Sauron: Embedded Single-Camera Sensing of Printed Physical User Interfaces

Valkyrie Savage, Colin Chang, Björn Hartmann
University of California, Berkeley
Berkeley Institute of Design, Computer Science Division
valkyrie@eecs.berkeley.edu, colinichang@berkeley.edu, bjoern@eecs.berkeley.edu

ABSTRACT

3D printers enable designers and makers to rapidly produce physical models of future products. Today these physical prototypes are mostly *passive*. Our research goal is to enable users to turn models produced on commodity 3D printers into interactive objects with a minimum of required assembly or instrumentation. We present Sauron, an embedded machine vision-based system for sensing human input on physical controls like buttons, sliders, and joysticks. With Sauron, designers attach a single camera with integrated ring light to a printed prototype. This camera observes the interior portions of input components to determine their state. In many prototypes, input components may be occluded or outside the viewing frustum of a single camera. We introduce algorithms that generate internal geometry and calculate mirror placements to redirect input motion into the visible camera area. To investigate the space of designs that can be built with Sauron along with its limitations, we built prototype devices, evaluated the suitability of existing models for vision sensing, and performed an informal study with three CAD users. While our approach imposes some constraints on device design, results suggest that it is expressive and accessible enough to enable constructing a useful variety of devices.

Author Keywords

Prototyping; Fabrication; 3D Printing; Vision-Based Sensing

ACM Classification Keywords

H.5.2 User Interfaces (D.2.2, H.1.2, I.3.6): Prototyping

General Terms

Design, Human Factors

INTRODUCTION

Our environment is replete with products that have dedicated physical user interfaces like game controllers, musical instruments or personal medical devices. While the ubiquity of smart phones has led to a rise in touchscreen applications, retaining physicality has important benefits such as tactile feedback and high performance manipulation [11]. For example, gamers prefer physical input for speed and performance, musicians for virtuosity and control.

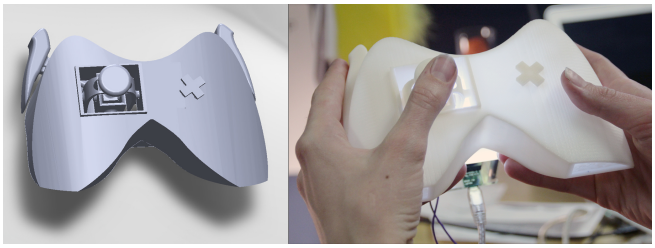

Figure 1. With Sauron, designers create a 3D CAD model of an input device and place a virtual camera in the model. Once printed, they attach a matching physical camera to sense user input on the device.

Rapid additive manufacturing technologies enable designers and makers (we will refer to both groups jointly as "designers" for the remainder of this paper) to quickly turn CAD models of such future devices into tangible prototypes. While such printed form prototypes can convey the *look and feel* of a physical device, they are fundamentally passive in that they do not sense or respond to manipulation by a user. Building integrated prototypes that also exhibit interactive *behavior* requires adding electronic sensing components and circuitry to the mechanical design.

Existing research has developed electronic toolkits that lower the threshold of making physical prototypes interactive [2, 7]. However, such toolkits still require designers to manually assemble printed parts and sensors. Such assembly may also require significant changes to a 3D model (e.g., to add fasteners or split an enclosure into two half shells). Detailed electro-mechanical co-design is time-consuming and cumbersome and mismatched with the spirit of rapid prototyping. Alternatively, designers may instrument the environment with motion capture [1] or depth cameras [25] to add interactivity, but these approaches limit designers to testing prototypes inside the lab in small, restricted areas.

Our research goal is to facilitate the creation of *functional* physical interface prototypes on commodity 3D printers with minimal additional instrumentation or assembly (Figure 1). In this paper, we present an embedded machine vision-based approach for sensing human input on 3D-printed physical

UIST'13, October 8–11, 2013, St. Andrews, United Kingdom.
ACM 978-1-4503-2268-3/13/10.
http://dx.doi.org/10.1145/2501988.2501992

prototypes. Our system, Sauron, enables designers to 3D print a complete interactive device in a single step. After printing, designers add a miniature camera with integrated ring light to their prototype. After an interactive registration step, Sauron can track the motion and position of buttons, sliders, joysticks, and other input devices through machine vision performed on the user's computer.

Sensing all input components on a device with complex shape can be challenging, as components may be outside the viewing frustum of a single camera or blocked by the device's geometry. To address such challenges, we introduce automatic visibility analysis and model modification to translate human input into visible movement that can be accurately tracked with standard computer vision algorithms. We first determine which components will be visible to the camera by placing a virtual camera into a CAD model during the design phase. For components that are not visible, Sauron can modify the component model's internal geometry to extend motion into the camera's viewing frustum using parameterized extrusions. Next, Sauron uses raytracing to determine how optical mirrors may be placed to make motion visible in cases where geometry modification fails because of mechanical interference. We implement these techniques by extending commercial parametric CAD software.

While computer vision research traditionally strives to uncover information about an unknown environment, our approach seeks to modify a known environment to facilitate computer vision. Prior work has demonstrated how 3D printed mechanisms can be used to detect physical motion with optical sensors [24]; we believe we are the first to automatically modify them based on analysis of a 3D design.

Our approach has some important assumptions and limitations: first, we require a 3D printer that can deposit sacrificial support material to print designs with moving parts in a single pass. Most professional machines support this, but few hobbyist machines do today. Second, for printers that cannot deposit multiple colors simultaneously, a user has to perform some manual marking of a printed model with either reflective or dark pigment. Third, our implementation of the CAD plugin can currently only process certain types of hollow models and is not guaranteed to succeed. Fourth, our current model modification techniques only work for a subset of input components. Despite these limitations, Sauron enables construction of a useful variety of devices.

To evaluate the expressivity of our approach, we describe functional prototypes created with Sauron. Three knowledgable CAD users were asked to design DJ mixing boards with our sensing approach in mind. In all cases the users were able to focus on the usability of their prototype interfaces without being impeded by the sensing techniques. We also evaluated ten pre-made models downloaded from the internet and determined that even designers who did not have vision sensing in mind while designing would have been able to use Sauron for their prototypes in seven of ten cases.

