Conversation: An Accessible CAD Program for Students and Hobbyists to Rapidly Design 3D-Printed Robots

James M. Bern, Michael Faulkner, Nathan J. Vosburg

***Abstract*— We observe a mismatch between the strengths of modern mechanical computer-aided design (CAD) software and the needs of educators teaching robotics with 3D-printing. In this paper we present a new, experimental CAD software called Conversation that maintains a complete separation between 2D drawings and 3D geometry, and demonstrate how this unique design decision can simplify and accelerate the design process for students and hobbyists. Additionally, we detail geometric algorithms that make Conversation faster and easier to use, and explain how tweening makes Conversation a useful pedagogical tool in lecture. Finally, we share results of using Conversation as the primary mechanical CAD software in a semester-long college-level course on robotics and digital fabrication.**

I. Introduction

This paper is about a new mechanical computer-aided design (CAD) program that makes it easier for students and hobbyists to rapidly design their own 3D-printed robots. Our program is targeted at non-mechanical engineering students and hobbyists, including middle-school or high-school students in a STEM summer camp, EE/CS college students in a mechatronics course, 3D-printing enthusiasts in a maker space, and Robotics graduate students in an academic lab. In this paper, we ask the question: **What is the right CAD tool to enable students and hobbyists to quickly design and 3D-print their own custom robots?**

Robots have long been a mainstay of STEM education, and a variety of kits like LEGO Mindstorm and VEX Robotics [1] give children the chance to assemble exciting robots from premade parts. The advent of cheap, reliable FDM (fused deposition modeling) 3D-printers promised students something even better: the ability to directly turn their own creative ideas into robotic reality. Unfortunately, early enthusiastic adopters of 3D-printers quickly learned a bitter lesson: *fabrication*–the actual building of things–is only half of the battle [2]. Without accessible *design* tools, a 3D-printer is basically just an expensive paperweight. Students and children might download and print a few 3D models from the Internet, but the 3D-printer's (actually quite real) promise of extraordinary creative freedom remained unrealized.

Designing relatively simple 3D-printed objects and machines in mainstream general-purpose mechanical CAD programs like SolidWorks can be a complex and frustrating process [2]. This fact is not entirely surprising, as mainstream CAD programs were designed to meet the extremely demanding needs of large-scale industries like automotive and aerospace design. One part of the problem is complexity; mainstream CAD programs have many, many features which students often use by accident. For example, in SolidWorks, it is easy to accidentally add unwanted sketch relations, creating a mess of conflicting constraints. Another part of the problem is audience; mainstream CAD programs were designed to appeal to professional mechanical designers, and, as such, lack the engaging colors and explanatory animations we would expect in a program specifically designed for children [3]. And finally, we believe part of the problem to be more fundamental, and stem from the fact that mainstream CAD programs tend to be oriented around the creation of a single part, rather than an assembly.

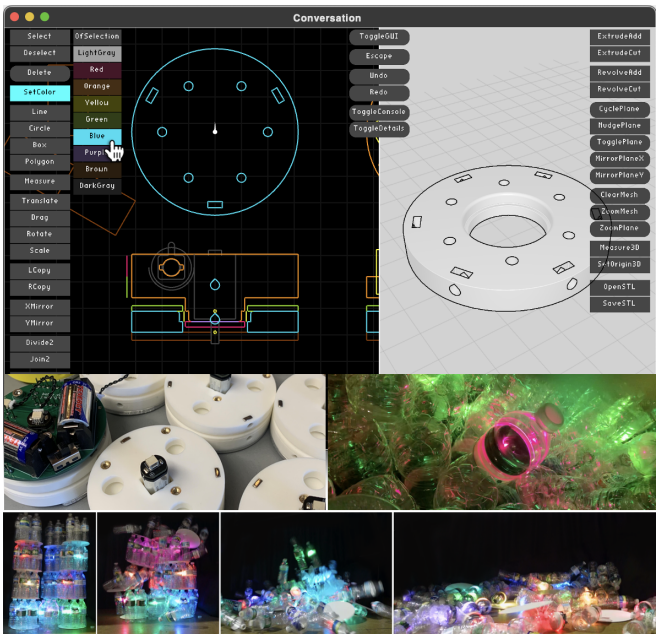

Fig. 1: We present Conversation, a new, accessible CAD software for rapidly designing 3D-printed robots. In Conversation, the user creates a large, 2D drawing of an entire robot (top, left pane). From this drawing, the user can create 3D-printable meshes of the robot's parts (top, right plane). Importantly, the 2D drawing and 3D geometry remain completely separate. In the screenshot, one of the authors has just finished modeling a 3D-printed plate used in the robotic art installation *Not-So-Micro Plastic* at ICRA 2025 (bottom).

Our goal–to make it easier for students and hobbyists to design and create their own 3D-printed robots–has been pursued by members of multiple academic communities. Some work from the computer graphics and robotics communities attempts to automate part or all of the design process. These papers usually present specific *algorithms* rather than

Computer Science Department, Williams College jmb15@williams.edu

general-purpose CAD software. These include algorithms for designing the bodies of 3D-printed robots [4], [5] and generating motions [6], [7]. Diffusion-type generative models for robot design also fall into this category [8], [9].

Other work, from industry and the open-source community, has produced full, "hobby-grade" CAD programs that try simplify the CAD experience. One path forward is to shun the otherwise ubiquitous "2D sketch → 3D geometry" workflow [10]. For example, AutoDesk's TinkerCAD [11] and Womp's eponymous program [12] both work by providing the user with a set of 3D primitives that can transformed and combined together using boolean operations. The open-source OpenSCAD [13] also works this way, albeit with a radically different text-based interface for creating and modifying geometry.

