A Computational Design Pipeline to Fabricate Sensing Network Physicalizations

S. Sandra Bae, Takanori Fujiwara, Anders Ynnerman, Ellen Yi-Luen Do, Michael L. Rivera, and Danielle Albers Szafir

Abstract—Interaction is critical for data analysis and sensemaking. However, designing interactive physicalizations is challenging as it requires cross-disciplinary knowledge in visualization, fabrication, and electronics. Interactive physicalizations are typically produced in an unstructured manner, resulting in unique solutions for a specific dataset, problem, or interaction that cannot be easily extended or adapted to new scenarios or future physicalizations. To mitigate these challenges, we introduce a computational design pipeline to 3D print network physicalizations with integrated sensing capabilities. Networks are ubiquitous, yet their complex geometry also requires significant engineering considerations to provide intuitive, effective interactions for exploration. Using our pipeline, designers can readily produce network physicalizations supporting *selection*—the most critical atomic operation for interaction—by touch through capacitive sensing and computational inference. Our computational design pipeline introduces a new design paradigm by concurrently considering the form and interactivity of a physicalization during fabrication. We evaluate our approach using (i) computational evaluations, (ii) three usage scenarios focusing on general visualization tasks, and (iii) expert interviews. The design paradigm introduced by our pipeline enables generalizability that can help lower the barrier to physicalization research, creation, and adoption.

Index Terms—Physicalization, tangible interfaces, 3D printing, computational fabrication, design automation, network data

1 Introduction

Interactions are critical for working with data. They help people build mental models as they explore a dataset by zooming into critical details, filtering for important information, or panning to view otherwise hidden surfaces [53, 92]. Manipulating data physically can further enhance our data understanding when compared to more traditional formats (e.g., 2D display) by being more natural [24], yielding better performance on certain tasks [48], and even improving information recall [50, 78, 79]. Despite these benefits, most physicalizations are static representations that do not support common data interactions (e.g., selecting, filtering). Without interactions, physicalizations are essentially static images of datasets, reducing data expressiveness and inhibiting data exploration.

Creating interactive physicalizations is challenging because it requires cross-disciplinary knowledge spanning visualization, fabrication, and electronics [8, 23]. Visualization enables us to effectively represent data. Fabrication converts data into physical objects using computer-aided designs (CAD). Electronics integrate interactive capabilities into the resulting objects. Most interactive physicalizations are produced in an ad hoc, unstructured manner, resulting in one-off solutions tailored to a specific dataset, problem, or interaction. Without generalizable approaches, these unstructured explorations mean that the form (i.e., physical structure) and function (i.e., interactive capabilities) of physicalizations are produced as separate workflows, leading to a lack of transferable techniques for future physicalizations.

To support generalizability, this work presents a computational design pipeline that enables us to 3D print various network physicalizations with integrated sensing capabilities. Given a network dataset, our pipeline produces a network physicalization that supports node *selection* using capacitive sensing—a common technique for capturing touch input on devices (e.g., smartphones) [34]. Leveraging multi-material 3D printing, we create capacitive sensors within the network that can be used to uniquely identify each node. By concurrently considering form and interactivity, our pipeline automates low-level hardware instrumentation to raise selection-driven interactions to the application level (e.g., AR/VR, shape-changing displays, or desktop visualizations). Our pipeline intentionally provides flexibility for interaction design: the node selection provides input, but how the selection is used to generate corresponding output on the application level is left to the developer.

We focus on networks as they have broad utility in almost all domains [9, 51, 57]. Past network physicalizations investigated layouts of physical networks (e.g., brain networks) [21] and how physicality can aid in data sensemaking [24] and common network tasks [55]. However, these artifacts lack interactivity. Networks have complex geometry that requires significant engineering considerations to make them physically interactive (e.g., circuit design, sensor integration). Our computational design pipeline—enabled by 3D printing—addresses these challenges by generating electrical circuits and 3D models in two steps.

First, given a network dataset, our pipeline automatically generates conductive traces (i.e., wires) with tuned electrical resistance that will be integrated into the links of a network physicalization. Second, it generates the necessary 3D geometry to fabricate the sensing network physicalization consisting of links and nodes. Conductive traces within the links act as resistors, and the nodes act as electrodes. The nodes and links are computationally designed to exploit a phenomenon called *resistor-capacitor (RC) delay* (cf. Sec. 3.1). Our pipeline generates unique RC time delays when a user touches any node within the network thereby enabling applications to identify the touched node using a single capacitance measure. Applications can use this identification to implement *selection* by touch. Selection—the ability to indicate a mark of interest—is a fundamental interaction primitive for more complex interaction designs, such as filtering or elaborating [4, 26, 92].

Sensing physicalizations, as a whole, provide a critical step toward creating future physicalizations with interactivity. To lay this foundation, we demonstrate the efficacy of our approach with computational evaluations, expert discussion, and three usage scenarios. These usage scenarios motivate the need for sensing networks and illustrate how selection can support three general visualizations tasks: exploration, explanation, and analytic provenance [30, 65, 93]. Furthermore, our network physicalizations serve as groundwork to support any visualizations modeled as a hub-and-strut form (a series of connected points, e.g., vertices in a Voronoi diagram, points on a line chart).

This work shows how a systematic approach can lead to generaliz-

- *S. Sandra Bae, Ellen Yi-Luen Do, Michael L. Rivera are with University of Colorado, Boulder. E-mail: {sandra.bae, ellen.do, mrivera}@colorado.edu*
- *Takanori Fujiwara, Anders Ynnerman are with Linköping University. E-mail: {takanori.fujiwara, anders.ynnerman}@liu.se*
- *Danielle Albers Szafir is with University of North Carolina-Chapel Hill. E-mail: danielle.szafir@cs.unc.edu.*

able techniques for physicalizations by integrating form and interaction into one cohesive fabrication workflow. Consequently, this new design paradigm can aid in lowering the barrier to physicalization research, creation, and adoption. The source code and supplemental materials (e.g., video demo) are available at our project website [1].

Contributions: The primary contributions of this paper are:

- A fabrication approach that integrates capacitive sensing into network physicalizations and supports the data interaction *selection*;
- Novel algorithms that (i) convert network datasets into electrical circuits of resistors, (ii) optimize the resistance selection, and (iii) perform network layout adjustments to satisfy fabrication constraints.

2 Related Work

Due to the cross-disciplinary nature of physicalizations [8], we build upon past work in (i) interaction primitives in visualization, (ii) fabrication approaches used in physicalizations, (iii) network physicalizations, and (iv) fabrication techniques to create capacitive sensors within 3D printed objects.

2.1 Select as an Interaction Primitive

To efficiently explore data (whether digitally or physically), people must be able to easily select different parts of the representation and inspect (or hide) details [53, 92]. Both human-computer interaction (HCI) and visualization recognize *select* as an interaction primitive that enables other interactions to take place. Foley et al. [26] places selection at the highest hierarchical level of input interactions for graphics, indicating its foundational role in supporting more complex interaction designs. Similarly, several visualization interaction taxonomies [4, 92] highlight the necessity of select for visualization systems to enable other data-related interactions. While select is traditionally achieved with the keyboard and mouse in desktop visualizations, we do not have a corresponding paradigm for physicalizations.

Despite the lack of interaction standards for physicalization, research identifies several benefits of interacting with data physically. Compared to desktop visualizations, direct manipulation with physicalizations has been shown to be more natural and preferred [24], yield better task performance [48], and improve information recall [50, 78, 79]. However, most physicalizations still have limited interaction capabilities. A key reason for this limitation is that the materials used to create physicalizations generally do not sense and respond to users.

Though some physicalizations' forms naturally afford interactions [18, 44], most offset this difficulty by incorporating off-the-shelf electronic components (e.g., motors, LEDs) into their form [52, 82] or use computer vision (CV) techniques [38, 55]. Electronic components have pre-defined scales and dimensions, ultimately affecting how they can be incorporated into a physicalization. Solutions leveraging electronic components do not generalize well to more complex physicalizations (e.g., complex spatial structures, large datasets). For example, electronic components are unlikely able to support interactivity for a network of 50 nodes and 70 links given the complex internal wiring. CV typically relies on visual tracking for gesture interactions, introducing challenges such as inaccurate alignment of the virtual and physical objects as objects move, occlusion from hands or other parts of the structure, and requiring users to stay in-frame [13]. Additionally, most devices using CV cannot reproduce the haptic benefits that we naturally leverage (i.e., holding, rotating, tracing) with our sense of touch. Past studies [11, 24, 41, 90] confirm the importance of tangible inputs when virtually exploring data. The existing practices for implementing these two approaches reflect the prevailing design paradigm of separating interactive physicalization design into two phases: first thinking about form and then interactions.

Separating form from interaction leads to a series of issues, including post-hoc instrumentation, the potential for conflicting constraints requiring costly design iterations, and a lack of generalizability across design instances. Most existing interaction infrastructures are designed for a specific implementation, resulting in one-off artifacts where the design insights and methods gained from a physicalization are specific to that instantiation [8, 23]. Our methodology integrates sensing capabilities directly into a physicalization's form, enabling designers to concurrently consider form and interactivity. This concurrent design enables us to address two key design and fabrication challenges in physicalization as outlined by Djavaherpour et al [23]: (i) designing for manufacture and assembly and (ii) prototyping and interactive design.

2.2 Digital Fabrication Approaches for Physicalizations

Digital fabrication uses a digital representation (e.g., 3D model) to produce a physical object. This procedure allows designers to use computational techniques to generate designs for physicalizations that would be otherwise difficult to manually create. 3D printing is a common digital fabrication technique used to make physicalizations [8, 23]. With 3D printing, thin amounts of material are layered to produce an object with complex geometries (e.g., overhangs, lattices, and internal structures). 3D printing can also employ multiple materials to achieve differences in color or material properties (conductivity, flexibility, etc.) [58]. These capabilities offer creative opportunities to represent data, including cartographic maps [3, 83], customized data representations [80], and common visualization idioms (e.g., bar charts [81], networks [21, 24, 55]). These works illustrate how 3D printing is a powerful representational medium and highlight its potential to produce more complex physicalizations.

Computational pipelines [20, 81] can also make designing physicalizations easier. MakerVis [81] imports a dataset to be fabricated as layered visualizations (e.g., bar, line) or prism maps. These physicalizations are passive, static objects. Abreu de Freitas et al. [20] extend MakerVis by enabling a designer to specify the intended behavior for bar charts (i.e., passive, reconfigurable, dynamic). Dynamic bars are printed but must be attached to various external electronic components for interaction, which present limitations (cf. Sec. 2.1).

We build on past pipelines in two ways: supporting structural complexity using 3D printing and integrating key interaction support during fabrication. Our approach explores complex geometries (i.e., networks), which offer unique structural challenges for fabrication that our pipeline dynamically resolves (see Sec. 3.4.1). Our pipeline also automatically specifies and produces electrical circuitry inside of a physicalization using a conductive filament, rather than designing around electrical components that would be integrated post-production. This approach eases the process of creating interactive physicalizations by linking electronic design and implementation with a physicalization's form.

