Computational Design and Single-Wire Sensing of 3D Printed Objects with Integrated Capacitive Touchpoints

S. Sandra Bae*
sandrabae@cs.arizona.edu
ATLAS Institute
University of Colorado
Boulder, CO, USA
& Dept. of Computer Science
University of Arizona
Tucson, AZ, USA

Takanori Fujiwara*
tfujiwara@cs.arizona.edu
Dept. of Science and Technology
Linköping University
Linköping, Sweden
& Dept. of Computer Science
University of Arizona
Tucson, AZ, USA

Ellen Yi-Luen Do
ellen.do@colorado.edu
ATLAS Institute
University of Colorado
Boulder, CO, USA

Danielle Albers Szafir
danielle.szafir@cs.unc.edu
Dept. of Computer Science
University of North Carolina
Chapel Hill, NC, USA

Michael L. Rivera
mrivera@colorado.edu
ATLAS Institute & Dept. of Computer Science
University of Colorado
Boulder, CO, USA

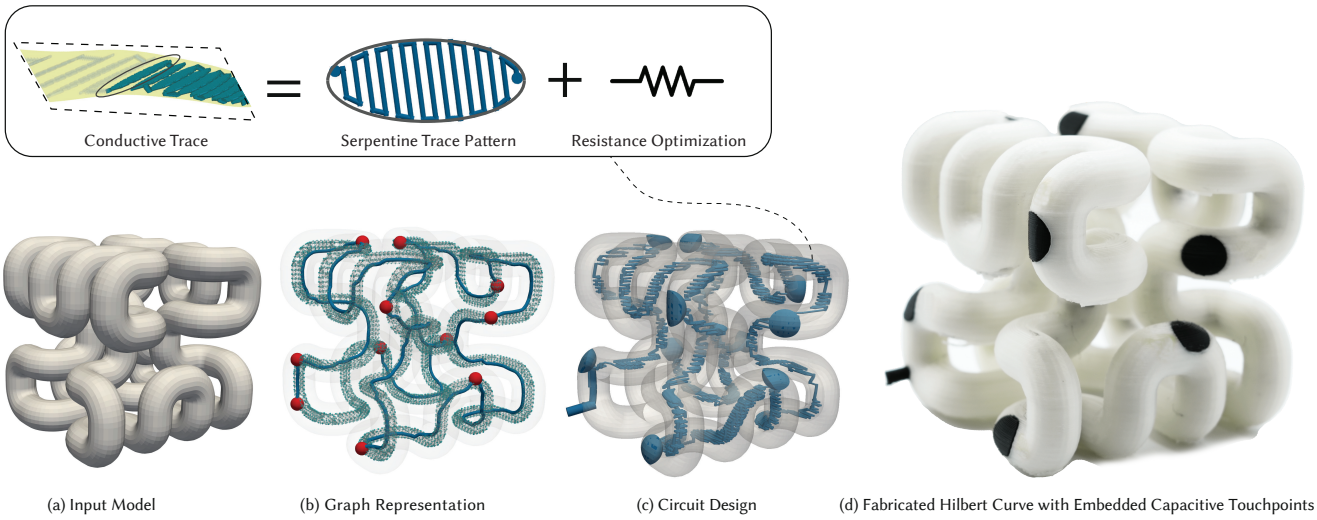


Figure 1: High-level overview of our computational design pipeline: (a) Our pipeline begins with an input model and the user selects different areas on the model's surface to turn into touchpoints; (b) computes a graph-based path to serially connect the touchpoints; (c) generates an internal circuit design to embed capacitive sensors inside the object; and (d) fabricates the object and internal circuit using multi-material 3D printing to be used as a sensing interface with only 1 or 2 wire connections.

*These authors contributed equally to this work and are listed alphabetically.

Authors' addresses: S. Sandra Bae, sandrabae@cs.arizona.edu, ATLAS Institute, University of Colorado, Boulder, CO, USA , & Dept. of Computer Science and University of Arizona, Tucson, AZ, USA; Takanori Fujiwara, tfujiwara@cs.arizona.edu, Dept. of Science and Technology, Linköping University, Linköping, Sweden , & Dept. of Computer Science and University of Arizona, Tucson, AZ, USA; Ellen Yi-Luen Do, ellen.do@colorado.edu, ATLAS Institute, University of Colorado, Boulder, CO, USA; Danielle Albers Szafir, danielle.szafir@cs.unc.edu, Dept. of Computer Science, University of North Carolina, Chapel Hill, NC, USA; Michael L. Rivera, mrivera@colorado.edu, ATLAS Institute & Dept. of Computer Science, University of Colorado, Boulder, CO, USA.

ABSTRACT

Producing interactive 3D printed objects currently requires laborious 3D design and post-instrumentation with off-the-shelf electronics. Multi-material 3D printing using conductive PLA presents opportunities to mitigate these challenges. We present a computational design pipeline that embeds multiple capacitive touchpoints into any 3D model that has a closed mesh without self-intersection.

0730-0301/2025/10-ART
https://doi.org/10.1145/3745778.3766650

With our pipeline, users define touchpoints on the 3D object's surface to indicate interactive regions. Our pipeline then automatically generates a conductive path to connect the touch regions. This path is optimized to output unique resistor-capacitor delays when each region is touched, resulting in all regions being able to be sensed through a double-wire or single-wire connection. We illustrate our approach's utility with five computational and sensing performance evaluations (achieving 93.35% mean accuracy for single-wire) and six application examples. Our sensing technique supports existing uses (e.g., prototyping) and highlights the growing promise to produce interactive devices entirely with 3D printing.

CCS CONCEPTS

• **Computing methodologies → Shape modeling**; • **Applied computing → Computer-aided design**; • **Hardware → Circuit optimization**; • **Human-centered computing → Interaction devices**.

KEYWORDS

computational design, 3D printing, sensors, capacitive sensing, input devices

ACM Reference Format:

S. Sandra Bae, Takanori Fujiwara, Ellen Yi-Luen Do, Danielle Albers Szafir, and Michael L. Rivera. 2025. Computational Design and Single-Wire Sensing of 3D Printed Objects with Integrated Capacitive Touchpoints. *ACM Trans. Graph.* 1, 1 (October 2025), 19 pages. https://doi.org/10.1145/3745778.3766650

1 INTRODUCTION

Despite recent advances, the overall design and manufacturing process to fabricate interactive 3D printed objects is time-consuming and fragmented. Embedding off-the-shelf electronic components (e.g., sensors [Wang et al. 2020; Zhu et al. 2020a] or LEDs [He et al. 2022; Savage et al. 2014]) into 3D prints is a popular approach, adhering to the traditional design process that separates form (i.e., designing a 3D model) and interactivity into two individual processes. However, this approach introduces two challenges. First, it requires users to design *around* the electronic components and their predefined shapes and dimensions. This constraint makes it difficult to integrate electronics into complex geometries, such as curved or organic shapes or thin-walled structures with limited bounding volume (e.g., sword, robotic tactile sensors [Kohlbrenner et al. 2025]). Second, it requires users to have extensive knowledge spanning electronics, computer-aided design, and fabrication. Each step is compartmentalized to a dedicated software, such that a change requires modifying subsequent steps through extensive trial and error. Our work is motivated by the following question: *how can we effectively streamline the process of manufacturing interactive 3D printed objects?*

Multi-material printing can help address these challenges while presenting new design opportunities. Namely, it can bridge the two aforementioned processes (i.e., form, interactivity) into one streamlined process. As one example, we can use conductive filaments to 3D print electronics, such as wires and resistors, directly into the target object and minimize post-instrumentation. Minimizing instrumentation can enhance durability [Zhu et al. 2020a], aesthetics [Olberding et al. 2013; Zhu et al. 2020a], and space efficiency [Dahiya et al. 2009] while simplifying (dis)assembly [He et al. 2022; Wen et al. 2025] and reducing costs [Rupavatharam et al. 2023]. This bridging can lead toward the broad vision of 3D printing objects that are fully interactive and ready to be used straight off the printer.

To this end, our primary contribution is a computational design pipeline that leverages multi-material printing to embed multiple capacitive touchpoints into any 3D model that has a closed mesh without self-intersection (Fig. 1). Our approach focuses on abiding by the given geometric constraint of a 3D model as opposed to modifying it. After users select touchpoints and wiring connection point(s) on the model's surface, our pipeline employs a graph-based pathfinding algorithm to serially connect the touchpoints (Fig. 1b) and then uses the resulting path to generate conductive traces (i.e., 3D printed resistors) between each pair of touchpoints through a serpentine trace space-filling algorithm (Fig. 1c). These conductive traces are optimized to achieve electrical resistance across the user-defined touchpoints to exploit a phenomenon called resistor-capacitor (RC) delays. By creating unique RC time delays for all touchpoints, each touchpoint can be capacitively sensed using only a **single-wire** or **double-wire** connection (Fig. 2).

Achieving interactivity with a single-wire is the extreme case of minimal instrumentation. Our secondary contribution is a thorough investigation of how to achieve this extrema and its mathematical and computational boundaries. This approach enables interactivity in 3D printed objects for an extensive range of 3D geometry while even improving overall sensing reliability (cf. Sec. 9.5). Prior works have also leveraged multi-material 3D printing with conductive materials to create interactive objects. However, these objects either still require significant instrumentation (e.g., n wires linked to a microcontroller to enable n touchpoints) [Palma et al. 2024; Pourjafarian et al. 2019; Schmitz et al. 2015, 2019]. Our work highlights how we can fabricate interactive objects irrespective of their complex geometry with minimal instrumentation.

We demonstrate the scalability, computational performance, robustness, accuracy, and applicability of our approach with corresponding technical evaluations. For scalability, our approach can embed 20 touchpoints into a 3D object using a single-wire connection when there is at least 40mm of distance between each pair of touchpoints. This distance can be further reduced to 12mm through parameter adjustments. Our sensing evaluation highlights real-time recognition of 93.35% mean accuracy for the single-wire connection and 89.49% mean accuracy for the double-wire connection by testing with 8 different objects. The higher accuracy of the single-wire connection is further validated by our robustness evaluation. Also, through this robustness evaluation, we further discuss how to better improve the recognition accuracy for both single-wire and double-wire connections. Our six applications—Stanford Bunny, MIDI Drumpads, Hilbert Curve, Chinese Character (Power), Chinese Lion, and Globe—demonstrate our method's flexibility in supporting a range of geometries with varying complexity (Fig. 9).

The source code of our computational design pipeline and supplemental materials can be found at https://github.com/d-rep-lab/3dp-singlewire-sensing. A video demonstration of our sensing technique can be found in the supplemental materials.

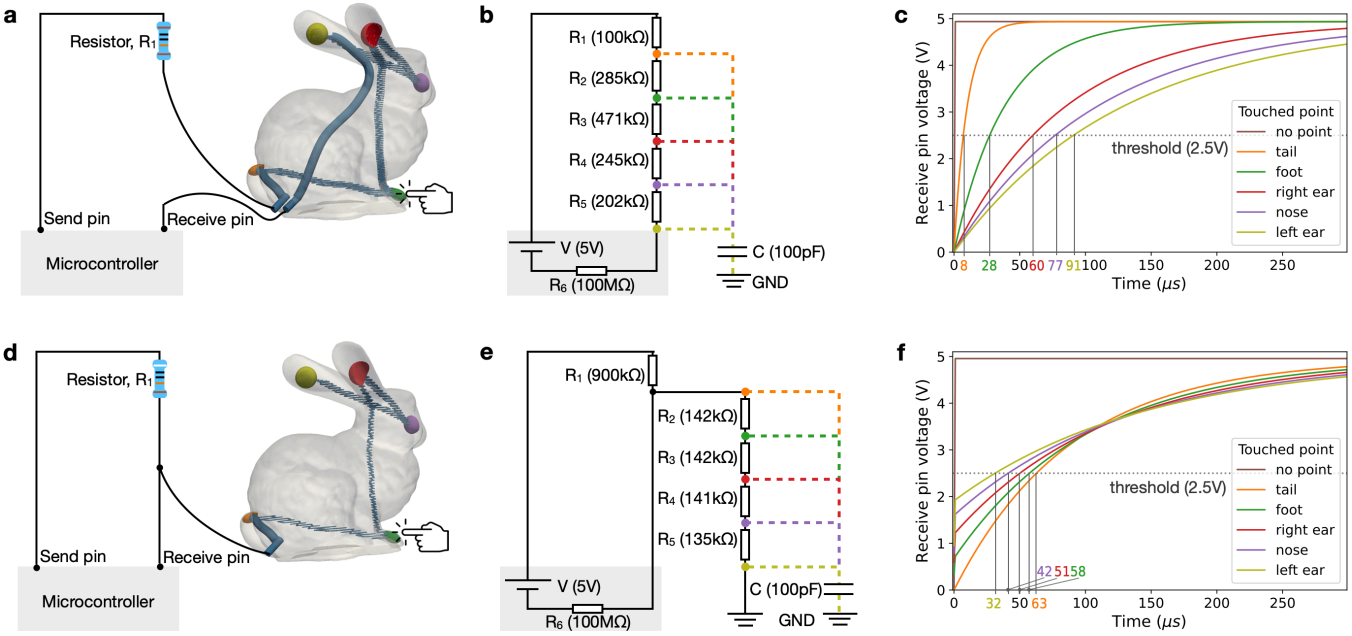


Figure 2: Capacitive sensing for the Stanford Bunny: (a) overall schematic on how the five touchpoints are connected via conductive traces (colored blue) and connected in series to a microcontroller's circuit using *two* wires; (b) the circuit diagram corresponding to (a) with the representative resistance and capacitance measurements. Each colored dashed wire corresponds to a case when a point is touched (e.g., orange: tail, green: foot); and (c) the voltage change measured at the microcontroller's receive pin when a point is touched. (d–f) correspond to (a–c) but the Stanford Bunny is connected in parallel to a microcontroller's circuit using *one* wire.

2 RELATED WORK

Our work builds on prior research that demonstrates ways to computationally design and fabricate interactive objects and capacitive sensors with 3D printing.