However, for our use case, what we really want is a tool that enables students to leverage modern CAD's powerful and expressive "2D sketch → 3D geometry" workflow. So, rather than seeking to automate or subvert this workflow, we instead simplify the overall experience of it in a divide an conquer fashion by creating a CAD program that maintains a complete separation between 2D and 3D.

Specifically, we contribute:

- Conversation, a new top-down CAD software that maintains a complete separation between 2D drawings and 3D geometry.
- A search-based algorithm for automatically registering 2D drawings to 3D geometry.
- Explanations of multiple sketching tools that are especially useful for 3D-printed robots.
- Description of a simple, effective approach to tweening (animating) many parts of a CAD program.
- An example use case of Conversation as the primary mechanical CAD software in a semester-long, project-based robotics course.

II. Motivation for Separating 2D and 3D

To understand why mainstream programs may not be ideal for teaching 3D-printed robot design, let us consider a typical modern parametric CAD program like SolidWorks or Fusion 360. Drawing an analogy to computer programming and the *object-oriented programming* (OOP) paradigm, we can say that SolidWorks is primarily *part-oriented* (*bottom-up*). Each part lives in its own file, with its own sketches and features. Once all parts have been designed in 3D, they can be put together into a virtual *assembly* of the overall robot. A major benefit of the part-oriented paradigm is that suppliers often provide downloadable part files for off-the-shelf parts.

While the part-oriented paradigm is ubiquitous among industry-grade CAD programs like SolidWorks, it has notable drawbacks for 3D-printing hobbyists and students designing their own robots. Part-oriented robot design can feel like trying to solve a crossword puzzle while only considering each clue in isolation, waiting until the very end to see if your answers actually fit together. Each part may seem to make sense on its own, but this illusion falls apart if we consider the whole.

Part-oriented design postpones consideration of a robot's interfaces until all parts have been designed, at which point students are already dreadfully deep into their design process. Furthermore, since students will be designing and 3D-printing their own brackets, rather than getting them from a kit, part-oriented CAD does not yield its typical benefits. Motivated by this shortcoming of modern, mainstream CAD when it comes to teaching 3D-printed robot design, we wonder what an alternative CAD program might look like. Specifically, **we want a CAD software that encourages students to consider the key interfaces of their design from the very beginning**. In this paper, we present Conversation, an experimental *assembly-oriented* CAD program designed from the ground up to be a faster, more accessible way to design 3D-printed robots.

III. System Overview

A. User Interface

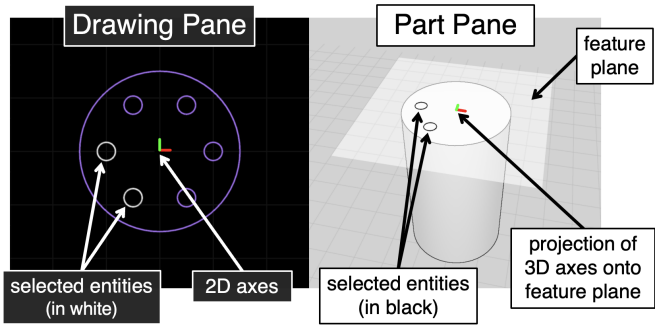

Fig. 2: Annotated screenshots of Conversation's user interface, which has two separate panes, one for the 2D drawing (Drawing Pane) and one for a 3D part (Part Pane). The user can select entities inside the Drawing Pane, which then show up on the feature plane inside the Part Pane.

Conversation's full user interface is shown in Figure 1, consisting of two main panes and buttons for choosing commands. All buttons have corresponding hotkeys, and students can hide the buttons once they learned the hotkeys in order to free up space. Conversation has two panes, which are detailed in Figure 2. The *Drawing Pane* contains a large 2D assembly drawing (sketch), and the *Part Pane* contains the single 3D part (mesh) currently being modeled. The Part Pane is also where the *feature plane* will show up when it is active. The feature plane is used to temporarily locate the drawing in 3D space in order to edit the mesh, a process we will describe further in Section III-B. Finally, in Sections III-C and III-D, we will explain Conversation's specific, unconventional implementations of 2D drawings and 3D parts, and argue why these design decisions make sense for our particular target audience. However, it is important to note here that Conversation's fundamental design decision–to completely separate 2D drawings from 3D parts, and relate them only temporarily through a dynamic feature plane–is agnostic to these specific implementation decisions.

B. Conversational Workflow

Conversation's core 2D → 3D workflow is shown by example in Figure 3 and consists of 1) choosing a feature plane, 2) selecting drawing entities, and 3) extruding (or revolving) those 2D entities into new 3D geometry. Conversation makes it convenient to put all of a robot's 2D geometry, including multiple views of multiple parts, into a single drawing by allowing the user adjust the 2D drawing's axes (coordinate system) with commands like `SetOrigin2D` and `RotateAxes2D`. If desired, the user can also adjust the feature plane and 3D axes with commands like `RotatePlane` and `SetOrigin3D`. Figure 3 includes an example of the user employing `SetOrigin2D` to switch from modeling off the top view to the side view.