Our main contribution is a design tool enabling users to rapidly turn 3D models of input devices into interactive 3D-printed prototypes where a single camera senses input. This comprises three parts:

1. A method for tracking human input on physical components using a single camera placed inside a hollow object.
2. Two algorithms for analyzing and modifying a 3D model's internal geometry to increase the range of manipulations that can be detected by a single camera.
3. An informal evaluation of Sauron, a system that implements these techniques for models constructed in a professional CAD tool.

The remainder of this paper is organized as follows: we present related work, then a description of our approach. We offer details of our initial implementation. We present a collection of prototypes, created by us, to test Sauron's CAD modification capabilities, and the results of an informal user study using Sauron. Finally, we discuss the limitations of Sauron and conclude with directions for future work.

RELATED WORK

Sauron is informed by prior work in three distinct areas: electronic toolkits for rapid prototyping of functional physical user interfaces; computer vision approaches for sensing interaction; and techniques and systems that leverage digital fabrication.

Electronic Prototyping Tools

Electronic prototyping platforms like Arduino [2], Calder [3], Phidgets [7], and d.tools [8] enable designers to quickly create new tangible computing devices by offering accessible software abstractions for working with sensing, actuation, and display hardware. These platforms focus on working with electronic components, but offer little support for integrating such components into printed devices. Modifying a device CAD model to accommodate such hardware can be a time-intensive process: users have to address constraints like mount points and clearances for each component. Enclosures and components must then be manually assembled, which may again require CAD redesign to make assembly possible, e.g., by breaking the shell into multiple pieces and adding fasteners.

Some recent projects seek to lower this barrier: .NET Gadgeteer's CAD plug-in automatically generates mounting bosses and cut-outs for connectors [21], and Enclosed allows users to interactively design enclosures around pre-designed electronics [22]. Sauron takes inspiration from such systems; in addition, its vision sensing can also eliminate electromechanical integration and assembly steps and enable designers to print an entire device as a single object.

Computer Vision

Vision-based design tools like Papier-Mâché offer high-level event models to facilitate design of tangible interfaces [12]. Eyepatch [13] and Crayons [6] offer direct-manipulation interfaces to train vision classifiers without programming.

Some systems place cameras in the environment to track objects and users' interaction with those objects for rapid prototyping, e.g., depth cameras in SketchSpace [9] or motion

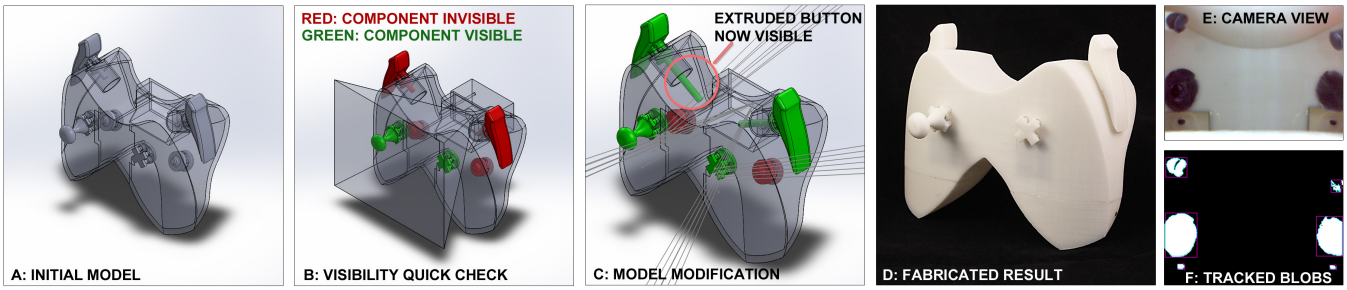

Figure 2. When designing with Sauron, a designer begins with his model (A), then inserts a virtual camera and runs quick check for visibility (B). A full model modification pass (C) performs extrusions and suggests mirror placement to bring invisible controls into the camera's view. He fabricates his design (D), then colors the inside and inserts the camera and mirrors (E). The computer vision software tracks the motion of components (F) and forwards events on to control software, such as a game.

capture systems in Display Objects [1]. In contrast to these projects, we embed the camera inside the designer's device so testing is not confined to a controlled laboratory setting. This embedding approach has also been taken by some hardware devices, e.g., the Rockin' Mouse [4], Mouse 2.0 [20], Digits [10] and Rock-Paper-Fibers [17], though these systems do not contribute general prototyping tools.

Sauron's approach shares the greatest similarity with Döring's work-in-progress on embedding a camera and projector system into product mockups [5]. We extend their work by providing visibility analysis and automatic CAD model modification techniques. We also share an approach with SLAP widgets [23] – fabricated controls that can be placed on an interactive tabletop, where controls are tracked by the tabletop's vision system. Sauron generalizes the intuition behind SLAP widgets beyond tabletop computing and offers a tool to design custom widget configurations in CAD software.

CAD & Digital Fabrication

Finally, Sauron relates to prior work in augmenting CAD tools and leveraging digital fabrication for prototyping.

ModelCraft [19] introduced techniques to edit digital 3D models based on digital pen annotations performed on folded paper versions of the model. Our goal of prototyping interactivity is orthogonal to providing new interaction techniques for model editing.

Willis's work on printed optics describes how to combine optical sensors with light guides printed using transparent materials to fabricate interactive components like buttons and sliders [24]. Their prototype requires inserting electronics at print time. With Sauron, a single camera is added after printing is complete. In addition, while Willis's work is inspirational in delineating a design space for optically sensed physical manipulation, the authors do not provide a tool for the design of such objects.

Midas fabricates capacitive touch sensors using a CNC cutter based on high-level specifications [18]. Our contribution is complementary, as Sauron focuses on vision sensing of mechanisms rather than touch input.

DESIGNING WITH SAURON

We will describe the process of designing and fabricating models for single-camera sensing with a running example: a designer wishes to prototype a new video game controller with buttons, a joystick, and a direction pad. He wants to explore ergonomics – how the controller feels to hold and how it will feel during gameplay. He follows the steps in Figure 2.

Modeling: The designer creates a 3D model of his controller in a CAD tool like SolidWorks, placing buttons and joysticks from a library of available controls Sauron provides (Figure 2A). Each library element is parameterized and customizable.

Adding a virtual camera: Using the Sauron CAD plug-in, he adds a 3D model of Sauron's camera to his assembly. This camera can be positioned anywhere on the model's surface, pointing inwards, into the interior of the hollow model. The designer then adds mount points for the camera so it can be attached with screws once he fabricates his controller.

