Programming Language Tools and Techniques for 3D Printing

Chandrakana Nandi, Anat Caspi, Dan Grossman, and Zachary Tatlock

**Paul G. Allen School of Computer Science & Engineering,
University of Washington, Seattle, WA, USA**
`{cnandi, caspian, djg, ztatlock}@cs.washington.edu`

──── **Abstract** ────

We propose a research agenda to investigate programming language techniques for improving affordable, end-user desktop manufacturing processes such as 3D printing. Our goal is to adapt programming languages tools and extend the decades of research in industrial, high-end CAD/CAM in order to help make affordable desktop manufacturing processes more accurate, fast, reliable, and accessible to end-users. We focus on three major areas where 3D printing can benefit from programming language tools: design synthesis, optimizing compilation, and runtime monitoring. We present preliminary results on synthesizing editable CAD models from difficult-to-edit surface meshes, discuss potential new compilation strategies, and propose runtime monitoring techniques. We conclude by discussing additional near-future directions we intend to pursue.

1 Introduction

Affordable desktop-class 3D printers, laser cutters, and Computer Numerical Control (CNC) mills will soon be available to millions of people [4]. The potential social benefits of broad, end-user access to these technologies have been much hyped, but the current reality is that desktop-class hardware and tools are often significantly less accurate, fast, and reliable than their industrial counterparts. In industry, these processes take place on expensive, high-end machines managed by trained experts. While desktop-class hardware will continue to improve, we believe that without key software improvements, democratized manufacturing practice by end-users on affordable hardware is bound to fall short of its ambitious promise.

Many programming language techniques can be adapted to address analogous problems in desktop manufacturing—developing Computer Aided Design (CAD) models and editing existing objects are analogous to synthesis; generating accurate, efficient tool paths (paths the print head of a 3D printer follows) from a CAD model is analogous to optimizing compilation; automatically tracking operations to ensure safety and halt before bogus operations is analogous to runtime monitoring. Much as RAID software [26] enabled cheap, unreliable storage hardware to compete with expensive, reliable alternatives, we believe that programming language tools can significantly improve the state of the art in desktop
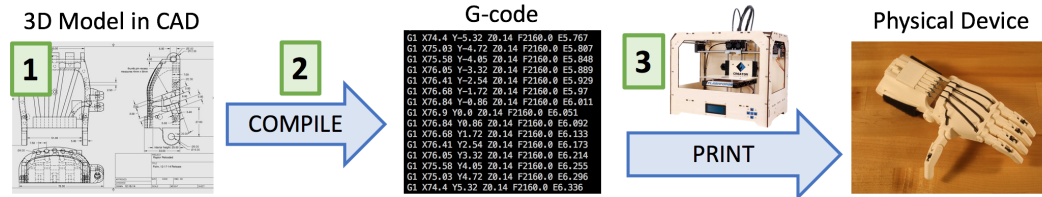
Figure 1 *The 3D Printing Development Cycle.* To 3D print a device: (1) An engineer first designs a 3D model using standard CAD tools (e.g., SolidWorks [13]). (2) This model is compiled into a sequence of low-level *G-code* commands that corresponds to basic actions the printer can take (move the print head, start/stop extrusion, lower the build plate, etc.). (3) The printer directly executes the G-code, producing a physical object.

manufacturing.[1] In the remainder of this paper we focus on one type of desktop manufacturing, 3D printing, but we believe the proposed research can be generalized to other processes like laser cutting and milling.

We propose a three-part research agenda to begin tackling these challenges for affordable, end-user desktop manufacturing:

- In Section 3, we discuss program synthesis techniques to generate easy-to-edit CAD models from difficult-to-edit surface meshes, and eventually to enable optimization and refactoring of complex CAD models.
- In Section 4, we discuss compiler techniques to improve printer performance via parallelization, and eventually to automatically account for observed errors in output prints.
- In Section 5, we discuss runtime monitoring for 3D printers to aid debugging, automatically detect printing errors, and eventually fix errors on the fly.