2.1 Interactive 3D Prints Using Electronics

Electronic components (e.g., motors, LEDs) offer versatile functionalities and are the backbone of most interactive devices we encounter. The most popular interactivity mechanism for modern 3D printed objects is embedding or attaching off-the-shelf electronic components to 3D printed objects [Ballagas et al. 2018]. However, integrating off-the-shelf electronic components into 3D prints can be challenging. An individual must design an object around these components, ensuring that they can be inserted and wired accordingly post-fabrication [Ballagas et al. 2018; Groeger et al. 2016; Palma et al. 2024; Peng et al. 2015; Savage et al. 2013; Swaminathan et al. 2020].

Computationally designing the location of electronic components can reduce design labor as well as minimize instrumentation. For example, SurfCuit [Umetani and Schmidt 2017] and MorphSensor [Zhu et al. 2020b] allow makers to computationally preview component placement (e.g., resistors and integrated circuits) on the exterior surface of an object and then manually connect them with conductive tape once the object is 3D printed. DefSense [Bächer et al. 2016] computationally designs channels so that wires and sensors can be embedded into a 3D print to enable deformation sensing. Similarly, ModElec [He et al. 2022] further reduces manual labor of wiring by generating 3D-printable conductive traces with A* search algorithm [Hart et al. 1968].

While these approaches help reduce design challenges, their methods still must account for the external physical electronic components. This reliance can influence the design of the object (e.g., prevent a small footprint) and still requires significant wiring and/or assembly, especially to integrate sensors. In contrast, our work aims to fabricate electronics as part of the 3D printing process, contributing to emerging research on 3D printable electronics [Espalin et al. 2014; Flowers et al. 2017; Goh et al. 2021; Macdonald et al. 2014]. In our approach, conductive traces are automatically generated and 3D printed inside an object to act as resistors. This approach supports any 3D models that have a closed mesh without self-intersection, reduces the need for manual assembly, and minimizes the use of additional electronic components.

2.2 3D Printed Capacitive Sensors

Capacitive sensing is a popular technique to capture touch input on devices by capacitively coupling the human body to a conductive material (e.g., an electrode or wire). We refer readers to Grosse-Puppendahl et al. [2017]'s survey highlighting how capacitive sensing has been used in various human-computer interaction (HCI) contexts. Embedding conductive materials—including conductive filament—into 3D printed objects can enable capacitive sensing. These 3D printed objects generally fall under two categories: they

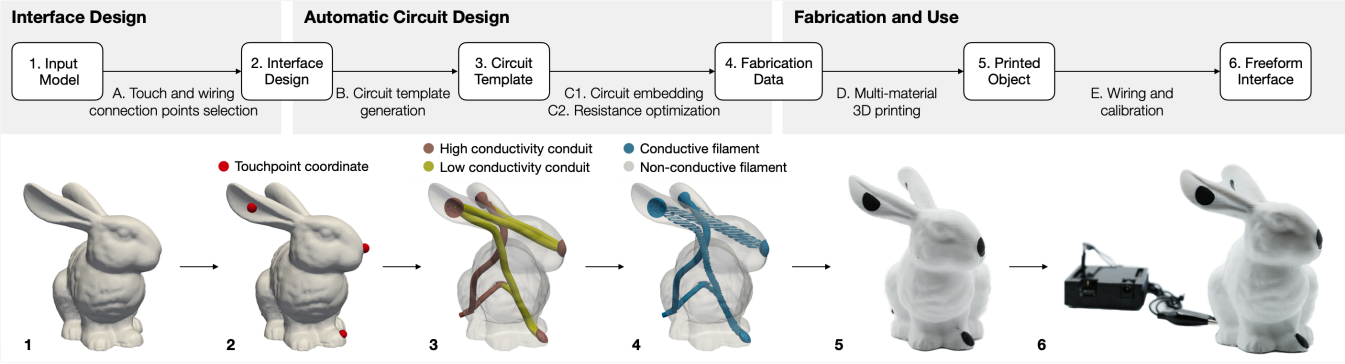


Figure 3: A computational design pipeline to create a freeform interface with embedded capacitive touchpoints. Rounded rectangles represent output data or physical objects. Arrows represent different processes. Each image at the bottom is the corresponding output data (i.e., rounded rectangles) from the computational pipeline.

are either designed with a conductive bottom surface that can be sensed on touchscreen devices [Schmitz et al. 2021, 2017] or with conductive regions that can be wired to an external microcontroller [Bae et al. 2024; Burstyn et al. 2015; Palma et al. 2024; Schmitz et al. 2015].

Our approach aligns with the second category. Several prior works [Alalawi et al. 2023; Bae et al. 2024; Burstyn et al. 2015; Ikematsu and Siio 2018; Kato et al. 2020; Palma et al. 2024; Schmitz et al. 2015, 2019; Takada et al. 2016] demonstrate techniques to generate electrical traces within a 3D model. The resulting 3D printed objects have multiple touchpoints for sensing once they are connected to a microcontroller. However, the majority of these approaches [Burstyn et al. 2015; Ikematsu and Siio 2018; Kato et al. 2020; Palma et al. 2024; Schmitz et al. 2015, 2019; Takada et al. 2016] still require significant instrumentation (e.g., n wires connected to a microcontroller to sense n touchpoints). In contrast, our work focuses on *minimal instrumentation*, requiring only either a single-wire or double-wire connection(s) (Fig. 2). Our previous work [Bae et al. 2024] also explored how to reduce instrumentation but is limited to only network-like geometry (i.e., spheres and cylinders) [Rossignac 2005]. This severe constraint cannot generalize to arbitrary 3D forms. Our current approach lifts this constraint by supporting any closed, non-self-intersecting mesh, regardless of geometric or topological complexity. This capability expands the design space to fabricate complex geometric models with intrinsic interactivity, eliminating the need for post-processing. Furthermore, we deepen the technical foundation of this minimal instrumentation approach by systematically optimizing the circuit design needed to enable accurate single-wire sensing of multiple touchpoints (cf. Sec. 6.3).

3 PRINCIPLE OF CAPACITIVE SENSING WITH RC DELAY

Our computational design pipeline enables embedding multiple capacitive touchpoints within a 3D object such that all touchpoints can be sensed using only a single-wire or double-wire connection (Fig. 2). We provide a short introduction to capacitive sensing using RC delay, which is the key principle underlying our approach.

In a capacitive sensing circuit, when a user touches a conductive element (e.g., electrode), the user's body and the element become capacitively coupled [Grosse-Puppendahl et al. 2017]. This coupling induces an *RC delay* in the sensing circuit. RC delay is the time required to charge a capacitor in a circuit through a particular amount of electrical resistance. Increasing the resistance in a circuit will generally increase the amount of time needed to charge the capacitor, thereby creating a larger RC delay. If each conductive element in a circuit needs a different amount of time to charge when touched, we can infer what is being touched by measuring the time needed to reach a predefined voltage threshold (e.g., 2.5V) on a microcontroller. In our pipeline, the electrical resistance for each conductive touchpoint is optimized to achieve a different RC delay by varying the length of the conductive trace between each pair of touchpoints.

Fig. 2 illustrates this sensing principle with the Stanford Bunny as our freeform interface. As shown in Fig. 2c, 0 µs indicates the baseline in which no touchpoints are touched. Touching the bunny's tail requires 8 µs to reach the voltage threshold, while its foot requires 28 µs. Using this approach, we can detect multiple capacitive touchpoints by connecting the 3D printed object to a microcontroller with either *two* (Fig. 2a–c) or *one* wire (Fig. 2d–f). A single-wire connection results in a parallel circuit, while the double-wire connection results in a series circuit. The difference between these two circuit configurations results in different possible ranges of RC delays (Fig. 2c vs. Fig. 2f).

4 COMPUTATIONAL DESIGN PIPELINE OVERVIEW

The main objective of our computational pipeline is to embed multiple capacitive touchpoints into a freeform model. Our pipeline can work for any 3D models that have a closed mesh without self-intersection. To achieve this goal, our computational pipeline (Fig. 3) is divided into three stages: interface design (Sec. 5), automatic circuit design (Sec. 6), and fabrication and use (Sec. 7).

Interface Design. In this first stage, the designer prepares a freeform model by either 3D modeling with a CAD software or uploading an existing 3D model. Afterward, the designer (1) selects where the

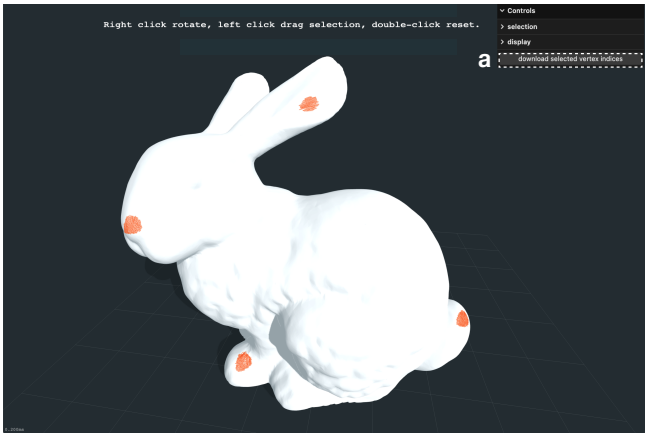


Figure 4: 3D model of the Stanford Bunny with five selected touchpoints (foot, nose, left ear, right ear, and tail) and two wiring connection points. The right ear and the two wiring points are hidden from the viewpoint. The orange meshes indicate the user's lassoed selections. The dashed lines in (a) show where users can download the selected mesh coordinates.

touchpoints will be on the 3D geometry's surface and (2) chooses the connection points (i.e., one or two) that will connect the 3D printed object to a microcontroller (see Fig. 2a,d).

Automatic Circuit Design. After selecting the touchpoints and wiring connection point(s) on the freeform model, the second stage uses our computational algorithms to generate the necessary geometry to route conductive traces throughout the freeform model. This step consists of two stages. First, our graph-based pathfinding algorithm computes a path to serially connect all the touch and wiring points. Next, to produce sufficient resistance between each touchpoint for the RC delay, our space-filling algorithm draws long, thin conductive traces within the path.

Fabrication and Use. At the final stage, the designer uses the fabrication data from the second stage to print the model. The 3D printed object is connected to a microcontroller with either one or two wires. After calibrating all touchpoints to sense touches, the freeform interface is ready for use.

In the following sections, we use the Stanford Bunny with a double-wire connection (Fig. 2a) to illustrate the computational design pipeline. The double-wire connection serves as the foundation to understand how we can implement a single-wire design (Fig. 2d).

5 INTERFACE DESIGN

The designer manually designs a freeform model or uses an existing model, and selects the coordinates of the touchpoints and wiring connection points on the 3D model's surface. Our pipeline allows a designer to use any software of their choice (e.g., Fusion360, Rhino) to generate the coordinates. We also provide a web-based user interface (UI) (Fig. 4) to help facilitate this step. To select the touchpoints and wiring connection points, the web-based UI requires the following steps.

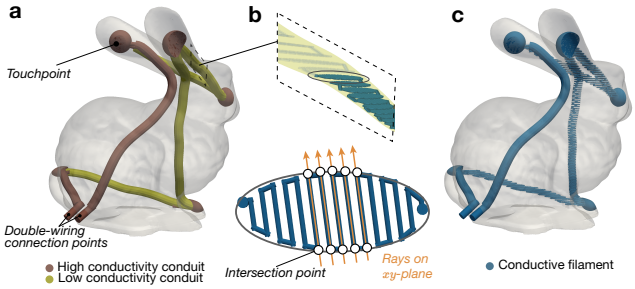


Figure 5: Automatic circuit design. (a) Our automatic circuit design first generates a circuit template, which outlines how we will embed the touchpoints and conductive traces inside a freeform interface. (b) During circuit embedding, serpentine trace patterns are generated inside low conductivity conduits by using a space-filling algorithm. (c) The output is fabrication data (STL files), which will be used for multi-material printing.

Upload STL File. The designer uploads the STL file of a freeform model to the web-based UI. The uploaded file is then rendered as a 3D model for the designer to pan, rotate, and view.

Select Touchpoints and Wiring Connection Points. After viewing the model, the designer can freely lasso different areas on the model's surface to indicate where the touchpoints and wiring connection points would be placed. Fig. 4 shows an example where the user has converted the Stanford Bunny's nose, foot, ears, and tail as touchpoints.

Export Coordinates. Once the point selection has been finalized, the designer can export the points' coordinates (Fig. 4a). The centroids of these coordinates will be used to generate touchpoints and wiring connection points on the surface of the freeform model.

6 AUTOMATIC CIRCUIT DESIGN

The objective of the automatic circuit design stage is to generate the appropriate internal circuit design that will be embedded in the freeform model. This stage is fully automatic and does not require any active involvement from the interface designer.

The automatic circuit design is broken into two sub-stages. The first step is to generate a circuit template (Sec. 6.1). A circuit template specifies the geometry of points (touchpoints, wiring points) and conduits (Fig. 5a). This information is used to embed the internal circuit design into the freeform model in the subsequent step. Next, in the circuit embedding step (Sec. 6.2), our algorithm uses the information from the circuit template to draw the conductive traces for 3D printing using a space-filling algorithm (Fig. 5b,c).

For both substages, we use the following terminology: *points* referring to the touchpoints and the wiring connection point(s); *conduits* as the generated pipes within the freeform models' volume.

6.1 Circuit Template Generation

Generating a circuit template requires two sets of information.

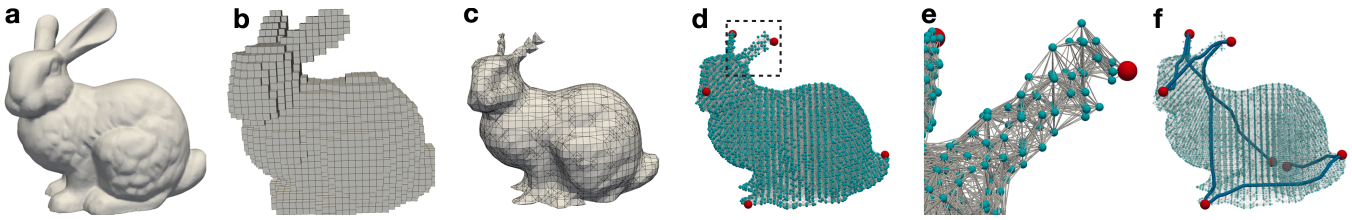


Figure 6: Representations involved in pathfinding: (a) input 3D model for pathfinding; (b) voxel representation of the input model; (c) voxel representation after trimming voxels close to the model's surface; (d) graph representation of the trimmed voxel representation, where a vertex closest to each touchpoint is colored red; (e) a close-up look of the graph representation; and (f) identified paths using Dijkstra's pathfinding algorithm. Except for (e), all figures share the same camera position and angle. For presentation purposes, the voxel and graph representations have lower resolution (i.e., smaller numbers of voxels and vertices) than implemented.