Visibility analysis: Sauron provides a "quick check" feature which allows the designer to quickly determine if components are directly within view of the camera or if they will require model modifications (Figure 2B). In our example, the joystick and direction pad in front of the camera are visible, so they are colored green. The bumper and rear buttons are not: they lie outside the camera's field of view and are marked red.

Model modification: To make the remaining components visible to the camera, the Sauron plugin automatically extrudes the interior portion of the bumper buttons to extend into the camera's field of view (Figure 2C). The rear buttons cannot be extended, as the extrusions would intersect the controller's shell. Detecting this interference, Sauron casts rays from the camera into the 3D scene, reflecting them off the interior of the body, and determines locations where placement of two small mirrors will make the rear buttons visible in the camera image. The plugin visualizes these locations to guide the designer during manual assembly.

Fabrication and assembly: The designer sends his file (without the camera model) to his 3D printer (Figure 2D). Once the print is completed, an automatically generated instruction sheet guides him through the process of marking the interior of input components, e.g., with black marker (Figure 2E). Last, he screws the camera into its mounts.

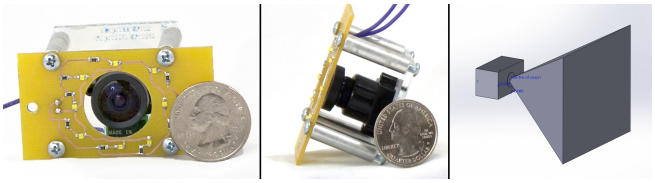

Figure 3. Left: Sauron's USB camera and ring light. Right: Our virtual model of the camera and its field of view.

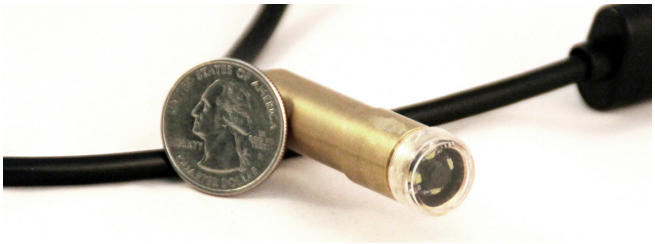

Figure 4. The hardware can be miniaturized, as in this pipe inspection camera with integrated LEDs.

Registration and Testing: Finally, the designer registers the components with the vision system one at at time: his CAD tool prompts him which component to move, and he moves each through its full range of motion to configure its component-specific recognizers. The system then tracks each component separately (in Figure 2E & F, components are: extruded bumper buttons on top; joystick and d-pad in the middle, reflected rear buttons in mirrors below). Once all the components are registered, he is ready to test his controller. Sauron sends input event data as OpenSoundControl messages, a format that software tools can understand and map to game controller events. Sauron can also deliver events over WebSockets to applications written in HTML and JavaScript.

IMPLEMENTATION

In this section, we describe Sauron's camera, CAD component architecture, algorithms for modifying internal geometry, and vision pipeline.

Physical and Virtual Cameras

Sauron uses a single camera to sense input on a physical device. In order to determine visibility of input components inside the CAD environment, Sauron uses a virtual camera that matches the physical camera's measurements and optical characteristics. We empirically measured the field of view of the camera with a geometric test pattern, and we then generated model geometry corresponding to this field of view as a reference for designers (Figure 3).

Our current implementation uses a 640x480 USB camera with a retrofitted 110 degree M12 lens (Sentech STC-MC36USB-L2.3). The interior of the model is illuminated by a ring light with eight surface-mount white LEDs. This hardware may be too bulky for handheld devices; however, there are no technological barriers to miniaturization. We have also built prototypes using a commercial pipe inspection camera (Figure 4) which is much smaller, but suffered from a low video frame rate and shallow depth of field .

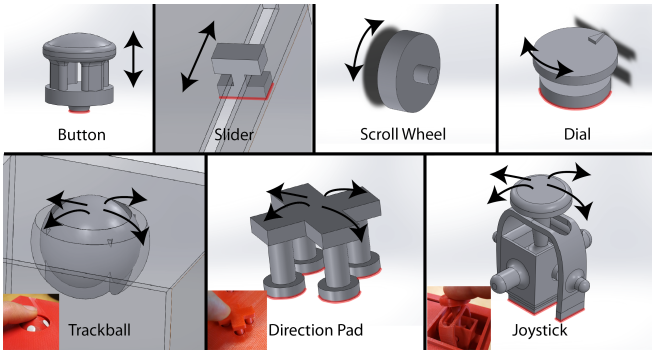

Figure 5. Sauron currently supports seven types of input components. The various components have different types of motion trackable by Sauron, from binary up/down of a button, to one-dimensional slider input, to two-dimensional input from a trackball or joystick. Extrusion features of components are highlighted in red.

Component Library and Architecture

Sauron provides a library of components with buttons, sliders, scroll wheels, dials, trackballs, direction pads, and joysticks (Figure 5). These components, when printed, will be tracked through contrasting material applied in a specific pattern or location. For many of the components, this location is in the base, which is tagged in our models. We require that designers use components with tagged geometry in their devices so our plugin understands which portions need to be visible to the camera as well as how to perform modifications. Our base components are parametric models for the SolidWorks CAD software.

Because Sauron models are parametric, designers already have significant freedom in modifying them to suit their needs. As long as the tagged geometry (on the interior, facing the camera) is kept, the exterior of the models can be changed. As an example, a designer creating a video game controller may make some buttons oblong rather than circular: the long buttons on the side of the controller in Figure 2 were built from the same parametric model as the rear circular buttons.

To create a new Sauron-compatible component, the component must exhibit visible motion on the inside of a prototype that can be tracked by the camera. Second, the component must be paired with a suitable vision algorithm to extract its state from visible changes. These two requirements can be decoupled. For example, both toggle switches and momentary switches can use the same algorithm extracting a single state bit from a change in position.

Modifying Components

Users' CAD models are modified based on an analysis of which input components fall within the field of view of the virtual camera. The two basic modifications our software considers are extrusion and mirror placement. The software which performs model modifications is implemented in C# as a SolidWorks 2012 plugin.

