# RetroDepth: 3D Silhouette Sensing for High-Precision Input On and Above Physical Surfaces

**David Kim[1,3], Shahram Izadi[1], Jakub Dostal[4], Christoph Rhemann[1], Cem Keskin[1]**
**Christopher Zach[1], Jamie Shotton[1], Tim Large[2], Steven Bathiche[2]**
**Matthias Nießner[5], D. Alex Butler[1], Sean Fanello[6], Vivek Pradeep[2]**

[1]Microsoft Research [2]Microsoft [3]Newcastle University
[4]University of St. Andrews [5]Stanford University [6]Italian Institute of Technology

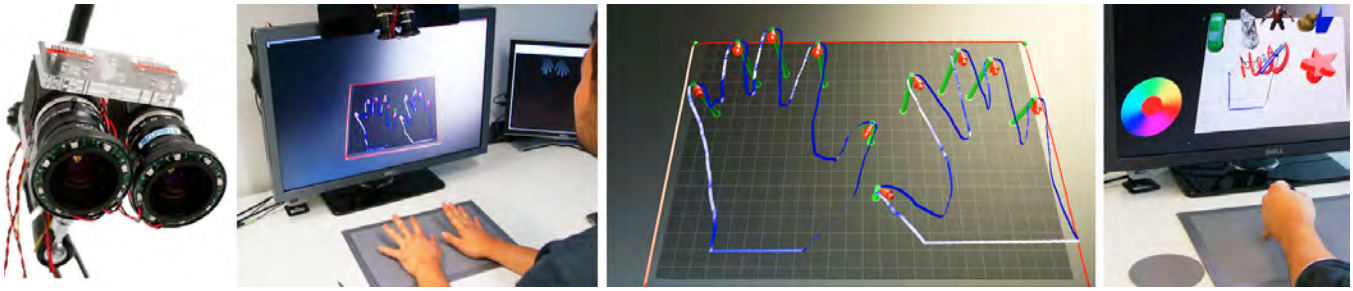

Figure 1: RetroDepth is a new 3D silhouette sensing system for high-precision interaction on and above physical surfaces.

## ABSTRACT

We present RetroDepth, a new vision-based system for accurately sensing the 3D silhouettes of hands, styluses, and other objects, as they interact on and above physical surfaces. Our setup is simple, cheap, and easily reproducible, comprising of two infrared cameras, diffuse infrared LEDs, and any off-the-shelf retro-reflective material. The retro-reflector aids image segmentation, creating a strong contrast between the surface and any object in proximity. A new highly efficient stereo matching algorithm precisely estimates the 3D contours of interacting objects and the retro-reflective surfaces. A novel pipeline enables 3D finger, hand and object tracking, as well as gesture recognition, purely using these 3D contours. We demonstrate high-precision sensing, allowing robust disambiguation between a finger or stylus touching, pressing or interacting above the surface. This allows many interactive scenarios that seamlessly mix together freehand 3D interactions with touch, pressure and stylus input. As shown, these rich modalities of input are enabled on and above *any* retro-reflective surface, including custom "physical widgets" fabricated by users. We compare our system with Kinect and Leap Motion, and conclude with limitations and future work.

### Author Keywords

NUI, 3D contours, depth sensing, 3D input, touch, stylus, vision-based UIs, stereo matching, contour classification


*CHI 2014*, April 26 - May 01 2014, Toronto, ON, Canada

http://dx.doi.org/10.1145/2556288.2557336

## INTRODUCTION

Natural user interfaces (NUI) have received much attention recently, as they provide rich modalities and possibilities for interaction, beyond mouse and keyboard. Whilst the term is broad and can encompass touch, gesture, gaze, voice, and tangible input, NUI often implies leveraging the dexterity and higher degrees-of-freedom (DoF) of our hands for interaction. However, even when focusing on hands, NUI has vastly different interpretations. A decade ago, the term predominately referred to emerging multi-touch interfaces. More recently, it has become more synonymous with 3D touchless input using fingers, hands or whole body interactions, popularized by products such as Kinect and Leap Motion.

This has resulted in the emergence of many underlying NUI technologies designed with very different scenarios in mind, each with strengths and weaknesses. For example, Leap Motion is only able to estimate fingertips of a hand in-air, but at a high precision, where Kinect provides more flexibility in sensing dense depth maps of arbitrary scenes and tracking a human skeleton, but at a relatively low precision.

In this paper we present RetroDepth, a system that unifies a variety of NUI modalities such as freehand touch, pressure, in-air interactions, as well as stylus and other tangible object input. Our system is designed specifically for scenarios where users interact on and above physical surfaces of different shapes and sizes, and allows seamless switching between these modes of input. RetroDepth provides a middle ground between sensors that extract high-level features such as fingertips (e.g. Leap Motion) and more flexible depth cameras (e.g. Kinect) that expose raw depth maps but at lower precision.

Our system is designed specifically to sense only the *3D silhouette* of interacting objects. The use of silhouettes greatly lowers the computation required for depth sensing, when compared to estimating a dense depth map. We present a novel

vision-based approach for silhouette sensing, which leads to extremely high-precision contour estimation, and demonstrate sufficient accuracy to distinguish between touch, hover and pressure. Interestingly, most consumer depth cameras, such as stereo cameras, Kinect or even time-of-flight sensors (ToF), struggle the most at object boundaries and edges, where large depth discontinuities lead to noise and imprecise depth estimation. In this paper, we highlight the importance of precision at these object edges, as ultimately these form the main point of contact between the user and the digital/physical world.

We highlight numerous examples of how the resulting 3D silhouette information can be exploited for interactive scenarios. In particular, we demonstrate just how much information a silhouette can encode, and present a novel pipeline that identifies and tracks fingers and objects in 3D, and infers hand parts, poses and gestures, using 3D contours only. We compare the accuracy of our system with Kinect and Leap Motion, and conclude with a discussion of limitations and future work.

## RELATED WORK

Early uses of 2D silhouettes and contours in the context of interaction coincide with pioneering work on NUI, indicating their importance for enabling such experiences. Krueger's seminal work [21] and SideShow [9] enables whole-body and gestural interactions using 2D silhouettes. In recent years, 2D silhouettes and contours have been exploited in the context of vision-based tabletop systems [39, 28, 37, 20, 41]

Top-down configurations that place a 2D camera above a physical surface have proved popular as they provide more flexibility in sensing, allowing any regular desk to be transformed into an interactive one. Because these systems rely on 2D cameras, segmentation of hands and objects from the (typically cluttered) desktop is non-trivial, and touch and hover is difficult to disambiguate. These challenges have led researchers to use stereo cameras for higher precision touch sensing, combined with simple "hardware" techniques for segmentation, e.g. colored backgrounds or polarizing filters [25, 2]. Izadi et al. [16] use the same configuration as [2] to combine stereo-based touch estimation with stylus input, object recognition, and depth sensing, but the latter is used for remote feedback only.