Preparing Point Geometry. The first step of the circuit template generation is to prepare the geometry of the points by using the coordinates of touchpoints and wiring connection points (cf. Sec. 5).

Touchpoints. We ensure that the geometry of touchpoints is bounded within the freeform model's volume with two steps. The first step involves generating a 3D geometry at the centroid of each touchpoint's coordinate. We default to a sphere with a 12mm diameter, but other sizes and types of 3D geometries can also be considered depending on the designer's needs. We then volumetrically clip the sphere based on its intersection with the freeform model's surface.

Wiring Connection Points. For the wiring connection point(s), we generate a cylinder that intentionally extrudes beyond the model's surface (Fig. 5a). A cylindrical design allows external wires (e.g., alligator clips) to easily connect to the freeform model at the fabrication and use stage. To achieve this cylindrical design, we first compute both the centroid and normal of a polygon at the specified coordinates of the wiring connection point(s). We then generate a cylinder with a 4mm diameter and that is 10mm long. These dimensions are arbitrary and sufficient for an alligator clip to grip onto. We use the centroid as the cylinder's center and the normal as the cylinder's axis. The cylinder's height, diameter, and axis direction can also be manually specified.

Routing Conduits with Pathfinding Algorithm. Next, we generate conduits (i.e., 3D pipes). The conduits serve two purposes. First, they will connect the points within the 3D model's bounding volume. Second, they will house the 3D printed conductive traces.

We first need to consider how to connect points. Although the connection of points can be either in series or parallel, we focus only on a series connection. A series connection is simpler to design for an RC circuit as well as controlling the RC delay. In addition, based on our assumption that the freeform model has a closed mesh without self-intersection, we expect the model to have enough volume to construct a series connection within the model.

To make a series connection, we need to decide the order of the points. In the case of the double-wire connection, the first and last points are the wiring connection points. For the single-wire connection, the first point corresponds to the wiring connection point. The designer can manually specify which points are used as the wiring connection point(s). By default, the remaining points (i.e., touchpoints) are connected in the selected order during the interface design stage. For example, the point order for Fig. 5a is the following: first wiring connection point, tail, foot, right ear, nose, left ear, and second wiring connection point.

We then route the conduits to connect the points in the specified order with our graph-based pathfinding algorithm. Our pathfinding algorithm consists of four steps: (1) voxelize the freeform model, (2) trim the voxels that are close to the freeform model's surface, (3) construct a weighted neighbor graph with the remaining voxels, and (4) find the shortest path between each point to connect all points. These steps are visually summarized in Fig. 6. The shortest path is used to ensure there is sufficient space to generate other conduits with the remaining bounding volume. To help distinguish from the terminology used for the circuit template design (i.e., points, conduits), we use *graph*, *vertices*, and *edges* as specific terminology for our pathfinding algorithm.

(1) *Voxelize the freeform model (Fig. 6b).* Voxelization is necessary to prepare a graph that will route the conduits inside the freeform model's bounding volume. We generate a voxel representation of the model using a specified voxel size. By default, our pipeline uses 0.5% of the maximum dimension of any side of the model's bounding box.
(2) *Trim the voxels close to the model surface (Fig. 6c).* If a conduit is placed too close to the model surface, it can introduce parasitic capacitance (i.e., unintentional capacitance) during use. Parasitic capacitance is non-ideal as it can influence the overall sensing capability. Thus, we trim the voxels that are too close to the model surface (by default, 3mm from the surface).
(3) *Construct a weighted neighbor graph (Fig. 6d,e).* We generate a weighted neighbor graph from the trimmed voxel representation. We first compute the distance between each voxel and then construct a k-nearest neighbor graph based on the distances ($k = 10$ by default). This process generates a graph consisting of vertices corresponding to the voxels, the edges of neighbor relationships, and the edge weights corresponding to the distances.
(4) *Find the shortest paths (Fig. 6f).* We can find the shortest path between a pair of vertices using a pathfinding algorithm such as Dijkstra's algorithm or A* [Hart et al. 1968]. By default, we

employ Dijkstra's algorithm, but our implementation is flexible to switch to other pathfinding algorithms. We iteratively perform this pathfinding step to find the route that connects all points in series. In parallel, our algorithm aims to avoid overlapping conduits with each other. Since conduits will house the conductive traces, an overlap can change the circuit design. After each iteration, we assign a large penalty for the edge weights (300 mm) that have already been used or are too close to the found path. Although the shortest path is generally preferable to ensure sufficient space, the path between two points may be too short to generate sufficiently large resistance in the circuit embedding step (Sec. 6.2). To resolve such a case, we provide two options. The first option is to randomly permute the connection order of touchpoints until all paths become longer than the designer-specified lengths. The second option is to run our shortest path finding algorithm multiple times. Due to the penalty added in the edge weights, the algorithm can make a path gradually longer.

After the route of the conduits is finalized, the conduits are then rendered as 3D pipes (5mm diameter by default). We chose 5mm as our default to provide enough space to generate the serpentine trace patterns with our nozzle's extrusion width (0.4mm).

6.2 Circuit Embedding

We use the circuit template to generate the freeform model's internal circuit design. 3D printing the circuit requires a combination of conductive and non-conductive materials. As mentioned, points are the touchpoints and wiring connection points are filled with a large amount of conductive filament (100% infill) by default. As a result, points have high conductivity and negligible resistance. In contrast, the generated conduits can have either *high conductivity* or *low conductivity* (Fig. 5a). The conductivity is determined based on the conduit's role in the internal circuit.

Conduits that originate from the wiring connection point(s) have high conductivity (brown links in Fig. 5a). These conduits are meant to act as wires (i.e., negligible resistance). Similar to points, the high conductivity conduits will also be 3D printed with a 100% infill with a conductive filament. In comparison, the conduits between each pair of touchpoints have low conductivity (yellow links in Fig. 5a). The low conductivity conduits act as *resistors*. To leverage RC delay for capacitive sensing, these low conductivity conduits need to achieve high resistance within their limited volume.

We achieve high resistance by drawing a thin, long trace of the conductive filament using a serpentine trace pattern [Soh et al. 2009] inside the low conductivity conduits (Fig. 5b). Due to the resistivity law, a thinner conductive trace will provide lower conductivity and higher resistance. The thickness of the conductive traces varies when drawing the trace on the xy-plane versus along the z-direction. The thickness for the xy-plane can be close to the printer's nozzle extrusion width (e.g., 0.8mm), while the thickness for z-direction should be at least twice the extrusion width (e.g., 1.2mm) to ensure contact with the previously printed layer. The variance in thickness is to account for the printing resolution of common FDM printers.

To support a wide range of geometry, our pipeline needs to be able to handle curved conduits. We designed a space-filling algorithm to draw serpentine trace patterns in these curved conduits. For a given z-coordinate along the xy-plane, we cast multiple rays that are parallel to each other. Each ray finds the intersection points with the conduit and creates line segments by connecting the intersection points. By alternatively connecting one line segment's endpoint and another line segment's start point, we can obtain a serpentine pattern for one layer (Fig. 5b). We repeat this process while gradually increasing (or decreasing) the z-coordinate, resulting in multiple layers of serpentine patterns. We then connect these serpentine patterns with a staircase pattern using vertical lines along the z-direction. We also need to consider how much of a margin should be between each ray as well as between each layer. The margin must be larger than the printer's nozzle extrusion width. Based on the specifications of most FDM printers, we use 1.2mm as the margin for both the ray and layer by default.

After generating all circuit components described above, the internal circuit design is output as STL files for multi-material 3D printing.

6.3 Supporting Cases Using a Single-Wire Connection

The serpentine pattern described in Sec. 6.2 aims to achieve high resistance for the conductive traces in the low conductivity conduits. Our analysis reveals that we can support the double-wire connection as long as *each* low conductivity conduit has sufficiently high resistance (Sec. 6.3.1). However, to support a single-wire connection, there are additional requirements: we need to carefully control the *interplay of all* resistances in the circuit. This additional requirement stems from how the single-wire connection results in a parallel circuit, while the double-wire connection results in a series circuit (see Fig. 2). To handle the differences in the overall circuit configuration, we first discuss the theoretical differences between the double-wire and single-wire connections. We then introduce an optimization method to support multiple capacitive touchpoints with a single-wire connection.

6.3.1 *Single-Wire vs. Double-Wire Connections.*

Double-Wire Connection. When a freeform model is connected to the microcontroller with two wires like Fig. 2a, we only need to ensure each low conductivity conduit has sufficiently large resistance (e.g., 50kΩ, cf. Sec. 9.1). The large resistance allows a microcontroller to have enough buffer to capture the RC delay differences among the touchpoints. We now discuss in detail the reasoning behind this simple requirement.

We assume the circuit shown in Fig. 7a, where R_1 is a resistor connected to a microcontroller and $R_2, \cdots, R_N$ (N: the number of touchpoints) are the resistors embedded into the 3D printed object. The microcontroller has a voltage source with v_{in} and a resistor between its voltage source and receive pin. For convenience, we denote this microcontroller's resistor as R_{N+1}. Also, we denote R_i's resistance as r_i ($i = \{1, \cdots, N+1\}$). In addition, the capacitor formed by touch has capacitance c.

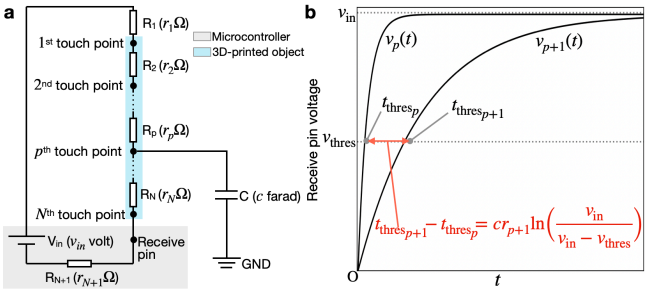


Figure 7: Analysis of a double-wire connection. (a) shows the circuit schematic for the double-wire connection, where the pth touchpoint is selected. (b) compares the voltage changes and time delays induced when pth and $(p+1)$th touchpoints are touched.

When touching the pth touchpoint ($1 \le p \le N$), the voltage change measured at the receive pin can be written as:

$$v_p(t) = v_{\text{in}} \left(1 - \exp\left(-\frac{t r_{\text{all}}}{c r_{\text{till}_p} r_{\text{after}_p}}\right)\right) \frac{r_{N+1} r_{\text{till}_p}^2 r_{\text{after}_p}^2}{r_{\text{all}} \left(\sum_{i=0}^{p} r_i r_{\text{after}_p}\right)^2} \quad (1)$$

where t is the time to charge a capacitor; $r_{\text{all}} = \sum_{j=1}^{N+1} r_j$; $r_{\text{till}_p} = \sum_{j=1}^{p} r_j$; and $r_{\text{after}_p} = \sum_{j=p+1}^{N+1} r_j$ (see Appendix A for the derivation). To induce a high-impedance state for the receive pin, r_{N+1} is usually very large (e.g., 100MΩ). This value is pre-determined by the microcontroller's manufacturer. When $r_1, \cdots, r_N$ are relatively small (e.g., 100kΩ), we can approximate Eq. 1 as:

$$v_p(t) \approx v_{\text{in}} \left(1 - \exp\left(-\frac{t}{c r_{\text{till}_p}}\right)\right) \quad (2)$$

From Eq. 2, we can approximate the time required to reach a microcontroller's logic threshold voltage, v_{thres} as followed:

$$t_{\text{thres}_p} \approx c r_{\text{till}_p} \ln\left(\frac{v_{\text{in}}}{v_{\text{in}} - v_{\text{thres}}}\right) \quad (3)$$

Fig. 7b summarizes the key theoretical relationships that we derived from the equations above. Using Eq. 3, we can consider that the RC delay only depends on r_{till_p}, the cumulative sum of resistance values involved from the voltage source to a selected touchpoint. We further derive that $t_{\text{thres}_{p+1}} - t_{\text{thres}_p} = c r_{p+1} \ln(v_{\text{in}}/(v_{\text{in}} - v_{\text{thres}}))$. This equation indicates that larger resistance for each of $r_1, \cdots, r_N$, results in larger differences between t_{thres_p} and $t_{\text{thres}_{p+1}}$. Note: $t_{\text{thres}_1} \le t_{\text{thres}_2} \le \cdots \le t_{\text{thres}_N}$ because $c \ge 0$, $r_p \ge 0$, and $\ln(v_{\text{in}}/(v_{\text{in}} - v_{\text{thres}})) \ge 0$.