Extrusion

In order to perform modifications, our initial step is to extend the virtual camera's field of view feature to infinity while maintaining its angles. We revert this after all modification

steps are complete. We determine visibility through collision detection between tagged model geometry and the virtual camera's field of view feature. When components are outside the field of view, e.g., on a side wall (Figure 6C), Sauron attempts to extend the component's base through extrusion (Figure 6A-B). This technique is not applicable to scroll wheels or trackballs. The model parts Sauron can extrude are shown in red in Figure 5.

To calculate extrusion depth, we first cast a ray from the component's base and determine if it intersects the field of view. If not, then we cannot reach the field of view with extrusion. We then measure the distance from the base along its normal to the field of view and update our extrusion to that depth. We next iterate through possible positions of the component (e.g., simulate a slider's motion along its track) and check that we are not intersecting any other components or the body of the device, and that we continue to meet the field of view. We iteratively extend our extrusion if we fall outside the cone and perform mechanical interference checks at all positions at each length. If we avoid collisions, the component has been successfully modified. Failure cases of this algorithm are shown in Figure 7.

Extrusion need not be limited to a single direction straight down from a component's base. We have built proof-of-concept components like the button in Figure 8, which have multiple possible extrusion directions. This increases the applicability of extrusion to more complicated geometries. Our prototype does not automatically extrude such components yet, but a designer using the camera's virtual field of view reference can make these modifications manually.

Visibility Check, Raytracing, and Mirror Placement

Designers can check visibility of their components by seeing whether they fall within the field of view geometry of the virtual camera. However, the virtual camera's field of view shown to the user has limited depth so it does not interfere with other modeling tasks. Using raycasting, Sauron provides immediate visibility feedback by highlighting all components that are directly visible to the camera. We cast a ray from the center of the camera to the bottom of each component and determine whether that ray falls inside the field of view. If so, we perform the same check in the maximum and minimum positions of the component (e.g., we slide sliders to each end of their tracks).

We use raytracing to determine how to place mirrors for components where extrusion failed (Figure 9). The designer has to manually insert these during post-print assembly.

Each ray is cast from the camera to the body of the device, and from there reflected based on the surface normal of the body at the intersection point: i.e., we assume that during assembly the mirror will be placed tangent to the body's inner face. The reflected rays are traced to determine if they intersect any components which were not successfully modified in the extrusion step. If such a component is encountered by the reflected ray, the location on the body that it was reflected from is marked. This leaves a cloud of points per component, which informs the designer where to place mirrors during assembly (see Figure 2). Our prototype traces a coarse grid of 20x20 rays because of limitations of the SolidWorks API, in which a single trace takes up to 250ms. A more efficient reimplementation can increase rays to one per camera pixel.

The raytracing algorithm also finds occlusions. If a component is not the first object hit by any direct rays cast or any rays reflected off the main body, the user is alerted that the component needs to be moved or manually modified because it is out of the camera's view. For example, in a case with two buttons in a row and the camera's view parallel to the row, if mirror placement is not possible then the rear button would

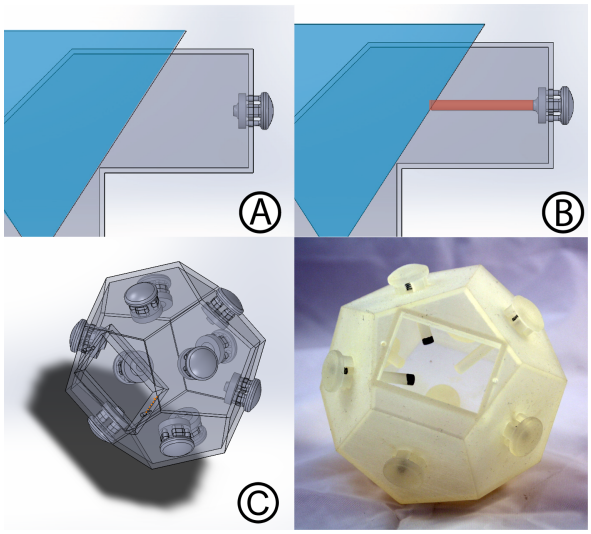

Figure 6. We measure the distance from the button to the virtual camera's field of view–highlighted in blue (A), then extrude the bottom of the button that distance (B). This technique is useful when creating objects where input components on many faces point different directions, like this dodecahedral ball of buttons (C).

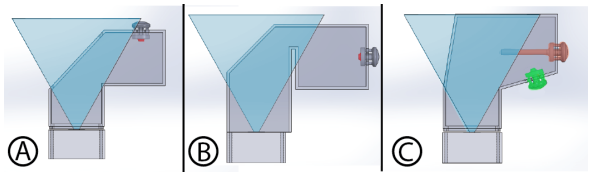

Figure 7. Extrusion does not work in some cases. The component's base may not point at the camera's field of view (A). The component's base may point at the field of view, but be blocked by the main body (B). One component's base (green), if extruded, would intersect the another component (red) (C).

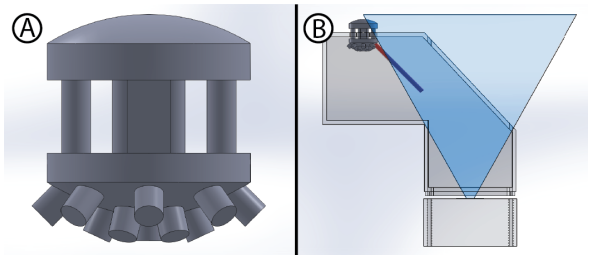

Figure 8. This prototype component (A) can be extruded in multiple directions to meet the camera's FOV cone (B).

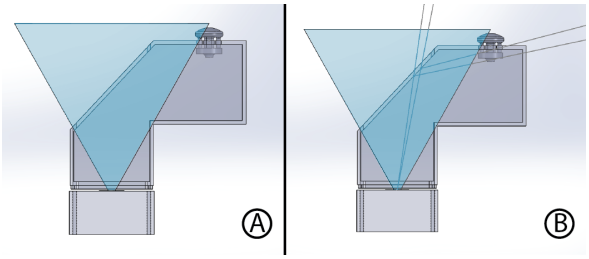

Figure 9. An illustration of the raytracing algorithm used for mirror placement. Note that the button in the figure cannot be extruded to meet the field of view cone. A mirror will be glued at the spots where the rays successfully reflect (seen in B) during assembly.

trigger this alert because all rays cast from the camera hit the front button first.

