 'Art after Philosophy''. English. In: *Oxford Art Journal* 29 (3). Copyright Oxford University Press(England) Oct 15, 2006; Last updated - 2016-10-01, p. 419 (cited on page 16).

Hundhausen, C., J. Vatrapu, and Wingstrom (Jan. 1, 2003). 'End-User Programming as Translation: An Experimental Framework and Study'. In: *The IEEE Symposium on Human Centric Computing Languages and Environments* (Jan. 1, 2003). IEEE, pp. 47–49 (cited on page 67).

Hundhausen, Christopher and Jonathan Brown (2007). 'An Experimental Study of the Impact of Visual Semantic Feedback on Novice Programming'. In: *Journal of Visual Languages and Computing* 18.6, pp. 537–559. DOI: 10.1016/j.jvlc.2006.09.001 (cited on page 88).

Hunt, Andrew and David Thomas (1999). *The Pragmatic Programmer : From Journeyman to Master*. USA: Addison-Wesley (cited on page 82).

Jacobs, Jennifer et al. (2018). 'Dynamic Brushes: Extending Manual Drawing Practices with Artist-Centric Programming Tools'. In: *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*. CHI EA '18. Montreal QC, Canada: ACM, D316:1–D316:4. DOI: 10.1145/3170427.3186492 (cited on page 67).

Jacobs, P. F. (1994). *StereoLithograpy and other RPM technologies*. New York: ASME Press (cited on page 40).

Jared, Bradley H. et al. (2017). 'Additive manufacturing: Toward holistic design'. In: *Scripta Materialia* 135, pp. 141–147. DOI: https://doi.org/10.1016/j.scriptamat.2017.02.029 (cited on page 11).

Jin, Yuan, Yong He, Guoqiang Fu, et al. (2017). 'A Non-Retraction Path Planning Approach for Extrusion-Based Additive Manufacturing'. In: *Robotics and Computer-Integrated Manufacturing* 48, pp. 132–144. DOI: 10.1016/j.rcim.2017.03.008 (cited on page 41).

Jin, Yuan, Yong He, Jian-Zhong Fu, et al. (2014). 'Optimization of Tool-Path Generation for Material Extrusion-Based Additive Manufacturing Technology'. In: *Additive Manufacturing*, pp. 1–4 (cited on page 41).

Johnson-Laird, P. N. (2013). 'Mental models and cognitive change'. In: *Journal of Cognitive Psychology* 25.2, pp. 131–138. DOI: 10.1080/20445911.2012.759935 (cited on pages 58, 60).

Johnson-Laird, Philip N. (2004). *The history of mental models*. Psychology Press (cited on pages 59, 60).

Johnson-Laird, Philip Nicholas (1983). *Mental models: Towards a cognitive science of language, inference, and consciousness*. 6. Harvard University Press (cited on pages 58–60).

Jonas, Wolfgang (2015). 'A cybernetic model of design research'. In: *The Routledge Companion to Design Research*. Ed. by Paul A. Rogers and Joyce Yee. Oxon, UK: Routledge. Chap. 2, pp. 22–36 (cited on page 51).

– (2007). 'Research through DESIGN through research: A cybernetic model of designing design foundations'. In: *Kybernetes* 36.9/10, pp. 1362–1380. DOI: 10.1108/03684920710827355 (cited on page 51).

Kahneman, Daniel (2011). *Thinking, fast and slow*. Macmillan (cited on page 60).

Keep, Jonathan (Dec. 2, 2020). *Studio Journal*. URL: http://www.keep-art.co.uk/journal_1.html (cited on pages 33, 44).

– (Nov. 2014). 'The Fourth Way'. In: *Ceramic Review* (270), pp. 32–37 (cited on pages 33, 44).

Kent, Sarah (July 2007). 'Sol LeWitt'. In: *Modern Painters*. based on an interview on 26 April, p. 69 (cited on page 16).

Khemlani, Sangeet and P. N. Johnson-Laird (2013). 'Cognitive changes from explanations'. In: *Journal of Cognitive Psychology* 25.2, pp. 139–146. DOI: 10.1080/20445911.2012.720968 (cited on pages 59, 60).

Kieras, David E. and Susan Bovair (Jan. 1, 1984). 'The role of a mental model in learning to operate a device'. In: *Cognitive Science* 8, pp. 255–273 (cited on page 59).