2   Background on 3D printing

3D printers come in a wide variety of designs, from lithography-based resin printers to inkjet-based powder printers. The challenges and techniques described in this paper apply to many of these designs, but to make the discussion more concrete, we focus on "cartesian fused filament fabrication" (FFF) printers (Figure 2, a), the most common and affordable type of printer. Figure 1 depicts the typical workflow for using such a device:

1. **Design** Users first design their model using CAD tools. There is a diverse array of available CAD tools including freely available options such as OpenScad [5] or SketchUp [11] and proprietary packages like Rhinoceros [8] and SolidWorks [13] which can cost thousands of dollars. In this paper, we focus on OpenSCAD since it conveniently represents CAD models as programs and is widely used on design sharing websites such as Thingiverse [14]. Figure 3 shows two CAD programs in OpenSCAD. OpenSCAD provides various primitive 3D structures (e.g., `cube`), transformations (e.g., `translate`) and combinators (e.g., `difference`), that can be used together to create complex 3D models. While OpenSCAD is programmatic, many CAD tools, such as Rhino [8], are GUI based. Irrespective of the interface, the models designed using these tools are declarative in nature, i.e., they only describe the 3D structure and parameters of a model, not how to manufacture it.

---

[1] Some gap between industrial and desktop manufacturing will always remain. Two ton CNC mills are inherently more rigid and stable than ten kilogram mini-mills after all.
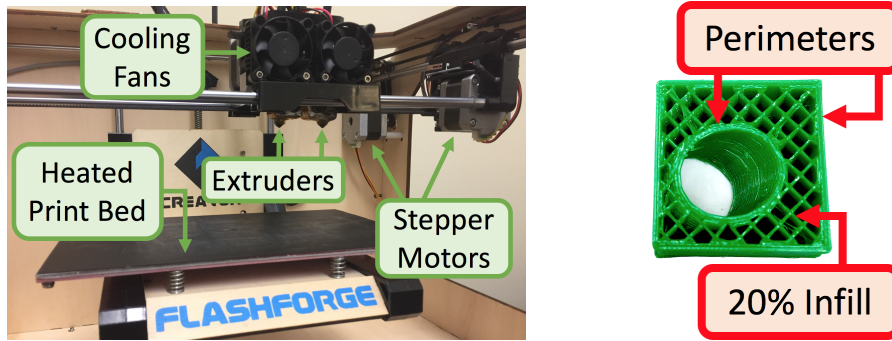
Figure 2 a) Major printer components. b) Perimeter and infill cross-section.
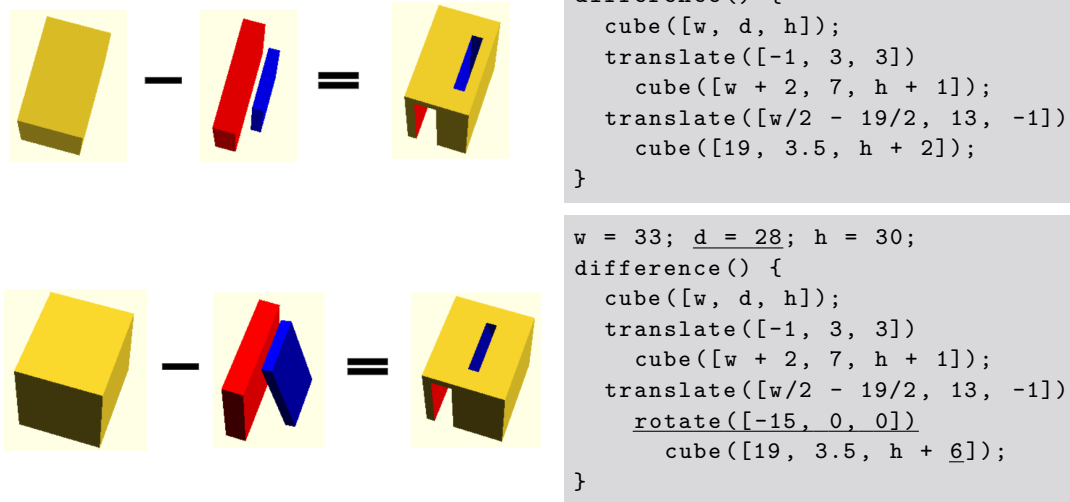
2. **Compilation** Compilation happens in two phases: (A) A CAD model is translated to an intermediate representation, typically in STereoLithography (STL) format [18]. This represents the surface mesh in the form of polygons in a 3D coordinate system. (B) The STL is "sliced" to obtain *G-code*. The G-code is a sequence of imperative commands that control extrusion, movement and temperature. The slicer determines the *tool path* which refers to the path the print head should follow while printing. A typical slicing strategy is discussed in Section 2.1 in more detail.

3. **Print** The printer runs firmware that interprets the G-code and sends low-level hardware control signals to motors, heating elements, and cooling fans. An extruder melts print material and pushes it through a nozzle to build up the part layer-by-layer starting with the first layer directly on the build plate. There are many physical phenomena involved in this step that affect the print quality—the inertia on the print head, thermal expansion of the print material, the temperature and humidity of the environment, etc.