2.3 Network Physicalizations

Current network physicalizations are passive objects with no innate sensing capabilities (e.g., [27, 33, 59]). Drogemuller et al. [24] investigate haptic and visual comprehension of a flat 2D network physicalization (16–24 nodes) using an external camera to log user interactions. Dehmamy et al. [21] introduce a network layout algorithm for physical networks that is optimized to avoid link intersection, but their algorithm relies on a rigorous trial-and-error process to determine layout parameters. Their 3D printed network (184 nodes, 176 links) does not support integrated sensing. McGuffin et al. [55] show the promise of physical network interactions (70 nodes, 140 links) mediated with augmented reality (AR) tracked using external cameras, which present interaction limitations (cf. Sec. 2.1). In contrast to these work, our approach supports network physicalizations with minimal parameter-tuning to produce free-standing networks with integrated sensing capabilities.

2.4 Integrating Interactivity into 3D Printed Objects

Most objects made on 3D printers are static forms that have no interactive capabilities. HCI research explored embedding electronic components (e.g., sensors and LEDs [37, 89, 91]) and designing internal structures (e.g., pipes for light [72, 91]) in printed objects to support interactivity through user input and display output. Past work also investigated how to create conductive traces within printed objects to achieve *capacitive sensing* [16, 73, 74], a common technique used in commercial devices (e.g., smartphones, tablets) to capture touch input [34]. A conductive material (e.g., conductive PLA) inside an object can create electrical traces for capacitive sensing. Capricate [73] and ./trilaterate [74] create capacitive sensors in 3D printed forms; however, both rely on having an individual electrical connection (i.e., wire) for

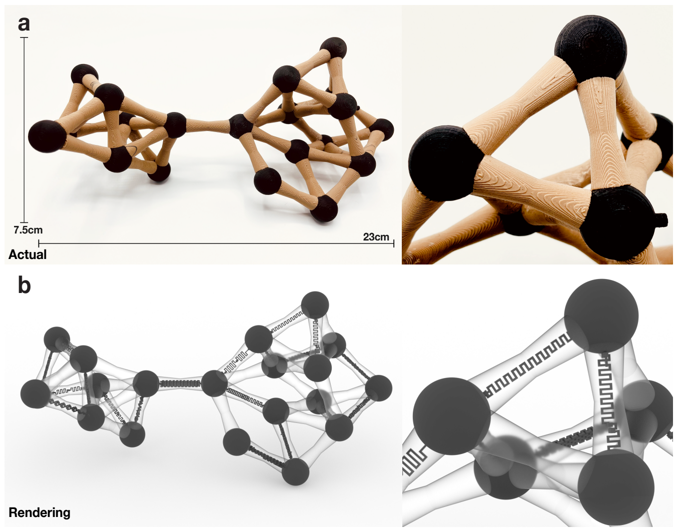

Fig. 1: A sensing network physicalization ($N = 20$, $L = 40$). (a) A multi-material 3D printed network physicalization produced by our computational design pipeline (Sec. 3.2). Conductive traces are embedded in the network's links which enables node selection via capacitive sensing. (b) A computational rendering of the network physicalization showcasing how the conductive traces are distributed throughout the network's links. The conductive traces use a serpentine pattern.

each sensor. This instrumentation can affect an object's mobility and scalability. Furthermore, neither system is designed to handle the complex structure of networks. Our approach requires a minimum of one wire to achieve sensing and optimizes the design of electrical traces, capacitive sensors, and physical geometry specifically for networks.

3 Methodology

Our computational design pipeline produces network physicalizations capable of node selection via capacitive sensing (Fig. 1a). When a user touches a node, the network physicalization outputs unique sensor readouts such that a visualization application (e.g., desktop, AR/VR) can identify the selected node. See Sec. 4.2 for usage scenarios. We first outline the basic principles of how we achieve capacitive sensing using multi-material 3D printing (Sec. 3.1). We then provide an overview of the pipeline (Sec. 3.2) and discuss each pipeline subcomponent in Sec. 3.3–Sec. 3.6. The pipeline involves various network representations. To distinguish these networks, we define and use the following terms:

- **Network Dataset:** Dataset pertaining to nodes and links.
- **Resistor Network:** A network of resistors derived from the network dataset that will be converted into conductive traces.
- **3D Representation Network:** A 3D representation of the network dataset, assigning spatial positions to nodes and links.
- **Fabrication-Ready Network:** A digital model for 3D printing that embeds the resistor network within the 3D representation network
- **Printed Network:** A 3D printed network produced from the fabrication-ready network 3D model
- **Sensing Network:** A printed network that has sensing capabilities.

Our supplementary materials summarize notations used in the paper [1].

3.1 Basic Principles of Sensing Network Physicalization

We use capacitive sensing to infer node selection by touch. This capacitive sensing is achieved through the conductive traces distributed throughout a set of a network's links (Fig. 1b). Conductive filament used in 3D printing can act as resistors in electrical circuits [22]. Our printed network has (i) conductive nodes and (ii) traces of specific electrical resistance integrated into its links. As shown in Fig. 2a (a two-node network), we connect this printed network to an electrical circuit that is managed by a microcontroller. This connection results in a combined circuit as shown in Fig. 2b.

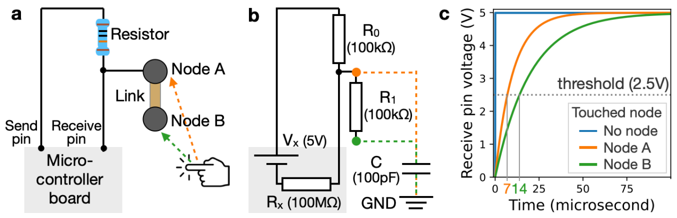

Fig. 2: Identifying node selection with capacitive sensing: (a) overall schematic where a two-node printed network is connected to a microcontroller's circuit; (b) the circuit diagram corresponding to (a) with representative resistance measurements; and (c) the voltage change measured at the receive pin when a different node is touched.

When a user touches a node (e.g., in Fig. 2b, the orange and green points are nodes A and B, respectively), the user's body and the node become capacitively coupled (i.e., energy moves between them [34]), inducing an *RC delay*. RC delay is the time required to charge a capacitor in a circuit through a particular amount of resistance. Increasing the resistance in the circuit will also increase the amount of time needed to charge the capacitor. Our pipeline is based on this principle: we can design networks where the RC delay varies based on which node is touched (cf. Fig. 2c). By computationally designing the geometry of conductive traces, different paths with unique amounts of resistance and regions can function as entry points of capacitive coupling for the RC delay.

Thus, each capacitive region takes a different amount of time to charge once a user touches it. This process enables us to infer a touched node by measuring the time taken to reach a predefined voltage threshold (e.g., 2.5V) on a microcontroller. For example, 0µs indicates no nodes are touched; 7µs is Node A, and 14µs is Node B. Our pipeline automatically converts network data into an electrical circuit that can uniquely identify each node with capacitive sensing with the following.

Conductive printed network. For a printed network, nodes should have high conductivity, so their resistance is essentially negligible. Links, in contrast, should have low conductivity so they can act as resistors. This process requires converting network data, specifically the links, into a network of resistors as *conductive traces* (i.e., electrical paths). The resistance of these traces can be tuned by varying their length and thickness [22]. In our case, low conductivity/high resistance can be achieved by printing a long, thin line using a conductive filament with specific geometry computed using the resistivity law:

$$r = \frac{\rho l}{a} \tag{1}$$

where r is the resultant resistance, ρ is a material's resistivity, and l and a are respectively the length and cross-sectional area of an object. Nodes are printed using a conductive filament, producing a large cross-sectional area per length, resulting in low resistance. Links enclose conductive traces using non-conductive filament, which can be varied to support different visual design parameters (e.g., length, width, color).

Capacitive sensing of different nodes. When an electrical circuit consists of a power source, a resistor, and a capacitor in series, the voltage change of the capacitor is expressed as:

$$v(t) = v_{\text{in}}\left(1 - e^{-\frac{t}{cr}}\right) \tag{2}$$

where t is the time from the start of charging the capacitor; v_{in} is the input voltage from the power source; c is the capacitor's capacitance; and r is the resistor's resistance. When we apply capacitive sensing to a printed network, the equation changes based on the number of links (i.e., resistors) and their resistance, the resistors' connections (i.e., series, parallel, or combination), and the touched node (i.e., a connection between a user/node).

The pipeline uses Eq. 2 to transform a network dataset into a resistor network, which is optimized to uniquely identify nodes through capacitive sensing. Then our pipeline automatically generates the necessary CAD geometry (i.e., fabrication-ready network) that can be exported as STL files for 3D printing. After 3D printing, we can directly produce a

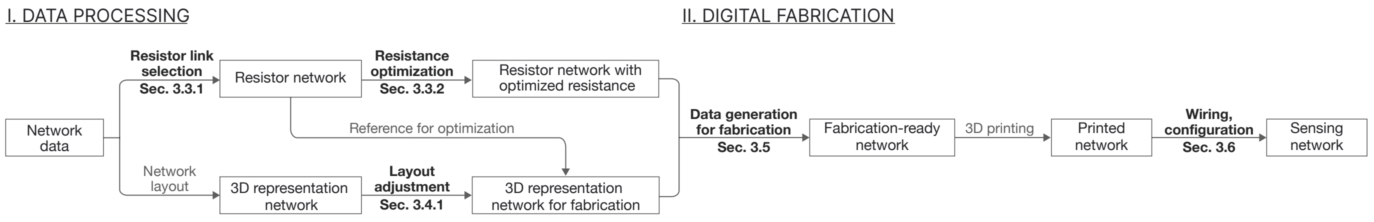
I. DATA PROCESSING
II. DIGITAL FABRICATION
Network data
Resistor link selection Sec. 3.3.1
Resistor network
Resistance optimization Sec. 3.3.2
Resistor network with optimized resistance
Reference for optimization
Network layout
3D representation network
Layout adjustment Sec. 3.4.1
3D representation network for fabrication
Data generation for fabrication Sec. 3.5
Fabrication-ready network
3D printing
Printed network
Wiring, configuration Sec. 3.6
Sensing network

Fig. 3: A computational design pipeline for producing sensing network physicalizations. Rectangles represent output data or physical objects (see Sec. 3 for definitions). Arrows represent different processes. Processes that are bolded are new methods that we introduce (Sec. 3.3–Sec. 3.6).

sensing network physicalization by connecting the printed network to an electrical circuit that is managed by a microcontroller (Fig. 2).

3.2 Computational Design Pipeline Overview

Our computational pipeline (Fig. 3) is divided into two stages: data processing (Sec. 3.3, Sec. 3.4) and digital fabrication (Sec. 3.5, Sec. 3.6).