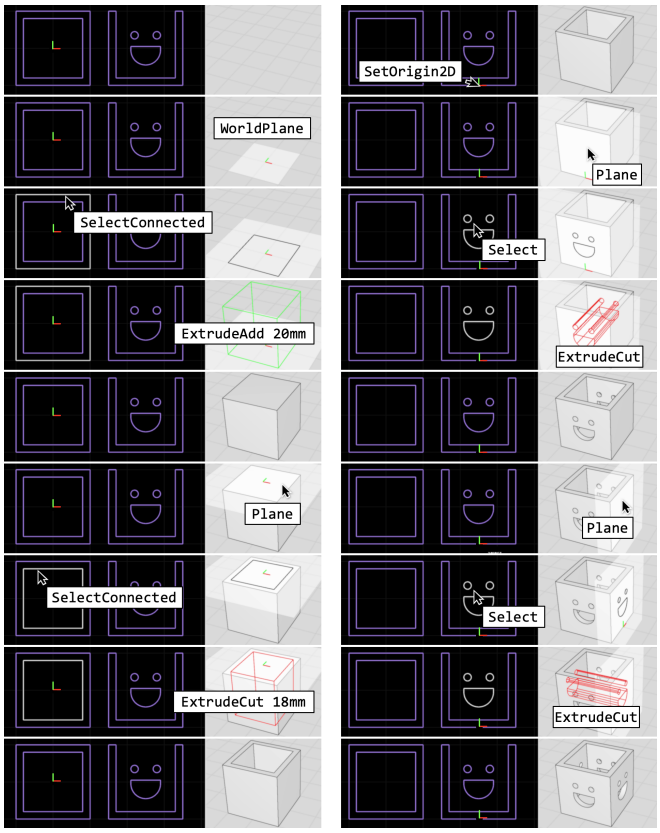

Fig. 3: Annotated sequence of screenshots of a user modeling *Happy Box* in Conversation, starting from a single drawing of the box's top and side views (shown in purple). Reading column by column, the user first chooses the world top plane to be the current feature plane, and selects the outer contour of the box's top view (shown in white). They then extrude the box (add preview shown in green wireframe). They then select the top of the box as the current feature plane, select the inner contour of the box's top view, and cut down into the box (cut preview shown in red wireframe). The user then moves the 2D origin to the middle of the bottom line in the side view–the use of snaps makes this precise; see Section III-C–and cut a happy smiley face through the box's sides. These steps are then repeated using another side of the box as the current feature plane.

We dub this overall 2D → 3D workflow "conversational" as a nod to conversational programming, a mode on some CNC (computer-controlled) machine tools wherein the machinist interactively specifies and executes cuts one at a time in "conversation" with the machine, rather than all at once in a large G-code file generated by a computer program. In Conversation, the user adds and subtracts 3D geometry one feature at a time, but does not build up a large, editable parametric history like they would in SolidWorks. We find Conversation's workflow, much like conversational CNC, to be fast and intuitive for making relatively simple parts, which fits our goal as educators teaching 3D-printed robot design.

C. Unconstrained 2D Drawing Engine

In keeping with our goal of making an accessible CAD program for rapid design, we opt to use an *unconstrained* (*non-parametric*) 2D drawing engine. This means that drawings in Conversation do not include explicit *constraints* (*sketch relations*) common to programs like SolidWorks. Rather, our user creates and transforms *entities* (lines, arcs, circles) in an unconstrained manner aided by various sketch tools (`Translate`, `Rotate`, etc.) and "*snaps*" (`Coincident`, `Concentric`, etc.) See Figure 4 for an example. Finally, we emphasize that snaps are not persistent. Internally, the drawing is always just a straight-forward list of entities; the drawing does not need to "solved."

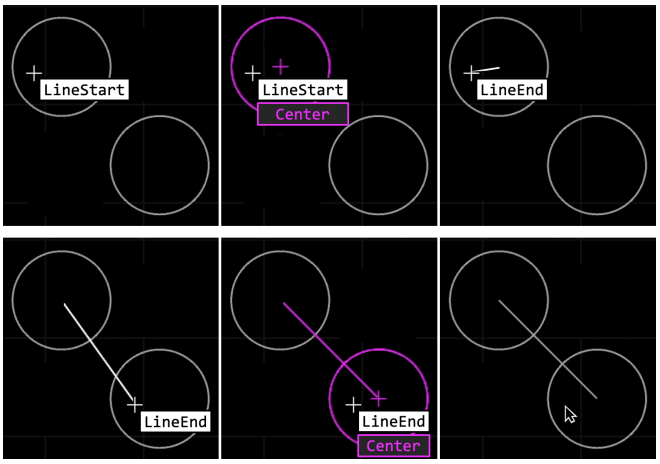

Fig. 4: Annotated sequence of screenshots showing a user sketching with the `Center` snap, which snaps to the center of the nearest circle or arc. When the `Center` snap is active, Conversation highlights the circle or arc nearest to the user's (white, crosshair-style) cursor in magenta, and also draws a second "snapped" cursor in magenta. When the user presses the left mouse button, Conversation will act as if their cursor had been the magenta one. Reading row by row, the user has activated the Line tool (by pressing `L` on the keyboard), activates the `Center` snap by pressing `C`, and then presses the mouse left mouse button to place the line's start point at the center of the top circle. The user then moves their mouse to be near the bottom circle, presses `C`, and clicks again. Note that snaps do not persist. If the user were now to move the line, the circles would stay put.

While mainstream 3D CAD programs like SolidWorks embrace a constrained 2D drawing engine, the unconstrained approach we take in Conversation is actually quite common in 2D CAD software like the open-source 2D CAD software QCAD [14] the proprietary waterjet software OMAX LAYOUT [15], and most notably Autodesk's AutoCAD, which was first released in 1982 and did not add parametric sketching until 2010 [16], [17]. Our use of an unconstrained drawing engine is certainly not novel, though our specific visualization of snaps using tweening and multiple crosshairs (see Section VI) is at least very uncommon.