4. **Iterate** Finally, there is an implicit fourth step which is to repeat the above steps until the 3D object comes out as expected.

2.1 Baseline Slicing

At a high level, common slicers [10, 9, 3, 12] take a 3D surface geometry in STL and divide it into a sequence of 2D slices parallel to the xy-plane at regular intervals of height h (typically $h \approx 0.1$mm). Thus the ith slice represents the perimeters of the object to be printed at height $i \times h$. To generate G-code, the slicer computes tool paths to trace the perimeters at each height and fill the space between perimeters with a regular pattern at some user-specified density (see Figure 2 b). Within these layers, the slicer inserts G-codes to start and stop extrusion during movements along perimeters and over fill areas. The slicer then inserts additional G-code between the instructions for each layer to increment the printer's z axis by h. Finally, the slicer inserts an initial preamble to set fan speeds as well as build plate and extruder temperatures to appropriate values for the material being printed. Throughout the slicing process, the compiler performs optimizations to minimize the travel time of the print head.

2.2 Challenges in 3D printing

CAD/CAM and related computer-aided manufacturing are some of the oldest areas of computer science [31]. However, work in this space is often targeted at an industrial setting

```
w = 33; d = 20; h = 30;
difference() {
  cube([w, d, h]);
  translate([-1, 3, 3])
    cube([w + 2, 7, h + 1]);
  translate([w/2 - 19/2, 13, -1])
    cube([19, 3.5, h + 2]);
}
```

```
w = 33; d = 28; h = 30;
difference() {
  cube([w, d, h]);
  translate([-1, 3, 3])
    cube([w + 2, 7, h + 1]);
  translate([w/2 - 19/2, 13, -1])
    rotate([-15, 0, 0])
      cube([19, 3.5, h + 6]);
}
```

Figure 3 Renderings of a tea scoop holder with the original CAD program and the modified CAD program with the wall thickness and angle of the holder changed (changes are underlined).

where accurate, fast (and therefore expensive) equipment is operated by highly motivated experts. The advent of affordable, desktop-class 3D printers for end-users gives rise to new challenges that we believe programming language techniques can help address. While improving hardware trends will inevitably ease some challenges with 3D printing, we believe that new software techniques will be essential for narrowing the gap between what's possible on affordable hardware and industrial practice. Toward that end, we propose initially focusing on three challenge areas:

Design CAD tools have been widely used for decades, but still present users with a steep learning curve—even if one can clearly describe the desired model in plain English, it is not obvious what buttons to click and menus to navigate in the CAD tool to actually make that model from scratch. One possibility is to customize existing CAD models to meet new requirements. Unfortunately, most of the models shared in large online repositories like Thingiverse [14] are not the CAD models—they contain only the surface mesh in the form of STL which is difficult to edit successfully since much of the high-level information about the design (e.g. structural constraints) has been compiled away.

Performance 3D printing is a slow process—it can take more than a day to print a large complex model. It is also generally unclear when and where the process can be made faster—going too fast in regions with fine detail can ruin a print because the material may not have time to cool sufficiently before the next layer.

Popular slicers such as Simplify3D [9], Cura [3], and ReplicatorG [7] cannot generate G-code that takes advantage of multiple print heads simultaneously, and thus in practice most printers with multiple print heads use only one. As discussed later, exploiting such latent parallelism significantly complicates the slicing strategy.

Furthermore, 3D printing typically involves manual inspection and tweaking. Users must often repeat the process several times to get the print they expected. Each iteration requires manually editing the CAD model, slicer parameters, or the printer settings. Ideally, users could avoid such manual fixes if slicers were able to automatically compensate for errors between iterations.

Reliability 3D printing depends on the type of printer, the material being printed, and environmental conditions such as temperature. Even with perfect designs that have been correctly sliced, some problems that arise during printing can only be noticed *while* printing. It would be ideal if whenever an error occurs, we could halt the print to avoid wasting time and material and then work backwards to identify which command in the G-code led to the failure. Ideally, this information could even be used to repair errors automatically on the fly.

3 Synthesis

For many users, designing a part from scratch is challenging due to CAD's steep learning curve. They avoid this challenge by downloading, slicing, and printing parts shared as STL files in online repositories like Thingiverse [14]. Some users scan parts they wish to print using 3D scanners which also produce STL-like representations. These approaches are sufficient when the part is standalone and fits the user's needs. However, it is insufficient when the user wants to combine or modify parts. This is because many modifications are difficult in surface geometry representations like STL. STL tools like Blender [2] or AutoDesk's MeshMixer [1] can easily scale and rotate a design, but cannot effectively modify parts where some aspects depend on others (e.g., a gear whose tooth count depends on its radius). Even when CAD programs are made available, they can still be difficult to edit as end users often do not parameterize their designs or incorporate the structural constraints that make expert-written models easy to modify.