Two recent notable products enable high-precision sensing using stereo cameras. The commercial Leap Motion sensor is a small, self-contained unit containing a pair of wide field-of-view (FoV) infrared (IR) cameras and IR LEDs pointing upwards on a desk. Whilst not published, the device provides tracking of fingertips and other strong peaks in the input signal, such as a pen, with high frame-rate and precision sensing. The device does not expose a full contour, only tracked 3D points mapping to high-level features such as fingertips. Further, segmentation is based on assuming that only freespace exists behind interacting objects. This limits the device to in-air input only, which can result in arm fatigue during prolonged interactions. Haptix [11] uses a similar small desktop unit with two IR cameras but instead points across the physical surface to detect multi-touch gestures only. Our approach combines these modalities of high-precision touch and in-air input, but also provides features such as pressure sensing, full contour-based interactions, gestures, and tracking of hands and tangible objects.

**Going beyond the tabletop** A logical next step for interactive surfaces is to explore in-air interactions, potentially in combination with touch. Systems have demonstrated stylus-based input using magnetic, inductive or vision-based systems (see [35, 40]). Many optical and non-optical techniques have been explored for freehand in-air input in the context of interactive surfaces. These include electric-field sensing techniques [30, 23], use of scanning lasers [30] and laser-line generators [36], or IR proximity sensors [29, 17, 4], either placed around the bezel or embedded behind a display. These techniques either coarsely sense 3D input, or have a limited interaction volume or are limited to 3D input only.

Other notable systems combine on-surface and in-air interactions using a variety of different hardware configurations. For example, the use of high-end (and costly) marker-based motion trackers used for 3D tabletops [26] or even 3D cardboard or foam prototypes [3]; as well as, synchronized IR cameras and switchable diffuser to image on and beyond the surface [18], where depth was coarsely estimated using diffuse IR light intensity falloff [13]; and light field imaging through an LCD to coarsely estimate depth [15]. Prior to the wide availability of depth cameras, researchers also explored ToF sensors placed behind [13] and above [43] projection surfaces. These sensors suffered from low resolution and high noise making interactions coarse and limited.

**The rise of consumer depth cameras** With the advent of consumer depth cameras, many new systems for in-air interactions coupled with surface-based interactions have appeared. Examples include situated augmented reality displays such as [14, 5], as well as augmented desktops [12, 24]. All these systems use a top-down Kinect camera to both sense touch [44] and in-air interactions. These systems inherently lack precision due to the quantization noise of the Kinect (see below) making such systems imprecise compared to other input devices [10]. 3Gear Systems [1] demonstrate tracking of specific hand gestures above a desktop, again using a top-down Kinect camera, based on early research of Wang et al. [38]. This method can be considered state-of-the-art in terms of Kinect-based hand tracking, and we provide quantitative comparisons later in this paper.

One of the main drawbacks with consumer depth cameras is high noise at object boundaries. However, for NUI, these boundaries are critical for sensing when the user's hands (or other physical tool) interacts with either digital or physical content. The original Kinect uses a method akin to stereo matching, where each small 2D patch of the reference dot pattern is matched by sweeping a window along the associated epipolar line in the observed IR image. This leads to ambiguities when matching at object boundaries, where large depth discontinuities lead to outliers, holes, and edge fattening [32, 8]. ToF sensors suffer from issues such as *multi-path* or *mixed pixels* [31], where light returns from secondary surfaces to the same pixel during a single exposure, as well as '*flying pixels*' where depth is estimated across multiple captures and scene motion results in foreground and background contributions being averaged together [31].

**Custom depth cameras** This fundamental issue of consumer depth cameras is a key motivation for RetroDepth, where we specifically design a system for real-time, high-

precision depth estimation at silhouette and object boundaries. Whilst consumer depth cameras lack precision, researchers have demonstrated much higher quality sensors, in particular by using dynamic structured light patterns (see [22] for a review). There are however limitations in such systems for our interactive scenarios. Most of these systems estimate depth by projecting multiple dynamic patterns into the scene and capturing multiple images with a camera. To deal with a moving scene (which is necessary in our interactive scenarios) and therefore limit motion artifacts, these patterns need to be projected and imaged at very high frame-rates. This necessitates costly high speed camera and projector hardware that can add considerably to the system cost. There is also the computational cost associated with estimating dense depth maps, which as demonstrated in this paper is unnecessary for many interactive scenarios.

### RetroDepth

In the following sections we present the physical RetroDepth setup and demonstrate the interactive capabilities of the system. The physical configuration is shown in Fig. 1. Two off-the-shelf monochrome cameras are positioned on a metal rail at a fixed baseline (6cm apart). Currently we use Lumenera PLT-425-NIR (running at 1024x512 and 60Hz) due to their IR response and global shutter capabilities, but there is no hard restriction on manufacturer or imaging sensor. A ring of 8 diffuse IR LEDs (OSRAM CHIPLED SFH 4053) operating at 850nm are attached around the lens of each camera. This placement and number of LEDs ensures that illumination is uniform and shadows are minimized. IR bandpass filters are placed on each camera lens.

This hardware setup is easily reproducible with only off-the-shelf components. The setup is powered entirely over USB (peak power for each LED is 260mW, and average power is 35mW). The LEDs are pulsed from each camera's General Purpose Input/Output (GPIO) pins, and only active during the camera exposure (2ms). A synchronization board is used to generate a 60Hz trigger to shutter the cameras simultaneously.

Figure 2: Left: passive retro-reflective keyboard, and series of other "physical widgets". Right: IR image of scene.

### Retro-reflective segmentation

Fig. 2 also shows retro-reflective surfaces (such as the off-the-shelf 3M Scotchlite-Part 8910), which are cut in various shapes and sizes, and placed on a table. Upon shuttering, the cameras both simultaneously capture an image of the scene. All IR LEDs are switched on during this exposure time, producing a bright uniform response from all the retro-reflective materials on the table, as shown in Fig. 2 (right). This makes these reflective surfaces readily distinguishable as bright silhouettes in the captured stereo images. When objects interact on or above these surfaces, a sharp contrast is created between the background and foreground, e.g. the hand or marker pen shown in Fig. 2 (right).

In a first pass, we extract contours of any bright silhouettes observed from each camera. We then perform stereo matching across these contour images. In a second pass, we *invert* the image within the contour boundaries, creating bright silhouettes within each retro-reflective region, which correspond to objects (e.g. hands or styluses), interacting on or above the surface (Fig. 2). We again extract contours and perform stereo matching in these regions. This creates a clean separation between sensing the retro-reflective objects, and the 3D interactions occurring on or above them. Optionally, we interpolate to fill in a dense 3D silhouette of foreground objects.

### Creating and tracking physical widgets

Retro-reflective material is cheap and readily procurable. It is also easy to cut into various shapes and sizes. Fig. 2 shows a variety of simple "*physical widgets*", made by cutting sheets of acrylic or cardboard into the desired shape and gluing the retro-reflective material on the upper-side. Additionally, a clear acetate sheet is used to print labels (most printer ink is invisible to IR) for the widget, such as keys for a 3D sensing touch keyboard (Fig. 2, left).