Kilpinen, Erkki (2009). 'The Habitual Conception of Action and Social Theory'. In: *Semiotica* 273, pp. 99–128 (cited on page 101).

Kirschner, Paul A., John Sweller, and Richard E. Clark (June 2006). 'Why Minimal Guidance During Instruction Does Not Work: An Analysis of the Failure of Constructivist, Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching'. In: *Educational Psychologist* 41.2, pp. 75–86. DOI: 10.1207/s15326985ep4102_1 (cited on page 59).

Koskinen, Ilpo et al. (2012). *Design Research Through Practice: From the Lab, Field, and Showroom*. First. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. (cited on pages 50, 63).

Kranz, J., D. Herzog, and C. Emmelmann (Feb. 2015). 'Design guidelines for laser additive manufacturing of lightweight structures in TiAl6V4'. In: *Journal of Laser Applications* 27, S14001. DOI: 10.2351/1.4885235 (cited on page 180).

Lawson, B. and S. Loke (1997). 'Computers, words and pictures'. In: *Design Studies* 18, pp. 171–183 (cited on pages 170, 172).

Leitão António, Cabecinhas Filipe and Susana Martins (2010). 'Revisiting the Architecture Curriculum. The programming perspective'. In: *ECAADe 2010 Conference. Future Cities*. Proceedings of the 28th Conference on Education in Computer Aided Architectural Design in Europe (Verlag der Fachvereine Hochschulverlag AG an der ETH Zurich, Zurich, 2010). Switzerland, pp. 81–88 (cited on page 17).

Lewitt, Sol (1967). 'Paragraphs on Conceptual Art'. In: *Artforum* 5.10, pp. 79–83 (cited on pages 6, 16).

Li, Jingyi et al. (2017). 'Direct and Immediate Drawing with CNC Machines'. In: *Proceedings of the 1st Annual ACM Symposium on Computational Fabrication*. SCF '17. Cambridge, Massachusetts: ACM, 11:1–11:2. DOI: 10.1145/3083157.3096344 (cited on page 39).

Linvega, Devine Lu (June 2019). *Orca TUTORIAL*. (Retrieved July 28, 2019). URL: https://github.com/hundredrabbits/Orca/blob/master/TUTORIAL.md (cited on page 28).

Livesu, Marco et al. (2017). 'From 3D models to 3D prints: an overview of the processing pipeline'. In: *Computer Graphics Forum* 36.2, pp. 537–564. DOI: 10.1111/cgf.13147 (cited on pages 33–36, 40, 41, 162).

Lutters, Eric et al. (2014). 'Tools and techniques for product design'. In: *CIRP Annals* 63.2, pp. 607–630. DOI: https://doi.org/10.1016/j.cirp.2014.05.010 (cited on pages 34, 43).

Magnusson, T. and E. Hurtado Mendieta (2008). 'The Phenomenology of Musical Instruments: A Survey.' In: *eContact* 10 (4): *Improv*. (Visited on 09/09/2020) (cited on page 166).

Magnusson, Thor (2010). 'Designing constraints: Composing and performing with digital musical systems'. In: *Computer Music Journal* 34.4, pp. 62–73 (cited on page 171).

– (2011). 'The IXI Lang: A SuperCollider Parasite for Live Coding'. In: *Proceedings of the Computer Music Conference (ICMC) 2011*. Michigan Publishing (cited on pages 18, 52).

Mathews, M. V. (Nov. 1, 1963). 'The Digital Computer as a Musical Instrument'. In: *Science* 142 (3592), pp. 553–557 (cited on page 14).

Mathews, M. V. and F. R. Moore (Dec. 1, 1970). 'GROOVE — a Program to Compose, Store, and Edit Functions of Time'. In: *Commun. ACM* 13.12, pp. 715–721. DOI: 10.1145/362814.362817 (cited on page 14).

Matter, 3d (2015). *what-is-the-influence-of-color-printing-speed-extrusion-temperature-and-ageing-on-my-3d-prints*. URL: https://my3dmatter.com/what-is-the-influence-of-color-printing-speed-extrusion-temperature-and-ageing-on-my-3d-prints/ (cited on page 166).

Matthews, Ben and Stephan Wensveen (Jan. 2015). 'Prototypes and prototyping in design research'. In: *The Routledge Companion to Design Research*. Ed. by Paul A. Rogers and Joyce Yee. Oxon, UK: Routledge, pp. 262–276 (cited on page 50).