From the observations above, we can support the double-wire connection by generating a serpentine trace pattern that is as long as possible within a conduit's volume (Sec. 6.2). Thus, we only need to ensure that $t_{\text{thres}_{p+1}} - t_{\text{thres}_p}$ is large enough for a microcontroller to measure (e.g., 200 clock cycles of a microcontroller's CPU). Note that when r_{till_p} is extremely large, t_{thres_p} may be too large to provide reasonable latency for interactivity (e.g., when $t_{\text{thres}_p} >$ 10ms). However, the possibility of this situation is rare. For example, a hypothetical scenario where $t_{\text{thres}_p} >$ 10ms would require

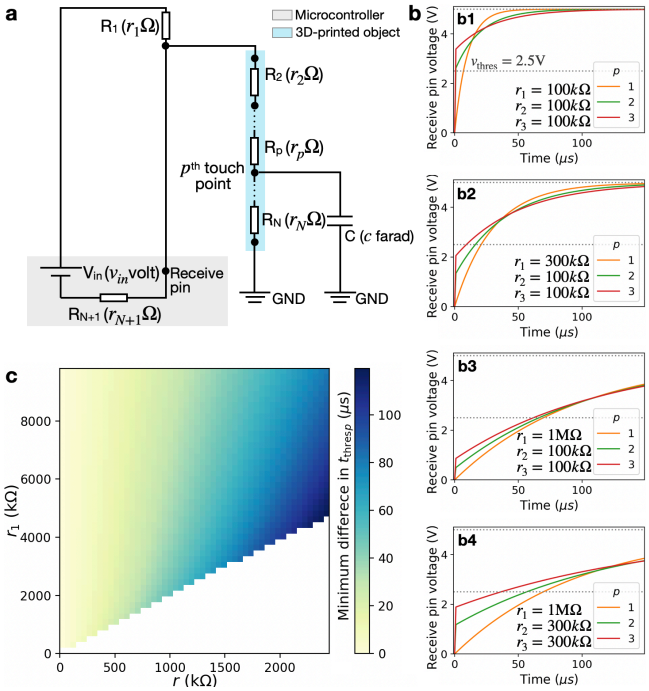


Figure 8: Analysis of a single-wire connection. (a) shows the circuit schematic for the single-wire connection where the pth touchpoint is selected. (b) compares the voltage changes across different variations of three resistance values (r_1, r_2, r_3). (c) shows the minimum differences among $t_{\text{thres}1}, t_{\text{thres}2}, t_{\text{thres}3}$ for the different variations of r_1 and r (note: here r_2=r_3=r). A white grid cell indicates a violation of the hard constraint of r_1. For (b) and (c), we assume a case using the Arduino UNO R4 (i.e., $v_{\text{in}} = 5$V, $v_{\text{thres}} = 2.5$V) and $c = 100$pF.

$r_{\text{till}_p} > 140$MΩ when using the Arduino UNO R4 as the microcontroller (v_{in}=5V, v_{thres}=2.5V) and c =100pF as a representative capacitance [ESD Association 2020]. Generating conductive traces where $r_{\text{till}_p} > 140$MΩ inside a freeform model is almost infeasible.

Single-Wire Connection. For the single-wire connection, we connect the freeform model to a microcontroller as shown in Fig. 8a. This connection makes a branch in the electrical path (i.e., forming a parallel circuit). Consequently, for the single-wire connection, we obtain the measured voltage change at the receive pin as:

$$v_p(t) = v_{\text{in}} \frac{1}{r_1 + r_{N+1}} \left(r_{N+1} - \frac{r_1 r_{N+1}^2}{r_1 r_{\text{till}_p} + r_{N+1} r_{\text{till}_p} - r_1^2} \exp\left(-\frac{t(r_1 + r_{N+1})}{c(r_1 r_{\text{till}_p} + r_{N+1} r_{\text{till}_p} - r_1^2)}\right)\right) \quad (4)$$

Similar to the double-wire connection, we can approximate Eq. 4 as:

$$v_p(t) \approx v_{\text{in}} \left(1 - \frac{r_1}{r_{\text{till}_p}} \exp\left(-\frac{t}{c r_{\text{till}_p}}\right)\right) \quad (5)$$

Then the approximate time required to reach a microcontroller's logic threshold voltage is:

$$t_{\text{thres}_p} \approx c r_{\text{till}_p} \ln\left(\frac{r_1}{r_{\text{till}_p}} \frac{v_{\text{in}}}{v_{\text{in}} - v_{\text{thres}}}\right) \quad (6)$$

We can observe that the difference between Eq. 5–6 and Eq. 2–3 is the coefficient r_1/r_{till_p} (or $r_{\text{till}_p}/r_{\text{r}_1}$). This coefficient introduces greater complexity for the single-wire connection compared to the double-wire connection. The most critical difference from the double-wire connection is the voltage measured at the receive pin at t=0: $v_p(0) = v_{\text{in}}(1 - r_1/r_{\text{till}_p})$. This indicates $v_p(0)$ changes depending on the relationships between r_1 and r_{till_p} (note: $r_{\text{till}_p} = r_1 + \cdots + r_p$). Additionally, the coefficient, r_1/r_{till_p}, influences the slope of the exponential function in Eq. 5. These facts suggest that we need to carefully select r_1 and $\{r_2, \cdots, r_n\}$ to support the single-wire connection. This selection requires two considerations.

First, we have a hard constraint for r_1. To sense a selected touchpoint, we must avoid where $v_p(0) \geq v_{\text{thres}}$ (i.e., $r_1/r_{\text{till}_p} \leq 1 - v_{\text{thres}}/v_{\text{in}}$). For example, Fig. 8-b1 reflects this violation, and a microcontroller would not be able to detect if a touchpoint has been selected for two of the points (i.e., $p = 1$ and $p = 2$). Thus, to ensure all touchpoints can be sensed, we must satisfy $r_1/r_{\text{till}_p} > 1 - v_{\text{thres}}/v_{\text{in}}$ for all touchpoints. If using the Arduino UNO R4, $1 - v_{\text{thres}}/v_{\text{in}} = 0.5$; thus, the hard constraint corresponds to $r_1 > r_2 + \cdots + r_N$. This indicates that r_1 must be greater than the cumulative sum of the resistance inside the freeform model.

Second, to maximize the difference in t_{thres_p} for each touchpoint (e.g., pth vs. $(p+1)$th touchpoint), we need to resolve the complex relationships among r_1 and $r_2, \cdots, r_N$. Fig. 8b shows four variations that use different resistance values for the same example. Among the four variations, Fig. 8-b4 achieves the maximum difference in t_{thres_p} for each touchpoint. To find the configuration with the largest difference in t_{thres_p} for each touchpoint, we introduce a heuristic resistance optimization for the single-wire connection.

6.3.2 Resistance Optimization. Our heuristic resistance optimization for the single-wire connection consists of two steps: (1) identify the appropriate resistance values and (2) adjust the geometry of the serpentine trace patterns. This adjustment will take place during the circuit embedding step described in Sec. 6.2.

Identify Appropriate Resistance Values. The objective of the first step is to optimize r_1 and $r_2, \cdots, r_N$. The goal is to *maximize the minimum difference among the time delays*. This goal ensures the time delay difference for each touchpoint is large enough for a microcontroller to measure. To heuristically achieve this goal, we perform a grid search utilizing a circuit simulator, specifically, Lcapy [Hayes 2022]. To make the search space reasonably small, we consider a case where all resistance values within the freeform model have the same value, i.e., $r_2 = \cdots = r_N = r$. We then only have two parameters, r_1 and r, to search for a given N (i.e., the number of touchpoints), v_{in}, and v_{thres}. We satisfy $r_2 = \cdots = r_N = r$ when adjusting the serpentine trace pattern's geometry in the subsequent step.

We must first specify the search range and step for each r_1 and r. For r_1, we set the search range as [200kΩ, 10MΩ] and the step increment as 200kΩ by default. These values were chosen in balance of computational performance and optimization quality. For r, we first identify the highest resistance each conduit can achieve with the serpentine trace pattern. Among these values, we select the lowest value as the upper limit for r. The step increment is set as 50kΩ. For each grid cell (e.g., r_1 = 1MΩ and r = 100kΩ), we use a circuit simulator to generate N circuits, each of which corresponds to a case where the pth point is touched ($1 \leq p \leq N$). Among the N different t_{thres_p} values, we select the pair that has the minimum difference. Each non-white cell in Fig. 8c illustrates this minimum difference. The optimal result in Fig. 8c is the cell with the largest r and smallest r_1 (i.e., r = 2500kΩ, r_1 = 4200kΩ). However, when r_1 is near the boundary of the hard constraint, $v_p(0)$ for the Nth touchpoint is also close to v_{thres} (e.g., Fig. 8-b2). Cases when $v_N(0)$ is close to v_{thres} are problematic: they can introduce violations where r may be larger than expected. This violation can be further exacerbated when 3D printing the conductive traces with poor precision. To account for general fault tolerance in 3D printers, we select a pair of r_1 and r that achieves a close-to-optimal result while satisfying the condition that $v_p(0) \leq 0.9 v_{\text{thres}}$.

Adjust the Serpentine Trace Patterns. After optimizing r_1 and r, we apply these values to the circuit design. r_1 is the resistance of an outside resistor connected to a microcontroller, and thus, it can be easily adjusted by hand. In contrast, r is the resistance value for each low conductivity conduit, and we can achieve r by adjusting the serpentine trace patterns. As discussed in Sec. 6.2, the circuit embedding step aims to generate the longest conductive trace (i.e., largest resistance) by filling a serpentine trace pattern with a given small margin (by default, 1.2mm). We can find the serpentine trace pattern that achieves $r_2 = \cdots = r_n = r$ by gradually increasing both the margin between each ray and the margin between each layer.

7 FABRICATION AND USE

Multi-Material Printing. After automatically designing the circuit, the computational pipeline outputs the fabrication data as four STL files: the original 3D model, the conductive traces, the touchpoints and wiring connection point(s), and the conduits to encase the conductive traces. We use all four files for fabrication. In Appendix B, we discuss in more detail our 3D print settings and the filaments that we use. See our Github repository[1] for the STL files of our freeform interfaces.

Wiring and Calibration. After 3D printing, we connect the printed object to a microcontroller. The schematic differs whether the freeform interface uses a single-wire or double-wire connection. Fig. 2a and Fig. 2d represents the schematic diagram for the double-wire and single-wire connection, respectively. Lastly, using an existing signal processing library [Bae et al. 2024], we calibrate the RC delay corresponding to each touchpoint by manually touching each touchpoint for five seconds and observing the time required to reach a microcontroller's logic threshold voltage. Calibration is necessary as each individual and external factors (e.g., clothing, temperature) may generate a different capacitance [Grosse-Puppendahl et al. 2017]. A video demonstration of our sensing technique can be found in the supplemental video.

8 APPLICATIONS

We demonstrate the applicability of our computational design pipeline with six freeform interfaces with different geometries. Besides the Stanford Bunny already mentioned, our other freeform

[1] https://github.com/d-rep-lab/3dp-singlewire-sensing

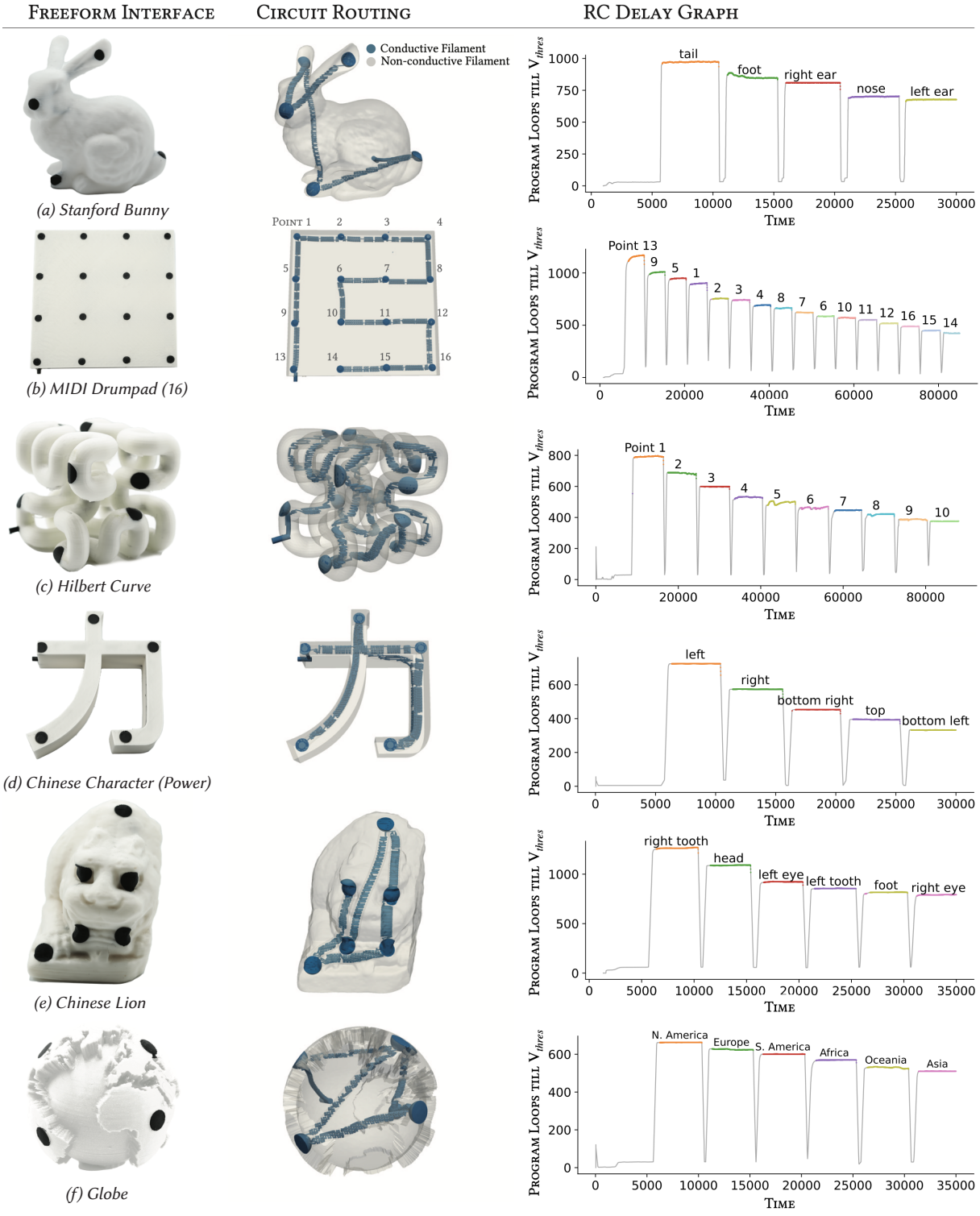


Figure 9: Examples of freeform interfaces with a single-wire connection designed with our computational design pipeline. Each row shows the 3D printed object, its internal circuit design, and its corresponding RC delay graph. The RC delay graph is made by touching each touchpoint one after another. The x-axis corresponds to the time elapsed since using the freeform interface. The y-axis corresponds to the number of program loops that elapsed until reaching the microcontroller's voltage threshold. The RC delay graph is colored based on each touchpoint's unique RC delay (e.g., gray: no touch, orange: first touchpoint).

interfaces include a MIDI Drumpad, a Hilbert Curve, a Chinese character (power), a museum artifact (Chinese lion), and a globe. Fig. 9 shows the 3D printed model, its internal circuit, and its respective RC delay graph with the single-wire connection. These objects were chosen to demonstrate our pipeline's ability to handle various geometries. The MIDI Drumpad is an example of a basic geometry; Hilbert Curve illustrates geometric curves; Power demonstrates working with a limited bounding volume; the Stanford Bunny and Chinese Lion are examples of freeform interfaces. Except for the Chinese Character (power) and MIDI Drumpad, the remaining four models were selected from the Thingi10K dataset repository [Zhou and Jacobson 2016]. See the supplemental materials to see the STL files for these freeform interfaces. The freeform interfaces represent different numbers of touchpoints with the required conduit length discussed in Fig. 10b.