Data processing. Data processing has two parallel tracks. The first track (top) designs the resistor network and the second track (bottom) adjusts the network layout to satisfy fabrication constraints. The resistor network design begins by selecting links from the network dataset to be used as resistors. We refer to the selected links as *resistor links*. Then our resistance optimization process identifies a suitable resistance for each resistor link such that the RC delay is different for every touched node. The network layout design begins by laying out the network dataset as a 3D representation network. For this process, we can use any existing network layout algorithms or use predefined positions specified by the source network data. Next, we apply a network layout adjustment method using neural networks. This adjustment is primarily to prevent conductive components (i.e., nodes and resistor links) from intersecting to avoid interference with the circuit (Fig. 5).

Digital Fabrication. The digital fabrication stage uses the outputs from the data processing stage to dynamically generate a fabrication-ready network (i.e., CAD files). This process involves computationally (i) drawing the resistor network as long, thin traces and (ii) updating the link geometry to structurally reinforce the connection between nodes and links. We fabricate our network using multi-material printing and then connect the printed network to a microcontroller.

3.3 Resistor Network Design

Designing the resistor network consists of selecting a subset of links to be used as resistor links and optimizing their resistance to achieve unique RC delays when a node is touched. Note that we only consider networks with a single connected component (i.e., no nodes are completely isolated from others).

3.3.1 Resistor Link Selection

Networks typically contain more links than nodes (e.g., 10 nodes, 30 links), but, in our approach, all links do not need to be resistors to electrically connect all nodes for sensing. See Fig. 1b. For a network with N nodes and L links, we only need to find $(N-1)$ resistor links that connect all N nodes. The minimum set of such links can be found by performing a traversal search such as breadth-first search (BFS) or depth-first search (DFS) over the network.

We use DFS to reduce the computational complexity of the resistance optimization process in Sec. 3.3.2. DFS selects a set of resistor links that generate fewer path branches compared to BFS. However, based on the starting node of the traversal, DFS may find a set of resistor links with more path branches. For example, in Fig. 4b, we show two different sets of resistor links selected from the network dataset in Fig. 4a. The green resistor links have the minimum number of branches (two branches: nodes 3–6 and 5–7). The red resistor links have three branches (nodes 3–6, 2–4, and 5–7). To find resistor links with a fewer number of path branches, we run DFS N times. Each iteration selects a different node as a starting node, and then we select the result with the minimum number of branches. Note that we use this DFS-based selection as a heuristic because finding the optimal solution (i.e., the fewest number of path branches) can be computationally expensive [31].

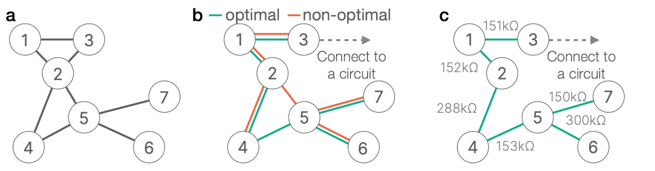

Fig. 4: Example of resistor network generation: (a) input network, (b) resistor network with optimal resistor links (green), while red links are non-optimal, (c) resistor network after assigning optimized resistance.

After determining the resistor links, we also select one or two leaf nodes as the connection point(s) to the microcontroller's circuit (e.g., Node 3 in Fig. 4). Then, for each resistor link, we calculate its appropriate resistance using the resistance optimization (cf. Fig. 4c).

3.3.2 Resistance Optimization

Optimization goal. We can identify which node is touched based on how long it takes for a capacitor to reach a certain voltage threshold (i.e., RC delay). When a user touches a node, a new connection to a capacitor (i.e., the user's body) and the ground is introduced into the electrical circuit. This connection varies based on the touched node (e.g., the orange vs. green dashed wires in Fig. 2b), inducing a different RC delay based on the circuit design. Although slight time delays are theoretically sufficient to recognize the selection of different nodes (e.g., 10 ns), we want to **maximize the minimum difference among the time delays** for fault tolerance in the microcontroller and 3D printer. First, we assume a microcontroller is used to capture time delays. The CPU clock of microcontrollers is typically slow (e.g., 16MHz for the Arduino Uno R3 [6]), which limits the precision of the measurement (e.g., 62ns for 16MHz CPUs). Second, errors can be introduced during the 3D printing process. Our optimization goal is formulated as:

$$\underset{\mathbf{r}\in\mathbb{R}^{N-1},\, r_{\min}\le r_k\le r_{\max}}{\arg\max} \quad \min\{d_{i,j}(\mathbf{r}) | 1\le i\le N, 1\le j\le N\} \tag{3}$$

where $\mathbf{r}$ is a vector of resistance assigned to $(N-1)$ resistor links, $r_{\min}$ and $r_{\max}$ specify the value boundary for each resistance r_k in $\mathbf{r}$, and $d_{i,j}(\mathbf{r})$ is the difference of the time delays between cases when nodes i and j are touched. We define the difference of the time delays as $d_{i,j}(\mathbf{r}) = \|t_i(\mathbf{r}) - t_j(\mathbf{r})\|$, where $t_i(\mathbf{r})$ is the time delay when node i is touched. We call t_i node i's *time delay function*.

The optimization for Eq. 3 has two challenges. First, the time delay function differs based on the selected node as touching a new node results in a different circuit. Consequently, we need to evaluate various different circuits. Second, we need to optimize $\mathbf{r}$, which has a large number of parameters (i.e., $N-1$). To address the first challenge, we leverage circuit simulators to computationally generate and evaluate a circuit corresponding to each touched node. For the second, we compute the derivative of each time delay function with respect to $\mathbf{r}$ and then apply gradient descent [69] to efficiently optimize all parameters.

Circuit generation and simplification. The pseudocode for our resistance optimization algorithm is shown in Alg. 1. The optimization starts by registering the circuit information (including the microcontroller's circuit and resistor links) with a circuit simulator (line 1). Afterward, we generate N different circuits, each of which corresponds to a case where one of N nodes is touched (line 3). Circuit simulators (e.g., SPICE [84]) typically have an analysis function to observe voltage

Algorithm 1 Resistance optimization.

Inputs: $\mathcal{N}$, network nodes; $\mathcal{R}$, resistor links; $\mathcal{M}$, microcontroller circuit information; C, artificial human capacitor; v_{thres}, the voltage threshold to measure the time delay; $[r_{\min}, r_{\max}]$, boundary of possible resistance
Outputs: $\mathbf{r}_{\text{best}}$, a set of optimized resistance

```
    Prepare touched circuits and corresponding symbolic functions
 1: Register $\mathcal{M}$ and $\mathcal{R}$ with a symbolic circuit simulator.
 2: for all $\mathcal{N}$ do in parallel
 3:     Create a touched circuit $\mathcal{M}'_i$ by adding C into $\mathcal{M}$.
 4:     Combine $\mathcal{M}'_i$'s resistors connected in series.
 5:     Get a mapping between resistance before ($\mathbf{r}$) and after ($\mathbf{r}'_i$) aggregation.
 6:     Generate a time-varying voltage function $v_i$ at the receive pin.
 7:     Obtain a symbolic time delay function $t_i$ by solving $\text{v}_{\text{thres}} = v_i(t_i)$.
 8: for $i = 1, \cdots, |\mathcal{N}|$ do
 9:     for $j = i+1, \cdots, |\mathcal{N}|$ do
10:         Compute the symbolic time delay difference function $d_{i,j} = \|t_i - t_j\|$.
11:         Compute $\nabla d_{i,j}$, the symbolic derivative of $d_{i,j}$ with respect to $\{\mathbf{r}'_i, \mathbf{r}'_j\}$.
    Iterative optimization using gradient descent
12: Initialize $\mathbf{r}$ (e.g., assigning random values in a range of $[r_{\min}, r_{\max}]$).
13: while not converged do
14:     Evaluate Eq. 3 with current $\mathbf{r}$ and keep the best result so far as $\mathbf{r}_{\text{best}}$.
15:     Find a bottleneck pair of nodes ($x$, $y$) that produced the minimum $d_{i,j}$.
16:     Update $\mathbf{r}$ based on $\nabla d_{x,y}(\{\mathbf{r}'_x, \mathbf{r}'_y\})$ and the mapping between $\mathbf{r}'_i$ and $\mathbf{r}$.
17:     Clip $\mathbf{r}$ to fit resistance into the boundary $[r_{\min}, r_{\max}]$.
18: return $\mathbf{r}_{\text{best}}$
```

changes. We thus avoid manually deriving all necessary equations and functions to calculate the time delay caused by each touch.

Among a variety of circuit simulators, we employ a simulator (e.g., Lcapy [36]) that accepts symbolic computations [17, 56] to derive derivatives. In symbolic computations, we can use mathematical symbols as-is to solve equations and compute derivatives without converting them to numerical values. Consequently, we can obtain precise derivatives for the optimization and also perform an efficient optimization by reusing the same symbolic functions across iterations.

Symbolic computing still suffers from computational costs when several symbols are involved, particularly when we compute the matrix inversion operation required for solving our optimization equations. The time complexity for this matrix inversion operation is $\mathcal{O}(S^3)$ where S is the number of symbols. To mitigate this complexity, we utilize the fact that resistors connected in series can be simplified as one resistor [15]. Given three resistors (i.e., r_1, r_2, r_3), the symbolic computation without combining these resistors would involve three symbols. In contrast, combining three resistors with $r' = r_1 + r_2 + r_3$ would only involve r' in the computation.

Combining resistors can radically reduce computational costs. To maximally reduce the number of symbols, we use the DFS-based resistor link selection as discussed in Sec. 3.3.1. We combine resistors for each of the N generated circuits, derive their voltage change functions, and solve each function for a given voltage threshold (lines 4–7 in Alg. 1). We further speed up these processes by solving the functions in parallel. For the circuit induced by touching node i, we denote a vector of resistance of the combined resistors as $\mathbf{r}'_i$, a bijective mapping from $\mathbf{r}$ to $\mathbf{r}'_i$ as $f_i : \mathbf{r} \to \mathbf{r}'_i$ and its inverse mapping as $f_i^{-1} : \mathbf{r}'_i \to \mathbf{r}$. Note: $|\mathbf{r}'_i| \ll |\mathbf{r}|$.

Gradient-based optimization. From all pairs of the time delay functions, we symbolically derive the time delay difference (i.e., $d_{i,j}$) and its derivative ($\nabla d_{i,j}$) with respect to $\{\mathbf{r}'_i, \mathbf{r}'_j\}$ (lines 8–11). We then assign concrete values to $\mathbf{r}$ and iteratively optimize these values.