For Conversation, we opt for the unconstrained approach because it simplifies the process of creating new drawings. There is simply less work for students to do; they only have to draw entities; they do not have to think about setting up compatible constraints. Additionally, students cannot inadvertently add conflicting constraints, so we avoid this common source of friction. A final win is that Conversation's drawing engine is *fast*. For reference, Conversation can render and edit drawings of $100k+$ entities at real-time rates on an M1 MacBook Pro. This is due to Conversation's lack of constraint solving, our data-oriented design of its entity struct (see section VIII-A), and our use of a sparse spatial grid data structure [18] to accelerate queries for nearby points.

We conclude by noting that unconstrained sketching itself also makes an important tradeoff: while it may be less cumbersome to make new drawings, it can be more cumbersome to edit them. This is because rather than simply changing a single dimension and having changes propagate, the user must manually move (or delete and redraw) all affected entities. We believe this tradeoff aligns with Conversation's goal of being a tool for rapid design for relatively simple 3D-printed robots, and additionally, in Section V-C show how to partially alleviate the editing burden.

D. Mesh-Based 3D Geometry Kernel

Conversation uses a *triangle mesh*-based 3D geometry kernel, specifically the fantastic open-source Manifold library [19], which can perform fast accurate boolean operations on triangle meshes. This means that the part displayed in Conversation's Mesh pane is literally a triangle mesh.

We note that using a triangle mesh to represent 3D geometry is a highly nonstandard design decision in the world of mechanical CAD software. Essentially all mainstream mechanical CAD software uses some sort of spline surface like NURBS. However, in digital sculpting software like ZBrush [20] and box modeling software like Blender [21], a mesh representation is the typical one. We chose to use a mesh-based 3D geometry kernel primarily because it made it easier to get a minimum viable product of Conversation working. We avoided having to work with NURBS (spline surfaces) at all, noting that even programs that use a NURBS-based kernel will still use triangle meshes to render. Using Manifold was very much a pragmatic decision that we have "managed to get away with," enabling us to get an otherwise "from scratch" CAD program up and running quickly. Still, our decision comes with a variety of pros and cons.

The main advantage of Conversation's mesh-based kernel is that it works seamlessly with organic, non-mechanical meshes, allowing users to add mechanical features to triangle meshes made in other programs such as the turtle's shell in Figure 5, which was initially box modeled in Blender. Doing operations like this in SolidWorks tends to be riddled with pitfalls. A minor win is that the user does not have to discretize their part into a triangle mesh in order to save to STL; in Conversation, the part already is a triangle mesh. This sidesteps confusing settings in STL export dialogues.

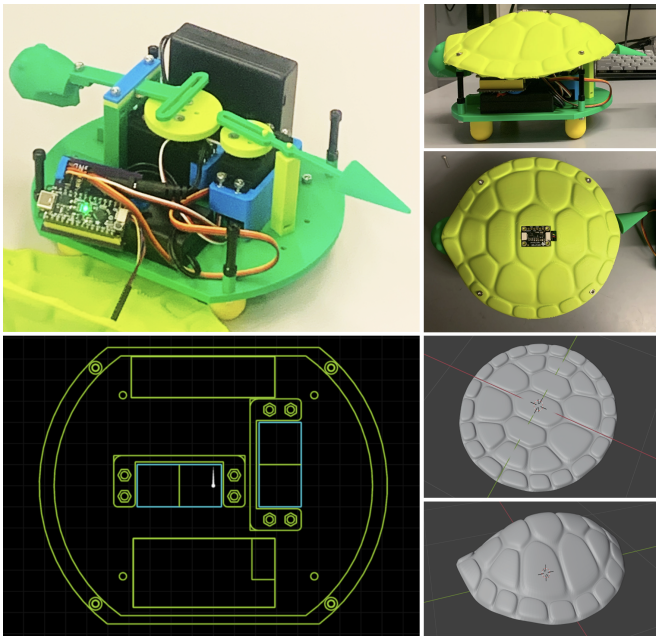

Fig. 5: *Terry the Turtle* by Anonymous Students, course midterm project. A team of two students independently designed and fabricated this servo-driven robotic toy turtle capable of hiding its head and wagging its tail in response to light. Scotch yoke mechanisms are used to move the head and tail. All 3D-printed parts were modeled by the students entirely in Conversation, except for the organic meshes of the turtle's head, which the students found online, and the turtle's shell, which the students initially box-modeled in Blender. Note that Conversation's mesh-based geometry kernel makes it easy to add mechanical features to organic sculpts. In this case, the students used Conversation to add precise holes for attaching standoffs and a light sensor to the shell.

The main disadvantages of using a mesh-based kernel seem to be accuracy and speed. We certainly lose some accuracy by "discretizing early and often," but this does not seem to be a problem in practice. Also, the Manifold library has trouble dealing with exactly aligned surfaces, and can produce "shards" and other artifacts [22]. While technically "correct," such results are inappropriate for a CAD program. However, we expect issues like these to improve as Manifold continues to be developed, and regardless these issues were not a major impediment to students in our demo deployment. This is likely due in part to the fact that the slicer we used (Bambu Studio) tends to silently ignore thin geometry.

Concerning speed, accurately triangle meshing curved surfaces requires huge numbers of triangles. For mechanical parts made with revolves and fillets, this fact slows down fundamental operations like ray casting–which we use to pick a feature plane when the user clicks on the mesh. However, in our demo deployment we found Conversation's performance acceptable, and furthermore there are ways to speed up ray casting that we can take in the future if needed [23].