Mayer, R. E. (1987). 'Cognitive aspects of learning and using a programming language'. In: *Interfacing Thought*. Ed. by J. M. Carroll. MIT Press, Cambridge, MA, pp. 61–79 (cited on page 25).

Mayring, Philipp (2014). *Qualitative content analysis: theoretical foundation, basic procedures and software solution*. URL: https://www.ssoar.info/ssoar/bitstream/handle/document/39517/ssoar-2014-mayring-Qualitative_content_analysis_theoretical_foundation.pdf (visited on 05/06/2021) (cited on pages 95, 99).

McCarthy, John (1981). 'History of Programming Languages'. In: ed. by Richard L. Wexelblat. New York, NY, USA: ACM. Chap. History of LISP, pp. 173–185. DOI: 10.1145/800025.1198360 (cited on page 14).

McCartney, James (2002). 'Rethinking the Computer Music Language: SuperCollider'. In: *Computer Music Journal* 26.4, pp. 61–68 (cited on page 27).

McLean, Alex (Jan. 2019). *TOPLAP 15TH ANNIVERSARY STREAM - 14–17TH FEBRUARY 2019*. URL: https://toplap.org/toplap-15th-anniversary-stream-14-17th-february-2019/ (cited on page 28).

McLean, Alex and Roger T. Dean (Feb. 2018). 'Algorithmic Trajectories'. In: *The Oxford Handbook of Algorithmic Music*. Ed. by Roger T. Dean and Alex McLean. Oxford University Press. DOI: 10.1093/oxfordhb/9780190226992.013.37 (cited on pages 15, 53).

McLean, Alex and Geraint A. Wiggins (2010). 'Bricolage Programming in the Creative Arts.' In: *PPIG*, p. 18 (cited on pages 176, 177).

Meinhardt, Hans (2009). *The Algorithmic Beauty of Sea Shells*. 4th ed. Springer Publishing Company, Incorporated (cited on page 17).

Meyer, L. B. (Feb. 1961). *Emotion and Meaning in Music*. Phoenix books. University of Chicago Press (cited on page 18).

Milkert, Heidi (Jan. 1, 2014). *3D Printers Play Music from MarioBros., Star Wars' Imperial March & More*. 3D Print. URL: https://3dprint.com/29244/3d-printer-music-songs/ (visited on 09/03/2019) (cited on page 45).

Molnar, Vera (1975). 'Toward Aesthetic Guidelines for Paintings with the Aid of a Computer'. In: *Leonardo* 8.3, pp. 185–189. DOI: https://doi.org/10.2307/1573236 (cited on page 18).

Mozilla (2019). *Iterators and Generators* (cited on page 159).

Mueller, Stefanie (Aug. 2017). '3D Printing for Human-computer Interaction'. In: *interactions* 24.5, pp. 76–79. DOI: 10.1145/3125399 (cited on pages 39, 40).

Mueller, Stefanie, Sangha Im, et al. (2014). 'WirePrint: 3D Printed Previews for Fast Prototyping'. In: *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*. UIST '14. Honolulu, Hawaii, USA: ACM, pp. 273–280. DOI: 10.1145/2642918.2647359 (cited on pages 53, 61, 162).

Mueller, Stefanie, Pedro Lopes, and Patrick Baudisch (2012). 'Interactive Construction: Interactive Fabrication of Functional Mechanical Devices'. In: *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*. UIST '12. Cambridge, Massachusetts, USA: Association for Computing Machinery, pp. 599–606. DOI: 10.1145/2380116.2380191 (cited on page 39).

Norman, D. A. (1983). 'Some observations on mental models'. In: *Mental Models*. Ed. by Dedre Gentner and A. L. Stevens. Hillsdale: Lawrence Erlbaum Associates, pp. 7–14. NJ (cited on pages 58, 59, 101).

Ogborn, David (2012). 'Composing for a Networked, Pulse-Based, Laptop Orchestra*'. In: *Organised Sound* 17.1, pp. 56–61. DOI: 10.1017/S1355771811000513 (cited on page 17).

Papacharalampopoulos, Alexios, Harry Bikas, and Panagiotis Stavropoulos (2018). 'Path Planning for the Infill of 3D Printed Parts Utilizing Hilbert Curves'. In: *Procedia Manufacturing* 21, pp. 757–764. DOI: 10.1016/j.promfg.2018.02.181 (cited on pages 41, 44, 158).