We first initialize $\mathbf{r}$ with user-specified or random values within a range of $[r_{\min}, r_{\max}]$ (line 12). We evaluate Eq. 3 with the current $\mathbf{r}$ (line 14) by inserting the values of $\mathbf{r}$ into $d_{i,j}$ while referring to the mappings f_i and f_j. We then update $\mathbf{r}$ to improve the minimum time delay difference with gradient descent. To achieve this, we first identify the bottleneck $d_{x,y}$, causing the minimum time delay difference (line 15). We then update $\mathbf{r}$ based on $\nabla d_{x,y}(\{\mathbf{r}'_x, \mathbf{r}'_y\})$. However, the obtained gradients are for the combined resistors. Thus, we evenly distribute each combined resistance's gradient to the set of original resistor links by referring to f_x^{-1} and f_y^{-1}. For example, when a combined resistor consists of $\{r_1, r_2, r_3\}$ and its gradient is 3, we assign 1 as a gradient for each of $\{r_1, r_2, r_3\}$. For convenience, we denote the gradients after this conversion as $\text{grad}(d_{x,y})(\mathbf{r})$. We can formulate the update of $\mathbf{r}$ by

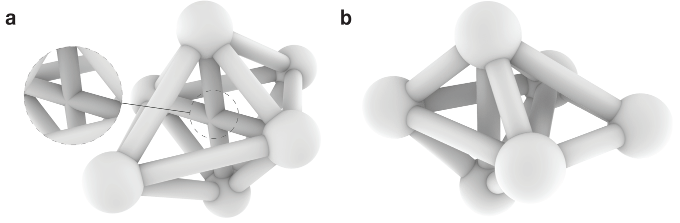

Fig. 5: A comparison of network layouts before and after adjustment. Two links intersect, creating potential circuitry issues in (a). Spatial adjustment resolves this issue (b).

gradient descent (line 16) as:

$$\mathbf{r} \leftarrow \mathbf{r} - \alpha\, \text{grad}(d_{x,y})(\mathbf{r}) \tag{4}$$

where $\alpha \in \mathbb{R}$ is a step size to control the pace of the optimization (e.g., 1% of $(r\_max - r\_min)$). We then clip each resistance of $\mathbf{r}$ in the user-specified boundary, $[r_{\min}, r_{\max}]$ (line 17). We repeat until the optimization reaches convergence and then use the value of $\mathbf{r}$ that achieves the best result for Eq. 3 (lines 13–18).

3.4 Network Layout Design

For the network layout design, we generate a 3D representation network. Our pipeline conducts post-hoc adjustments that work with any existing network layout algorithm. By default, we use the force-directed Fruchterman-Reingold layout [28]. We can also use predefined positions if they are available (e.g., networks with molecular structures, brain regions, geospatial information). This process extends to non-network visualizations deconstructed as hubs and struts (e.g., vertices in a Voronoi diagram, points in a line graph) by using the same process as a network with predefined positions. In this work, we mainly consider networks only with straight links.

3.4.1 Network Layout Adjustment

The 3D representation network generated from the above process might have an undesirable structure for fabrication. Since most existing layout algorithms do not consider 3D volumes of nodes and links, the conductive nodes and resistor links might intersect with each other, resulting in unexpected changes in the circuit. Thus, we introduce a network layout adjustment algorithm to adjust network geometry to avoid such issues. Fig. 5 shows an example of 3D representation networks before (Fig. 5a) and after (Fig. 5b) applying our layout adjustment.

Optimization goals. Our layout adjustment has the following goals:

- Primary goal: All resistor links *must* not intersect with each other. This restriction also applies to all nodes.
- Secondary goal: The adjusted layout should preserve the structure of the original layout as much as possible.
- Optional aesthetic goals:
  - Intersections of non-conductive links should also be minimized.
  - Link lengths should be closely uniform [10].

To efficiently satisfy these multiple goals, we employ a neural network (NN). An NN enables us to conveniently perform gradient-based optimization over loss functions corresponding to these goals [2]. The optional goals do not fulfill all graph aesthetic criteria (e.g., maximizing link angles). However, our NN approach can readily extend to support such options by designing additional corresponding loss functions.

Loss functions. Node-link diagrams are intuitively represented as cylinders (links) and spheres (nodes). We can detect the intersection of two cylinders if the minimum distance of their axes is smaller than the sum of their base radii. Similarly, two spheres intersect if the distance between their centers is smaller than the sum of their radii. Let $\text{dist}(\cdot)$ be a generic function computing the distance of links' cylindrical axes or the distance of nodes' spheres; $\text{radius}(\cdot)$ be a generic function returning a radius of the link's cylinder base or node's sphere. We define an intersection loss for all links and nodes as:

$$J_{\text{int}} = \sum_i \sum_{j, j \neq i} \text{ReLU}\left(\text{dist}(e_i, e_j) - \text{radius}(e_i) - \text{radius}(e_j)\right) \tag{5}$$

where $\{e_i\}$ is a set of nodes or links and $\text{ReLU}(\cdot)$ is the rectified linear unit function ($\text{ReLU}(x) = \max(0,x)$). We use ReLU, instead of thresholding, to train the NN more easily. If there is no intersection, ReLU returns 0; otherwise, it returns the intersected length. We compute the intersection loss separately for resistor links ($J_{\text{int}}^{\text{res}}$), non-resistor links ($J_{\text{int}}^{\text{non\_res}}$), and nodes ($J_{\text{int}}^{\text{node}}$).

Similar to Wang et al. [88], we measure the network layout change before and after adjusting node positions with Procrustes distance. Procurstes distance computes the sum of Euclidean pairwise distances of nodes after aligning two sets of node positions with translation, uniform scaling, rotation, and axis flipping [29]. This measure can eliminate the influence of non-essential layout differences. Let $\mathbf{P}^{\text{o}}$ and $\mathbf{P}^{\text{u}}$ be the node positions of the 3D representation network before and after the layout adjustment. Then the loss for the layout change can be written as: J_{pos}=Procrustes($\mathbf{P}^{\text{o}}, \mathbf{P}^{\text{u}}$) where Procrustes($\cdot$) computes the Procrustes distance. Lastly, we measure a loss corresponding to non-uniform link lengths, J_{len}, as the standard deviation of link lengths.

NN architecture and optimization procedure. We use an NN to find appropriate updated 3D node positions, $\mathbf{P}^{\text{u}}$. Thus, the NN outputs $\mathbf{P}^{\text{u}}$ from I_N, an input identity matrix with the size of N. We also input the node and link information required to compute the loss functions. We expect most network physicalizations to be relatively small (e.g., less than 1000 nodes). Therefore, we use a simple multilayer perceptron (MLP) consisting of three 100-neuron hidden layers by default.

Our training consists of two phases: (i) learning the original layout and (ii) adjusting the layout. The first phase learns neuron weights such that $\mathbf{P}^{\text{u}}$ becomes the same layout as $\mathbf{P}^{\text{o}}$ by minimizing J_{pos} during training. The second phase then adjusts the learned neuron weights to satisfy our optimization goals. For this second phase, we minimize the sum of multiple weighted loss functions. To place a higher priority on removing intersections, we assign larger weights to $J_{\text{int}}^{\text{res}}$ and $J_{\text{int}}^{\text{node}}$. For the loss functions corresponding to the secondary goal of layout preservation and optional aesthetic goals, we can select weights based on the design needs. For example, when generating the network layout shown in Fig. 1a, we used a loss function, $3J_{\text{int}}^{\text{res}} + 3J_{\text{int}}^{\text{node}} + J_{\text{int}}^{\text{non\_res}} + J_{\text{pos}} + J_{\text{len}}$. If $J_{\text{int}}^{\text{res}} > 0$ or $J_{\text{int}}^{\text{node}} > 0$, the optimization result does not satisfy the primary goal. To resolve such cases, we can retrain the NN with larger weights for $J_{\text{int}}^{\text{res}}$ and $J_{\text{int}}^{\text{node}}$ and/or reduce the links' base radii relative to the node radii.

3.5 Data Generation for Fabrication

The digital fabrication stage creates the necessary CAD files to 3D print the updated network. We generate a fabrication-ready network by embedding the resistor network into the 3D representation network.

Resistor network embedding. Nodes are fabricated with a conductive filament to provide high conductivity. To materialize resistor links, they need to have high resistance within a limited volume of each cylindrical link. This is accomplished by drawing a thin, long trace of the conductive filament using a serpentine trace pattern [77]. Given a surface area, the resistivity is maximized by drawing on the xy-plane layers with a serpentine trace pattern and connecting the endpoints in the z-direction line, resulting in a 3D zig-zag structure (Fig. 1b).

To achieve the resistance specified in the resistor network, we determine the length of each resistor link's trace based on the resistivity law (Eq. 1). Due to the printing resolution of most FDM printers, we suggest using different thicknesses for the line on xy-plane and the line along z-direction. The line thickness for the xy-plane can be close to the printer's nozzle extrusion width (e.g., 0.4 mm), while the thickness for z-direction should be at least twice the extrusion width (e.g., 0.8 mm) to ensure contact from the previous layer. A conductive filament's resistivity (i.e., ρ in Eq. 1) along the xy-plane and z-direction is typically provided by the filament makers. Then, with the given line thickness, filament's resistivity, and link's cylindrical shape (radius, angle, and length), we can computationally identify the appropriate length and drawing pattern of the trace. However, various external conditions may influence the line area and resistivity (e.g., printing precision, nozzle temperature, and filament production errors). Thus, we recommend identifying resistance per length (i.e., ρ/a) for the xy-plane and z-direction under the expected printing condition. The supplementary materials include our process used to obtain this information.

Structural support. Our sensing network is a physical interface, and thus needs to be structurally sound. We add cone-shaped structural support to the edges to increase contact.

3.6 Wiring and Calibration

We 3D print a fabrication-ready network to produce a printed network. We connect it to a microcontroller circuit using the same schematic diagram as Fig. 2a. We calibrate time delays corresponding to all nodes by manually touching each node and observing the time required to reach a microcontroller's logic threshold voltage (e.g., 2.5V in Fig. 2c). Calibration is necessary as each individual and external factors (e.g., clothing, temperature) may generate a different capacitance [34].

3.7 Implementation

We implemented the data processing stage with Python 3 and libraries for matrix computations such as NumPy/SciPy [87]. We used NetworkX [35] for the DFS-based resistor link selection. For the resistance optimization, we used Lcapy [36] and SymPy [56] for the circuit simulation and symbolic computation and Pathos for multiprocessing. The network layout adjustment is implemented with PyTorch [60].

For the digital fabrication stage, we used Rhinoceros 7 [66] as the CAD software and its programming environment, Grasshopper [19], to computationally build 3D models. We used a Prusa i3 MK3S+ 3D printer coupled with a Mosaic Palette Pro 2 to enable multi-material printing. Our conductive filament is Protopasta's conductive PLA (1.75 mm) [64]. 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. We used iSANMATE Wood Filament PLA+ (1.75 mm) [45] to emphasize color contrast between the nodes and links. For our microcontroller, we tested both the Arduino Uno R3 (16MHz CPU) and R4 WiFi (48MHz CPU) [6]. Both have a 5V power source and a 2.5V logic threshold. To constantly measure the time delays, we utilized signals from the microcontroller's digital I/O pins [1].

4 Evaluations

We evaluate our computational pipeline in three ways: (i) a quantitative scalability evaluation, (ii) three usage scenarios driven by three general visualization tasks, and (iii) an interview with six domain experts.

4.1 Scalability Evaluation

We conduct a two-part scalability evaluation. First, we establish the practical performance limitations of the data processing stage (i.e., resistor network and network layout design) for networks of various sizes and confirm the effectiveness of our resistance optimization in determining a user's touch selection (Sec. 4.1.1). Second, we analyze how large of a network we can fabricate with our approach (Sec. 4.1.2).