MIDI Drumpad and Hilbert Curve. We envision our technique can be used to quickly prototype tangible interfaces that require many touchpoints. As discussed in Sec. 2, one of the limitations of directly embedding electronics is that it requires iterations of post-processing. To illustrate this vision, we provide two examples. Fig. 9c is a Hilbert curve with touchpoints. The Hilbert curve aims to show how our technique can enable touchpoints even for complex geometry. In contrast, Fig. 9b is a MIDI trackpad with a box shape. The trackpad was modeled in a commercial CAD tool. The trackpad emphasizes how we can enable various touchpoints (i.e., 16 in this case). Both examples show how users can quickly create different tangible prototypes while minimizing the use of electronics and post-processing.

Chinese Character (Power). Research highlights how tangible artifacts can increase learning engagement while also presenting the learning materials in a different manner [Schneider et al. 2010]. For example, Fig. 9d shows how the five touchpoints can be used to learn the sequential stroke order for the Chinese character for 'power'. The touchpoints are embossed so a user can trace their finger along the surface of the character to learn how to write the character.

Cultural Heritage Artifacts. In most museum settings, visitors cannot directly touch historical artifacts. In these cases, visitors can only inspect the historical artifacts from afar. 3D printing technology can create replicas of cultural artifacts that visitors can engage with and provide a more interative way to learn [Neumüller et al. 2014]. While visitors may not be able to directly inspect historical artifacts, the replicas can have different embedded touchpoints that users can select for further inspection. Fig. 9e shows where a visitor selects the Chinese lion's paw for closer details.

Globe. Fig. 9f shows a globe with six touchpoints to help learners identify the different continents.

9 EVALUATION

The goal of our evaluation is to enable a deeper understanding of this RC-delay capacitive sensing technique. Thus, our evaluations focus on the sensing technique rather than the pipeline as a tool. The theoretical, experimental, and computational investigation provides the groundwork for this objective, and are highly recognized methods for technical HCI work [Hudson and Mankoff 2014].

As a step toward this goal, we evaluate the efficacy of our approach with five technical evaluations and six applications. We evaluate the practical constraints of our pipeline, specifically the number of touchpoints that we can fabricate while ensuring each touchpoint is distinguishable. We perform a computational performance evaluation of the algorithms used in the automatic circuit design stage. We showcase six freeform interfaces made with our pipeline. We measured the signal-to-noise ratio of this sensing technique. We conduct a user study to assess real-time recognition accuracy. We conducted a computational experiment and mathematical analysis to evaluate the robustness of the single-wire connection.

9.1 Fabrication Scalability

To determine the scalability of the capacitive touchpoints we can fabricate, we evaluate what the *minimum* length of each conduit (mm) between two touchpoints should be to distinguish each selected point. Determining the minimum length of each conduit between two touchpoints can infer the smallest possible volume of a freeform interface. We do not evaluate the maximum size of a freeform interface as the maximum size is restricted by the build volume of a given 3D printer.

Double-Wire Connection. We first consider the double-wire connection condition. Based on Sec. 6.3.1, the time difference required to reach a microcontroller's logic threshold voltage when touching pth and $(p+1)$th touchpoint can be written as: $t_{\text{thres}p+1} - t_{\text{thres}p} = cr_{p+1}\ln(v_{\text{in}}/(v_{\text{in}} - v_{\text{thres}}))$. To analyze $t_{\text{thres}p+1} - t_{\text{thres}p}$, we place three assumptions that would represent common use:

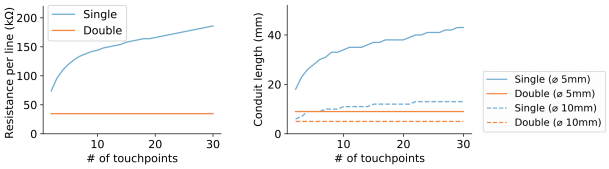
- c = 100pF, as a representative capacitance for a human body when selecting a touchpoint [ESD Association 2020].
- v_{in} = 5V and v_{thres} = 2.5V, following the technical specifications of the Arduino UNO R4.
- 5µs as the minimum value required for $t_{\text{thres}p+1} - t_{\text{thres}p}$, which corresponds to 240 clock cycles of the Arduino UNO R4's CPU.

From these conditions, we can derive $r_{p+1} \geq$35kΩ.

We need to determine the minimum length of a conduit (3D pipe) that can house conductive traces with over 35kΩ. To determine such length, we introduce the following technical assumptions:

- The use of Snapmaker J1S, a 3D printer with a 0.4 mm nozzle (standard for consumer FDM 3D printers)
- Using a 0 4mm nozzle, the thickness of the conductive trace for the xy-plane is set to 0.8mm; the ray and layer margins are set to 1.2mm (refer to Sec. 6.2).
- Protopasta's conductive PLA (1 75mm) as the conductive filament [ProtoPasta 2023].

We measured the resistance of a conductive trace per length along the xy-plane and z-direction using Protopasta's conductive PLA. The results are 256Ω/mm for the xy-plane and 1013Ω/mm for the z-direction (Appendix C). We focus only on the conductive traces on the xy-plane because our circuit embedding mainly relies on traces on the xy-plane (Sec. 6.2). Thus, to achieve 35kΩ, we need to print approximately 137mm of a conductive trace along the



(a) Required resistance **(b) Required conduit length**

Figure 10: Fabrication scalability evaluation of the single- and double-wire connections with different numbers of touchpoints. In (b), we assume that a conduit has a fixed diameter of 5mm **(our pipeline's default) or** 10mm**.**

xy-plane. This length can be achieved by drawing the conductive trace within a conduit that has a 5mm diameter (our pipeline's default value for the conduits) and 9mm length. This result infers the minimum length of a conduit should be 9mm between each pair of touchpoints.

Note that different technical assumptions can lead to different results. One significant but easily changeable assumption is the minimum value required for $t_{\text{thres}p+1} - t_{\text{thres}p}$ (i.e., 5µs). We consider 5µs to be a relatively safe value corresponding to over 200 clock cycles for the Arduino UNO 4. The value provides enough of a buffer to account for errors in the conductive trace's resistance (e.g., due to poor 3D printing precision). However, if a user can confirm one's 3D printing errors are small (e.g., high-precision printing), the minimum required time delay difference can be radically reduced (e.g., 1µs). This condition significantly reduces the required horizontal length of a conductive trace for each conduit (e.g., from 137mm to 27mm).

Single-Wire Connection. To understand the fabrication scalability for the single-wire connection, we place the same assumptions we listed for the double-wire connection condition. Similar to Sec. 6.3.2, we apply the constraint of $r_2 = \cdots = r_n = r$ and perform a grid search of r and r_1 to find the minimum value of r that satisfies $t_{\text{thres}p+1} - t_{\text{thres}p} \geq 5\mu s$ for all p. Unlike the double-wire connection, the required r for the single-wire connection varies based on the number of touchpoints. Fig. 10a summarizes the value of r depending on the different number of touchpoints. Following the same procedure as the double-wire connection, we derive the minimum length of a conduit with a 5mm diameter, as shown in Fig. 10b. The derived minimum length for 20 touchpoints is 40mm. However, as shown with the dashed lines in Fig. 10b, if we increase the diameter of the conduit to 10mm diameter, this minimum length of a conduit can be reduced to 12mm. For the single-wire connection, we can infer that (1) the required length of a conduit between two touchpoints is longer than the double-wire connection and (2) the required length follows a close-to-logarithmic function as we increase the number of touchpoints. Therefore, the single-wire connection places a stronger constraint to fabricate a small freeform interface with numerous touchpoints.

9.2 Computational Performance

We conducted a performance evaluation of the algorithms used in the automatic circuit design stage (Sec. 6). We first analyzed the time complexity of the algorithms to uncover potential performance bottlenecks when dealing with complex or large 3D objects. We then ran an experimental evaluation on different 3D models shown in Table 1. The results revealed the automatic circuit design stages can be completed between 16s and 240s.

Time Complexity. The computationally demanding steps are the (1) voxelization of the freeform model (Fig. 6b), (2) Dijkstra's pathfinding (Fig. 6f), and (3) circuit embedding (Fig. 5b).

Our voxelization uses the implementation provided by PyVista [Sullivan and Kaszynski 2019], which checks whether each position of voxel grids is within an object surface. Thus, the voxelization has $O(TG)$ where T is the number of triangles constructing a surface and G is the number of grid points. Note that G is roughly proportional to the number of the resulting voxels, V (i.e., $O(TG) \approx O(TV)$). For each pair of touchpoints, Dijkstra's pathfinding algorithm is performed with the weighted graph of the trimmed voxel representation (Fig. 6-d). In total, this pathfinding has $O(NV \log V)$ where N is the number of touchpoints. The circuit embedding finds a line segment for each ray on a conduit using the space-filling algorithm. With S triangles on a conduit's surface, the line segmentation can be performed with $O(S \log S)$. When several rays are generated to slice a conduit (modeled as a 3D pipe) with a small margin, the circuit embedding for each conduit has a time complexity of $O(US \log S)$ where U is the volume of a 3D object. We apply the serpentine trace pattern only for low conductivity conduits. In total, the circuit embedding takes $O(NUS \log S)$. However, if we model a conduit's surface with a fixed small number of triangles, we can simplify the complexity to $O(NU)$.

In sum, the automatic circuit design involves $O(TV)$ (i.e., voxelization), $O(NV \log V)$ (i.e., Dijkstra's), and $O(NU)$ (i.e., circuit embedding) computations. The results highlight that the critical parameters for computation are U (volume), T (the number of triangles), V (the number of voxels), and N (the number of touchpoints).

Experimental Evaluation. We used a MacBook Pro with 2.3 GHz 8-Core Intel Core i9 and 64 GB 2,667 MHz DDR4 (no GPU use). We collected and modeled eight 3D objects as seen in Table 1. For each object, we ran the automatic circuit design stage five times and measured the average completion time. The breakdown and total completion times are shown in Table 1. The miscellaneous steps shown in Table 1 include the clipping touchpoints, the conversion from the trimmed voxel to the graph representation, and the resistance optimization. As expected from the time complexity analysis, completion time differs based on the number of triangles, voxels, touchpoints, and model's volume. However, for the selected 3D models, the automatic circuit design is completed in less than 4 minutes for all objects.

9.3 Signal-to-Noise Ratio

Signal-to-Noise Ratio (SNR) is a quality metric that measures signal strength versus noise influence. This information can infer the likelihood of a false touch selection. SNR for capacitive sensing systems measures how robust the signals produced by the sensing technique (i.e., active signal) are compared to disturbances of background noise (i.e., inactive signal).

We employed a similar approach to past work [Palma et al. 2024], where we also computed SNR by repeatedly touching objects with

the index finger. Three objects (Stanford Bunny, Hilbert Curve, and MIDI Drumpad 16) were chosen based on the different number of touchpoints and geometry complexity. We touched all touchpoints for a given object (refer to Fig. 9 to see touchpoint placements). For each touchpoint, we adhered to a three-part process that lasted 9 seconds. First, we did not touch for 3 seconds. Then we touched the designed touchpoint for 3 seconds and lastly let go for 3 seconds. This process was repeated for 3 trials. During this process, we measured the raw capacitive values from the Arduino Uno R4. From these raw values, we used Davidson's proposed formula [Davison 2010] to compute SNR (Eq. 7). μ_U is the mean value when the touchpoint is not pressed. μ_P is the mean value the touchpoint is pressed. σ_U is the standard deviation of values when the touchpoint is not pressed.

$$SNR = \frac{|\mu_U - \mu_P|}{\sigma_U} \tag{7}$$

Traditionally, μ_U, σ_U for most capacitive sensing systems (e.g., [Palma et al. 2024; Pourjafarian et al. 2019]) represents the inactive signal (i.e., background noise). These systems cast a binary judgment of whether a touchpoint has been selected or not. However, in our case, any other touchpoint besides the target touchpoint is also considered background noise. Our capacitive sensing technique relies on a *categorical* judgment of determining which touchpoint is selected based on the RC Delay. As such, a system can wrongly judge a touchpoint selection if there is not enough difference in the RC delays among the touchpoints. Hence, for our SNR calculations, we computed a pairwise calculation ($n \times n$ matrix) between the target touchpoint (μ_P) and all of the other touchpoints (μ_U, σ_U). We report the *minimum* SNR value from this pairwise computation to illustrate the smallest gap between a pair of touchpoints (Table 2). See the supplemental material for the full pairwise computations. Davidson notes that the SNR threshold should at a minimum be 7, but ideally at least 15 for for robust sensing in real-world applications [Davison 2010]. Our results highlight how all combinations of the objects and wiring conditions satisfy this threshold, providing a high-level of reliability.

Table 1: Computational performance evaluation.