Papert, Seymour (1980). *Mindstorms: children, computers, and powerful ideas.* Basic Books. NY (cited on pages 52, 116).

Pattinson, Sebastian et al. (2019). 'Additive manufacturing of biomechanically tailored meshes for compliant wearable and implantable devices'. In: *Advanced Functional Materials* 29.32, p. 1901815 (cited on page 44).

Peano, G. (1890). 'Sur une courbe qui remplit toute une aire plane'. In: *Math. Ann* 36, pp. 157–160 (cited on page 146).

Peirce, Charles S. (1934). *The Collected Papers of Charles Sanders Peirce, Vol. V: Pragmatism and Pragmaticism*. English. Ed. by Charles Hartshorne and Paul Weiss. Cambridge: Harvard University Press (cited on page 26).

Peng, Huaishu et al. (2018). 'RoMA: Interactive Fabrication with Augmented Reality and a Robotic 3D Printer'. In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. CHI '18. Montreal QC, Canada: ACM, 579:1–579:12. DOI: 10.1145/3173574.3174153 (cited on pages 39, 67, 162, 176).

Pennington, N. (1987). 'Stimulus structures and mental representations in expert comprehension of computer programs'. In: *Cognitive Psychology* 19, pp. 295–341 (cited on page 25).

Perera, Roland (2013). 'Interactive functional programming'. PhD thesis. University of Birmingham (cited on page 20).

Pickering, A. (1995). *The Mangle of Practice*. Chicago: Chicago University Press (cited on pages 17, 179).

Pickering, Andrew (2011). *The Cybernetic Brain: Sketches of Another Future*. Chicago, IL, USA: University of Chicago Press (cited on pages 17, 156).

– (2012). 'The Robustness of Science and the Dance of Agency'. In: *Characterizing the Robustness of Science: After the Practice Turn in Philosophy of Science*. Ed. by Léna Soler et al. Dordrecht: Springer Netherlands, pp. 317–327. DOI: 10.1007/978-94-007-2759-5_13 (cited on page 17).

Podshivalov, Lev et al. (2013). 'Design, Analysis and Additive Manufacturing of Porous Structures for Biocompatible Micro-Scale Scaffolds'. In: *Procedia CIRP* 5. First CIRP Conference on BioManufacturing, pp. 247–252. DOI: https://doi.org/10.1016/j.procir.2013.01.049 (cited on page 43).

Prusinkiewicz, Przemyslaw and Aristid Lindenmayer (2012). *The algorithmic beauty of plants*. Springer Science & Business Media (cited on page 53).

Rael, Ronald et al. (2016). *GCODE.Clay*. URL: http://www.emergingobjects.com/project/gcode-clay/ (cited on page 9).

Raskob, Evan (Feb. 7, 2020). 'Filling In: Livecoding musical, physical 3D printing tool paths using space filling curves'. In: *ICLC*. ICLC 2020. Limerick, Ireland. DOI: 10.5281/zenodo.3939527 (cited on pages 147, 156).

Rein, Patrick et al. (2019). 'Exploratory and Live, Programming and Coding: A Literature Study Comparing Perspectives on Liveness'. In: *The Art, Science, and Engineering of Programming* 3.1. DOI: https://doi.g/10.22152/programming-journal.org/2019/3/1 (cited on pages 18–20).

Rittel, Horst W. J. and Melvin M. Webber (June 1973). 'Dilemmas in a general theory of planning'. In: *Policy Sciences* 4.2, pp. 155–169. DOI: doi:10.1007/BF01405730 (cited on pages 21, 22, 50, 55).

Roberts, Charles and Graham Wakefield (Feb. 1, 2018). 'Tensions and techniques in livecoding performance'. In: *The Oxford Handbook of Algorithmic Music*. Ed. by Alex McLlean Roger T. Dean. Oxford University Press. DOI: 10.1093/oxfordhb/9780190226992.001.0001 (cited on pages 14, 64, 66, 67).

Rodgers, Paul (Dec. 1, 2020). *AHRC Design Leadership Fellowship Final Report*. Lancaster: Lancaster University (cited on pages 49, 156).

Rose, J. (John) (1974). *The cybernetic revolution*. eng. London: Elek (cited on page 16).

Sagan, Hans (1994). 'Space-Filling Curves'. In: *Space-Filling Curves*. New York, NY: Springer. DOI: 10.1007/978-1-4612-0871-6 (cited on page 147).