Past research in computer graphics and animation has focused on obtaining higher level representations from low level polygon meshes. For example, Krishnamurthy et al. [20] have shown how to fit smooth surfaces to irregular polygon meshes using B-splines and displacement maps. While smoothing can convert dense polygon meshes to aesthetically pleasing and more user-friendly representations, the output of these manipulations contains limited structural information about the model (e.g., if the model is a gear, what are the dimensions, orientations, and angles of its teeth?). Having this information is particularly important for desktop class 3D printing where users might want to individually customize functional parts by changing the relationships between its subcomponents (by varying the design parameters). On the other hand, in graphics and animation, aesthetics and performance are of key importance. The main difference between the idea we present here and prior work on fitting surfaces to polygon meshes is that we are primarily interested in recovering underlying structural information from polygon meshes and presenting it in the form of editable CAD models so that modifications and manipulations of subparts becomes straightforward. With this motivation, we propose *synthesizing* well-engineered and easy-to-edit CAD models from surface geometry representations like STL.

For example, consider the model of a tea scoop holder in Figure 3. In order to make the tea scoop fit better, we wanted to change the angle of the holder which required increasing the thickness of the base so that the holder would not cut through the walls due to the rotation. As Figure 3 shows, this change was very easy to make in the CAD model (changes underlined). In general, such changes are difficult to make by editing the STL surface mesh because some parts of the object remain unchanged while others are scaled and yet others are independently rotated.

As another example, consider the model of a chicken in Figure 4. The legs of this model are too thin and hence broke easily during printing. We wanted to make them thicker while keeping the rest of the model at the same scale and ensuring that the chicken still balances
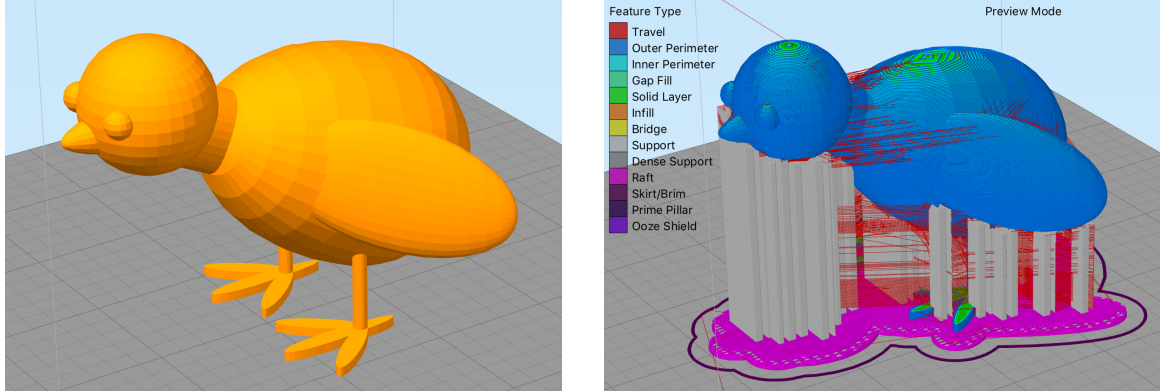
Figure 4 a) CAD model of a chicken. b) Tool path produced by slicer

stably on its feet. The most convenient way to do this is to simply increase the radius of the leg cylinders in CAD. However, the CAD model for the chicken was not available—we only had access to the surface mesh in the form of an STL file. By reverse engineering the CAD from the surface mesh, we were able to easily thicken the legs and successfully print the model.

We have designed and implemented an early prototype synthesis algorithm (Algorithm 1) that achieves some of the goals above. This is essentially a form of decompilation: given an STL file S, find a simple CAD model which, when rendered, yields S. The algorithm is based on the principle that every CAD model can be synthesized by either subtracting one part from another part or unioning two parts together. Like many early program synthesis projects, this algorithm is a combinatorial search that is intractable for models with more than a dozen parts. However, in our problem domain, that is often plenty—OpenSCAD for example has only 4 types of primitive solid objects that can be combined to build various complex models. Figure 3 (column 2) shows example outputs of our algorithm.

Future directions

We believe that the intersection of CAD modeling and program synthesis is ripe with interesting problems. As one concrete example, our prototype synthesis algorithm tends to produce overly verbose CAD programs for highly symmetric parts since the algorithm's search omits looping constructs. More generally, we believe research should explore synthesis techniques for minimizing CAD models, similar to copy paste detection [21], and also for superoptimizing G-code, similar to techniques used in highly parallel low-power architectures [27].