	Object Information				Completion Time (s)				
	vol (mm^3)	triangles	voxels	# points	voxelize	Dijkstra	circuit embed	misc	total
 *Stanford Bunny*	302391	259898	444696	5	60	53	12	28	155
 *MIDI Drumpad (4)*	78284	2116	50176	4	1	3	6	5	16
 *MIDI Drumpad (9)*	241964	7336	155232	9	2	7	15	15	42
 *MIDI Drumpad (16)*	458519	10316	320211	16	5	20	149	44	223
 *Hilbert Curve*	173667	39936	285897	10	12	1	51	49	117
 *Globe*	317412	29472	325651	6	19	23	9	23	74
 *Chinese Lion*	613578	69994	384825	6	23	133	23	22	201
 *Chinese Character (Power)*	132102	2256	295056	5	6	156	59	18	240

Table 2: Signal-to-noise ratio for Stanford Bunny, Hilbert Curve, and MIDI Drumpad (16) under the two wiring conditions: single wire and double wire. n represents the number of touchpoints.

Object & Wiring Condition	Trial 1	Trial 2	Trial 3
Stanford Bunny (n = 5)			
Double Wire	49.993	55.179	45.654
Single Wire	369.131	323.638	408.109
Hilbert Curve (n = 10)			
Double Wire	18.779	17.930	17.921
Single Wire	131.987	132.576	124.476
Drumpad 16 (n = 16)			
Double Wire	49.930	44.784	49.263
Single Wire	22.432	22.099	25.876

9.4 Recognition Accuracy

We evaluate two of the freeform interfaces discussed in Sec. 8. We conducted a controlled study to measure the real-time recognition accuracy of the freeform interfaces. The single-wire and double-wire connections are our independent variables. The accuracy of recognizing touchpoints is our dependent variable. Based on our discussion in Sec. 6.3 and insights from Sec. 9.1, we expect the single-wire connection to be more sensitive if a freeform interface has too many touchpoints (i.e., unable to distinguish between touchpoints). To validate this hypothesis, we measure our approach's real-time recognition accuracy with 8 different objects.

9.4.1 Objects. Eight objects were selected based on the different number of touchpoints. The objects were generated from four different models: MIDI Drumpad with 4 keys (PAD_4), MIDI Drumpad with 9 keys (PAD_9), MIDI Drumpad with 16 keys (PAD_16), and a Hilbert Curve with 10 touchpoints (HILBERT_10). Each model was printed twice to represent the single-wire and double-wire connections. Each touchpoint was labeled with a number to indicate its touchpoint ID.

We chose the MIDI Drumpads and Hilbert curve to represent both simple and complex geometry. The MIDI Drumpad's overall geometry as a box is simple enough that it would be easy for users to recognize and select the touchpoints. The simplicity in the geometry also lends well to how the same design can be easily scaled accordingly to generate different numbers of touchpoints. However, the design of the MIDI Drumpad limits assessing whether geometric complexity can also affect real-time recognition accuracy. As a result, we include a Hilbert curve with 10 touchpoints. We anticipate the Hilbert Curve is one of the most complex geometries that fits our modeling criteria.

9.4.2 Protocol. We recruited 10 participants for the study. Each study session was held in the same location to account for environmental factors. The 3D printed object is connected to an Arduino UNO 4 microcontroller, which is connected to a laptop. The laptop was powered by a wall outlet (earth ground). Participants were first given an overview of our capacitive sensing mechanism, and the experimenter demonstrated the sensing of touchpoints using the Stanford Bunny as an example.

After the demonstration, participants began the formal study. We alternated the order of the two conditions (i.e., single-wire and double-wire connection) per participant. For each object, the participants were given verbal instructions for the calibration process (Sec. 7). They were instructed on the order of the touchpoints (left to right; row by row). For the MIDI pads, they held onto each touchpoint for 7s to account for fluctuation and noise. For the Hilbert Curve, participants were given 3s additional seconds to find the appropriate touchpoint because of the object's complex geometry. After calibration, the performance evaluation started. For each object, we randomly generated the order of the touchpoint ID the participants should touch (i.e., target touchpoint). Three trials were performed for each touchpoint per object.

9.4.3 Data Collection and Analysis. For each object, we collected the participant's calibration and performance data with a signal processing library [Bae et al. 2024]. The performance data is a time series with ~64 samples per second. Each instance has a timestamp, the target touchpoint, and the classified touchpoint. For each touchpoint, we trim the first 3.5 seconds during our analysis to account for the transition time between touchpoints or look-up time to find the target touchpoint. This decision is based on our pilot study where we observed that a participant generally took about 2–3 seconds for transitions and look-up time. We added one additional second to provide a safe margin. Within the remaining time, we set a 70% threshold to determine the dominant touchpoint classification. The classification can fluctuate between two or more different touchpoint IDs if the RC delays are not distinct enough. If there is no dominant value within those remaining seconds, we consider the result to be *not recognized* (i.e., no convergence to a value).

9.4.4 Findings. Overall, our method yields an average accuracy of 91.42% (SE=1.329) across the 8 freeform interfaces. Fig. 11 shows that 6 out of 8 freeform interfaces achieve real-time accuracy of 90% or higher. Most objects in the single-wire connection demonstrated higher accuracy compared to their double-wire connection.

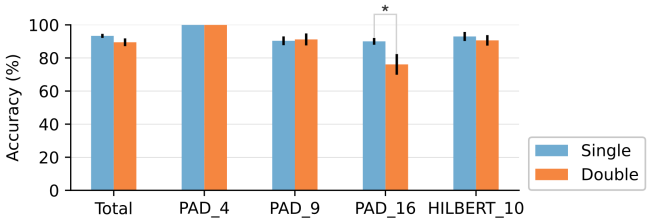


Figure 11: Results from the real-time recognition accuracy. Bar charts show the average accuracy across $n = 10$ participants with a 70% threshold. Error bars represent standard errors. There is a significant difference between the single-wire and double-wire connections for PAD_16

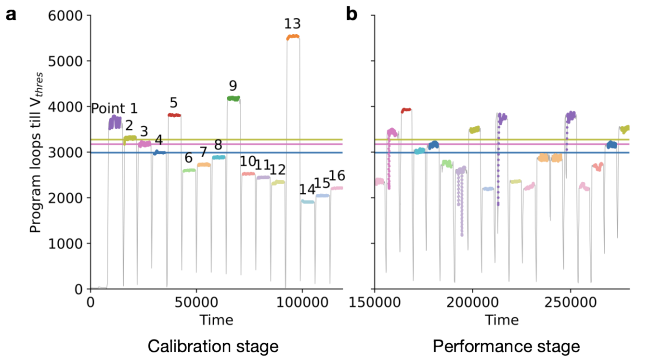


Figure 12: The RC delay graphs from P5's calibration and performance data for PAD_16 with the double-wire connection. Each graph is colored based on the target touchpoint. The colored horizontal lines highlight the mean RC delays measured during the calibration stage (yellow: Point 2, pink: Point 3, blue: Point 4). In (b), we observe that the RC delay of each target node is shifted upwards. For example, a horizontal line shows how Point 4 (blue) in the performance data corresponds to Point 3 (pink), resulting in a misrecognition.

Overall mean accuracies are 93.35% (SE=1.223) for the single-wire connection and 89.49% (SE=2.339) for the double-wire connection. However, a Wilcoxon signed-rank test did not show a statistically significant difference between the two conditions' accuracies ($Z = 1.376$, $p = 0.08445$).

The model that had the greatest difference was PAD_16. A Wilcoxon signed-rank exact test showed there is a significant difference ($p = 0.03723$) in the accuracies between the single-wire and double-wire connections. Only one participant had an accuracy of less than 80% with PAD_16 in the single-wire connection. In contrast, five participants had an accuracy of less than 80% with the double-wire connection.

A reason for the misrecognition errors is due to participants' RC delays shifting from the calibration stage to the performance stage. For example, we examined the collected data from P6 who scored only 33% accuracy for PAD_16 in the double-wire connection. Fig. 12 shows how P6's performance data for all touchpoints has shifted from their calibration stage. Consequently, most of P6's

touchpoint selections were misclassified (e.g., Point 4 in Fig. 12a was misclassified as Point 3 during the performance stage).

9.5 Robustness to Capacitance Shift

The results in Sec. 9.4 indicate that a shift in capacitance, c, can significantly influence the recognition accuracy. Given how we observed an overall higher accuracy of the single-wire connection than double-wire, we hypothesize that the single-wire connection is more robust to a capacitance shift. To validate this hypothesis, we conduct a computational experiment perturbing the capacitance and perform a mathematical analysis.

9.5.1 Computational Experiment. Utilizing a circuit simulator, Lcapy [Hayes 2022], we compare the robustness of the double-wire's and single-wire's connections to a capacitance shift. The circuit models correspond to the 8 freeform interfaces evaluated in Sec. 9.4.

We first resembled the calibration stage (Sec. 7) as follows. With the circuit models, for each touchpoint, p, we derived the time required to reach a microcontroller's logic threshold voltage (refer to Eq. 3 and Eq. 6) by setting c =100pF as a representative capacitance [ESD Association 2020]. We denote the derived time as $t_{\text{cal}p}$.

To mimic the recognition stage, we perturbed c from 100pF by randomly sampling c from a Gaussian distribution with $\mu = 100pF$ and σ. We evaluated different σ values ranging from 0pF to 5pF. For each σ, we sampled c 100 times, which corresponds to testing the recognition with 100 participants. For each sampled c, we derived the corresponding time required to reach the threshold voltage, $t_{\text{rec}p}$, for all touchpoints. Then, for each touchpoint, we judged whether the touch recognition was correct if $\arg\min_{q\in\{1,\cdots,N\}} |t_{\text{rec}p} - t_{\text{cal}q}| = p$ (note: N is the number of touchpoints). We computed the average accuracy for the 100 sampled c values and N touchpoints for all 8 freeform interfaces.

Fig. 13 summarizes the results: the single-wire connection is significantly more robust to a capacitance shift than the double-wire for the 8 freeform interfaces. By comparing Fig. 11 and Fig. 13, we can deduce that when the participants interacted with PAD_16, the capacitance shift has an approximate strength of $\sigma = 2pF$.

9.5.2 Mathematical Analysis. To theoretically validate the hypothesis of the single-wireconnection being more robust, we perform a mathematical analysis. To correctly recognize a touchpoint, p, we must satisfy two conditions:

$$|t_{\text{rec}p} - t_{\text{cal}p}| < |t_{\text{rec}p} - t_{\text{cal}p+1}| \tag{8}$$

$$|t_{\text{rec}p} - t_{\text{cal}p}| < |t_{\text{rec}p} - t_{\text{cal}p-1}| \tag{9}$$

These conditions avoid misrecognizing a touchpoint, p, with its two other adjacent touchpoints, $p-1$ and $p+1$. Note that when $p = 1$, we only need to satisfy Eq. 8; similarly, when $p = N$, only Eq. 9 is required.

9.5.3 Double-wire connection. We let c be the capacitance during the calibration stage and $(c+\epsilon)$ be the capacitance during the recognition stage. The goal is to derive the range of ϵ that satisfies both Eq. 8 and Eq. 9. By referring to Eq. 3, we can derive the range of ϵ as:

$$-\frac{c}{2}\frac{r_p}{r_{\text{till}p}} < \epsilon < \frac{c}{2}\frac{r_{p+1}}{r_{\text{till}p}} \tag{10}$$

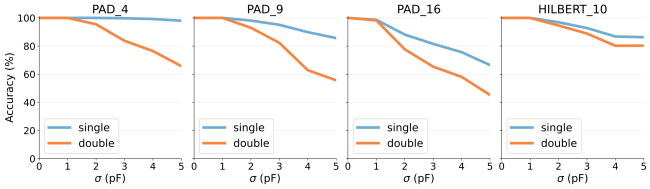


Figure 13: Results from the capacitance perturbation simulation for the freeform interfaces. σ represents the strength of the perturbation (specifically, capacitance values are sampled from a Gaussian distribution with $\mu = 100pF$ and σ). The single-wire connection is more robust to the capacitance shift than the double-wire connection.

The upper and lower bounds correspond to satisfying Eq. 8 and Eq. 9, respectively.

Single-wire connection. We derive the range of ϵ using Eq. 6. Unlike the double-wire, the single-wire's logarithmic term depends on a touchpoint, p. To simplify the math expression, we denote the logarithmic term in Eq. 6 as L_p. To satisfy Eq. 8, ϵ must satisfy:

$$\begin{cases} \epsilon < \frac{c}{2}\left(\frac{r_{\text{till}p+1}L_{p+1}}{r_{\text{till}p}L_p} - 1\right), \text{ if } r_{\text{till}p+1}L_{p+1} - r_{\text{till}p}L_p > 0 \\ \epsilon > -\frac{c}{2}\left(1 - \frac{r_{\text{till}p+1}L_{p+1}}{r_{\text{till}p}L_p}\right), \text{ otherwise} \end{cases} \tag{11}$$

Similarly, Eq. 9 corresponds to:

$$\begin{cases} \epsilon < \frac{c}{2}\left(\frac{r_{\text{till}p-1}L_{p-1}}{r_{\text{till}p}L_p} - 1\right), \text{ if } r_{\text{till}p-1}L_{p-1} - r_{\text{till}p}L_p > 0 \\ \epsilon > -\frac{c}{2}\left(1 - \frac{r_{\text{till}p-1}L_{p-1}}{r_{\text{till}p}L_p}\right), \text{ otherwise} \end{cases} \tag{12}$$

When we apply our resistance optimization (Sec. 6.3.2) to find r_1, $v_N(0)$ is only slightly smaller than v_{thres} (e.g., Fig. 8-b4). In this case, we can assume $t_{\text{cal}p+1} < t_{\text{cal}p} < t_{\text{cal}p-1}$. By concurrently considering this assumption with Eq. 11 and Eq. 12, we can derive:

$$-\frac{c}{2}\left(1 - \frac{r_{\text{till}p+1}L_{p+1}}{r_{\text{till}p}L_p}\right) < \epsilon < \frac{c}{2}\left(\frac{r_{\text{till}p-1}L_{p-1}}{r_{\text{till}p}L_p} - 1\right) \tag{13}$$

Comparison. For the double-wire connection, as indicated by Eq. 10, we can make the range of ϵ larger (i.e., more robust to the capacitance change) by increasing r_p and r_{p+1} while keeping $r_{\text{till}p}$ as small as possible. Also, since $r_{\text{till}p}$ (i.e., $r_1 + \cdots + r_p$) increases as p increases, generally, a larger p is more difficult to achieve with a wider range of ϵ. Based on these observations, to create a more robust interface, the resistance values should be designed to have $r_1 < \cdots < r_N$ while ensuring a large difference between the adjacent resistance values. For example, we can set $r_p = ar_{p-1}$ where $a > 1$. In this case, Eq. 10 becomes $-\frac{c(a^p - a^{p-1})}{2(a^p-1)} < \epsilon < \frac{c(a^{p+1}-a^p)}{2(a^p-1)}$. Ultimately, when $a \to \infty$, $-c/2 < \epsilon < \infty$. However, *in practice*, the minimum resistance (r_1) and the maximum resistance (r_N) should be large and small enough, respectively. These considerations are due to the limitations of the microcontroller's measurement of time delays, conduit volumes, and 3D printing resolutions. For example, in a practical setting, we can set $N = 10$ and $a = 1.1$. This configuration setting leads to $r_N \approx 2.4r_1$ and the average range of ϵ is $0.28c$ for touchpoints $p = \{2, \cdots, 9\}$ (i.e., $\frac{1}{8}\sum_{p=2}^{9} c\frac{r_{p+1}+r_p}{2r_{\text{till}p}}$). Note

that when we do not apply this optimization using $a > 1$, this range becomes smaller: e.g., $0.23c$ when $a = 1$.