Our stereo matching algorithm estimates the precise metric depth of the outlines of these physical widgets. After depth estimation, a machine learning algorithm classifies the widget based on silhouette shape. This classification occurs per frame in real-time and is robust to large parts of the widget being occluded by interacting hands and objects. Once classified the widget can be tracked in 6DoF. Given that new physical widgets are easy to fabricate, our machine learning algorithm is extensible in order to add new training classes at runtime.

### Freehand 3D interactions and gestures

As the user's hand or physical tool interacts above a retro-reflective widget, a clear internal silhouette is visible (Fig. 2, right). By inverting the image, our stereo matching algorithm can precisely estimate the depth of these internal silhouettes. Even a small occluding region of the finger can be sensed accurately. This creates an *interaction volume* above each widget that allows freehand 3D input. Depending on the size of widget this can be either bimanual or single hand input, with fine-scale sensing of individual fingers (Fig. 3).

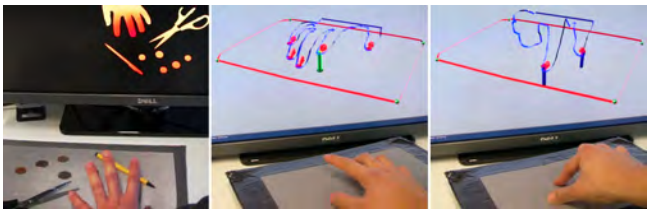

Figure 3: Left: Recovering the precise 3D contour of objects placed on the retro-reflector, including hands and other physical objects. Middle: Distinguishing between the widget (red contour) and the interacting objects (blue contour). Middle & Right: Sensing in-air (green), touch (red) and pressure (blue) input.

We repurpose the same machine learning pipeline to detect distinct hand shapes (i.e. when performing different gestures). Our system currently identifies a number of shapes including pointing, pinching, and whole hand interactions. Our approach can robustly handle variations across poses, and also estimate whether the left or right hand is observed.

After classifying the hand into distinct shapes, a machine learning classifier robustly identifies whether contour points belong to fingertips or the thumb, based on a curvature metric. These classified contour points can be clustered, and the individual fingertip peak identified. Simple line-fitting can be used to determine orientation of the fingertip in 3D. As shown in Fig. 1, Fig. 3 and accompanying video this allows for robust real-time identification of the digits of the hand.

### Stylus input and tangible tools

As shown in Fig. 3 and Fig. 4 a variety of objects placed on top of a retro-reflective surface can be clearly identified by their contours. These objects could either invoke specific UI commands, or be used for continuous input. One example of the latter is a stylus, which can be used for fine grained manipulations in 3D as shown in Fig. 1 and accompanying video. During contour extraction, we look for very sharp peaks in the signal to identify the pen tip, and estimate the pen location and orientation in 3D (using line fitting).

### Touch & Pressure

Beyond the 3D and tangible interactions outlined above, each retro-reflective widget has the capability of supporting touch and pressure input purely with our depth estimation method. To detect when finger or stylus touches a flat surface, a plane is estimated based on the widget's contour and intersections with any classified tips identified. One interesting possibility as shown in the accompanying video and Fig. 3 is to use a malleable material for the widget allowing users to press fingers into the substrate. Here the precision of our contour-based depth estimation method allows very small changes in the 3D location of the fingertip to be identified when the user presses the surface to approximate *pressure*.

### Example physical widgets

Fig. 2 shows a variety of different widgets that take minutes to fabricate. As highlighted just by virtue of covering an object with retro-reflective material, a interactive volume is created above the object, where touch, pressure, in-air and stylus input can be sensed readily, without any direct computational augmentation of the widget. Widgets that allow for 3D touch pads, mice, and keyboards can be easily created. Sheets of retro-reflector can be rolled out to enable ad-hoc 3D input devices. Physical toolbars and marking menus can be mapped to specific 2D UI functions, e.g. color selection.

## SOFTWARE PIPELINE

So far we have provided a high level view of the RetroDepth system and its sensing and interactive capabilities. In the next sections, we describe the software processing pipeline that takes pairs of synchronized images and estimates depth, and classifies widgets, hand shapes and salient hand features. The pipeline is composed of fairly standard image processing tasks, followed by a new contour-based stereo and classification algorithm. We briefly discuss these initial steps before focusing on the novel components.

First an offline calibration process is performed which comprises of: 1) *intrinsic calibration* to compute the geometric parameters of each IR camera lens (focal length, principal point, radial and tangential distortion); 2) *stereo calibration* to compute the geometric relationship between the two cameras, expressed as a rotation matrix and translation vector; 3) *stereo rectification* to correct the camera image planes to ensure they are scanline-aligned to simplify disparity computation. At runtime, the synchronized input IR images are *undistorted* given intrinsic lens parameters, scanline rectified, and finally cropped to ignore non-overlapping parts.


Figure 4: Early stages of processing pipeline. Top left: scanline-rectified stereo images. Bottom left: Foreground segmentation (in red) of internal objects within convex hull (blue). Note convex hull maps to each physical widget. Top right: extracted foreground contours. Bottom right: estimated depth of foreground silhouette (shown in false color).

The remainder of this online pipeline works in two steps. First, the depth for the background objects is computed, i.e. the visible widgets that are brightly lit. Second, depth is estimated for foreground objects, i.e. internal silhouettes within each widget. Because of the sparse nature of the contour-based algorithm, we can efficiently compute depth in two-passes.

To compute depth for background objects, we threshold both scanline-rectified images to remove low intensity pixels. We trace the contour of each binary image, which efficiently computes a set of 2D pixel coordinates corresponding to contour points for each connected component. At this point, stereo is computed on these background contours to estimate the 3D silhouettes of the widgets.

What remains is to compute the depth for internal foreground physical objects such as hands (the dark silhouettes inside the widget). We compute the convex hull for each background object, and generate a binary mask (1 within the convex hull, and 0 outside) for both stereo images. For each valid pixel in the mask, we invert the binary image, and repeat contour extraction on these points. This leaves us with two pairs of contour images; one corresponding to bright regions in the input images, and the other the silhouettes (of foreground objects) within these bright regions. Finally, our efficient stereo matching algorithm is computed for the foreground. (Fig. 4) shows an example of this processing pipeline.

### Efficient Contour-based Stereo

Stereo algorithms identify corresponding points in both images that are projections from the same scene point. We refer to [32, 8] for an in-depth discussion of stereo matching approaches. Because our input images are rectified, corresponding points are known to lie on the same horizontal scanline in the left and right image (see Fig. 5), which reduces the depth estimation to a 1D search task. The horizontal displacement is known as disparity $d$ and is inversely proportional to depth. In the following, we focus on the key problem of determining the correspondences between contour points that lie on a particular scanline $S$.