We also propose further synthesis of high level models from surface meshes, but for more constrained targets than full CAD models. In particular, "peeling" based modeling where an object is approximated by composing interlocking flat sheets. Such designs have the advantage of being printable as only flat sheets, which is faster and packs tighter than traditional solid printing designs. Another natural generalization of this approach is exploring how designs can be synthesized to take advantage of flexible filaments, e.g., by generating origami-inspired hinged designs.

Algorithm 1 Synthesis algorithm for generating CAD models

---

procedure SEARCH(model)
    if empty(model) **then return** [*Empty*()]
    else
        candidates = [ ]
        for b **in** PRIMITIVEBOUNDS(model) **do**
            diff = SUBTRACT (b, model)
            for c **in** SEARCH (diff) **do**
                candidates.append(*Diff*(b,c))
        for m1, m2 **in** SPLIT (model) **do**
            cs1 = SEARCH (m1)
            cs2 = SEARCH (m2)
            for c1 **in** cs1 **do**
                for c2 **in** cs2 **do**
                    candidates.append(*Union*(c1, c2))
        return candidates

---

4   Compilers for 3D Printing

3D printing seeks to efficiently compile an abstract object description to an actual, physical object. As described in Section 2, this compilation is typically composed of three stages: (1) CAD to STL, (2) STL to G-code, and (3) G-code to low-level hardware control signals. Just as traditional compiler research often focuses on middle-ends, here we focus on stage (2), also known as the *slicer*. The slicer is an ideal target as it typically has the greatest impact on print time and quality and also translates between standard languages independent of front-end CAD details and back-end printer firmware details.

In addition to the core compilation strategy presented in Section 2, slicers provide additional important features, as shown in Figure 4 (right). These include inserting support structures under part overhangs beyond some threshold angle d (typically $d \approx 45°$) and inserting a "raft," a thick set of initial layers to improve part adhesion to the print bed. These additions are often essential for successfully printing complex parts and we hope to explore their design space in future work. However, we propose that initial PL research in this area should begin by focusing on the core compilation challenges of performance, accuracy, and correctness.

Parallelization

Many desktop class printers have multiple extruders which, in principle, should enable parallelism during the printing process. In practice, these extruders are only used one at a time to support features such as multi-color prints or using dissimilar raft and support materials. Exploiting the latent parallelism of multiple print heads requires extending the slicing algorithm to partition the tool paths within each layer to sets of paths for each head. This is challenging because the print heads are in a fixed orientation relative to one another (typically mounted linearly along the printer's upper gantry). Thus, all extruders move together at fixed offsets from one another. Correctly generating G-code to manage the timing of all the coordinated movements presents a significant compilation challenge. Recently, some researchers have started focusing on parallelizing 3D printing for specially-built industrial printers [6], but the techniques used are proprietary and it is still unclear how they can be

applied to help end-users operating desktop class printers.

Our goal is to explore how classic compiler techniques such as peephole optimizations can be applied to achieve parallelism. We suggest building directly upon the simple baseline slicing strategy without making additional assumptions about printer hardware, since, as mentioned above, an important objective is maintaining accessibility for end users. As a first step, we will develop a G-code analysis to identify situations where a secondary extruder will entirely traverse an extrusion path parallel to one that the primary extruder would eventually extrude anyway. In these scenarios, we can keep the G-code produced by the traditional slicing algorithm and merely tweak it to both (1) enable the secondary extruder as it traverses the parallel future path, (2) remove the G-code for the primary extruder following the subsequent path, and (3) patch up movements to connect the G-code before and after the removed path.

Future Directions

Analogous to the bad old days of early compilers, users must occasionally *manually* tweak the generated G-code to fix some print errors. This can be due to misbehavior of the printer hardware (e.g., certain movements may cause a stepper motor to "skip" a position, especially at high speeds, and require a small G-code tweak to mitigate) or bugs in the slicer (e.g., failure to retract the filament before a long movement, leading to smearing). We propose that future slicer research investigate improving accuracy by incorporating error from earlier trials into subsequent re-slicings. For example, if a generated part is 0.3mm too narrow in the x direction due to printer hardware inaccuracy, the slicer could automatically insert "padding" in x movements to compensate. We also hope to explore formalizing both STL and G-code in order to reason formally about the correctness of slicing algorithms. Such a formal foundation will also enable implementing more sophisticated slicing algorithms with confidence by proving them equivalent to simpler strategies.