For the single-wire connection, by referring to Eq. 13, we can infer that increasing the range of ϵ can be achieved by having relationships $r_2 > \cdots > r_N$ while keeping r_1 as small as possible. However, as discussed in Sec. 6.3.1, r_1 must also satisfy $r_1 > (1 - v_{\text{thres}}/v_{\text{in}})r_{\text{till}\,N}$. To perform a fair comparison with the double-wire, we set $N = 10, r_{p-1} = 1.1r_p$ for $p \geq 2$ (i.e., corresponding to $a = 1.1$), and $r_1 = 1.01(r_2 + \cdots r_N)$. Here we assume the use of Arduinno UNO R4 as a microcontroller, i.e., v_{in}=5V, and v_{thres}=2.5V. Note that r_1 is resistance of a resistor connected to a microcontroller and can be easily adjusted and large unlike the other resistance values. Then, this setting derives $0.33c$ as the average ϵ range for touchpoints $p = \{2, \cdots, 9\}$. When $r_{p-1} = r_p$ (i.e., a non-optimal case), this range becomes $0.28c$. These results support that the single-wire connection can create more robust freeform interfaces than the double-wire for our expected usage.

10 LIMITATIONS AND FUTURE WORK

This work introduces a computational design pipeline that embeds multiple capacitive touchpoints into any 3D model that has a closed mesh without self-intersection. Our method exploits RC Delay so that all touchpoints within our freeform interface can be capacitively sensed using only a single-wire or double-wire connection. Our six evaluations enable a thorough understanding of the RC Delay capacitive sensing technique, highlighting areas of improvement.

10.1 Supporting Smaller Objects

Our fabrication scalability evaluation demonstrates that our approach for the double-wire connection could potentially support embedding a touchpoint for every 9mm distance. The single-wire connection places stricter constraints on fabricating a freeform interface with a smaller footprint (e.g., requiring over 30mm distance between each pair of touchpoints). While our approach can generally support fabricating small objects (e.g., the smallest volume we fabricated is 78284mm^3 for four touchpoints), future research is necessary on fabricating smaller objects (e.g., robotic grippers).

Our current fabrication scalability is largely dictated by the resistivity of the Protopasta conductive filament. Conductive filaments (including the Protopasta's) are typically used to connect electronic components (i.e., the role of wires), and are manufactured to have low resistivity. In contrast, our approach uses the conductive filament to create 3D printed resistors, requiring a different need of electrical properties. In our case, as long as the filament is conductive, a larger resistivity is generally preferable. A larger resistivity can help achieve the target resistance with a shorter conductive trace length. Designing such a filament would make it possible to fabricate smaller freeform interfaces.

10.2 Supporting More Distinct Signals

Our SNR evaluation highlights the robustness of our technique. All of the reported values in Table 2 are above the minimum SNR threshold (> 7) and achieve the standard for real-world applications (> 15) [Davison 2010]. The single-wire condition for Stanford Bunny and Hilbert Curve significantly outperforms the double-wire condition. Though we see a drop in performance for the PAD_16 for the single-wire condition, this result also matches our discussion in Sec. 6.3 and insights from Sec. 9.1. As expected, the single-wire connection becomes more sensitive to noise if a freeform interface has too many touchpoints. This limitation is enforced by the resistance optimization discussed in Sec. 6.3.2. One possible improvement could be relaxing the resistance value constraint we made for the efficient optimization (i.e., $r_2 = \cdots = r_N$). Optimizing each individual resistance value would create more distinct signals. However, we expect this approach would be subject to a much higher computational cost. Similar to fabricating smaller objects, addressing this challenge requires producing higher resistance within a small volume. This could be achieved through the use of conductive filaments that have higher resistivity and shorter conductive trace lengths.

10.3 Supporting Real-Time Calibration Adjustment

Though the SNR results highlight the robustness of our technique to background noise, our user study highlights a limitation of our pipeline. Currently, touchpoint selection is fully dependent on the calibration data. During the time gap between the calibration stage and touchpoint selection, if a change is introduced (e.g., a change in participant's capacitance or microcontroller performance), this dependency without real-time adjustment can introduce significant recognition errors.

The double-wire connection is more susceptible to errors due to this dependency. In Sec. 6.3.1, we originally hypothesized that the double-wire would perform better given how we can generate numerous unique RC delays. In contrast, generating unique RC delays with the single-wire connection is more restricted. However, our user study with PAD_16 highlighted the trade-offs of the double-wire connection in real-world conditions. As discussed in Sec. 9.5, our computational experiment and mathematical analysis validate that if a change occurred after calibration, then the double-wire connection has a more critical effect. This calibration shift did not occur in the SNR tests given its short time duration (9 sec). Thus, enabling more robust sensing will require future research on improving calibration, such as including a real-time adaptive baseline for adjustment.

We imagine such solutions can include internally routing an additional wire that can gather baseline capacitance readings. Another solution is to use swept-frequency capacitive sensing [Sato et al. 2012] with our technique. This combination can sweep different frequencies over a time window to sense whether the user configuration has changed. Regardless, future solutions would require the system to automatically detect the user's capacitance and re-calibrate by referring to Eq. 3 or Eq. 6.

10.4 Supporting Multi-touch

Our approach demonstrated overall 91.42% recognition accuracy (SE=1.329) for various objects across 10 participants. However, this technique is currently limited to one touch selection at a time. Future work should examine how to extend this technique to multi-touch. This will require modifying the algorithms in the automatic circuit design to account for multiple simultaneous touches. Given

that RC delay values can be optimized with our approach, it is possible to design the traces such that the sum of each combination of RC delay values can also be a unique value. This approach is similar to creating a resistor ladder [Chris 2018] in which different combinations of resistors are used to uniquely identify multiple switches in a single circuit. Another interesting research direction would be to leverage machine learning to predict simultaneously activated touchpoints based on changes in the RC delay signal.

11 CONCLUSION

We introduce a computational design pipeline that embeds multiple capacitive touchpoints into any 3D model that has a closed mesh without self-intersection. The core of our approach is optimizing a phenomenon called RC Delay so that all touchpoints within our freeform interface can be capacitively sensed using only a single-wire or double-wire connection. By leveraging multi-material printing, we achieve our research goal of streamlining fabricating interactive 3D printed objects with complex geometry with minimal instrumentation. The strengths of our pipeline (scalability, computational performance, robustness, accuracy, and applicability) work towards 3D printing objects that are fully interactive and ready for use straight off the printer as final products in real-world contexts.

ACKNOWLEDGMENTS

This research is sponsored in part by the U.S. National Science Foundation through grants IIS-2040489, IIS-2320920, STEM+C 1933915, the Knut and Alice Wallenberg Foundation through Grant KAW 2019.0024, and the CU Boulder Engineering Education and AI-Augmented Learning Interdisciplinary Research Theme Seed Grant. This work was also supported by an agreement with the National Renewable Energy Laboratory (NREL) under Alliance Partner University Program (APUP) No. UGA-0-41026-191. Most of the work was done while S. Sandra Bae and Takanori Fujiwara were respectively at CU Boulder and Linköping University.

REFERENCES

Marwa Alalawi, Noah Pacik-Nelson, Junyi Zhu, Ben Greenspan, Andrew Doan, Brandon M Wong, Benjamin Owen-Block, Shanti Kaylene Mickens, Wilhelm Jacobus Schoeman, Michael Wessely, et al. 2023. MechSense: A Design and Fabrication Pipeline for Integrating Rotary Encoders into 3D Printed Mechanisms. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–14.

Moritz Bächer, Benjamin Hepp, Fabrizio Pece, Paul G. Kry, Bernd Bickel, Bernhard Thomaszewski, and Otmar Hilliges. 2016. DefSense: Computational design of customized deformable input devices. In *Proc. CHI*. ACM, New York, NY, USA, 3806–3816. https://doi.org/10.1145/2858036.2858354

S. Sandra Bae, Takanori Fujiwara, Anders Ynnerman, Ellen Yi-Luen Do, Michael L. Rivera, and Danielle Albers Szafir. 2024. A computational design pipeline to fabricate sensing network physicalizations. *IEEE Transactions on Visualization and Computer Graphics* 30, 1 (2024), 913–923. https://doi.org/10.1109/TVCG.2023.3327198

Rafael Ballagas, Sarthak Ghosh, and James Landay. 2018. The design space of 3D printable interactivity. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2, 2 (2018), 1–21.

Jesse Burstyn, Nicholas Fellion, Paul Strohmeier, and Roel Vertegaal. 2015. PrintPut: Resistive and capacitive input widgets for interactive 3D prints. In *Proc. INTERACT*. Springer, Cham, 332–339. https://doi.org/10.1007/978-3-319-22701-6_25

Chris. 2018. The perfect multi-button input resistor ladder. http://www.ignorantofthings.com/2018/07/the-perfect-multi-button-input-resistor.html

Ravinder S Dahiya, Giorgio Metta, Maurizio Valle, and Giulio Sandini. 2009. Tactile sensing—from humans to humanoids. *IEEE transactions on robotics* 26, 1 (2009), 1–20.

Burke Davison. 2010. Techniques for robust touch sensing design. *AN1334 Microchip Technology Inc* 53 (2010).

ESD Association. 2020. Fundamentals of electrostatic discharge: Part five-device sensitivity and testing. https://www.esda.org/esd-overview/esd-fundamentals/part-5-device-sensitivity-and-testing/.

David Espalin, Danny W Muse, Eric MacDonald, and Ryan B Wicker. 2014. 3D Printing multifunctionality: Structures with electronics. *The International Journal of Advanced Manufacturing Technology* 72 (2014), 963–978.

Patrick F. Flowers, Christopher Reyes, Shengrong Ye, Myung Jun Kim, and Benjamin J. Wiley. 2017. 3D printing electronic components and circuits with conductive thermoplastic filament. *Additive Manufacturing* 18 (2017), 156–163. https://doi.org/10.1016/j.addma.2017.10.002

Guo Liang Goh, Haining Zhang, Tzyy Haur Chong, and Wai Yee Yeong. 2021. 3D printing of multilayered and multimaterial electronics: a review. *Advanced Electronic Materials* 7, 10 (2021), 2100445.

Daniel Groeger, Elena Chong Loo, and Jürgen Steimle. 2016. HotFlex: Post-print customization of 3D prints using embedded state change. In *Proc. CHI*. ACM, New York, NY, USA, 420–432. https://doi.org/10.1145/2858036.2858191

Tobias Grosse-Puppendahl, Christian Holz, Gabe Cohn, Raphael Wimmer, Oskar Bechtold, et al. 2017. Finding common ground: A survey of capacitive sensing in human-computer interaction. In *Proc. CHI*. ACM, New York, 3293–3315. https://doi.org/10.1145/3025453.3025808

Peter E Hart, Nils J Nilsson, and Bertram Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4, 2 (1968), 100–107.

Michael Hayes. 2022. Lcapy: Symbolic linear circuit analysis with Python. *PeerJ Computer Science* (2022), e875:1–e875:30. https://doi.org/10.7717/peerj-cs.875

Liang He, Jarrid A. Wittkopf, Ji Won Jun, Kris Erickson, and Rafael Tico Ballagas. 2022. ModElec: A design tool for prototyping physical computing devices using conductive 3D printing. *Proc ACM Interact Mob Wearable Ubiquitous Technol* 5, 4, Article 159 (2022), 20 pages. https://doi.org/10.1145/3495000

Scott E Hudson and Jennifer Mankoff. 2014. Concepts, values, and methods for technical human–computer interaction research. In *Ways of Knowing in HCI*. Springer, 69–93.

Kaori Ikematsu and Itiro Siio. 2018. Ohmic-Touch: Extending Touch Interaction by Indirect Touch through Resistive Objects. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/3173574.3174095

Garrett Johnson. 2023. three-mesh-bvh. https://github.com/gkjohnson/three-mesh-bvh. Accessed: 2023-09-10.

Kunihiro Kato, Kaori Ikematsu, and Yoshihiro Kawahara. 2020. CAPath: 3D-Printed Interfaces with Conductive Points in Grid Layout to Extend Capacitive Touch Inputs. *Proc. ACM Hum.-Comput. Interact.* 4, ISS, Article 193 (nov 2020), 17 pages. https://doi.org/10.1145/3427321

Carson Kohlbrenner, Caleb Esobedo, S. Sandra Bae, Alex Dickhans, and Alessandro Roncone. 2025. GenTact Toolbox: A Computational Design Pipeline to Procedurally Generate Context-Driven 3D Printed Large-Area Tactile Skins. In *Proceedings of the 2025 IEEE International Conference on Robotics and Automation* (Atlanta, Georgia).

Eric Macdonald, Rudy Salas, David Espalin, Mireya Perez, Efrain Aguilera, Dan Muse, and Ryan B. Wicker. 2014. 3D Printing for the rapid prototyping of structural electronics. *IEEE Access* 2 (2014), 234–242. https://doi.org/10.1109/access.2014.2311810

Aaron Meurer, Christopher P Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B Kirpichev, et al. 2017. SymPy: Symbolic computing in Python. *PeerJ Computer Science* (2017).