Schön, Donald A. (1991). *The reflective practitioner : how professionals think in action*. eng. Farnham: Ashgate (cited on pages 62, 169, 170).

Selman, Bill (Aug. 2013). *Firefox User Types in North America*. URL: https://blog.mozilla.org/ux/2013/08/firefox-user-types-in-north-america/comment-page-1/ (cited on page 140).

Sheil, Beau (1983). *Power Tools for Programmers* (cited on page 18).

Shneiderman, B. (Aug. 1, 1983). 'Direct Manipulation: A Step Beyond Programming Languages'. In: *Computer* 16.8, pp. 57–69. DOI: 10.1109/MC.1983.1654471 (cited on page 38).

Sicchio, Kate (2014). 'Hacking Choreography: Dance and Live Coding'. In: *Computer Music Journal* 38.1, pp. 31–39. DOI: 10.1162/COMJ_a_00218 (cited on page 16).

– (June 2019). *Terpsicode – first performance with my new live coding language for choreography*. (Accessed July 8, 2019). URL: https://twitter.com/sicchio/status/1135924100130623488 (cited on page 16).

– (2016). 'Vibe Shirt: Reflections on a Duet for Coder and Dancer'. In: *Proceedings of the International Conference on Live Coding (ICLC) 2016* (cited on page 16).

Sorensen, Andrew and Henry Gardner (2010). 'Programming with Time: Cyber-Physical Programming with Impromptu'. In: *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications*. OOPSLA '10. Reno/Tahoe, Nevada, USA: Association for Computing Machinery, pp. 822–834. DOI: 10.1145/1869459.1869526 (cited on page 19).

Sorensen, Andrew, Ben Swift, and Alistair Riddell (Mar. 2014). 'The Many Meanings of Live Coding'. In: *Computer Music Journal* 38.1. "In most traditional formal systems contexts (e.g., GenJam, cf. Biles 2007) the rules and axioms are unalterable, once the system is defined it cannot be altered in playback. Live coding breaks from this rigidity by supporting a human composer who operates "above-the-loop," in that live coding allows for the run-time modification of the system's axioms and rules." page 3, pp. 65–76. DOI: 10.1162/COMJ_a_00230 (cited on pages 17, 18, 45).

Sorensen, Andrew Carl (2018). 'Extempore: The design, implementation and application of a cyber-physical programming language'. PhD thesis. College of Engineering and Computer Science, The Australian National University (cited on pages 19, 20, 27, 28).

Sung-Hoon, Ahn et al. (Jan. 1, 2002). 'Anisotropic material properties of fused deposition modeling ABS'. In: *Rapid Prototyping Journal* 8.4 (4), pp. 248–257. DOI: 10.1108/13552540210441166 (cited on pages 33, 34, 180).

Sweeny, James Johnson (1973). 'A conversation with Marcel Duchamp'. In: *Salt Seller: The Writings of Marcel Duchamp*. Ed. by Michel Sanouillet and Elmer Peterson. New York (cited on page 8).

Sweller, J. (2003). 'Evolution of human cognitive architecture'. In: *The psychology of learning and motivation* 43. Ed. by B. Ross, pp. 215–266 (cited on page 60).

Tanimoto, Steven L. (2013). 'A Perspective on the Evolution of Live Programming'. In: *Proceedings of the 1st International Workshop on Live Programming*. LIVE '13. San Francisco, California: IEEE Press, pp. 31–34. DOI: 10.1109/LIVE.2013.6617346 (cited on pages 14, 18, 19).

Taylor, Astra (Aug. 1, 2018). 'The Automation Charade'. In: *Logic* (5) (cited on page 11).

Teboul, EJ (2020). 'A Method for the Analysis of Handmade Electronic Music as the Basis of New Works'. PhD thesis. Ph. D diss., Rensselaer Polytechnic Inst., Troy, NY (cited on pages 45, 53).

Thompson, Rob (2007). *Manufacturing processes for design professionals*. eng. Lomdon: Thames & Hudson (cited on pages 33, 41).

Tompson, M. K. et al. (2016). 'Design for Additive Manufacturing: Trends, opportunities, considerations, and constraints'. English. In: *CIRP annals : manufacturing technology* 65.2, pp. 737–760. DOI: 10.1016/j.cirp.2016.05.004 (cited on pages 40, 43).