Mirrors can also be used to redirect motion to increase its visibility. For example, buttons moving along the Z-axis (toward the camera) are harder to track than buttons that move in the XY plane. A 45 degree mirror placed next to the button can redirect visible motion. Our prototype does not automatically calculate the locations of these mirrors yet.

Post-Print Assembly

Due to the nature of our sensing approach, we require that designers' models be hollow and contain a hole of suitable size for the lighting and camera rig to be inserted. Many prototypes are designed to be hollow because such designs conserve printing material. However, this requirement places some restrictions on how other elements, e.g., an LCD screen, can be placed inside the model.

We also require a few steps of assembly to make the prototype suitable for use with our vision pipeline. To increase visibility of the input components versus the background, we require the addition of some distinctive material to the input components. This material can be printed in multi-color 3D printers. Alternatively, coloring the bottoms of the input components with a pen is sufficient. We use a silver permanent pen on dark model material or a black permanent pen on light model material (see Figure 10).

Because most current materials used for 3D printing are too brittle to create small compliant parts, users must add springs manually after printing (e.g., to restore buttons after being pressed). This limitation is not unique to Sauron. We designed our buttons to allow for insertion of springs using tweezers. Any mirrors will need to be inserted as well. We

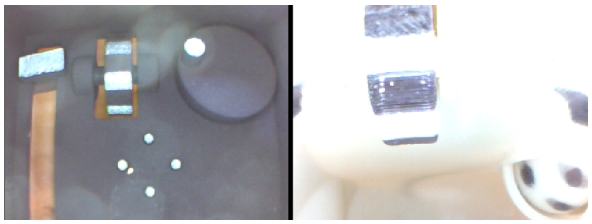

Figure 10. Components with reflective ink on black material (left) and black ink on white material (right).

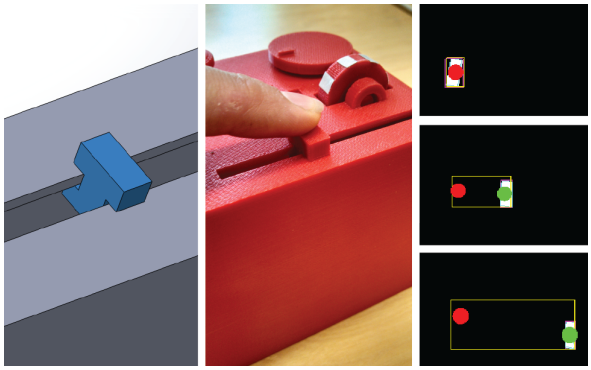

Figure 11. Sauron's SolidWorks plugin highlights each model component in turn and asks the designer to move it. The vision software creates a bounding box as the component moves through its range and also saves any information required by the component type. For example, to determine slider position later the vision software saves the two most extreme tracked center points (the red and green dots).

use small craft mirrors which we affix to the printed device's interior surface with epoxy.

Sauron generates a basic set of step-by-step instructions, automatically displayed in the designer's browser, to assist in correct model assembly. These instructions include automatically-created screenshots of the model highlighting parts that require their attention and example images showing them how to apply mirrors and how to mark components.

Machine Vision

A computer vision pipeline tracks user manipulations of components once they have been printed. We run each camera frame through a series of steps: binarization, connected components detection, and previous frame differencing. This highlights movement of components between frames.

Registration

Users have to register components before they can be tracked. During the registration process, regions of interest for each component are determined. A designer is prompted by SolidWorks to actuate each of his components in turn, and a bounding region is created that encompasses all the points through which the component moved (Figure 11). These regions determine the relative position of the component within its bounds during the testing phase.

In future work, we would like to explore more detailed simulation in the CAD environment. This could eliminate the physical registration phase by either generating and printing visual markers (and using sensing similar to [5]) in a contrasting material, or by predicting the position of the components in the camera's image and sending that information to the vision software.

Tracking

After registration, different detection algorithms apply to each input component. The techniques we use for each component are visualized in Figure 12. For *buttons*, we extract one bit of status from movement of its tracked blob. The direction pad generalizes this approach to track four cardinal directions, while the *joystick* tracks movement of X and Y

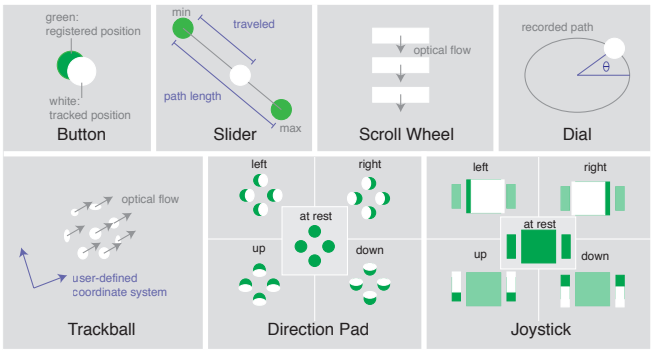

Figure 12. The different types of components in the Sauron library require tailored computer vision tracking approaches to extract state information.

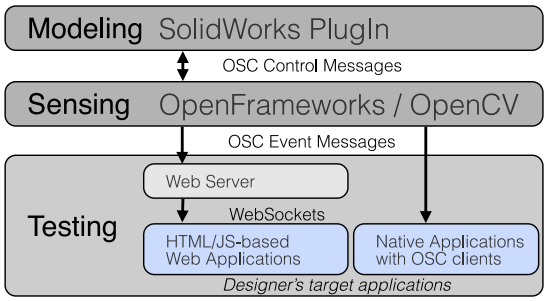

Figure 13. SolidWorks and OpenFrameworks exchange messages via OpenSoundControl. OpenFrameworks also sends OSC messages containing processed data to a WebSockets server to deliver events to a user's application.

axes separately. We find the absolute position of a *slider* in a unit interval by finding its blob on a line connecting the minimum and maximum positions observed during calibration (see also Figure 11). The *dial* tracks position as orientation of a blob around an elliptical path, while *scroll wheel* and *trackball* use optical flow to determine amount and direction of relative movement.

We currently do not correct for perspective in our images, which leads to non-linear behaviors in components like the slider and dial. It would be possible to account for perspective analytically since we know position and orientation of a component with respect to the camera in the model. For example, a slider follows a known line through $(x_{min}, y_{min}, z_{min})$ and $(x_{max}, y_{max}, z_{max})$ in the CAD model. Given a slider located at (u, v) in the image, we can find the point that is mutually closest (in a least-squares sense) to the line from the focal point through the image plane at (u, v) and the line of the slider's movement.