Moritz Neumüller, Andreas Reichinger, Florian Rist, and Christian Kern. 2014. 3D printing for cultural heritage: Preservation, accessibility, research and education. *3D research challenges in cultural heritage: a roadmap in digital heritage preservation* (2014), 119–134.

Simon Olberding, Nan-Wei Gong, John Tiab, Joseph A Paradiso, and Jürgen Steimle. 2013. A cuttable multi-touch sensor. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*. 245–254.

Gianpaolo Palma, Narges Pourjafarian, Jürgen Steimle, and Paolo Cignoni. 2024. Capacitive Touch Sensing on General 3D Surfaces. *ACM Transactions on Graphics* 43, 4 (2024), 1–20.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

Tiago P. Peixoto. 2014. The graph-tool python library. *figshare* (2014). http://figshare.com/articles/graph_tool/1164194.

Huaishu Peng, Jennifer Mankoff, Scott E. Hudson, and James McCann. 2015. A layered fabric 3D printer for soft interactive objects. In *Proc. CHI*. ACM, New York, NY, USA, 1789–1798. https://doi.org/10.1145/2702123.2702327

Narjes Pourjafarian, Anusha Withana, Joseph A. Paradiso, and Jürgen Steimle. 2019. Multi-Touch Kit: A do-it-yourself technique for capacitive multi-touch sensing using a commodity microcontroller. In *Proc. UIST* (New Orleans, LA, USA). ACM, New York, NY, USA, 1071–1083. https://doi.org/10.1145/3332165.3347895

ProtoPasta. 2023. Conductive PLA. https://www.proto-pasta.com/products/conductive-pla. Accessed: 2023-06-28.

Jordi-Roger Riba, Francesca Capelli, and Manuel Moreno-Eguilaz. 2019. Analysis and mitigation of stray capacitance effects in resistive high-voltage dividers. *Energies* 12, 12 (2019). https://doi.org/10.3390/en12122278
Jarek Rossignac. 2005. Shape complexity. *The Visual Computer* 21 (2005), 985–996.
Siddharth Rupavatharam, Xiaoran Fan, Caleb Escobedo, Daewon Lee, Larry Jackel, Richard Howard, Colin Prepscius, Daniel Lee, and Volkan Isler. 2023. AmbiSense: Acoustic Field Based Blindspot-Free Proximity Detection and Bearing Estimation. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 5974–5981. https://doi.org/10.1109/IROS55552.2023.10341766
Munehiko Sato, Ivan Poupyrev, and Chris Harrison. 2012. Touché: enhancing touch interaction on humans, screens, liquids, and everyday objects. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Austin, Texas, USA) *(CHI '12)*. Association for Computing Machinery, New York, NY, USA, 483–492. https://doi.org/10.1145/2207676.2207743
Valkyrie Savage, Colin Chang, and Björn Hartmann. 2013. Sauron: Embedded single-camera sensing of printed physical user interfaces. In *Proc. UIST*. ACM, New York, 447–456. https://doi.org/10.1145/2501988.2501992
Valkyrie Savage, Ryan Schmidt, Tovi Grossman, George Fitzmaurice, and Björn Hartmann. 2014. A series of tubes: Adding interactivity to 3D prints using internal pipes. In *Proc. UIST*. ACM, New York, 3–12. https://doi.org/10.1145/2642918.2647374
Martin Schmitz, Mohammadreza Khalilbeigi, Matthias Balwierz, Roman Lissermann, Max Mühlhäuser, and Jürgen Steimle. 2015. Capricate: A fabrication pipeline to design and 3D print capacitive touch sensors for interactive objects. In *Proc. UIST*. ACM, New York, 253–258. https://doi.org/10.1145/2807442.2807503
Martin Schmitz, Florian Müller, Max Mühlhäuser, Jan Riemann, and Huy Viet Viet Le. 2021. Itsy-Bits: Fabrication and recognition of 3D-printed tangibles with small footprints on capacitive touchscreens. In *Proc. CHI*. ACM, New York, NY, USA, Article 419, 12 pages. https://doi.org/10.1145/3411764.3445502
Martin Schmitz, Jürgen Steimle, Jochen Huber, Niloofar Dezfuli, and Max Mühlhäuser. 2017. Flexibles: Deformation-aware 3D-printed tangibles for capacitive touchscreens. In *Proc. CHI*. ACM, New York, NY, USA, 1001–1014. https://doi.org/10.1145/3025453.3025663
Martin Schmitz, Martin Stitz, Florian Müller, Markus Funk, and Max Mühlhäuser. 2019. ./trilaterate: A fabrication pipeline to design and 3D print hover-, touch-, and force-sensitive objects. In *Proc. CHI*. ACM, New York, 1–13. https://doi.org/10.1145/3290605.3300684
Bertrand Schneider, Patrick Jermann, Guillaume Zufferey, and Pierre Dillenbourg. 2010. Benefits of a tangible interface for collaborative learning and interaction. *IEEE Transactions on Learning Technologies* 4, 3 (2010), 222–232.
Will Schroeder, Ken Martin, and Bill Lorensen. 2006. *The Visualization Toolkit (4th ed.)*. Kitware.
Wei-Shan Soh, Kye-Yak See, Weng-Yew Chang, Manish Oswal, Lin-Biao Wang, et al. 2009. Comprehensive analysis of serpentine line design. In *Proc. APMC*. IEEE, USA, 1285–1288. https://doi.org/10.1109/APMC.2009.5384455
Bane Sullivan and Alexander Kaszynski. 2019. PyVista: 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK). *Journal of Open Source Software* 4, 37 (May 2019), 1450. https://doi.org/10.21105/joss.01450
Saiganesh Swaminathan, Jonathon Fagert, Michael Rivera, Andrew Cao, Gierad Laput, Hae Young Noh, and Scott E. Hudson. 2020. OptiStructures: Fabrication of room-scale interactive structures with embedded fiber Bragg grating optical sensors and displays. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 2, Article 50 (jun 2020), 21 pages. https://doi.org/10.1145/3397310
Ryosuke Takada, Buntarou Shizuki, and Jiro Tanaka. 2016. MonoTouch: Single Capacitive Touch Sensor that Differentiates Touch Gestures. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI EA '16)*. Association for Computing Machinery, New York, NY, USA, 2736–2743. https://doi.org/10.1145/2851581.2892350
three.js authors. 2023. three.js. https://github.com/mrdoob/three.js. Accessed: 2023-09-10.
Nobuyuki Umetani and Ryan Schmidt. 2017. SurfCuit: Surface-mounted circuits on 3D prints. *IEEE Computer Graphics and Applications* 37, 3 (2017), 52–60. https://doi.org/10.1109/MCG.2017.40
Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, et al. 2020. SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods* 17 (2020), 261–272. https://doi.org/10.1038/s41592-019-0686-2
Guanyun Wang, Fang Qin, Haolin Liu, Ye Tao, Yang Zhang, Yongjie Jessica Zhang, and Lining Yao. 2020. MorphingCircuit: An integrated design, simulation, and fabrication workflow for self-morphing electronics. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 4 (2020), 1–26.
Xin Wen, S. Sandra Bae, and Michael L. Rivera. 2025. Enabling Recycling of Multi-Material 3D Printed Objects through Computational Design and Disassembly by Dissolution. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan).
Qingnan Zhou and Alec Jacobson. 2016. Thingi10k: A dataset of 10,000 3D-printing models. *arXiv preprint arXiv:1605.04797* (2016).
Junyi Zhu, Lotta-Gili Blumberg, Yunyi Zhu, Martin Nisser, Ethan Levi Carlson, Xin Wen, Kevin Shum, Jessica Ayeley Quaye, and Stefanie Mueller. 2020a. CurveBoards: Integrating breadboards into physical objects to prototype function in the context of form. In *Proc. CHI*. ACM, New York, NY, USA, 1–13. https://doi.org/10.1145/3313831.3376617
Junyi Zhu, Yunyi Zhu, Jiaming Cui, Leon Cheng, Jackson Snowden, Mark Chounlakone, Michael Wessely, and Stefanie Mueller. 2020b. MorphSensor: A 3D electronic design tool for reforming sensor modules. In *Proc. UIST* (Virtual Event, USA). ACM, New York, NY, USA, 541–553. https://doi.org/10.1145/3379337.3415898

A EQUATION DERIVATION WITH SYMBOLIC PROGRAMMING

We derived Eq. 1 and Eq. 4 by utilizing Lcapy [Hayes 2022], a Python library that can perform symbolic circuit analysis. Symbolic circuit analysis can derive equations from given circuits and mathematical symbols (in our case, v_{in}, c, t, $r_1, \cdots, r_{n+1}$)

B IMPLEMENTATION

B.1 Software Implementation

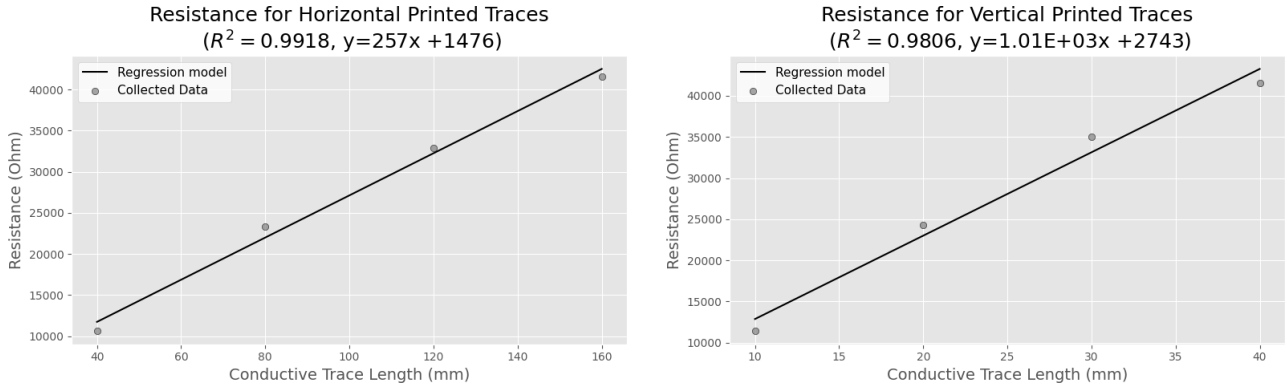
The user interface to select the touchpoints in Fig. 4 is a web application made using three.js [three.js authors 2023] and the three-mesh-bvh [Johnson 2023] libraries. We implemented the algorithms used in the automatic circuit design stage (Sec. 6) with Python 3 and libraries for matrix computations and machine learning methods such as NumPy/SciPy [Virtanen et al. 2020] and scikit-learn [Pedregosa et al. 2011]. We used graph-tool [Peixoto 2014] to use algorithms such as Dijkstra's and A* for path finding. We used PyVista [Sullivan and Kaszynski 2019] (a Python API for Visualization Toolkit [Schroeder et al. 2006]) for 3D graphics-related operations such as clipping, voxelization, and ray tracing. For the resistance optimization for the single-wire condition, we used Lcapy [Hayes 2022] and SymPy [Meurer et al. 2017] for the circuit simulation and symbolic computation and Pathos for multiprocessing. For the calibration, we utilized the sensing-network library [Bae et al. 2024].

Table 3: Resistance values of conductive traces. Three samples (S1-S3) for each measurement. For the thickness of the conductive trace, we followed our computational design pipeline default (i.e., horizontal: 0.8mm**, vertical:** 1.2mm**).**

Conductive Trace Length (mm)	S1 (Ω)	S2 (Ω)	S3 (Ω)	Avg (Ω)
Horizontal				
40	10600	10630	10600	10610
80	23330	23630	23110	23357
120	33290	33020	32470	32927
160	40200	43070	41600	41623
Vertical				
10	10730	12500	10970	11400
20	26070	22670	24080	24274
30	37700	38830	28540	35023
40	43830	42600	38330	41587

B.2 3D Printing Hardware and Materials

To fabricate freeform interfaces with embedded multi-points within one pass, we rely on a multi-material FDM 3D printer using non-conductive and conductive filaments. We use a Snapmaker J1S 3D printer, which supports dual-nozzle printing. Both nozzles are



(a) Resistance measurement for horizontal traces

(b) Resistance measurement for vertical traces

Figure 14: Two linear regression models for (a) horizontal traces and (b) vertical traces.

0.4 mm standard soft brass. We set our print speed to 60 mm/s for both conductive and non-conductive filaments. The layer height of all prints is 0.24 mm.

Our conductive filament is Protopasta's conductive PLA (1.75 mm) [ProtoPasta 2023]. This filament is commonly available and provides a good balance of conductivity and resistivity to design a sensing network. The non-conductive filament can be any standard PLA filament. The print and build plate temperatures for both filaments used were based on vendor recommendations.

The infill percentage differs for the four files discussed in Sec. 7. The original body uses an infill of 20% using the gyroid pattern. In some of our preliminary tests, we found that not having enough infill (e.g., 0%) can cause parasitic capacitance [Riba et al. 2019] where the coupled charge dissipates due to the air inside the model's body. As such, we recommend choosing a range between 5–20% infill to provide steady sensor readouts. The other STL files all have an infill percentage of 100% using the rectilinear pattern.

B.3 Sensing System

For our microcontroller, we tested using the Arduino Uno R4 WiFi (48MHz CPU), which has a 5V power source and a 2.5V logic threshold. To constantly measure the time delays, we utilized digital signals from the microcontroller's digital I/O pins.

C MEASUREMENT OF CONDUCTIVE TRACE'S RESISTANCE

We conducted an empirical investigation to understand the resistivity properties of the conductive filament. We measured the conductive traces horizontally and vertically as our 3D printed conductive traces are drawn in a serpentine trace pattern. We produced three samples of each measurement, and we used Fluke 115 Digital Multimeteter to measure each object's resistance (Table 3). See the supplemental materials to see the STL files.

In Fig. 14, we used the average values in (Table 3) to plot the relationship between the conductive trace's length and measured resistance. The two linear regression models informed how to generate the conductive traces during the circuit embedding step (Sec. 6.2).