A core contribution of our work is extending the standard dynamic programming dense stereo algorithms to sparse contours, which greatly reduces the amount of computation, whilst increasing precision of disparity estimation.

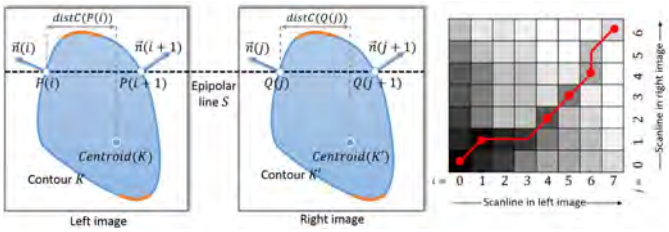


Figure 5: Left: contour-based depth estimation for rectified images. For notation please see text. Right: Cost Matrix $\mathcal{C}$. The brightness of each cell indicates the accumulated costs at $\mathcal{C}(i,j)$ (dark cells have low costs and bright ones high costs). The path of lowest costs is indicated in red. Matching pixels are marked with a red circle. All other pixels along the red path are occluded in the left or right image.

Let $p$ and $q$ be contour points in the left and right image, respectively. $P$ and $Q$ are lists of length $|P|$ and $|Q|$ that store the image $x$-coordinate of those contour points lying on scanline $S$ in the left and right image, respectively. $P(i)$ denotes the $i^{\text{th}}$ element in the list and we assume that the lists $P$ and $Q$ are sorted. For better understanding, Fig. 5 illustrates rectified stereo images that show a contour as well as lists $P$ and $Q$ for scanline $S$.

To find correspondences between the points in $P$ and $Q$, we compute the minimum-cost path through a matrix $\mathcal{C}$, illustrated in Fig. 5 (right). $\mathcal{C}$ is of size $|P| \times |Q|$ and $\mathcal{C}(i,j)$ stores the minimum accumulated cost of an optimal path from $(0,0)$ to $(i,j)$. The path of minimum cost is indicated in red in Fig. 5. Every path is restricted to start at cell $(0,0)$ and to end at cell $(|P|,|Q|)$, because we require a mapping for all contour points. Furthermore, only three moves are possible to construct a path: a diagonal $45°$ move that indicates a match, as well as horizontal and vertical moves that represent points that are only visible in the left or right image, respectively. The restrictions imposed on the path imply the following properties of the solution:

**Uniqueness property:** Every (contour) point in $P$ can only match to one point in $Q$ and vice versa. **Ordering property:** If point $P(i)$ matches $Q(j)$ than $P(i+1)$ can only match to $Q(j+\triangle)$ where $\triangle > 0$.

We enforce a third constraint, which we empirically found beneficial to improve robustness: Let $K$ be the contour that $P(i)$ belongs to and $K'$ the contour that $Q(j)$ belongs to. Then if $P(i)$ matches $Q(j)$ the height of the bounding boxes of $K$ and $K'$ must not differ by more than 50 pixels.

### Energy Function

The (accumulated) cost of a path is defined recursively (for $i > 0$ and $j > 0$) as:

$$\mathcal{C}(i,j) = \min \begin{cases} \mathcal{C}(i-1,j-1) + C_{\text{match}}(i,j) + C_{\text{smooth}}(i,j) \\ \mathcal{C}(i-1,j) + \lambda_{\text{occlusion}} \\ \mathcal{C}(i,j-1) + \lambda_{\text{occlusion}} \end{cases} \tag{1}$$

where the three different cases correspond to the three permitted moves discussed above. The boundary conditions are

$$\begin{aligned} \mathcal{C}(0,0) &= C_{\text{match}}(0,0) \\ \mathcal{C}(i,0) &= i \cdot \lambda_{\text{occlusion}} \qquad i > 0 \\ \mathcal{C}(0,j) &= j \cdot \lambda_{\text{occlusion}} \qquad j > 0 \end{aligned}$$

In Eq. 1, $\lambda_{\text{occlusion}}$ is a constant occlusion penalty and the remaining terms are defined as follows.

The data term $C_{\text{match}}$ measures the compatibility of putatively matching contour points:

$$\begin{aligned} C_{\text{match}}(i,j) = &\|\vec{n(i)} - \vec{n(j)}\| \\ &+ |\text{distC}(P(i)) - \text{distC}(Q(j))| \,. \end{aligned} \tag{2}$$

Here, the first part measures the Euclidean distance of the normal vectors $\vec{n(i)}$ and $\vec{n(j)}$ at contour points indexed by $P(i)$ and $Q(j)$, respectively. The second part compares the horizontal distance of point $P(i)$ and $Q(j)$ to the centroid of their corresponding contours $K$ and $K'$, respectively: $\text{distC}(P(i)) = P(i) - \text{Centroid}(K)$.

The pairwise term $E_{\text{smooth}}$ encourages solutions where the horizontal distance between two neighboring *matching* points $P(i)$ and $P(\varphi(i))$ is similar to the distance of their matching points $Q(j)$ and $Q(l)$:

$$E_{\text{smooth}}(i,j) = \|P(i) - P(\varphi(i)) - (Q(j) - Q(\varphi(j)))\|, \tag{3}$$

where function $\varphi()$ returns the closest previous matching point for the current path.

### Optimization via Dynamic Programming

Dynamic programming is a well established technique to find the path of minimal cost through $\mathcal{C}$. The optimization consists of two consecutive steps. First, the cumulative costs $\mathcal{C}(i,j)$ are computed recursively for every pair $(i,j)$ as per Eq. 1. At each recursion we also store the best "move" (i.e. the $\arg\min$ of Eq. 1) at $(i,j)$ in a matrix $\mathcal{M}$ that has the same dimensions as $\mathcal{C}$. In the second step we reconstruct the best path by tracing back the best moves stored in $\mathcal{M}$ starting from $(|P|,|Q|)$ until we reach the origin $(0,0)$ of $\mathcal{M}$. Once the best path is computed the disparity $d(i)$ (with respect to the left image) can be derived as $d(i) = P(i) - Q(j)$ for matching points $P(i)$ and $P(j)$.

### Post Processing

Although the estimated disparity map is usually of high quality, there might be outliers and contour points where no depth could be estimated (e.g. due to occlusion of this point in the other image). In the post-processing stage we aim to filter out wrong matches and assign *all* points along the contour to a depth value. In particular we apply the following steps:

**Invalidation of horizontal lines.** The depth estimation can be unreliable at contour points whose normal is almost perpendicular to the scanlines (such regions are marked in orange in Fig. 5). This is because the term $C_{\text{match}}$ is ambiguous in those regions. Therefore, we invalidate contour points whose normal vector has an angle of $\geq 85°$.

**Outlier invalidation.** The depth map obtained with dynamic programming is computed independently for each scanline. As a consequence, the depth at neighboring contour points that lie across scanlines might be inconsistent. Thus we invalidate points whose depth differs from those of the closest neighboring points along the contour by more than 3mm.