IV. Automatic Registration Algorithm

Conversation's core workflow requires the users to frequently place 2D drawings in 3D space. This can be done by clicking on the 3D part to choose a feature plane, and manually transforming the 2D axes to register the 2D drawing to the 3D part. These manual adjustments can become cumbersome for more complex parts, especially when the user has to repeatedly switch between different portions of the 2D drawing. This burden motivates the development of an automatic registration algorithm that infers the user's desired 2D drawing transform when they select a new feature plane. We call this feature `SmartOrigin`. For real-time footage of the algorithm running, please see our video; screenshots are shown in Figure 7. For a preliminary user study demonstrating this algorithm's effectiveness, see Section VII.

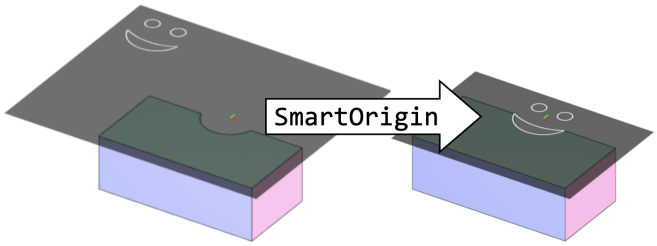

Fig. 6: Here the user has chosen the top face of the 3D part to be the feature plane. An automatic registration algorithm should be able to detect that the top arc of the 2D drawing's smiley face perfectly matches the curved face of the 3D part, and align the drawing accordingly.

We cast automatic registration as an optimization problem by defining a scoring function that measures how well the 2D drawing aligns with a projection of the 3D model onto the feature plane. We seek an optimal registration transform

$$\boldsymbol{R}^* = \arg\max_{\boldsymbol{R}} S(\boldsymbol{R}; \boldsymbol{D}, \boldsymbol{M}, \boldsymbol{P}) \tag{1}$$

that maximizes the scoring function S, where D is the 2D drawing, M is the 3D mesh, and P is the feature plane.

The scoring function S mathematically formalizes the intuition from Figure 6. We calculate S by summing up the lengths of overlapping entities in the transformed drawing and the projection of the 3D part onto the feature plane. We accumulate scores for all line pairs, allowing multiple feature correspondences to reinforce a particular transform.

In our optimization, we encode a registration transform as

$$\boldsymbol{R} = (\boldsymbol{t}, \theta, m), \tag{2}$$

where $\boldsymbol{t} = (t_x, t_y) \in \mathbb{R}^2$ is 2D translation, $\theta \in \mathbb{R}$ is 2D rotation in radians, and $m \in \{0, 1\}$ is a flag indicating whether to mirror mirror the drawing about the x-axis. Following standard computer graphics conventions, we apply these operations as an (optional) mirroring, followed by rotation, followed by translation.

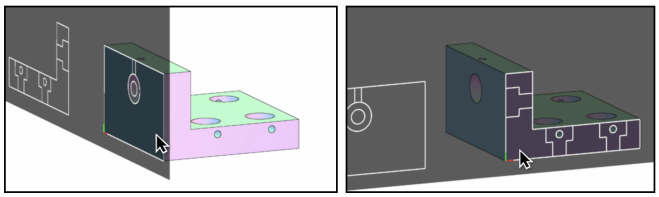

Fig. 7: Screenshots of a user selecting different feature planes with our automatic registration feature `SmartOrigin` turned on. Note how in each case the drawing has been automatically positioned to match the 3D part's geometry as well as possible.

To sample candidate transforms, we identify corresponding line segments between the user's drawing and the projected 3D model. For each pair of lines with the same length, our algorithm will consider the family of four affine transformations that map one line onto the other, allowing for reversal of direction and mirroring.

To account for floating-point precision issues that cause nearly identical transforms to be treated as distinct, we round all transform components to the thousandths place. Transforms that become identical after rounding are merged, with their scores combined. This approach provides sufficient precision for 3D-printed parts and ensures equivalent transforms are recognized as such.

V. Drawing Tools for 3D-Printed Robots

No manufacturing process is perfect, and parts created with FDM 3D-printing tend to have certain defects, also called *artifacts* [24]. Our CAD software tries to make it easy to create designs that account for the inherent limitations of FDM 3D-printing by providing several useful sketch tools.

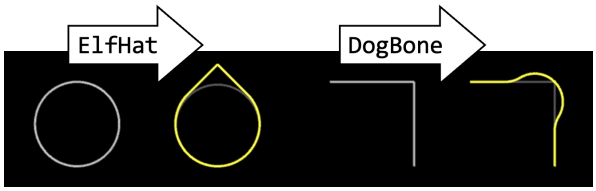

Fig. 8: Conversation's `ElfHat` and `DogBone` commands make it easier to design around the inherent limitations of consumer-grade 3D-printers.

A. Elf Hat

The top surfaces of transverse holes in FDM 3D-printed parts tend to sag down without the inclusion of support material [24]. A convenient alternative shown in Figure 8 is to add a triangular "hat" or "point" to the top of the hole's cross section, rendering the hole self-supporting [25].