The vision component of our prototype is implemented in C++ and runs at interactive speeds (>32fps) on a 2011 Macbook Pro. We rely on the open-source computer vision library OpenCV [15] and OpenFrameworks [16]. Messages are passed between SolidWorks and OpenFrameworks via the OpenSoundControl (OSC) protocol. OSC messages are sent over UDP and contain an address (e.g., *"/button/1"*) and payload (e.g., "on" or "off"). Our prototype uses these messages to communicate processed events, to start and stop test mode, and to start and stop registration of a particular component (see Figure 13).

Testing

Sauron can deliver input events to application using either OpenSoundControl or WebSocket messages.

Existing third-party tools can transform OSC messages into keyboard, mouse, or game controller events, without the need to write code. For example, using the third-party program OSCulator, a designer could simply assign messages coming from */joystick/x* to move the mouse cursor in the X direction and from */joystick/y* to move it in the Y direction. This strategy can also be employed to generate USB HID game controller events and key presses automatically without code.

For designers who wish for more control and who are familiar with programming, we enable event consumption in web applications written in HTML and Javascript. Leveraging web applications as a platform allows interface prototyping on any device with an internet-connected web browser. We use a node.js server which exposes processed events over WebSockets. We adopt this strategy from Midas [18].

EVALUATION

To evaluate the feasibility and expressivity of the Sauron sensing approach, we asked experienced CAD users to design Sauron devices; we also checked the suitability of various models found online for vision sensing; and we printed and tested several prototypes.

Modeling with Sauron

We performed an informal evaluation with three mechanical engineers. All were proficient SolidWorks users. We first explained how Sauron works and demonstrated a printed prototype containing examples of all our input components. We then asked them to prototype a disk jockey (DJ) controller that could be tested with Sauron. Common functions on such controllers are volume and EQ control knobs, large "scratch" wheels for two audio channels, and a crossfader. We emphasized thinking aloud, as we wished to determine how the constraints of our vision-based system affected their design process. Participants did not run the plug-in itself during the modeling sessions due to time constraints, but we ran it on the resulting models and fabricated one of their designs (see Figure 17).

All of our participants modeled DJ mixing boards that could be successfully used with our vision-based sensing approach (Figure 14). They followed different approaches to place the camera – though all showed concern for the aesthetics of their design and accordingly tried to mount the camera inside the main enclosure or otherwise out of the way. One user mounted the camera sideways (Figure 14A), but at a location such that the mixer's components would not occlude each other; another created a very deep box at the start, stating that he preferred "to focus on the user side, rather than the camera because I don't care about the box size" (Figure 14B). The most ingenious design mounted the camera on the top, pointing down, so that all components would be visible in

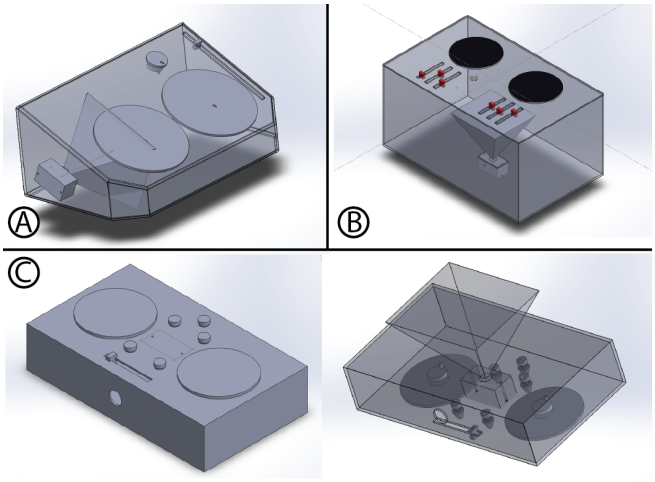

Figure 14. Our three user study participants prototyped DJ mixing boards using our component library. Each had a very different strategy for ensuring the camera could see all components. The assembly on the bottom (with interior cutaway view at right) was designed to have the camera inside reflecting off mirrors placed on the back wall.

a single large mirror placed at the bottom of the controller (Figure 14C). In aggregate, while users had to plan for the visibility constraints of camera sensing in their design, these constraints were not seen as overly burdensome.

One user wished that an interactive design checker was available to test his design iteratively for visibility. A complete model modification pass currently requires ∼5 minutes to process a non-rectilinear model with 10 components, because of slow calls to the SolidWorks API. Based on this feedback we implemented the "quick check" feature which highlights components that are immediately within the viewing area without performing ray-tracing or extrusion.

Participants also successfully modified the library of parameterized components. One participant stated that it was important to her that the sensing portion of each component was decoupled from the user-facing portion. For example, the scratch wheels are large on the user's side to enable users to place their entire hand on them, while the internal dial diameter is small so it can be seen by the camera in its entirety (see Figure 14C). The same user also wished that there was better documentation for the component library, describing how large holes for mounting needed to be.

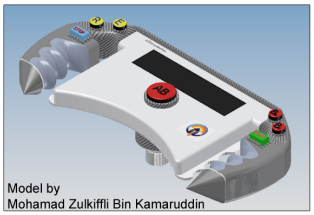

Figure 15. These models found online were too shallow to sense with Sauron—occlusion and curvature would prevent correct sensing with computer vision.

Analysis of Pre-Designed Models

To determine if designers working without our constraints in mind would create prototypes that are compatible with our vision-based system, we downloaded several online 3D models and analyzed them. The models, which comprised a deduplicated set of all models with keywords "interactive" or "controller" on the model-sharing site grabCAD.com, ranged from XBOX and Guitar Hero controllers to interactive desks with keyboards. None of the devices that we analyzed were designed for 3D printing, but rather for rendering or as engineering drawings. Our first step in processing them was estimation of the internal geometry of the bodies, for which we assumed simple shelling (i.e., no internal supports, wall thickness approximately .1", interior curves following the curves of the outside of the body). After this was done, we selected several candidate camera locations which would not interfere with what we understood to be the user-facing functions of the device, and we measured which components would be visible to the camera directly, which via extrusion, and which via reflection.