**Contour Smoothing.** We further smooth the depth values along the contour with a 1D mean filter of size 25 pixels. Note this filter can be implemented very efficiently using a sliding window technique with two operations per contour pixel.

**Filling.** Finally, we assign occluded and invalidated contour points to a depth value. This depth value is computed as a linear combination of the depth assigned to the closest valid contour points.

### Discussion

One important feature of our algorithm is the underlying computational cost. In general, the complexity of a pixel-wise stereo matching algorithm can be characterized with $O(WHD)$, where $W$ and $H$ are the width and height of the images, respectively, and $D$ is the number of tested depth hypotheses. In many scenarios $D = O(W)$, i.e. higher resolution images imply a larger set of depth hypothesis to be considered. Coarse-to-fine frameworks for stereo (e.g. [34]) and randomized stereo correspondence algorithms [6] (largely) remove the dependency of the runtime complexity on $D$. Since in our hardware setup the most discriminative regions in the image are the silhouettes, we restrict the computation of depth to pixels lying on the silhouette contours. Further, we obtain a substantial reduction in the search range $D$, since only a small set of extracted contour pixels in both images can be potential matches ($\sim$2-10 points).

## 1D HAND POSE AND FINGERTIP RECOGNITION

Once we have efficiently estimated the 3D silhouette, this high-quality contour contains valuable information about the scene objects, and in particular on the state and pose of any interacting hands – a key to enable truly interactive scenarios.

There are many options for detecting gestures and salient features of the hand. One simple approach detects points of high curvature as fingertip candidates (during contour tracing). Whilst a heuristic, these types of rules can produce convincing results [25]. However, there are often failure cases, such as complex poses where knuckles, styluses or other high curvature features confuse the peak detector. To address some of these issues, we present a novel contour (1D) based hand state and part classification algorithm, which is complimentary to more regular peak detection in specific scenarios.

For hand state classification we use a random decision forest (RDF) [7], which has been shown to be effective for both body [33] and hand [19] pose estimation using depth images. These approaches are trained to be invariant to pose and shape variations, use scale-invariant features, and achieve a robust result by averaging over multiple pixels. For RetroDepth, we adapt these approaches to silhouettes, keeping their benefits, while further reducing the complexity from 2D to 1D.

### RDF based contour classification

As a key contribution of this work, our classifiers operate on 1D contours instead of 2D images. These contours are essentially 1D sequences of 3D points in world space, sampled with a rate determined by the camera resolution and setup. For notational convenience, and without loss of generality, we will assume that the contours can be parameterized by a variable $s$, such that $X(s)$ gives the world coordinates of the point $s$ on the curve, and a unit step in $s$ on the 1D sequence corresponds to a unit step along the contour in 3D world space.

Each point on the contour is associated with two class labels $y^{\mathrm{s}}$ and $y^{\mathrm{f}}$, where $y^{\mathrm{s}}$ is a hand shape label (such as pointing, pinching, grasping or open hand) and $y^{\mathrm{f}}$ is a fingertip label (such as index, thumb, or non-fingertip). The task of the RDF is to classify the hand shape for the entire contour, and localize and identify the fingertips.


Figure 6: Left: For a given point $s$ on the contour, we move by the offsets $u_1$ and $u_2$ to reach points $X(s+u_1)$ and $X(s+u_2)$. Middle: Hand shape class label $y^{\mathrm{s}}$ (one shared label for all pixels). Right: Fingertip class labels $y^{\mathrm{f}}$.

Decision trees use the internal split nodes to test and guide the input to one of the leaves, where class distributions of the output labels are stored. Test functions at split nodes are of the form: $f(F) < T$, where the function $f$ maps the features $F$ onto a line, and $T$ acts as a threshold. We use the following test function:

$$f(s, u_1, u_2, \vec{p}) = [X(s+u_1) - X(s+u_2)]_{\vec{p}} \qquad (4)$$

where $[.]_{\vec{p}}$ is a projection onto the vector $\vec{p}$, and $\vec{p}$ is one of the primary axes $\vec{x}$, $\vec{y}$ or $\vec{z}$. This test probes two offsets on the contour, gets their world distance in the direction $\vec{p}$, and this distance is compared against the threshold $T$ (Fig. 6). Because the parameterization of $s$ is normalized, the offsets $u_1$ and $u_2$ are scale invariant. The test function splits the data into two sets and sends each to a child node. The quality of a split is determined by the information gain defined as follows:

$$G(u_1, u_2, T) = H(C) - \sum_{s \in \{\mathrm{L,R}\}} \frac{|C_s|}{|C|} H(C_s) \qquad (5)$$

where $H(C)$ is the Shannon entropy of the class label distribution of the labels $y$ in the sample set $C$, and $C_{\mathrm{L}}$ and $C_{\mathrm{R}}$ are the two sets of examples formed by the split. At training time, multiple features and thresholds are uniformly sampled from a large range for each split node, and the one with the highest information gain is selected. Once leaves are reached, the class distribution of the remaining pixels in the node are saved. As each pixel has two labels from distinct sets in our case, we keep two histograms at each leaf: one for hand shape and one for fingertips.

At run-time, every pixel is evaluated independently with each tree in the forest and they descend into one of the leaves. For hand shape classification, the associated distributions read from the leaves are pooled across the contour to form a final set of state probabilities. The mode of this distribution is selected as the hand shape for the contour $X$. The fingertip localization algorithm is instead more akin to the part classification method of [33]. After classification, we apply a 1D running mode filter to the labeled contour to filter out the noisy labels. We finally apply connected components to give labels to the fingers. The system selects the point with the largest curvature as the fingertip.

To train a single forest that jointly handles shape classification and fingertip localization, we first disregard the $y^{\mathrm{f}}$ labels and calculate the information gain only using $y^{\mathrm{s}}$ until we reach a certain depth $m$. From then on, we switch to using $y^{\mathrm{f}}$ labels

instead, and do not use $y^s$ to select features. This has the effect conditioning each subtree that starts at depth $m$ to the shape class distributions at their roots. This is similar to the idea presented in [19], in that the low level features are conditioned on the high level feature distribution. However, instead of using a multi-layered approach, we handle the classification jointly in a single forest.

We use a realistic synthetic hand model to automatically generate high-quality hand contours with labels. The hand model has 32 DoF, which can be used to automatically pose the object in a range of parameters, with added random noise on every joint angle. We generate a total of around 8000 left-hand images for each hand state class. These are then mirrored and given right hand labels. Fingertips are labeled by mapping the model with a texture that signifies different regions with separate colors. An example of the hand shape and fingertip labels for a certain pose can be seen in Fig. 6.

#### Classifying other object contours

Our method for hand posture classification can also be leveraged to classify retro-reflective widgets. Even though the shapes are mostly primitive shapes e.g. rectangles and circles, applying template matching is not optimal. This is mainly because the contour of the object changes while the user is interacting with it, and also because for every type of object, a new set of heuristics or rules need to be generated. An off-the-shelf RDF on the other hand can learn all the variation of the contours for every shape. It is accurate and fast, and can learn to recognize new objects at run-time.