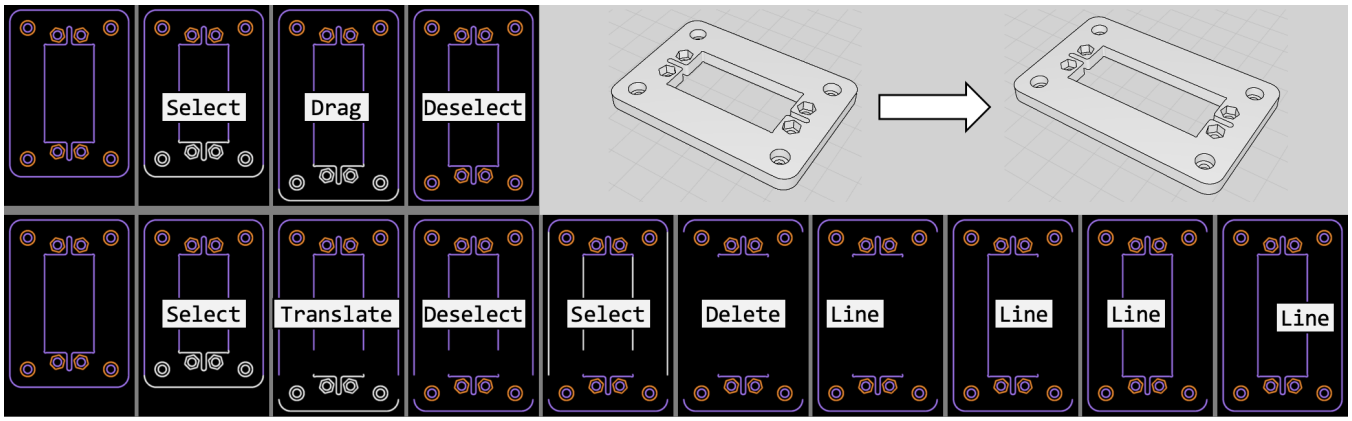

Fig. 9: Annotated screenshots of a user editing the drawing a servo bracket to accommodate a wider servo as shown in 3D in the top-right. The top row shows how this can be accomplished easily with the higher-level `Drag` command, while the bottom row shows how cumbersome it can be to do just with lower-level commands.

B. Dog Bone

Hard corners in FDM 3D-printed parts can get rounded out by the melting action of the printer. This means that polygonal posts may not fit through corresponding holes and off-the-shelf parts like bearings may not seat properly. Related issues arise in processes like CNC routing and waterjetting, and a typical solution shown in Figure 8 is to add small circular cutouts called *dog-bone fillets* at the corners of polygonal holes [26].

C. Drag

The relative imprecision of consumer-grade FDM 3D-printing means that students often need to slightly adjust their designs and reprint. Unfortunately, Conversation's direct sketching engine can make such edits tedious. To simplify the process of making routine edits, we provide a sketch command called `Drag`–similar to AutoCad's `Stretch` [27]–shown in Figure 9.

VI. Tweening

The use of *tweening* (*easing*, *animation*) can make user interfaces easier to understand and more engaging [28], [29]. We *tween* numerous elements of Conversation, including the snapping of all draw commands, the position and orientation of the feature plane, and the previewing and application of all mesh commands including extrusions and revolves. All tweening is done with the simple update

$$a \leftarrow a + 0.1 * (b - a), \tag{3}$$

which each frame moves some quantity a a fixed fraction of the way to the target b. This update goes by numerous names including exponential decay and "lerp smoothing" [30].

In lecture, the tweening of sketch commands helps students understand what we are clicking on, and we also discovered that by mashing `Undo` and `Redo` we could overview the sequence of features used to create a part.

Conversation uses tweening extensively. Please see Figures 10 to 13 and our video for examples.

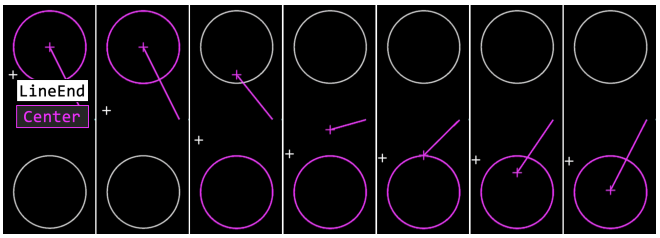

Fig. 10: Conversation tweens the previews of all draw commands. Here, the preview of the `Line` command is tweened as the user moves their mouse down and the end of the line snaps from the center of top circle to bottom one.

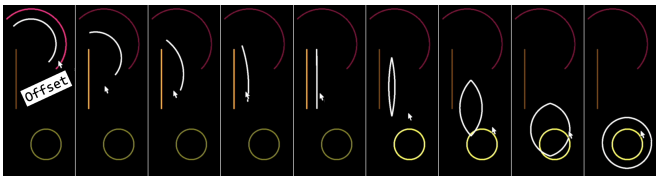

Fig. 11: With the `Offset` command active, the user moves the mouse from the red arc to the orange line, and then from the orange line to the yellow circle. Conversation tweens the preview of the resulting shape, shown in white.

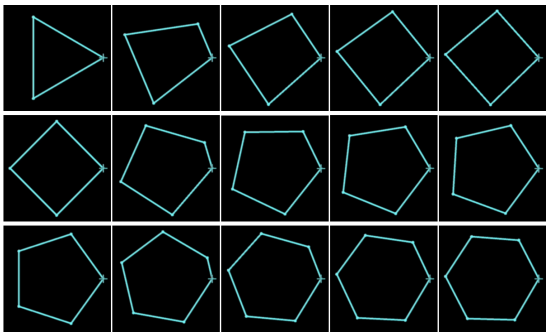

Fig. 12: As the user increments the `Polygon` command's number of sides with the side-length fixed, Conversation tweens the preview of the resulting shape.

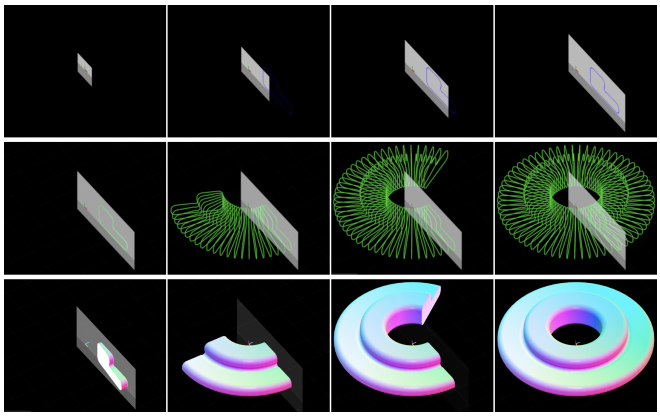

Fig. 13: Here we show all the different types of tweening involved to help visualize a single use of the `Revolve` command. Reading row by row, the top row tweens the size of feature plane to accommodate the selected drawing entities, the middle row tweens the preview of the revolve, and the final row tweens the application of the revolve using clip planes [31] and also tweens the opacity of the feature plane.