4.1.1 Computational Scalability

Time complexity analysis. The DFS for resistor link selection $\mathcal{O}(N(N+L))$ where N and L are the numbers of nodes and links, respectively. The main computation for resistance optimization is the matrix inversion, which is required to solve the equation for each touched node (line 7 in Alg. 1). This has $\mathcal{O}(NS^3)$ where S is the number of involved symbols. The number of symbols is linearly correlated to the number of resistor path branches, B (e.g., B=2 for Fig. 4c). We can thereby rewrite the time complexity as $\mathcal{O}(NB^3)$. The network layout process's time complexity depends on existing algorithms. Our default layout is the Fruchterman-Reingold layout, which has $\mathcal{O}(N^2+L)$. Lastly, the network layout adjustment's time complexity varies based on which loss functions are employed. $J_{\text{int}}^{\text{res}}$ has $\mathcal{O}(N^2)$ as it involves comparing all resistor links. Similarly, $J_{\text{int}}^{\text{non\_res}}$ is and $J_{\text{int}}^{\text{node}}$ have $\mathcal{O}(L^2)$ and $\mathcal{O}(N^2)$, respectively. The time complexities for J_{pos} and J_{len} are almost negligible compared to the other loss functions. For each iteration performed by an NN, the network layout adjustment has $\mathcal{O}(N^2)$ at minimum as we must include $J_{\text{int}}^{\text{res}}$ and $J_{\text{int}}^{\text{node}}$. When using all the loss functions, the time complexity becomes $\mathcal{O}(N^2+L^2)$.

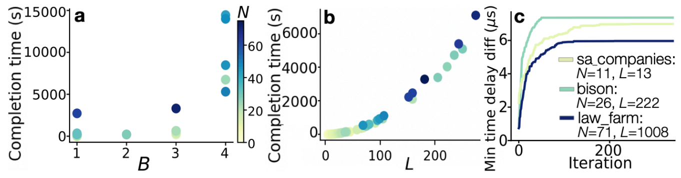

Fig. 6: Computational evaluation results: (a, b) performance of resistance optimization and network layout adjustment; (c) progress of resistance optimization over iterations. Because of long completion time, we only show the results for networks with $B \leq 4$ in (a) and networks with $L \leq 300$ in (b). For (c), we only show the results for three different networks.

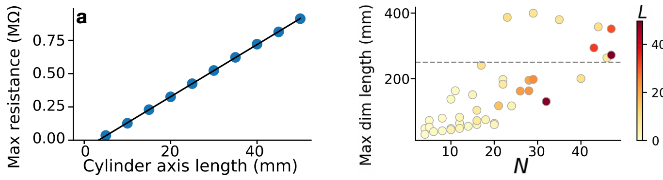

Fig. 7: Fabrication scalability analysis: (a) the maximum resistance we can create for different cylinder lengths; (b) the longest dimension lengths required for printing space, where 250 mm corresponds to the maximum printable size using a Prusa i3 MK3S+.

From this analysis, the network layout adjustment to avoid link intersection has the largest computational cost ($\mathcal{O}(N^2+L^2)$). The matrix inversion, which uses symbolic computing, also usually requires time-consuming computations even when the number of symbols is very small (e.g., 1000 s when S=10 [36]). Resistance optimization ($\mathcal{O}(NB^3)$) can be a bottleneck when a network dataset has a large number of path branches, B (e.g., a network with many star shape sub-networks).

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 networks from the graph-tool's dataset collection [62] and the Netzschleuder network repository [63] to evaluate our pipeline with various real-world and synthetic networks. Given current limitations on 3D printing resolution, we selected all networks with less than 100 nodes, resulting in 57 networks with the ranges of N: [4, 96] and L: [5, 2539]. For layout adjustment, we measured the time needed to process a network with all loss functions.

As expected from the time complexity analysis, the resistance optimization and the network layout adjustment had longer completion times than other pipeline phases. The friendship network of New Guinea Highlands tribes (N=16, L=58) took 0.7 ms for resistor link selection, 50 s for resistance optimization, 4 ms for network layout, and 300 s for network adjustment. From this bottleneck, we focus on evaluating the completion times of the resistance optimization and network adjustment.

The results of resistance optimization and network adjustment are shown in Fig. 6a and b. The x-axis reflects the most influential variable for their time complexity (e.g., B for resistance optimization due to $\mathcal{O}(NB^3)$). The sequential colormap reflects N as the secondary influential variable. These results follow the theoretical time complexity. The resistance optimization finished in approximately 4 hours for a network with N=47 and L=504 (B=4). The retwork layout adjustment completed in 2 hours for a network with N=70 and L=274. While these completion times are nontrivial, it would take much longer and be significantly more error-prone to design appropriate resistance and layout by hand.

Lastly, we evaluated the quality of resistance optimization to examine its effectiveness in practically distinguishing touched nodes with a microcontroller. We initialized the resistance of resistor links randomly within a range of [50 kΩ, 300 kΩ] and performed optimization. Fig. 6c shows the transition of the minimum difference among the time delays. Optimization significantly increases the minimum difference (e.g., from 0.8 μs to 7.4 μs for the bison network) and quickly converges. As mentioned in Sec. 3.3.2, our resistance optimization is designed such that it maximizes the minimum difference of the time delays. An order of magnitude increased time delays significantly improves the sensitivity of node recognition. This improvement enables the Arduino UNO R4 to further distinguish touched nodes by the difference of 355 clock cycles (i.e., 7.4 μs) instead of 38 clock cycles (i.e., 0.8 μs).

4.1.2 Fabrication Scalability

To determine the largest network we can fabricate, we assume a 3D printer with a 0.4 mm nozzle (standard for consumer FDM 3D printers) using Protopasta's conductive PLA (1.75 mm). We first measured the maximum resistance we can produce with the conductive traces using the serpentine trace pattern. Because the maximum resistance depends on the cylindrical volume of a resistor link, we applied our resistor network embedding method to cylinders with different axis lengths and a fixed base radius of 3 mm. In Fig. 7a, we see a linear relationship between the maximum achievable resistance and the cylinder length.

Next, we identified the required resistance for sensing networks' resistor links to be able to distinguish each selected node. To keep the evaluation concise, we assume the following: the sensing networks are paired with the Arduino Uno R4 (one clock cycle per 21 ns); the time delay to distinguish a touched node is at least a difference of 2.1 μs (100 clock cycles); the involved capacitance for each touch selection is 100 pF (a representative value for a human body [25]); and all resistor links are connected in series. With these assumptions and Eq. 2, the derived required resistance for each resistor link is 30 kΩ.

We use this resistance value to derive the required print sizes to fabricate sensing networks. As shown in Fig. 7a, the minimum cylindrical length to satisfy 30 kΩ is 17 mm. We set the radius of each node to be 6 mm (twice the base radius of a cylinder). Following our network layout and adjustment processes, we laid out the networks such that their minimum resistor link length matched the identified cylinder length. Note that we only used $J^{\text{res}}_{\text{int}}$ and $J^{\text{node}}_{\text{int}}$ to avoid intersections of conductive materials. We then obtained the maximum distance of all possible pairs of nodes to compute the required length for the printing area.

Based on these computations (Fig. 7b), we estimate that the largest printable sensing network size is between 20–30 nodes when using a consumer 3D printer such as the Prusa i3 MK3S+. This result implies that the scale of our approach is currently limited by the size of the 3D printer rather than the computational complexity of the algorithms.

4.2 Usage Scenarios

We present three example usage scenarios to illustrate how our sensing networks can be used in practice. These usage scenarios motivate how selection can support three general visualization tasks: exploration, explanation, and analytic provenance. The scenarios are contextualized in the educational context of a student exploring disease spread using a social network, and are driven by the concept of *exploranation* [93] (i.e., how exploratory and explanatory visualization techniques can inform each other). The exploratory case demonstrates how our approach aligns with emerging uses of tangible 3D input devices [12, 42].

Various studies confirm the benefits of using 3D input devices, such as reducing mental effort, compared to the traditional keyboard and mouse setup [11, 41, 70]. The explanatory case aligns with how physical models have also been used to richly explain abstract concepts (e.g., visualization [7, 43], biology [5]) for educational purposes and general science communication [68, 75]. The third usage scenario emphasizes our network physicalization's capabilities as a sensor. Building upon literature on analytic provenance [30, 65], our networks can richly and passively capture how users are interacting with the data physically, serving as a tangible data log for their physical navigation of the data.

We derived these scenarios from discussions with subject matter experts who use networks in their everyday practice (e.g., chemistry, computational biology, network science). The usage scenarios summarize core functionality that experts currently rely on alternative solutions to support and demonstrates the broad utility of our sensing network by enabling serial communication with other devices. Each scenario is based on the parameters described during our interviews (Fig. 8), and uses the network physicalization in Fig. 1. Details of these discussions are stated in Sec. 4.3. See the supplemental video for demo [1].

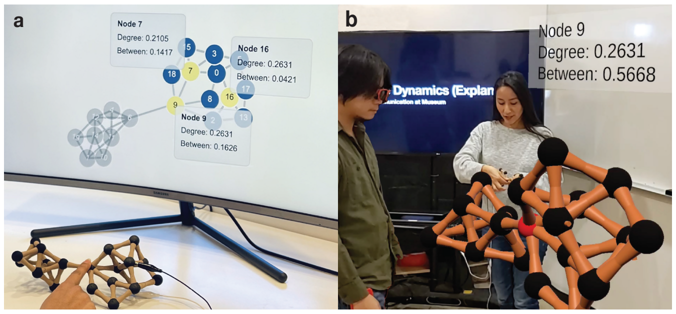

Fig. 8: Three usage scenarios (Sec. 4.2) with a sensing network physicalization ($N = 20$, $L = 40$; see Fig. 1). (a) Exploratory: Alice analyzes the network using graph centrality and concludes Node 9 should be vaccinated; (b) Explanatory: Alice touches Node 9 to explain her findings. Network is serially connected to audiences' AR glasses and mobile AR applications. Note: displayed degree and betweenness centralities are normalized by dividing by the highest possible values for this network.

4.2.1 Exploration and Discovery

Background. Students can learn about viral transmission through social networks, which model the connections between students. A professor prepares a synthetic social network ($N = 20$, $L = 40$) where each student node has a set of attributes such as age, gender, and number of social connections. The network is represented in 2D (digital) and 3D (physical). Students apply graph theory concepts to analyze the social interactions that determine disease dynamics.

A student named Alice assesses the network to identify individuals who have a high potential to spread the disease. Her task is to determine which centrality measure would lead to the best vaccination strategy. She looks at the 2D and 3D representations of the same network. With both representations, she immediately notices there are two distinct clusters, but the link intersections of the righthand cluster are difficult to distinguish on screen. Alice connects the physical model to a desktop visualization for navigation and initial exploration. She wants to find Node 2 who is identified as the one infected person. The desktop currently displays all nodes and their ID. She notices Node 2 is within the right cluster. She **filters** out all nodes in the left cluster by selecting each node. These filtered nodes turn grey and their opacity reduces.