Out of 10 devices we analyzed, we believe that 7 of them could be successfully processed by Sauron. Three devices were too thin – this caused serious occlusion problems between components. Their bodies also did not allow space for the inclusion of mirrors to solve the occlusion problem (Figure 15). One of the failing devices, a steering-wheel-style device, had two handle areas with buttons at their far ends and thin, continuously-curving surfaces bending away from the main body. Using just one mirror bounce, it would be impossible to see around these bends to the buttons at the ends.

Example Devices

We also fabricated three prototypes that display the range of interactive components our prototype system offers.

Ergonomic Mouse

Our ergonomic mouse (see Figure 16) has a trackball the user can manipulate with his thumb as well as two buttons and a scroll wheel. We configured the mouse to control the mouse cursor on a laptop using OSCulator. Due to large tolerances in our model, the scroll wheel tended to oscillate between "up" and "down" states after being released. This problem could either be addressed through modifications to the model or by double thresholding in our computer vision component.

DJ Mixer

We constructed a DJ mixing board – based on a study participant's design – in two pieces to fit on our 3D printer's bed

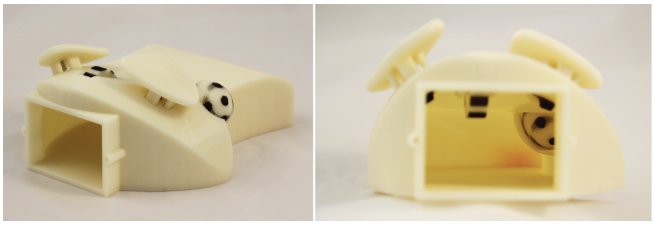

Figure 16. Our ergonomic mouse prototype has a trackball the user can manipulate with his thumb as well as two buttons and a scroll wheel. On the right is the camera's view of the inside of the mouse.

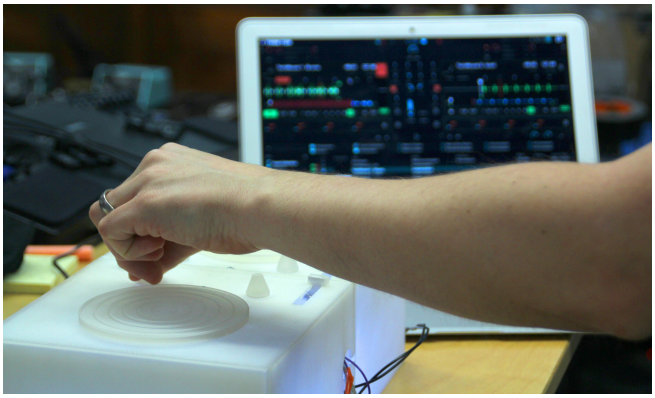

Figure 17. Our DJ mixing board, based on one of our users's designs, has sliders and two dial configurations: raised knobs for easy manipulation of volume, and a larger flat wheel for seeking and scratching songs. The different types of dials share a sensing algorithm, however, as their interior parts are similar.

size. We converted the OSC messages sent out by Sauron's vision software to MIDI messages to control Traktor, a professional DJ application (see Figure 17). One issue this prototype raised was that disparities between the virtual and physical camera parameters affected visibility. While the components were designed to fit within the virtual camera's field of view, an offset between the lens axis and the center of the sensor on our (manually-modified) camera led to some components falling outside the physical field of view. We are confident that better calibration and measurement can overcome such problems.

Game Controller

We developed two versions of a video game controller, shown in Figures 1 and 2. To test responsiveness, we built a simple browser-based game to accept data from Sauron's WebSockets server. The controller moves the player character around (joystick) and shoots fireballs (buttons). We found the game was playable, although detection of the joystick position was noisy. This seems to be due to the fact that the blobs tracked for the main base and the two flanks were lumped together when the joystick was in certain configurations, e.g., at extreme right. We believe this is not a fundamental issue and could be mitigated by iterating on the joystick's interior design or by using a higher-resolution camera.

LIMITATIONS

Currently, our prototypes still require some post-printing assembly for marking components and inserting mirrors. However, we believe this step is significantly less time-consuming than the process of wiring up a prototype with discrete electronic components.

A second limitation is the required registration process after printing. In future work we plan to create more sophisticated algorithms which can pre-determine bounding boxes of printed components using the digital model, or which can generate visual markers to denote end points and motion type. This would allow designers to skip the registration step.

Because we currently use visible light sensing, environmental lighting can interfere with our algorithms. For example, our prototypes behave erratically when tested with bright fluorescent lights directly overhead. Some components, like the slider and joystick, require a certain amount of clearance around them to move properly. When bright light shines through these gaps, vision tracking can become problematic. One remedy is to move sensing into the infrared spectrum.

Our algorithms do not deal with cases where chaining of model modifications is required: i.e., if a component could be seen by first extruding, then reflecting, it will not be correctly processed by our algorithm. We provide the field of view of our camera as a reference to designers so that they can correct cases like this on their own, however more complex automatic interior geometry modifications are possible.

Finally, we support only a limited library of components, and not all components can be modified through extrusions. However, this library is extensible by expert users who can define and label faces for extrusion and who can choose or program appropriate tracking algorithms. Our informal evaluation suggests though that configuring and changing existing components to suit the needs of a particular prototype may be sufficient to cover a useful design space.

FUTURE WORK

Current Sauron prototypes are all tethered to a PC. There are opportunities to explore interactive devices not connected to computers. For example, tangible peripherals for mobile device could also be prototyped using our system. Modern smartphones have on board cameras and LED flashes, and enough on-board processing to perform computer vision. Modeling the phone and its camera parameters could enable mobile prototypes designed to encase the phone.

We believe that an exciting use for Sauron is in the development of entirely novel input devices which are not supported by traditional electronics. One example is a curved slider: electronics typically measure either linear or rotary motion, but a slider on a curved or irregular track would be easily prototyped using Sauron.

The creation of interactive prototypes also need not be limited to 3D printed plastic. Digital fabrication opens the doors to many new areas of exploration: any process which fabricates material according to a model created in software could be processed similarly. One such promising technology is laser cutting, where we already see the ability to create 3D models through sliceforms or layering of 2D cross-sections of an object. Laser Origami [14] has pushed the bounds further, and it is not difficult to imagine fully laser-cuttable mechanisms that could be tracked by Sauron.