Adding tweening to Conversation was generally quite straight-forward, and we find it notable that no mainstream CAD program with which we are familiar uses tweening even close to as much as we do. We hypothesize that this is because mainstream CAD programs target professionals, and animations may be perceived as unserious. Because we focus specifically on students and hobbyists, we are free to add tweening throughout Conversation.

VII. Use Case: College-Level Robotics and Fabrication Elective

We used Conversation as the primary mechanical CAD software in our semester-long, project-based elective *Robotics and Digital Fabrication*, which we taught in the computer science department at a small liberal arts college. The vast majority of the 23 students in the course had no prior CAD or engineering experience, yet they successfully used Conversation to independently design and fabricate custom robots. We include images of some of their creations in Figures 5 and 14; please see our video for more.

A full-fledged user study of Conversation was out of scope for this paper, but we can glean some initial insights from student course evaluations. Students on average rated the course's contribution to their education as $6.0/7.0$, significantly above the average other STEM classes taught the same semester ($5.5/7.0$). Students rated the lab part of the class–which used Conversation most weeks–as $7.0/7.0$. Additionally, 21 students provided freeform written feedback in their course evaluations; all feedback specifically addressing Conversation was positive. In summary, Conversation works and is an effective way to teach 3D-printed robot design to students without prior CAD experience.

To validate our automatic registration algorithm, we conducted a preliminary user study at the end of the semester. Eight student participants were given a modeling task involving recreating a mechanical part from a technical drawing. Each participant performed this task twice: once with access to Conversation's `SmartOrigin` feature and once without. The `SmartOrigin` feature demonstrated significant impact on user productivity. Every participant rated the modeling task as easier when the `SmartOrigin` tool was available. This matched our measurements of their completion times, which averaged 2.1 times faster working with the tool than without. We also had participants report task difficulty of task on a 5-point Likert scale. Without `SmartOrigin` available, 6 participants rated the task as 3-Neutral and 2 participants rated it 4-Difficult. With `SmartOrigin` enabled, 2 participants rated the task 1-Very Easy and 6 students rated it 2-Easy. These results indicate that Conversation's automatic registration algorithm can significantly ease the process of modeling in a split 2D/3D CAD program.

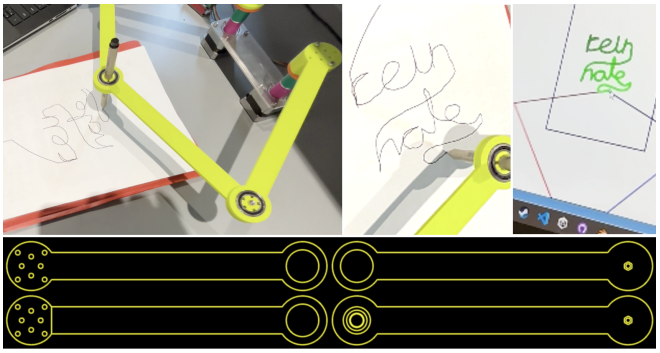

Fig. 14: *Five-Bar Linkage Arm* by Anonymous Students; course final project. A team of two students independently designed, fabricated, and did the trajectory planning for this stepper-driven robotic arm capable of writing their names ("rein" and "nate"). All yellow 3D-printed parts were designed by the students in Conversation; the stepper motor assembly was designed by the course instructor, also in Conversation. Note how Conversation's unconstrained sketching engine makes it easy to reuse 2D geometry. In this case, the outline and mounting holes have been reused across links.

VIII. Implementation Details

A. Entity Implementation

Some of the speed of Conversation's 2D drawing engine is attributable to the simple, data-oriented design of its internal `Entity` struct. While very simple, the approach we take is extremely different than a typical OOP approach [32], so we include it here. The core idea, inspired by an article by Ryan Fleury [33], is to store each entity as three endpoints, like this: `struct Entity { vec2 start, end, center; };` No type field is necessary. A circle has `end` set exactly to `start`, a line has `center.x` set to `INFINITY`, and an arc is neither a circle nor a line. We find that representing lines, circles, and arcs in this unified way enables us to compress CPU and GPU code-paths. For example, we draw all entities using the same shader program.

B. Codebase

Conversation is developed largely from scratch in C99. Currently, the only external libraries we use are *GLFW* for cross-platform windowing, OpenGL for cross-platform rendering, and Manifold for mesh boolean operations. Excluding external libraries, the source code is about 12.5k lines, and compiles in approximately 1 second.

IX. Discussion and Future Work

This is an incredibly exciting time for STEAM education. Cheap, consumer-grade 3D-printers actually work now! As an anecdote, the authors ran their semester-long course using a total of six 3D-printers, and only had to do minor repairs twice. Astoundingly, in many ways, our fabrication machines have outpaced our design tools. So, what should future educational CAD software look like?