Alice double-taps all the nodes within the righthand cluster to **inspect local details** for their centrality measures. A tooltip appears. She analyzes the degree centrality of each node. Nodes 7, 9, 16 have high degree centrality, representing individuals with many connections in the networks. Alice holds onto these respective nodes for three seconds to **highlight** them in yellow (Fig. 8a). She then analyzes the betweenness centrality of the highlighted nodes, which represents the extent to which individuals lie on the shortest paths between pairs of other individuals in the network. She **compares** the betweenness centrality among the three nodes. From the tooltip, Node 9 has the highest betweenness centrality. She concludes that Node 9 should be vaccinated.

4.2.2 Explanation and Communication

Alice presents her findings (i) to her classmates and teachers remotely and (ii) during a live presentation at a science center.

Remote presentation. Remote presentations lack shared physical context, making deictic gestures, such as pointing to items of interest, difficult to interpret [86]. To provide context for her findings, Alice uses her model as a physical prop during a video call with her class to present her findings. She can use deictic gestures to indicate key nodes and links during her presentation by pointing, with the physical model providing a shared context between Alice and her remote audience. Alice can also use the desktop visualization as a digital twin of the network [40] and use the sensing network to interact with the twin. As she touches different nodes, the sensor network allows the twin to highlight those nodes in red and display additional information about the corresponding student in the network. When she releases the node, the information disappears, allowing her to continue her discussion.

Science communication in public museums. Science communication at museums or local science centers bridges researchers and the general public. Visitors can engage in research data with algorithms and software similar to what was used by researchers. However, public displays impose installation challenges in terms of reliability and interaction. To amend this challenge, museums often use touch surfaces that enable robust and intuitive multi-user interaction [94]. Despite the documented success of touch interfaces, tangible interfaces and physicalizations can better scaffold knowledge and promote learning [75]. Alice uses her physicalization in a workshop presentation at the local science center. AR headsets and mobile AR are used to display augmented information on the network model. She presents her findings and lets the participants handle the model and study the responses to the augmented information. Rather than relying on the cameras on headsets and phones for interaction, which suffer from occlusion, she uses her digital twin configuration to bridge the physical model with the audience members' devices. As Alice is walking through her data exploration (Fig. 8b), the sensing capabilities of the network inform the headsets of the different node selections and display the corresponding node information. Audience members can see her node selections even when the model is outside of the headset's and phone's field of view.

4.2.3 Provenance and Documentation

As Alice explores and explains her data, she physically interacts with the network. What nodes she touched, for how long, and in what order can all be logged to replay her physical interactions with the data [30]. The innate sensing capability challenges conventional approaches to capture interactive data exploration (i.e., data sensemaking). Based on Alice's exploration (Sec. 4.2.1), her professor analyzes her logged interactions to understand her data sensemaking process. He wants to use this information to better inform his teaching curriculum. He visualizes the logged data as a timeline visualization and a heatmap.

In the visualizations, he notes that Alice kept "bookmarking" [24,48] Node 9 by leaving her finger in place. The lingering action confirms that Alice was thoroughly exploring the righthand cluster. He notices that she only engaged with the left cluster only in the beginning of her exploration. He plans to note that if Alice did not immediately filter out the left cluster's nodes, she would have noticed that Node 6 also has a high betweenness centrality and thus should also be vaccinated.

4.3 Expert Discussion on Utilization

We discussed our sensing networks with six domain experts (1 female, 5 males; age reported in bins of 26–55 years). Their expertise varied in AR/VR (3), material science (1), computational biology (1), and high-dimensional data (1). Four are working professionals in academia or national labs. Two are Ph.D. students. Two interviews were conducted remotely and four were in person. See the supplementary materials [1] for more details.

We first explained the basic principles of the pipeline (Sec. 3.1) and demonstrated the networks either in-person or remotely. Within the physical condition, we asked the domain experts to interact with the physical network themselves and demonstrated how it can be connected to other devices. Within the remote condition, we demonstrated the sensing capability by screen-sharing the model and playing videos we captured in advance. We then asked questions designed to elicit both insights into potential uses as well as preliminary perceptions of the pipeline and resulting networks.

Overall, the domain experts reacted positively to our produced networks, with E6 (an AR/VR developer) saying, "*Currently, there are only a few ways objects can meaningfully respond to touch, and your technique is changing that [paradigm].*" Five of the six experts noted the sensing network's potential for supplementing existing exploratory workflows. E1 (expert in uncertainty in high-dimensional data) expressed how it could potentially be used as a parameter tuning control for high-dimensional spaces (e.g., producing an interactive Morse-Smale Complex [32]). E1 felt she could "*play with the graph of the [high-dimensional] parameter space to [move] to new locations in the output space, [which can] give us a better understanding of how parameters are related and ways to navigate the parameter space.*" E2, a material scientist, uses CAVEs to immersively see static 3D molecular compounds (e.g., solar cells). However, he often faces difficulties interacting with the CAVE to filter certain sub-components to visualize further. He envisioned that the networks can possibly act as interactive "motifs" (similar to graphlets [76]) within the large molecular polymers he analyzes. The network's sensing capabilities would enable more intuitive interactions, while the pipeline would enable him as a domain scientist to readily produce polymer physicalizations on demand.

Experts who develop AR/VR provided similar comments on how this can help supplement existing interaction workflows for immersive spaces. E4-E6, all AR/VR developers, stated that the ability to produce haptic objects that share the form of critical virtual objects and can act as input sensors will benefit the field. E6 specified that the biggest issues with current haptic proxies stem from how they are either completely passive or require too much instrumentation to make them interactive. Our technique would enable developers to more easily produce responsive controllers shaped like networks or other hub-and-strut forms, which can complement immersive analytics tools.

E3 (a computational biologist) and E6 provided useful feedback for future work. Despite explaining the scalability of the technique, E3 noted that, "*I'm not sure if I want to even hold a 50+ node network*". This comment aligns with findings that the physical scale of a physicalization needs to be chosen carefully for ease of manipulation and representation legibility [54]. E6 pointed out how the current network could be more powerful by being modular and reconfigurable.

5 Discussion

Need for computational fabrication processes. By concurrently generating data representations (form) alongside sensing capabilities (interaction), our pipeline addresses several low-level engineering challenges associated with physicalization production (e.g., layout, circuit design, and sensor integration) for networks and any visualization using a hub-and-strut structure (see Sec. 3.4). This design paradigm shift enables integrating sensing capabilities without disturbing a physicalization's form and can help achieve more complex physicalizations.

Past work in visualization authoring tools (e.g., D3 [14] and Vega-lite [71]) parallels our goal to make it easier to produce interactive data representations. The power of these toolkits is their ability to concurrently consider interactions and visual representations, similar to our pipeline. However, prior to these tools, developers used systems designed for general graphics applications, which lacked support for designing interactive visualizations. Consequently, developers had to develop custom solutions for common operations (e.g., mapping data to visual elements and event handlers). Digital toolkits reduced the time and labor associated with visualization development, leading to readily reusable and reliable components. Toolkits for physicalizations have the potential to achieve similar goals, enabling developers to focus more on *what* analytical scenarios they want to explore with a physicalization rather than *how* to build one. We contend that our pipeline can provide an important step towards more robust toolkits for interactive physicalization. As a result, this work lowers the barrier of entry for people to incorporate physicalizations into their work.

Enabling output. Interactive objects receive input (e.g., from touch) and produce output (e.g., light, sound, color change) in a controlled manner. Our sensing network currently addresses the first part of the interaction loop by responding to touch inputs and can serve as a foundational step to truly realize interactive physicalizations. One way to enable display output in our sensing networks is to integrate trace routing approaches for 3D printed optical fibers [91] or color-changing dyes [49]. Color is one of the main visual channels when representing data, and the existing work in this space (e.g., visualizations with color-changing inks [61]) suggests its potential for physicalizations. Another possibility is to explore sound output with 3D printed speakers [46], which lends well to existing research efforts on sonification [39].

Working alongside other visualizations. Our computational pipeline currently supports selection by touch through capacitive sensing and computational inference. Though limited, select is a fundamental interaction primitive (Sec. 2.1) that enables designers to implement more complex interactions at the application layer (see Sec. 4.2). Currently, the output of our interaction loop is achieved through digital visualizations. Though future work should investigate how to enable physical output, experts saw value in this current digital-physical paradigm. This value stems from how our experts work with different media (e.g., CAVEs, desktops, AR/VR headsets) and how our sensing network physicalizations can be easily integrated with these different devices through serial communication. This integration allows systems to simultaneously leverage the responsiveness of digital displays with the intuitive interactions and cognitive benefits of physicalizations.

Nonetheless, future work is necessary to see which display methods (e.g., tabletop, shape-changing, AR/VR) best complement interactive physicalizations. In turn, this hybrid modality can shed a more critical perspective on analyzing the strengths and weaknesses of digital and physical approaches, possibly leading to a stronger post-WIMP (windows icon menu pointer) paradigm [47, 67]. Future work should continue to investigate how to integrate input and output modalities into physicalizations by concurrently considering form and interactivity.

6 Future Work

Improving scalability. Our fabrication process is the primary bottleneck for scalability (Sec. 4.1.2). While FDM 3D printers are more affordable and approachable to end-users, they have a smaller build volume, slower speed, and less printing resolution compared to other processes [58]. For example, the network shown in Fig. 1 (N=20, L=40) took 50 hours to print. A multi-nozzle 3D printer [85] could print much larger networks at a faster rate while having more precision in the design of conductive traces. This process can help fully explore the benefits of larger, physical networks (e.g., extracting depth information) while still supporting interactivity. To support higher precision and larger scale, our algorithms' performance must also improve. We could improve resistance optimization by simplifying resistor links connected in parallel. This would require future research into distributing gradients for the combined resistor to the original resistor links.

Improving calibration. Currently, our sensing technique requires manual calibration for each printed network and user due to the differences in capacitance. In addition, external environmental factors (e.g., temperature, surrounding materials) can affect the capacitive sensors. One way to partially automate this process is to measure a user's capacitance as they touch a few nodes and compute the corresponding time delays with the circuit simulator used for resistance optimization. However, this optimization would require high precision while printing conductive traces to ensure the sensing network and the circuit simulator have the same resistance for each resistor link.

Improving design flexibility. Our approach currently does not support link selection. To support concurrent node and link selection, we need to imbue both surfaces with high conductivity while creating conductive traces within. These constraints bring additional complexity that will need to be addressed by the resistance optimization, layout adjustments, and fabrication process. We also plan to support different shapes for nodes and links to allow more flexible physical encodings, such as arbitrarily-shaped nodes, curved and bundled links. These designs can take further advantage of the computational process of 3D printing, which can help physicalizations represent high-dimensional, multivariate datasets. While the nodes of our sensing networks are uniform spheres, we can add additional visual channels by varying their size, color, texture, and other physical and visual properties.