5   Runtime Monitoring

Desktop-class 3D printers are currently affordable because they use inexpensive stepper motors, basic extruder designs, and lightweight frames. While these economical choices are precisely what make the technology broadly available, they also lead to unreliability and error. Even with a perfect CAD design and error-free slicing, prints can still fail due to motors skipping steps, nozzles clogging, and environmental variations in temperature and humidity. Future hardware improvements may mitigate some of these concerns, but even experts operating high-end equipment still must often iteratively refine their process to get the best results. Debugging failures and making performance tweaks is difficult because the operation of the printer is opaque as it interprets tens of thousands of lines of G-code to generate a part. In this section we propose video-based runtime monitoring techniques to help debugging, detect printing errors, and address failures on the fly. These techniques take inspiration from traditional programming language runtimes which aid debugging and ensure safety by providing facilities to handle exceptions, prevent errors like division by zero or array out of bounds accesses, and dynamic type checking.

Record. Resemble? Respond!

Sitthi-Amorn et al. [29] have shown the use of depth cameras in runtime monitoring to repair height errors *on the fly* in their 3D printing platform, MultiFab. They use image processing

and 3D scanning to identify pixels with varying depths and add pixel-wise corrective layers to the printing process. We propose extending these techniques to validate and repair other properties (e.g., dimensional accuracy) while also keeping the hardware requirements affordable and performance overhead low.

A first step in aiding low cost print failure debugging is to log operations using commodity cameras to record prints and tag each frame with the currently executing G-code instruction. The logs can help users identify where the G-code may need to be tweaked to address poor printer performance. These logs can also help printer firmware developers tune and debug the low-level control code that translates G-code operations into carefully timed motor commands.

With this simple foundation laid, the next natural step would be to develop analyses which compare G-code programs and printing video streams to ensure that execution correctly matches expected behavior. Such analyses can be used to abort print jobs early or selectively disable printing in independent regions where something has gone wrong. This can be useful when a long-running job printing multiple copies of a complex part goes wrong for just one of the copies. Currently, the printer blindly continues executing G-code, oblivious to the small localized failure. This often causes cascading errors as subsequent extrusions over the failed area do not adhere correctly and are dragged over to interfere with the printing of other copies which, independent of the initial failure, would have otherwise successfully printed. If instead a runtime monitor could detect that an execution is no longer faithfully simulating the behavior specified by the input G-code program, printing could be halted early for failed parts, allowing other parts to successfully finish printing and to avoid wasted material.

Future Directions

A major challenge with video-based runtime monitoring for 3D printers is that responses must be carried out quickly in order to be effective, but printers typically only contain cheap microcontrollers for executing firmware. Future research should explore hybrid analysis techniques where partial evaluation of a video-based analysis is carried out at slicing time, before the first G-code instruction for a part is ever sent to the printer. Video-based analyses should also be investigated to enable coordination of printers with other manufacturing processes, e.g., a robotic arm. Such coordination could enable more sophisticated multi-process desktop manufacturing, e.g., by enabling a pick-and-place machine to embed magnets or metal fixtures within a part as it is being printed.

6 Related work

Several projects have explored new analyses of 3D models, slicing techniques, and user interfaces to help mitigate current limitations in 3D printing. These results appear across a diverse array of venues, from graphics to HCI, and many focus on industrial settings or specialized hardware which future economies of scale or hardware improvements may make broadly accessible. The programming languages community has only recently started looking into these problems, e.g., in OpenFab [34], a framework for programmatically specifying material and texture with the help of a domain specific language. Below we highlight some noteworthy and inspirational examples from other communities attacking 3D printing challenges.

The strength of a 3D printed part is non-uniform due to stronger adhesion within a layer than across layers. Umetani et al. developed a static analysis of CAD models to determine optimal printing orientations for maximizing mechanical strength [33]. Galjaard et

al. explored optimizing CAD models to minimize material use while maintaining key strength performance properties [17]. Teibrich et al. [32] introduced a patching technique to repair already printed objects to avoid printing again from scratch, thereby saving material. Delfs et al. [15] developed a tool that can optimize the orientation of a part during 3D printing in order to make the surface smoother.

In terms of speeding up early prints, Mueller et al.'s work on WirePrint [23] and faBrickator [25] provide creative examples of how non-uniform height slicing and hybrid build approaches (in this case using Lego^{TM}) can radically reduce turnaround time when developing prototypes. Mueller et al. [24] have also introduced laser cutting based techniques for rapid prototyping using folding and stretching of an object instead of cutting joints.