In this paper, we presented our vision for a new, accessible CAD program designed to enable students to more easily design 3D-printed robots. We have validated that our program is sufficiently functional to be used as the primary CAD program in a college-level robotics course, but a full-fledged quantitative evaluation of the program's efficacy is still called for. Do students build competency at Conversation faster than SolidWorks?–How much faster? And at what point do parts become too complex for Conversation to handle? Additionally, we are curious whether Conversation can serve as a stepping stone to mainstream CAD. That is to say, does learned Conversation first enable students to more quickly learn mainstreams programs like SolidWorks?

Looking forward, we are also curious about more artistic tools, which would enable students to blend engineering and art even further than they did in our class. Conversation already demonstrates exciting potential for editing organic meshes, and we are curious how far we can take this ability by developing new algorithms and interaction modalities. **How can we make it easier for all students to bring their own creative ideas to 3D-printed life?**

Fig. 15: *Orbie* by Anonymous Students; summer research. A team of three students modeled the physical robot in Conversation, and sculpted, rigged, and rendered the virtual character in ZBrush, Blender, and Unity. Their robot was used in the mixed reality robotic art piece *Soft Contact* at ICRA 2025.

References

[1] E. Susilo, J. Liu, Y. A. Rayo, A. M. Peck, J. Montenegro, M. Gonyea, and P. Valdastri, "Stormlab for stem education: An affordable modular robotic kit for integrated science, technology, engineering, and math education," *IEEE Robotics & Automation Magazine*, vol. 23, no. 2, pp. 47–55, 2016.

[2] R. Wade, N. P. Garland, and G. Underwood, "Challenges of 3d printing for home users," *International Conference on Engineering and Product Design Education*, 2017.

[3] N. C. Borg, "Creating multimedia presentations with kid pix studio," *University of Malta. Faculty of Education*, 1997.

[4] S. Coros, B. Thomaszewski, G. Noris, S. Sueda, M. Forberg, R. W. Sumner, W. Matusik, and B. Bickel, "Computational design of mechanical characters," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, pp. 1–12, 2013.

[5] Y. Yuan, C. Zheng, and S. Coros, "Computational design of transformables," in *Computer Graphics Forum*, vol. 37, no. 8. Wiley Online Library, 2018, pp. 103–113.

[6] V. Megaro, B. Thomaszewski, M. Nitti, O. Hilliges, M. Gross, and S. Coros, "Interactive design of 3d-printable robotic creatures," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 6, pp. 1–9, 2015.

[7] M. Geilinger, R. Poranne, R. Desai, B. Thomaszewski, and S. Coros, "Skaterbots: Optimization-based design and motion synthesis for robotic creatures with legs and wheels," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 4, pp. 1–12, 2018.

[8] T.-H. J. Wang, J. Zheng, P. Ma, Y. Du, B. Kim, A. Spielberg, J. Tenenbaum, C. Gan, and D. Rus, "Diffusebot: Breeding soft robots with physics-augmented generative diffusion models," *Advances in Neural Information Processing Systems*, vol. 36, pp. 44 398–44 423, 2023.

[9] B. Kim, T.-H. Wang, and D. Rus, "Generative-ai-driven jumping robot design using diffusion models," in *2025 International Conference on Robotics and Automation (ICRA)*, 2025.

[10] J. Micallef, *Beginning design for 3D printing*. Apress, 2015.

[11] AutoCAD Incorporation, "TinkerCAD," https://www.tinkercad.com.

[12] Womp 3D Inc., "Womp," https://www.womp.com.

[13] J. Gohde, "OpenSCAD," https://openscad.org.

[14] RibbonSoft, GmbH, "QCAD," https://www.qcad.org/en/.

[15] OMAX Corporation, "LAYOUT," https://www.omax.com/en/us/intellimax-software.

[16] Autodesk Incorporation, "AutoCAD," https://www.autodesk.com/products/autocad/.

[17] R. Ellis, "A Practical Guide to Parametric Drawing in AutoCAD."

[18] R. Nystrom, *Game programming patterns*. Genever Benning, 2014.

[19] E. Lalish, "Manifold," https://github.com/elalish/manifold.

[20] Maxon Computer GmbH, "ZBrush," https://www.maxon.net/en/zbrush.

[21] The Blender Foundation, "Blender," https://www.blender.org.

[22] "Dealing with extra triangles after boolean comparison..." https://github.com/elalish/manifold/discussions/1165.

[23] "Tutorial 29 - 3D Picking — ogldev.org," https://ogldev.org/www/tutorial29/tutorial29.html.

[24] B. Lab, "How to print overhangs," https://wiki.bambulab.com/en/filament-acc/filament/print-quality/overhang.

[25] Slant 3D, "Design better holes," 2023, https://www.youtube.com/watch?v=Bd7Yyn61XWQ.

[26] Imran Anees, "Everything you need to know about dogbone fillets," 2020, https://www.bricsys.com/en-eu/blog/everything-you-need-to-know-about-dogbone-fillets.

[27] Autodesk Inc., "The stretch and lengthen commands," 2024.

[28] R. Fleury, "UI, Part 1: The Interaction Medium," 2022, https://www.rfleury.com/p/ui-part-1-the-interaction-medium.

[29] M. Jonasson and P. Purho, "Juice it or lose it," https://www.youtube.com/watch?v=Fy0aCDmgnxg.

[30] F. Holmér, "Lerp smoothing is broken," 2024, https://www.youtube.com/watch?v=LSNQuFEDOyQ.

[31] Andrei Gradinari, "Advanced clipping techniques," https://glbook.gamedev.net/GLBOOK/glbook.gamedev.net/moglgp/advclip.html.

[32] A. Mustun, "qcad," 2025, https://github.com/qcad/qcad.

[33] R. Fleury, "The Codepath Combinatoric Explosion," 2023, https://www.rfleury.com/p/the-codepath-combinatoric-explosion.