7 Conclusion

We introduce a computational design pipeline to 3D print network physicalizations with integrated sensing capabilities. This pipeline inputs network data, computes the internal circuitry for capacitive sensing, adjusts the layout to support fabrication, and produces a sensing network physicalization. Our methodology concurrently considers form and interactions for network physicalizations. By automating low-level hardware design challenges in favor of application-level implementations, our approach can create more complex and powerful tools for designing physicalizations. With this design paradigm shift, we can produce generalizable techniques that lower the barrier to physicalization research, creation, and adoption.

Acknowledgments

This research is sponsored in part by the U.S. National Science Foundation through grants IIS-2040489, IIS-1764089, IIS-2320920, IIS-1933915, IIS-2233316, 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.

References

[1] Supplemental materials. https://sandrabae.github.io/sensing-network, 2023.

[2] R. Ahmed, F. De Luca, S. Devkota, S. Kobourov, and M. Li. Multicriteria scalable graph drawing via stochastic gradient descent, $(SGD)^2$. *IEEE Trans Vis Comput Graph*, 28(6):2388–2399, 2022. doi: 10.1109/TVCG.2022.3155564

[3] K. Allahverdi, H. Djavaherpour, A. Mahdavi-Amiri, and F. Samavati. Landscaper: A modeling system for 3D printing scale models of landscapes. *Comput Graph Forum*, 37(3):439–451, 2018. doi: 10.1111/cgf.13432

[4] R. Amar, J. Eagan, and J. Stasko. Low-level components of analytic activity in information visualization. In *Proc. INFOVIS*, pp. 111–117. IEEE, USA, 2005. doi: 10.1109/INFVIS.2005.1532136

[5] K. D. Ang, F. F. Samavati, S. Sabokrohiyeh, J. Garcia, and M. S. Elbaz. Physicalizing cardiac blood flow data via 3D printing. *Comput Graph*, 85:42–54, 2019. doi: 10.1016/j.cag.2019.09.004

[6] Arduino. Arduino Documentation. , 2023. Accessed: 2023-03-24.

[7] S. S. Bae, R. Vanukuru, R. Yang, P. Gyory, R. Zhou, E. Y.-L. Do, and D. A. Szafir. Cultivating visualization literacy for children through curiosity and play. *IEEE Trans Vis Comput Graph*, 29(1):257–267, 2023. doi: 10.1109/TVCG.2022.3209442

[8] S. S. Bae, C. Zheng, M. E. West, E. Y.-L. Do, S. Huron, and D. A. Szafir. Making data tangible: A cross-disciplinary design space for data physicalization. In *Proc. CHI*. ACM, New York, 2022. doi: 10.1145/3491102.3501939

[9] A.-L. Barabási. *Network Science*. Cambridge University Press, 2016.

[10] C. Bennett, J. Ryall, L. Spalteholz, and A. Gooch. The aesthetics of graph visualization. In *Proc. CompAesth*, pp. 57–64. EG, Goslar, 2007. doi: 10.2312/COMPAESTH/COMPAESTH07/057-064

[11] L. Besançon, P. Issartel, M. Ammi, and T. Isenberg. Mouse, tactile, and tangible input for 3D manipulation. In *Proc. CHI*, pp. 4727–4740. ACM, New York, 2017. doi: 10.1145/3025453.3025863

[12] L. Besançon, A. Ynnerman, D. F. Keefe, L. Yu, and T. Isenberg. The state of the art of spatial interfaces for 3D visualization. *Comput Graph Forum*, 40(1):293–326, 2021. doi: 10.1111/cgf.14189

[13] M. Billinghurst, A. Clark, and G. Lee. A survey of augmented reality. *Found Trends Hum-Comput Interact*, 8(2-3):73–272, 2015. doi: 10.1561/1100000049

[14] M. Bostock, V. Ogievetsky, and J. Heer. D³ data-driven documents. *IEEE Trans Vis Comput Graph*, 17(12):2301–2309, 2011. doi: 10.1109/TVCG.2011.185

[15] R. Brown. Series and parallel resistors and capacitors. *Phys Teach*, 41(8):483–485, 2003. doi: 10.1119/1.1625209

[16] J. Burstyn, N. Fellion, P. Strohmeier, and R. Vertegaal. PrintPut: Resistive and capacitive input widgets for interactive 3D prints. In *Proc. INTERACT*, pp. 332–339. Springer, Cham, 2015. doi: 10.1007/978-3-319-22701-6_25

[17] J. S. Cohen. *Computer Algebra and Symbolic Computation: Mathematical Methods*. CRC Press, 2003. doi: 10.1201/9781439863701

[18] F. Daneshzand, C. Perin, and S. Carpendale. KiriPhys: Exploring new data physicalization opportunities. *IEEE Trans Vis Comput Graph*, 29(1):225–235, 2023. doi: 10.1109/TVCG.2022.3209365

[19] S. Davidson. Grasshopper: Algorithmic modeling for Rhino. https://www.grasshopper3d.com/, 2023. Accessed: 2023-03-24.

[20] A. A. de Freitas, W. C. Monteiro, T. A. Soares de Sousa, V. F. Queiroz, T. D. Oliveira de Araújo, and B. S. Meiguins. A flexible pipeline to create different types of data physicalizations. In *Proc. IV*, pp. 73–78. IEEE, USA, 2022. doi: 10.1109/IV56949.2022.00021

[21] N. Dehmamy, S. Milanlouei, and A.-L. Barabási. A structural transition in physical networks. *Nature*, 563(7733):676–680, 2018. doi: 10.1038/s41586-018-0726-6

[22] A. Dijkshoorn, M. Schouten, G. Wolterink, R. Sanders, S. Stramigioli, and G. Krijnen. Characterizing the electrical properties of anisotropic, 3D-printed conductive sheets for sensor applications. *IEEE Sens J*, 20(23):14218–14227, 2020. doi: 10.1109/JSEN.2020.3007249

[23] H. Djavaherpour, F. Samavati, A. Mahdavi-Amiri, F. Yazdanbakhsh, et al. Data to physicalization: A survey of the physical rendering process. *Comput Graph Forum*, 40(3):569–598, 2021. doi: 10.1111/cgf.14330

[24] A. Drogemuller, A. Cunningham, J. A. Walsh, J. Baumeister, R. T. Smith, and B. H. Thomas. Haptic and visual comprehension of a 2D graph layout through physicalisation. In *Proc. CHI*. ACM, New York, 2021. doi: 10.1145/3411764.3445704

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

[26] J. D. Foley, V. L. Wallace, and P. Chan. The human factors of computer graphics interaction techniques. *IEEE Comput Graph Appl*, 4(11):13–48, 1984. doi: 10.1109/MCG.1984.6429355

[27] C. Freksa, T. Barkowsky, Z. Falomir, and J. van de Ven. Geometric problem solving with strings and pins. *Spat Cogn Comput*, 19(1):46–68, 2019. doi: 10.1080/13875868.2018.1531415

[28] T. M. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software Pract Exper*, 21(11):1129–1164, 1991. doi: 10.1002/spe.4380211102

[29] T. Fujiwara, J.-K. Chou, S. Shilpika, P. Xu, L. Ren, and K.-L. Ma. An incremental dimensionality reduction method for visualizing streaming multidimensional data. *IEEE Trans Vis Comput Graph*, 26(1):418–428, 2020. doi: 10.1109/TVCG.2019.2934433

[30] T. Fujiwara, T. Crnovrsanin, and K.-L. Ma. Concise provenance of interactive network analysis. *Visual Informatics*, 2(4):213–224, 2018. doi: 10.1016/j.visinf.2018.12.002

[31] M. Fürer and B. Raghavachari. Approximating the minimum degree spanning tree to within one from the optimal degree. In *Proc. SODA*, p. 317–324. SIAM, USA, 1992.

[32] S. Gerber, P.-T. Bremer, V. Pascucci, and R. Whitaker. Visual exploration of high dimensional scalar functions. *IEEE Trans Vis Comput Graph*, 16(6):1271–1280, 2010. doi: 10.1109/TVCG.2010.213

[33] Grabkowska, D. What Made Me. https://www.grabkowska.com/what-made-me, 2023. Accessed: 2023-03-24.

[34] T. Grosse-Puppendahl, C. Holz, G. Cohn, R. Wimmer, O. Bechtold, et al. Finding common ground: A survey of capacitive sensing in human-computer interaction. In *Proc. CHI*, p. 3293–3315. ACM, New York, 2017. doi: 10.1145/3025453.3025808

[35] A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proc. SciPy*, pp. 11–15, 2008.

[36] M. Hayes. Lcapy: Symbolic linear circuit analysis with Python. *PeerJ Comput Sci*, pp. e875:1–e875:30, 2022. doi: 10.7717/peerj-cs.875

[37] L. He, J. A. Wittkopf, J. W. Jun, K. Erickson, and R. T. Ballagas. ModElec: A design tool for prototyping physical computing devices using conductive 3D printing. *Proc ACM Interact Mob Wearable Ubiquitous Technol*, 5(4):159:1–159:20, 2022. doi: 10.1145/3495000

[38] B. Herman, M. Omdal, S. Zeller, C. A. Richter, F. Samsel, et al. Multi-touch querying on data physicalizations in immersive AR. *Proc ACM Hum-Comput Interact*, 5(ISS):497:1–497:20, 2021. doi: 10.1145/3488542

[39] T. Hermann and H. Ritter. Listen to your data: Model-based sonification for data analysis. *Advances in Intelligent Computing and Multimedia Systems*, 8:189–194, 1999.

[40] G. Hu, Y. Wang, M. Mao, and Y. Zhao. Remote care and collaboration for empty nest family: Smart home, digital twin and mixed reality. In *Proc. ICVR*, pp. 126–134, 2022. doi: 10.1109/ICVR55215.2022.9847779

[41] H. H. Huang, H. Pfister, and Y. Yang. Is embodied interaction beneficial? A study on navigating network visualizations. *Inf Vis*, 22(3):169–185, 2016. doi: 10.1177/14738716231157082

[42] Y.-J. Huang, T. Fujiwara, Y.-X. Lin, W.-C. Lin, and K.-L. Ma. A gesture system for graph visualization in virtual reality environments. In *Proc. PacificVis*, pp. 41–45. IEEE, USA, 2017. doi: 10.1109/PACIFICVIS.2017.8031577

[43] S. Huron, S. Carpendale, A. Thudt, A. Tang, and M. Mauerer. Constructive visualization. In *Proc. DIS*, p. 433–442. ACM, New York, 2014. doi: 10.1145/2598510.2598566

[44] J. Hurtienne, F. Maas, A. Carolus, D. Reinhardt, C. Baur, and C. Wienrich. Move&Find: The value of kinaesthetic experience in a casual data representation. *IEEE Comput Graph Appl*, 40(6):61–75, 2020. doi: 10.1109/MCG.2020.3025385

[45] iSANMATE. Wood filament PLA+. https://www.isanmate.com/product-category/3d_filaments/pla/wood_filament, 2023. Accessed: 2023-06-28.