Stava et al. [30] proposed a technique based on structural analysis that automatically detects and fixes structural problems in models. Zhou et al. [35] proposes another structural analysis for 3D printable objects that uses material and geometric properties. FlatFitFab [22] is an interactive interface that allows users to specify functional parts and provides real-time simulations that visualize stress. Dumas et al. [16] recently proposed a texture synthesis algorithm that takes a surface mesh and an example pattern as inputs and generates a texture.

New 3D printing applications are also constantly emerging, particularly within medical contexts such as tissue and organ fabrication; customized prosthetics and implants; and drug manufacturing, dosage forms, delivery, and discovery [19, 28].

7 Conclusion

In this paper, we proposed an early research agenda for using programming language techniques to help make affordable, desktop-class manufacturing processes (such as 3D printing) more accurate, fast, and accessible to end-users. Even as the available hardware improves, we believe there will continue to be opportunities for software to narrow the gap between expensive, high-end processes and the widely available, democratized means of production. Here, we discussed three major domains where 3D printing in particular can benefit from such research—applying program synthesis techniques to improve the design process, applying compiler techniques to speed up and improve prints, and applying runtime monitoring approaches to ease debugging. We are eager to further explore these particular lines of work and look forward to seeing how the PL community can help address these challenges more broadly.

8 Acknowledgements

References

1 Autodesk. Meshmixer. http://www.meshmixer.com/.
2 blender. Creative freedom starts here. https://www.blender.org/.
3 Cura Software. https://ultimaker.com/en/products/cura-software.

4   Forecast: 3D Printers, Worldwide, 2015. `https://www.gartner.com/doc/3132417`.

5   OpenSCAD. `http://www.openscad.org/`.

6   Project Escher. `http://projectescher.com/`.

7   ReplicatorG lowering the barrier to 3d printing. `http://replicat.org/`.

8   Rhinoceros. `https://www.rhino3d.com/`.

9   SIMPLIFY3D. `https://www.simplify3d.com/`.

10   Skeinforge. `http://reprap.org/wiki/Skeinforge`.

11   SketchUp. `http://www.sketchup.com/`.

12   Slic3r. `http://slic3r.org/`.

13   Solidworks. `http://www.solidworks.com/`.

14   Thingiverse. `http://www.thingiverse.com/`.

15   P. Delfs, M. $\ddot{T}$ows, and H.-J. Schmid. Optimized build orientation of additive manufactured parts for improved surface quality and build time. *Additive Manufacturing*, 12, Part B:314 – 320, 2016. Special Issue on Modeling & Simulation for Additive Manufacturing. URL: `http://www.sciencedirect.com/science/article/pii/S2214860416301142`, `doi:http://dx.doi.org/10.1016/j.addma.2016.06.003`.

16   Jérémie Dumas, An Lu, Sylvain Lefebvre, Jun Wu, and Christian Dick. By-example synthesis of structurally sound patterns. *ACM Trans. Graph.*, 34(4):137:1–137:12, July 2015. URL: `http://doi.acm.org/10.1145/2766984`, `doi:10.1145/2766984`.

17   Salomé Galjaard, Sander Hofman, and Shibo Ren. *New Opportunities to Optimize Structural Designs in Metal by Using Additive Manufacturing*, pages 79–93. Springer International Publishing, Cham, 2015. URL: `http://dx.doi.org/10.1007/978-3-319-11418-7_6`, `doi:10.1007/978-3-319-11418-7_6`.

18   T. Grimm. *User's Guide to Rapid Prototyping*. Society of Manufacturing Engineers, 2004. URL: `https://books.google.com/books?id=o2B7OmABPNUC`.

19   G.T. Klein, Y. Lu, and M.Y. Wang. 3d printing and neurosurgery– ready for prime time? *World Neurosurgery*, 80(3):233–235, 9 2013.

20   Venkat Krishnamurthy and Marc Levoy. Fitting smooth surfaces to dense polygon meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 313–324, New York, NY, USA, 1996. ACM. URL: `http://doi.acm.org/10.1145/237170.237270`, `doi:10.1145/237170.237270`.

21   Zhenmin Li, Shan Lu, Suvda Myagmar, and Yuanyuan Zhou. Cp-miner: A tool for finding copy-paste and related bugs in operating system code. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 20–20, Berkeley, CA, USA, 2004. USENIX Association. URL: `http://dl.acm.org/citation.cfm?id=1251254.1251274`.

22   James McCrae, Nobuyuki Umetani, and Karan Singh. Flatfitfab: Interactive modeling with planar sections. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, pages 13–22, New York, NY, USA, 2014. ACM. URL: `http://doi.acm.org/10.1145/2642918.2647388`, `doi:10.1145/2642918.2647388`.