Trenouth, J. (Jan. 1991). 'A Survey of Exploratory Software Development'. In: *The Computer Journal* 34.2, pp. 153–163. DOI: 10.1093/comjnl/34.2.153 (cited on page 18).

Trevino, Jeffery Robert (2013). 'Compositional and analytic applications of automated music notation via object-oriented programming'. PhD thesis. UC San Diego (cited on page 177).

Turkle, Sherry and Seymour Papert (1990). 'Epistemological pluralism and the revaluation of the concrete.' In: *Signs: Journal of Women in Culture and Society* 16 (1), pp. 128–157 (cited on page 176).

– (1991). 'Epistemological pluralism: styles and voices within the computer culture'. In: *Constructionism*. Ed. by I. Harel and S. Papert. Ablex, pp. 161–192. Norwood, MA (cited on page 176).

Various (Apr. 2019a). *RepRap*. URL: https://reprap.org/wiki/RepRap (cited on page 33).

– (Apr. 2019b). *Ultimaker 2 Marlin Firmware*. URL: https://github.com/Ultimaker/Ultimaker2Marlin (cited on page 35).

Wang, Ge (2008). 'The Chuck Audio Programming Language. "a Strongly-Timed and On-The-Fly Environ/Mentality"'. AAI3323202. PhD thesis. Princeton, NJ, USA: Princeton University, USA (cited on page 72).

Wenger, Etienne (1999). *Communities of practice : learning, meaning, and identity*. eng. Learning in doing : social, cognitive and computational perspectives. Cambridge: Cambridge University Press (cited on page 32).

Westcott, Matt (Sept. 3, 2015). *MIDI-TO-CNC*. URL: https://github.com/gasman/MIDI-to-CNC/blob/master/mid2cnc.py (visited on 09/03/2019) (cited on pages 45, 145).

Wiener, Norbert (1961). *Cybernetics: or control and communication in the animal and the machine*. eng. 2nd ed. Cambridge, Mass.: M.I.T. Press (cited on page 16).

Wieser, Renate (Feb. 2018). 'Deautomatization of Breakfast Perceptions'. In: *The Oxford Handbook of Algorithmic Music*. Ed. by Roger T. Dean and Alex McLean. Oxford University Press. DOI: 10.1093/oxfordhb/9780190226992.013.28 (cited on page 16).

Wikipedia (2019a). '3D Manufacturing Format'. In: *Wikipedia* (cited on page 40).

– (Oct. 29, 2021). *I know it when I see it*. URL: https://en.wikipedia.org/wiki/I_know_it_when_I_see_it (cited on page 15).

– (2019b). 'STL File Format'. In: *Wikipedia* (cited on pages 35, 40).

Willis, Karl D. D. et al. (2011). 'Interactive Fabrication: New Interfaces for Digital Fabrication'. In: *Proceedings of the Fifth International Conference on Tangible, Embedded, and Embodied Interaction*. TEI '11. Funchal, Portugal: ACM, pp. 69–72. DOI: 10.1145/1935701.1935716 (cited on pages 38, 39, 67).

Winn, W. (2004). 'Cognitive Perspectives in Psychology'. In: *Handbook of Research on Educational Communications and Technology*. Ed. by D. H. Jonassen. Mahwah, NJ, Lawrence Erlbaum Associates (cited on page 59).

Winter, Sebastian, Stefan Wagner, and Florian Deissenboeck (2007). 'A comprehensive model of usability'. In: *IFIP International Conference on Engineering for Human-Computer Interaction*. Springer, pp. 106–122 (cited on page 102).

Wolfram, Stephen (2002). *A new kind of science*. eng. Champaign, IL: Wolfram Media (cited on page 156).

Wright, Matthew (2005). 'Open Sound Control: an enabling technology for musical networking.' In: *Organised Sound* 10 (3), pp. 193–200 (cited on page 27).

Zhao, Haisen et al. (July 2016). 'Connected Fermat Spirals for Layered Fabrication'. In: *ACM Trans. Graph.* 35.4. DOI: 10.1145/2897824.2925958 (cited on page 44).

Zuza, Mikolas (June 7, 2018). *Everything about nozzles with a different diameter*. Prusa Printers. URL: https://blog.prusaprinters.org/everything-about-nozzles-with-a-different-diameter_8344/ (visited on 03/23/2021) (cited on page 122).