[46] Y. Ishiguro and I. Poupyrev. 3D printed interactive speakers. In *Proc. CHI*, pp. 1733–1742. ACM, New York, 2014. doi: 10.1145/2556288.2557046

[47] R. J. Jacob, A. Girouard, L. M. Hirshfield, M. S. Horn, O. Shaer, E. T. Solovey, and J. Zigelbaum. Reality-based interaction: A framework for post-WIMP interfaces. In *Proc. CHI*, p. 201–210. ACM, New York, 2008. doi: 10.1145/1357054.1357089

[48] Y. Jansen, P. Dragicevic, and J.-D. Fekete. Evaluating the efficiency of physical visualizations. In *Proc. CHI*, p. 2593–2602. ACM, New York, 2013. doi: 10.1145/2470654.2481359

[49] Y. Jin, I. Qamar, M. Wessely, A. Adhikari, K. Bulovic, P. Punpongsanon, and S. Mueller. Photo-Chromeleon: Re-programmable multi-color textures using photochromic dyes. In *Proc. UIST*, p. 701–712. ACM, New York, 2019. doi: 10.1145/3332165.3347905

[50] J. W. Kelly, M. N. Avraamides, and N. A. Giudice. Haptic experiences influence visually acquired memories: Reference frames during multimodal spatial learning. *Psychon Bull Rev*, 18:1119–1125, 2011. doi: 10.3758/s13423-011-0162-1

[51] A. Kerren, H. Purchase, and M. Ward. *Multivariate Network Visualization*. Springer, 2014. doi: 10.1007/978-3-319-06793-3

[52] M. Le Goc, L. H. Kim, A. Parsaei, J.-D. Fekete, P. Dragicevic, and S. Follmer. Zooids: Building blocks for swarm user interfaces. In *Proc. UIST*, p. 97–109. ACM, New York, 2016. doi: 10.1145/2984511.2984547

[53] Z. Liu and J. Stasko. Mental models, visual reasoning and interaction in information visualization: A top-down perspective. *IEEE Trans Vis Comput Graph*, 16(6):999–1008, 2010. doi: 10.1109/TVCG.2010.177

[54] I. López García and E. Hornecker. Scaling data physicalization–how does size influence experience? In *Proc. TEI*, pp. 1–14. ACM, New York, 2021. doi: 10.1145/3430524.3440627

[55] M. J. McGuffin, R. Servera, and M. Forest. Path tracing in 2D, 3D, and physicalized networks. *IEEE Trans Vis Comput Graph*, pp. 1–14, 2023.

[56] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, et al. SymPy: Symbolic computing in Python. *PeerJ Comput Sci*, 2017.

[57] M. Newman. *Networks*. Oxford University Press, 2018.

[58] T. D. Ngo, A. Kashani, G. Imbalzano, K. T. Nguyen, and D. Hui. Additive manufacturing (3D printing): A review of materials, methods, applications and challenges. *Compos B Eng*, 143:172–196, 2018. doi: 10.1016/j.compositesb.2018.02.012

[59] Norén, L. Infographics inspire art | R. Justin Stewart. https://thesocietypages.org/graphicsociology/tag/r-justin-stewart/, 2011.

[60] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, et al. PyTorch: An imperative style, high-performance deep learning library. In *Proc. NeurIPS*, 2019 (12 pages). doi: 10.48550/arXiv.1912.01703

[61] B. Patnaik, H. Peng, and N. Elmqvist. Sensemaking sans power: Interactive data visualization using color-changing ink. *IEEE Trans Vis Comput Graph*, 2022 (Early Access). doi: 10.1109/TVCG.2022.3209631

[62] T. P. Peixoto. The graph-tool python library. *figshare*, 2014. doi: 10.6084/m9.figshare.1164194.v14

[63] T. P. Peixoto. The Netzschleuder network catalogue and repository. https://networks.skewed.de/, 2020. Accessed: 2023-03-28.

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

[65] E. D. Ragan, A. Endert, J. Sanyal, and J. Chen. Characterizing provenance in visualization and data analysis: An organizational framework of provenance types and purposes. *IEEE Trans Vis Comput Graph*, 22(1):31–40, 2016. doi: 10.1109/TVCG.2015.2467551

[66] Robert McNeel & Associates. Rhinoceros. https://www.rhino3d.com/, 2023. Accessed: 2023-03-24.

[67] J. C. Roberts, P. D. Ritsos, S. K. Badam, D. Brodbeck, J. Kennedy, and N. Elmqvist. Visualization beyond the desktop–the next big thing. *IEEE Comput Graph Appl*, 34(6):26–34, 2014. doi: 10.1109/MCG.2014.82

[68] H. Rosling. Global population growth, box by box. https://www.ted.com/talks/hans_rosling_global_population_growth_box_by_box, 2014.

[69] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv:1609.04747*, 2017. doi: 10.48550/arXiv.1609.04747

[70] K. A. Satriadi, J. Smiley, B. Ens, M. Cordeil, T. Czauderna, et al. Tangible globes for data visualisation in augmented reality. In *Proc. CHI*. ACM, New York, 2022. doi: 10.1145/3491102.3517715

[71] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-Lite: A grammar of interactive graphics. *IEEE Trans Vis Comput Graph*, 23(1):341–350, 2017. doi: 10.1109/TVCG.2016.2599030

[72] V. Savage, R. Schmidt, T. Grossman, G. Fitzmaurice, and B. Hartmann. A series of tubes: Adding interactivity to 3D prints using internal pipes. In *Proc. UIST*, p. 3–12. ACM, New York, 2014. doi: 10.1145/2642918.2647374

[73] M. Schmitz, M. Khalilbeigi, M. Balwierz, R. Lissermann, M. Mühlhäuser, and J. Steimle. Capricate: A fabrication pipeline to design and 3D print capacitive touch sensors for interactive objects. In *Proc. UIST*, p. 253–258. ACM, New York, 2015. doi: 10.1145/2807442.2807503

[74] M. Schmitz, M. Stitz, F. Müller, M. Funk, and M. Mühlhäuser. ./trilaterate: A fabrication pipeline to design and 3D print hover-, touch-, and force-sensitive objects. In *Proc. CHI*, pp. 1–13. ACM, New York, 2019. doi: 10.1145/3290605.3300684

[75] K. J. Schönborn, G. E. Höst, and K. E. Lundin Palmerius. Interactive visualization for learning and teaching nanoscience and nanotechnology. In K. Winkelmann and B. Bhushan, eds., *Global Perspectives of Nanoscience and Engineering Education*, pp. 195–222. Springer, 2016. doi: 10.1007/978-3-319-31833-2_7

[76] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt. Efficient graphlet kernels for large graph comparison. In *Proc. AISTATS*, pp. 488–495, 2009.

[77] W.-S. Soh, K.-Y. See, W.-Y. Chang, M. Oswal, L.-B. Wang, et al. Comprehensive analysis of serpentine line design. In *Proc. APMC*, pp. 1285–1288. IEEE, USA, 2009. doi: 10.1109/APMC.2009.5384455

[78] S. Stusak, M. Hobe, and A. Butz. If your mind can grasp it, your hands will help. In *Proc. TEI*, p. 92–99. ACM, New York, 2016. doi: 10.1145/2839462.2839476

[79] S. Stusak, J. Schwarz, and A. Butz. Evaluating the memorability of physical visualizations. In *Proc. CHI*, p. 3247–3250. ACM, New York, 2015. doi: 10.1145/2702123.2702248

[80] S. Stusak, A. Tabard, F. Sauka, R. A. Khot, and A. Butz. Activity Sculptures: Exploring the impact of physical visualizations on running activity. *IEEE Trans Vis Comput Graph*, 20(12):2201–2210, 2014. doi: 10.1109/TVCG.2014.2352953

[81] S. Swaminathan, C. Shi, Y. Jansen, P. Dragicevic, L. A. Oehlberg, and J.-D. Fekete. Supporting the design and fabrication of physical visualizations. In *Proc. CHI*, p. 3845–3854. ACM, New York, 2014. doi: 10.1145/2556288.2557310

[82] F. Taher, Y. Jansen, J. Woodruff, J. Hardy, K. Hornbæk, and J. Alexander. Investigating the use of a dynamic physical bar chart for data exploration and presentation. *IEEE Trans Vis Comput Graph*, 23(1):451–460, 2017. doi: 10.1109/TVCG.2016.2598498

[83] M. C. Thrun and F. Lerch. Visualization and 3D printing of multivariate data of biomarkers. In *Proc. WSCG*, pp. 7–16, 2016.

[84] P. W. Tuinenga. *SPICE A Guide to Circuit Simulation and Analysis Using PSpice*. Prentice-Hall, 1995.

[85] J. Vasquez, H. Twigg-Smith, J. Tran O'Leary, and N. Peek. Jubilee: An extensible machine for multi-tool fabrication. In *Proc. CHI*, p. 1–13. ACM, New York, 2020. doi: 10.1145/3313831.3376425

[86] G. Verhulsdonck. Issues of designing gestures into online interactions: Implications for communicating in virtual environments. In *Proc. DOC*, pp. 26–33. ACM, New York, 2007. doi: 10.1145/1297144.1297151

[87] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, et al. SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nat Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2

[88] Y. Wang, Z. Jin, Q. Wang, W. Cui, T. Ma, and H. Qu. DeepDrawing: A deep learning approach to graph drawing. *IEEE Trans Vis Comput Graph*, 26(1):676–686, 2020. doi: 10.1109/TVCG.2019.2934798

[89] M. P. Weller, E. Y.-L. Do, and M. D. Gross. Posey: Instrumenting a poseable hub and strut construction toy. In *Proc. TEI*, p. 39–46. ACM, New York, 2008. doi: 10.1145/1347390.1347402

[90] M. Whitlock, S. Smart, and D. A. Szafir. Graphical perception for immersive analytics. In *Proc. VR*, pp. 616–625. IEEE, USA, 2020. doi: 10.1109/VR46266.2020.00084

[91] K. Willis, E. Brockmeyer, S. Hudson, and I. Poupyrev. Printed Optics: 3D printing of embedded optical elements for interactive devices. In *Proc. UIST*, pp. 589–598. ACM, New York, 2012. doi: 10.1145/2380116.2380190

[92] J. S. Yi, Y. a. Kang, J. Stasko, and J. Jacko. Toward a deeper understanding of the role of interaction in information visualization. *IEEE Trans Vis Comput Graph*, 13(6):1224–1231, 2007. doi: 10.1109/TVCG.2007.70515

[93] A. Ynnerman, J. Löwgren, and L. Tibell. Exploranation: A new science communication paradigm. *IEEE Comput Graph Appl*, 38(3):13–20, 2018. doi: 10.1109/MCG.2018.032421649

[94] A. Ynnerman, T. Rydell, D. Antoine, D. Hughes, A. Persson, and P. Ljung. Interactive visualization of 3D scanned mummies at public venues. *Commun ACM*, 59(12):72–81, 2016. doi: 10.1145/2950040