To reduce the variation due to occlusion, we first estimate the convex hull of the objects and evaluate this simplified contour. We use real data for training, since labeling the contours is straightforward in this case. The same features and objective function are used as in Equation 4 and 5. As the contours are much simpler in this case, much shallower forests are sufficient for online training.

#### Recognizing a stylus

Pens are typically much sharper than fingers and therefore easier to recognize with heuristics. On both small objects where we have partial hand contours, as well as on larger surfaces where we have the full hand contour, finding the sharpest point on the contour (and thresholding it) is a good estimator for the pen tip location. However, the pen changes the contour of the hand shape considerably, which reduces the accuracy of the hand state classifier and causes the fingertip localizer to fail. Therefore, when a pen is detected on a large surface, the pen contour points are first removed from the contour. This can be done by following the pen contour in both directions until the gradient of the contour changes abruptly. The corresponding points are then connected to form the hand contour.

#### Discussion and results

When users interact with smaller widgets, where only a few fingers or the pen is visible, hand state recognition is not desired as we have only a partial hand contour. In these cases, the pen and fingers can be found by simple peak detection. When the hand is on the large rectangular surface however, simple heuristics fail to identify or localize fingers robustly (e.g. finding peaks at knuckles). Examples that would very likely be failure cases for standard heuristic-based approaches such as detecting pinch [42] or fingertips [25] are shown in Fig. 7. Whilst these reinforce the need for a machine learning approach, our method is also complimentary to more heuristic-driven approaches.

| Class | Thumb | Index | Middle | Ring | Pinky | Reject |
|---|---|---|---|---|---|---|
| Accuracy | 0.848 | 0.754 | 0.615 | 0.594 | 0.796 | 0.882 |

Table 1: Per-pixel classification accuracy for the individual fingertip classes, before filtering.

Figure 7: Examples of results based on complex poses and visually ambiguous poses. Ground truth data shown left and test results shown right. Poses shown from left to right: two pinch gestures, a splayed hand with an occluded finger, and fused fingers on a splayed hand.

We conducted experiments with eight hand states, corresponding to left and right handed versions of four hand poses, namely "pointing", "pinching", "open hand looking down", and "open hand looking up". For each hand state we have 4000 synthetic left hand images, which are mirrored to retrieve data for the right hand. We train on half of the samples and test on the remaining ones.

We train an RDF with a single tree until depth 18 with the state labels only, and calculate the confusion matrix over all hand states. The RDF reaches perfect accuracy for both left and right handed versions of three of the command modes, and only shows confusion between the left– and right–handed pointing gestures. The success rate of left–handed pointing is 93% and right–handed pointing is 92%, and the corresponding left-right confusion is 7% for the former and 6% for the latter case. This gives us an overall command mode classification accuracy of 100% and left-right hand classification accuracy of 98% when all states are considered.

The synthetic images also have labels for the five fingertips and the reject class, corresponding to the non-fingertip areas. After the state classification training concludes at depth 18, we continue training with the fingertip labels until depth 25. The leaves are assigned the distributions of both state and fingertip labels. The success rates are given in Table 1. Even though we are operating on 1D contours, with much lower computation, the per-pixel classification rates are similar to the reported results in [33] and [19].

### CONTOUR INPAINTING

So far we have purposefully pushed the limits of what can be achieved purely based on silhouettes. However, there are certain scenarios and legacy applications that require fully dense depth maps. In this section, we show a robust and fast method for inpainting the contour of objects. The results of our contour inpainting algorithm can be found in Fig. 8.

In order to fill the interior $\Omega$ of our segmented contour $\partial\Omega$, we formulate the interpolation problem as a Laplace equation with Dirichlet boundary condition. Hence, contour depth values define a known scalar-valued boundary function $\hat{f}(x,y)|_{\partial\Omega}$ and we need to solve for the interior depth values $f(x,y)$ over $\Omega$ with:

$$\Delta f(x,y) = 0 \qquad (x,y) \in \Omega - \delta\Omega$$
$$f(x,y)|_{\partial\Omega} = \hat{f}(x,y)|_{\partial\Omega} \qquad otherwise$$

Then, we use a parallel Jaccobi/Gauss-Seidel solver on the GPU to solve the Laplace equation. In order to speed up convergence, we initialize the interior using a hierarchical pull-push interpolation with bi-cubic interpolation weights [27]. Note that we mark pixel values of $\partial\Omega$ with a negative sign, which allows us (except for the pull-push hierarchy) to solve the system in place. The contour inpainting results can be seen in the supplemental video.

## EXPERIMENTS

We now evaluate the performance of our system compared to three baseline measures. As a first baseline, we choose Leap Motion (**Leap**) given the widely publicized precision of the system, and its availability as a commodity sensor. The other baselines are based on another, arguably more ubiquitous depth sensor: the Kinect. With this sensor, we chose to run two conditions. The first (**KinectLight**) uses the RetroDepth hand posture and fingertip tracking pipeline described previously, and uses contour-traced Kinect depth maps as input (instead of RetroDepth stereo data). This in part evaluates our software pipeline with a commodity sensor, but also gives us an indication of the precision of the stereo matching algorithm and hardware setup that the full RetroDepth system supports. The second Kinect baseline (**3Gears**) evaluates the hand tracker from 3Gear systems [1]. This can be considered the current widely-available state-of-the-art hand tracker for Kinect with scenarios close to RetroDepth.

### 3D Target Acquisition

In our first task, we wanted to evaluate one of the core features of RetroDepth, its 3D input precision. For all conditions, the same physical setup was used as shown in Fig. 9. This provided a physical input space of 30x30x30cm. Leap was placed on the table, Kinect and RetroDepth mounted top down. All cameras were configured to share the same center of origin. A large table was used with retro-reflective material visible for the RetroDepth condition. This material was removed for the other conditions to avoid IR interference. A PC with a 6-core 2.8GHz processor and Nvidia GTX480 GPU, and large display were used for all conditions. Our pipeline ran on this hardware at 60Hz.

**Participants** Twelve participants (9 male, 3 female) between the ages of 21 and 39 were recruited for the study. Participants were daily computer users, 2 were left handed. All had normal vision. Participants had no experience of using 3D hand gesture-systems, but some had experiences playing Kinect games.

**Task** We conducted a 3D targeting task where the physical interaction volume was subdivided into 27 equally sized cells

Figure 8: Contour inpainting results in false color.

Figure 9: Study setup. Left: Kinect, Leap and RetroDepth setups for 3D acquisition task. Middle: Kinect touch evaluation. Right: Leap touch evaluation.