23   Stefanie Mueller, Sangha Im, Serafima Gurevich, Alexander Teibrich, Lisa Pfisterer, François Guimbretière, and Patrick Baudisch. Wireprint: 3d printed previews for fast prototyping. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, pages 273–280, New York, NY, USA, 2014. ACM. URL: `http://doi.acm.org/10.1145/2642918.2647359`, `doi:10.1145/2642918.2647359`.

24   Stefanie Mueller, Bastian Kruck, and Patrick Baudisch. Laserorigami: Laser-cutting 3d objects. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 2585–2592, New York, NY, USA, 2013. ACM. URL: `http://doi.acm.org/10.1145/2470654.2481358`, `doi:10.1145/2470654.2481358`.

25 Stefanie Mueller, Tobias Mohr, Kerstin Guenther, Johannes Frohnhofen, and Patrick Baudisch. fabrickation: Fast 3d printing of functional objects by integrating construction kit building blocks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 3827–3834, New York, NY, USA, 2014. ACM. URL: `http://doi.acm.org/10.1145/2556288.2557005`, `doi:10.1145/2556288.2557005`.

26 David A. Patterson, Garth Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (raid). In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, SIGMOD '88, pages 109–116, New York, NY, USA, 1988. ACM. URL: `http://doi.acm.org/10.1145/50202.50214`, `doi:10.1145/50202.50214`.

27 Phitchaya Mangpo Phothilimthana, Aditya Thakur, Rastislav Bodik, and Dinakar Dhurjati. Scaling up superoptimization. *SIGPLAN Not.*, 51(4):297–310, March 2016. URL: `http://doi.acm.org/10.1145/2954679.2872387`, `doi:10.1145/2954679.2872387`.

28 C. Schubert, M.C. van Langeveld, and L.A. Donoso. Innovations in 3d printing: a 3d overview from optics to organs. *British Journal of Ophthalmology*, 98(2):159–161, 2014.

29 Pitchaya Sitthi-Amorn, Javier E. Ramos, Yuwang Wangy, Joyce Kwan, Justin Lan, Wenshou Wang, and Wojciech Matusik. Multifab: A machine vision assisted platform for multi-material 3d printing. *ACM Trans. Graph.*, 34(4):129:1–129:11, July 2015. URL: `http://doi.acm.org/10.1145/2766962`, `doi:10.1145/2766962`.

30 Ondrej Stava, Juraj Vanek, Bedrich Benes, Nathan Carr, and Radomír Měch. Stress relief: Improving structural strength of 3d printable objects. *ACM Trans. Graph.*, 31(4):48:1–48:11, July 2012. URL: `http://doi.acm.org/10.1145/2185520.2185544`, `doi:10.1145/2185520.2185544`.

31 Ivan E. Sutherland. Sketch pad a man-machine graphical communication system. In *Proceedings of the SHARE Design Automation Workshop*, DAC '64, pages 6.329–6.346, New York, NY, USA, 1964. ACM. URL: `http://doi.acm.org/10.1145/800265.810742`, `doi:10.1145/800265.810742`.

32 Alexander Teibrich, Stefanie Mueller, François Guimbretière, Robert Kovacs, Stefan Neubert, and Patrick Baudisch. Patching physical objects. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, UIST '15, pages 83–91, New York, NY, USA, 2015. ACM. URL: `http://doi.acm.org/10.1145/2807442.2807467`, `doi:10.1145/2807442.2807467`.

33 Nobuyuki Umetani and Ryan Schmidt. Cross-sectional structural analysis for 3d printing optimization. In *SIGGRAPH Asia 2013 Technical Briefs*, SA '13, pages 5:1–5:4, New York, NY, USA, 2013. ACM. URL: `http://doi.acm.org/10.1145/2542355.2542361`, `doi:10.1145/2542355.2542361`.

34 Kiril Vidimče, Szu-Po Wang, Jonathan Ragan-Kelley, and Wojciech Matusik. Openfab: A programmable pipeline for multi-material fabrication. *ACM Trans. Graph.*, 32(4):136:1–136:12, July 2013. URL: `http://doi.acm.org/10.1145/2461912.2461993`, `doi:10.1145/2461912.2461993`.

35 Qingnan Zhou, Julian Panetta, and Denis Zorin. Worst-case structural analysis. *ACM Trans. Graph.*, 32(4):137:1–137:12, July 2013. URL: `http://doi.acm.org/10.1145/2461912.2461967`, `doi:10.1145/2461912.2461967`.