For future work we hope to test our tool more extensively with designers in the context of a workshop or class. We are also planning to explore tools to simplify the physical design process for users unfamiliar with CAD tools.

CONCLUSION

In this paper, we presented Sauron, a system to assist designers creating interactive physical prototypes without the need

for wiring or an augmented environment. Sauron uses a camera to track input on moving physical components. Sauron can check visibility of model components at design time and modify component geometry. Several 3D printed objects created by the authors demonstrate the range of capabilities of the prototype system. Feedback from early users also suggests improvements to be made as the work continues. Beyond Sauron, we will expand our future research agenda to assist people in the creation of interactive objects by leveraging the ubiquity and power of digital fabrication tools.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant No. DGE 1106400, a Microsoft Graduate Women's Fellowship and an Alfred P. Sloan Fellowship. The authors would also like to thank Mark Oehlberg for help with electronics.

REFERENCES

1. Eric Akaoka, Tim Ginn, and Roel Vertegaal. DisplayObjects: prototyping functional physical interfaces on 3D styrofoam, paper or cardboard models. In *Proceedings of the International Conference on Tangible, Embedded, and Embodied interaction*, TEI '10, pages 49–56. ACM, 2010.
2. Arduino : an open-source electronics prototyping platform. http://arduino.cc/. Accessed: July 2013.
3. Daniel Avrahami and Scott E. Hudson. Forming interactivity: a tool for rapid prototyping of physical interactive products. In *Proceedings of the ACM Conference on Designing Interactive Systems*, DIS '02, pages 141–146. ACM, 2002.
4. Ravin Balakrishnan, Thomas Baudel, Gordon Kurtenbach, and George Fitzmaurice. The Rockin'Mouse: integral 3D manipulation on a plane. In *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*, CHI '97, pages 311–318. ACM, 1997.
5. Tanja Doering, Bastian Pfleging, Christian Kray, and Albrecht Schmidt. Design by physical composition for complex tangible user interfaces. In *CHI '10 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '10, pages 3541–3546. ACM, 2010.
6. Jerry Fails and Dan Olsen. A design tool for camera-based interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '03, pages 449–456. ACM, 2003.
7. Saul Greenberg and Chester Fitchett. Phidgets: easy development of physical interfaces through physical widgets. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, UIST '01, pages 209–218. ACM, 2001.
8. Björn Hartmann, Scott R. Klemmer, Michael Bernstein, Leith Abdulla, Brandon Burr, Avi Robinson-Mosher, and Jennifer Gee. Reflective physical prototyping through integrated design, test, and analysis. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, UIST '06, pages 299–308. ACM, 2006.
9. David Holman and Hrvoje Benko. SketchSpace: designing interactive behaviors with passive materials. In *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '11, pages 1987–1992. ACM, 2011.
10. David Kim, Otmar Hilliges, Shahram Izadi, Alex D. Butler, Jiawen Chen, Iason Oikonomidis, and Patrick Olivier. Digits: freehand 3D interactions anywhere using a wrist-worn gloveless sensor. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, UIST '12, pages 167–176. ACM, 2012.
11. Scott R. Klemmer, Björn Hartmann, and Leila Takayama. How bodies matter: five themes for interaction design. In *Proceedings of the ACM Conference on Designing Interactive Systems*, DIS '06, pages 140–149. ACM, 2006.
12. Scott R. Klemmer, Jack Li, James Lin, and James A. Landay. Papier-Mâché: toolkit support for tangible input. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 399–406. ACM, 2004.
13. Dan Maynes-Aminzade, Terry Winograd, and Takeo Igarashi. Eyepatch: prototyping camera-based interaction through examples. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, UIST '07, pages 33–42. ACM, 2007.
14. Stefanie Mueller, Bastian Kruck, and Patrick Baudisch. LaserOrigami: Laser-Cutting 3D Objects. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 2585 – 2592. ACM, 2013.
15. OpenCV : open source computer vision. http://opencv.org/. Accessed: July 2013.
16. OpenFrameworks : an open-source C++ toolkit for creative coding. http://www.openframeworks.cc/. Accessed: July 2013.
17. Frederik Rudeck and Patrick Baudisch. Rock-paper-fibers: bringing physical affordance to mobile touch devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 1929–1932. ACM, 2012.
18. Valkyrie Savage, Xiaohan Zhang, and Björn Hartmann. Midas: fabricating custom capacitive touch sensors to prototype interactive objects. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, UIST '12, pages 579–588. ACM, 2012.
19. Hyunyoung Song, François Guimbretière, Chang Hu, and Hod Lipson. ModelCraft: capturing freehand annotations and edits on physical 3D models. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, UIST '06, pages 13–22. ACM, 2006.
20. Nicolas Villar, Shahram Izadi, Dan Rosenfeld, Hrvoje Benko, John Helmes, Jonathan Westhues, Steve Hodges, Eyal Ofek, Alex Butler, Xiang Cao, and Billy Chen. Mouse 2.0: multi-touch meets the mouse. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, UIST '09, pages 33–42. ACM, 2009.
21. Nicolas Villar, James Scott, Steve Hodges, Kerry Hammil, and Colin Miller. .NET gadgeteer: a platform for custom devices. In *Proceedings of the International Conference on Pervasive Computing*, Pervasive '12, pages 216–233, Berlin, Heidelberg, 2012. Springer-Verlag.
22. Christian Weichel, Manfred Lau, and Hans Gellersen. Enclosed: a component-centric interface for designing prototype enclosures. In *Proceedings of the International Conference on Tangible, Embedded and Embodied Interaction*, TEI '13, pages 215–218. ACM, 2013.
23. Malte Weiss, Julie Wagner, Yvonne Jansen, Roger Jennings, Ramsin Khoshabeh, James D. Hollan, and Jan Borchers. SLAP widgets: bridging the gap between virtual and physical controls on tabletops. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 481–490. ACM, 2009.
24. Karl Willis, Eric Brockmeyer, Scott Hudson, and Ivan Poupyrev. Printed optics: 3D printing of embedded optical elements for interactive devices. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, UIST '12, pages 589–598. ACM, 2012.
25. Andrew D. Wilson. Using a depth camera as a touch sensor. In *ACM International Conference on Interactive Tabletops and Surfaces*, ITS '10, pages 69–72. ACM, 2010.