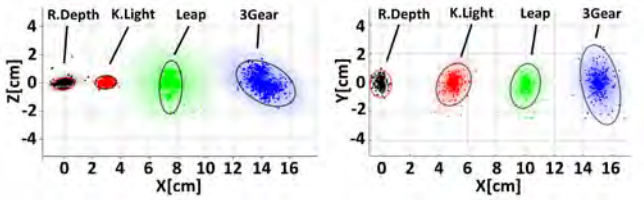


Figure 10: Left: touch targeting results, and right: 3D acquisition targeting results.

(3x3x3 grid). The user began by holding their finger in a 3D transparent box appearing centered and at the front of the 3D scene. After a fixed time, a billboarded target shown in Fig. 9 (left screen) appeared centered in one cell for users to select. 3 trials were performed per condition. Starting and target positions were coupled a priori to reduce in-condition variance. This set was used across all conditions, with the order randomized, ensuring that the total distance traveled was exactly equal across all users, and each condition. Measurement of task completion was triggered once the fingertip left the starting position and ended once the user entered the target triggering area. The targets were 5x5cm high.

**Procedure** The experiment employed a within-subject repeated measure design. The independent variable was the input type: RetroDepth, Leap, KinectLight, 3Gears. The dependent measure was the targeting accuracy (measured as a distance of the fingertip to the target center upon selection). The presentation order of the conditions was counterbalanced using a Latin Square design across users. A short training phase was first performed, after which each user was asked to perform the task as accurately as possible. The entire experiment lasted about 60 minutes. Participants filled out a post-experiment questionnaire upon completion. Users were observed at all times as the task was performed, and notes on subjective experiences were taken.

### Results

Our 12 participants produced a total of 3888 selections in this task. No effect was found between the four blocks of trails (e.g. from fatigue or learning effects). Additionally there was no significant differences between participants. This allows us to combine data to cluster per condition results as shown in Fig. 10 (right). The plot shows the average of the target hits across all participants, clustered by condition. Note we ignore the Z contribution as the targets were planar, and depth is measured along the Y axis in our 3D scene. The more compact the cluster the more accurate the result. 95% confidence ellipses are shown for each condition.

We compute the minimum target sizes required to achieve 95% accuracy for each condition. RetroDepth achieves the

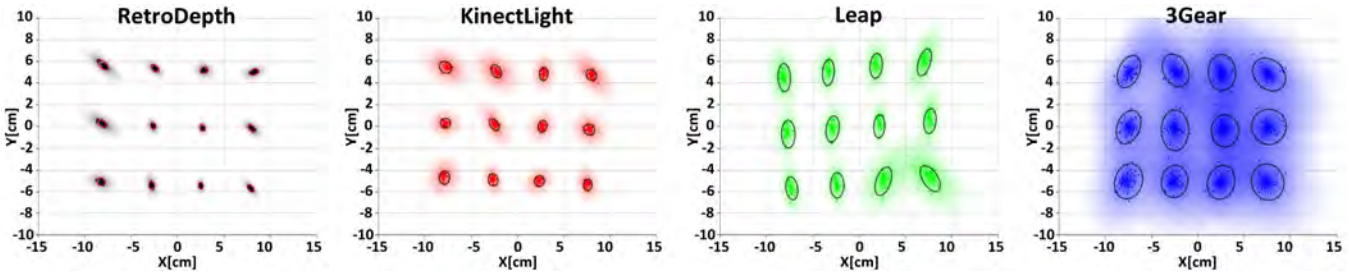


Figure 11: 2D touch results across all four conditions.

smallest minimum target size of 1.95x1.55cm with error X=7.89mm and Y=10.90mm. 3Gears achieves the largest minimum target size of 5.30x3.15cm with error X = 20.86mm and Y=29.74mm. KinectLight achieves a size of 2.89x2.08cm with error X= 15.24mm and Y=24.87mm. Leap performs similarly achieving a size of 3.18x2.00cm with error X=17.53mm and Y=22.67mm.

### Physical Touch Performance

Another important feature of RetroDepth is the ability to support touch sensing. To evaluate the performance of the system against our baseline, we ran a further experiment, where each camera (Kinect, Leap and RetroDepth) was setup in top down arrangement Fig. 9 (middle and right). For Leap this meant adding a sheet of IR absorbing film to the table. A 20cm$\times$25cm acrylic sheet was used as a touch surface, with small laser etched targets (spaced 5cm apart), and sized for the user's fingertips. 12 participants, 3 women and 9 men, were asked to perform a physical touch task. The aim was to measure system performance in terms of touch accuracy, and overall sensor precision. Participants were asked to touch all 12 targets in turn over 5 trials. The 3D point was recorded once the user's self reported. Fig. 11 shows the X-Y 2D accuracy plots for each of the sensors. Fig. 10 (left) shows the Z-Y accuracy plots (Z axis corresponds to depth precision). Based on these statistics it is possible to define the minimum 3D target bounds to capture 95% of the touches. For RetroDepth this is 1.69x0.98x0.75cm, KinectLight is 1.49x1.29x0.85cm, Leap is 3.88x2.91x 1.63cm, 3Gear is 4.41x3.71x2.78cm.

### Discussion

We found the results extremely encouraging given the precision of RetroDepth compared to these baselines. It is however important to note that we are not comparing like-for-like necessary. In terms of fidelity, for example, Leap uses lower cost cameras, and we use more costly research cameras. Nor was the Leap designed for touch-like scenarios (i.e. pointing downwards). Although we feel that the touch accuracy of the Leap is good enough to warrant future work on this scenario, based on our IR absorbing idea. On the other hand, RetroDepth is a research prototype whilst others are engineered products. Our system also offers different data (silhouettes) than the Leap and the Kinect, and none of these elements are being measured. However, with all this in mind our results are encouraging in terms of precision and performance.

An interesting finding is how well Kinect performed in the KinectLight condition, particularly for touch sensing. This illustrates benefits of our software pipeline for improving touch and 3D acquisition. Another interesting finding is that user's perceptions and performance can be very different. Leap surprisingly did not perform as well as expected, yet it was ranked a close second to RetroDepth in terms of preference. Many users described it as fluid, which indicates that the higher frame-rate has a strong impact in terms of NUI experience, potentially even distorting views of sensor accuracy. The 3Gear performance also gives an interesting insight. Whilst the system appeared to be performing well, showing compelling 3D models fitting the Kinect data, this model fitting approach ultimately led to severe inaccuracies for touch and selection, as there will always be discrepancies between a generic model and the user's hand.

Overall we feel that RetroDepth offers an exciting new research platform for future work on NUI interfaces, leading to new applications that incorporate touch, pressure, 3D, and even tangible input. There are of course limitations with our approach. Ultimately, 3D silhouettes are a lower dimensional representation of the real world, and in that regard data can be lost. The interesting finding from our experiments is just how much information is encoded in the silhouette allowing high accuracy rates for hand pose and part classification. Another clear finding is around frame rate, where user feedback suggests that trading some precision for speed may be desirable, and where there are significant computational advantages of working with silhouettes.

## CONCLUSIONS

In this paper we have presented RetroDepth a system that allows for high-precision 3D sensing on and above physical surfaces. We have provided several key contributions: 1) A fully working real-time 3D silhouette sensing technology which combines (for the first time) the common NUI modalities of touch, pressure, stylus, objects, and 3D, into a single system. 2) a new stereo matching algorithm for computing fast and precise 3D contours. 3) a new 1D contour classification algorithm, for robustly identifying physical tools, hand poses and salient features. 4) A software pipeline with practical impact even for regular Kinect sensors.

## REFERENCES

1. 3Gear Systems Inc. http://threegear.com/, 2013.
2. Agarwal, A., Izadi, S., Chandraker, M., and Blake, A. High precision multi-touch sensing on surfaces using overhead cameras. In *Tabletop'07*, 197–200.
3. Akaoka, E., Ginn, T., and Vertegaal, R. Displayobjects: prototyping functional physical interfaces on 3d styrofoam, paper or cardboard models. In *TEI'10*, ACM (2010), 49–56.
4. Annett, M., Grossman, T., Wigdor, D., and Fitzmaurice, G. Medusa: a proximity-aware multi-touch tabletop. In *UIST'11*, ACM (2011), 337–346.

5. Benko, H., Jota, R., and Wilson, A. Miragetable: freehand interaction on a projected augmented reality tabletop. In *CHI'12*, ACM (2012), 199–208.
6. Bleyer, M., Rhemann, C., and Rother, C. Patchmatch stereo - stereo matching with slanted support windows. In *BMVC* (2011), 1–11.
7. Breiman, L. Random Forests. *Machine Learning 45*, 1 (Oct. 2001), 5–32.
8. Brown, M. Z., Burschka, D., and Hager, G. D. Advances in computational stereo. *PAMI 25*, 8 (2003), 993–1008.
9. Davis, J. W., and Bobick, A. F. Sideshow: A silhouette-based interactive dual-screen environment. Tech. rep., MIT, 1998.
10. Dippon, A., and Klinker, G. Kinecttouch: accuracy test for a very low-cost 2.5 d multitouch tracking system. In *ITS'11*, ACM (2011), 49–52.
11. Haptix. http://www.haptixtouch.com/, 2013.
12. Haubner, N., Schwanecke, U., Dörner, R., Lehmann, S., and Luderschmidt, J. Detecting interaction above digital tabletops using a single depth camera. *Machine Vision and Applications* (2013), 1–13.
13. Hilliges, O., Izadi, S., Wilson, A. D., Hodges, S., Garcia-Mendoza, A., and Butz, A. Interactions in the air: adding further depth to interactive tabletops. In *UIST'09*, ACM (2009), 139–148.
14. Hilliges, O., Kim, D., Izadi, S., Weiss, M., and Wilson, A. Holodesk: direct 3d interactions with a situated see-through display. In *CHI'12*, ACM (2012), 2421–2430.
15. Hirsch, M., Lanman, D., Holtzman, H., and Raskar, R. Bidi screen: a thin, depth-sensing lcd for 3d interaction using light fields. In *TOG*, vol. 28, ACM (2009), 159.
16. Izadi, S., Agarwal, A., Criminisi, A., Winn, J., Blake, A., and Fitzgibbon, A. C-slate: a multi-touch and object recognition system for remote collaboration using horizontal surfaces. In *Tabletop'07*, IEEE (2007), 3–10.
17. Izadi, S., Hodges, S., Butler, A., West, D., Rrustemi, A., Molloy, M., and Buxton, W. Thinsight: a thin form-factor interactive surface technology. *CACM 52*, 12, 90–98.
18. Izadi, S., Hodges, S., Taylor, S., Rosenfeld, D., Villar, N., Butler, A., and Westhues, J. Going beyond the display: a surface technology with an electronically switchable diffuser. In *UIST'08*, ACM (2008), 269–278.
19. Keskin, C., Kirac, F., Kara, Y. E., and Akarun, L. Hand Pose Estimation and Hand Shape Classification Using Multi-layered Randomized Decision Forests. In *ECCV'12* (2012).
20. Koike, H., Sato, Y., and Kobayashi, Y. Integrating paper and digital information on enhanceddesk: a method for realtime finger tracking on an augmented desk system. *TOCHI 8*, 4 (2001), 307–322.
21. Krueger, M. W. *Artificial reality II*, vol. 10. Addison-Wesley Reading (Ma), 1991.
22. Lanman, D., and Taubin, G. Build your own 3d scanner: 3d photography for beginners. In *ACM SIGGRAPH 2009 Courses*, ACM (2009), 8.
23. Lee, J., Park, K. S., and Hahn, M. The 3d sensor table for bare hand tracking and posture recognition. In *Advances in Multimedia Modeling*. Springer, 2006, 138–146.
24. Liu, Y., Weibel, N., and Hollan, J. Interactive space: A framework for prototyping multi-touch interaction on and above the desktop. In *CHI'13 EA*, ACM (2013).
25. Malik, S., and Laszlo, J. Visual touchpad: a two-handed gestural input device. In *ICMI'04*, 289–296.
26. Marquardt, N., Jota, R., Greenberg, S., and Jorge, J. A. The continuous interaction space: interaction techniques unifying touch and gesture on and above a digital surface. In *INTERACT'11*, Springer (2011), 461–476.
27. Marroquim, R., Kraus, M., and Cavalcanti, P. R. Efficient point-based rendering using image reconstruction. In *SPBG*, IEEE (2007), 101–108.
28. Matsushita, N., and Rekimoto, J. Holowall: designing a finger, hand, body, and object sensitive wall. In *UIST'97*, ACM (1997), 209–210.
29. Moeller, J., Kerne, A., and Moeller, J. Zerotouch: An optical multi-touch and free-air interaction architecture. In *CHI'12*, ACM (2012), 2165–2174.
30. Paradiso, J. A. Several sensor approaches that retrofit large surfaces for interactivity. In *UbiComp 2002* (2002).
31. Remondino, F., and Stoppa, D. *Tof range-imaging cameras*. Springer, 2013.
32. Scharstein, D., and Szeliski, R. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. In *IJCV* (2002).
33. Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., and Blake, A. Real-time Human Pose Recognition in Parts from Single Depth Images. In *CVPR* (2011).
34. Strecha, C., and Van Gool, L. PDE-based multi-view depth estimation. In *Proc. 3DIMPVT* (2002), 416–425.
35. Subramanian, S., Aliakseyeu, D., and Lucero, A. Multi-layer interaction for digital tables. In *UIST'06*, ACM (2006), 269–272.
36. Takeoka, Y., Miyaki, T., and Rekimoto, J. Z-touch: an infrastructure for 3d gesture interaction in the proximity of tabletop surfaces. In *ITS'10*, ACM (2010), 91–94.
37. Ullmer, B., and Ishii, H. The metadesk: models and prototypes for tangible user interfaces. In *UIST'97*, ACM (1997), 223–232.
38. Wang, R., Paris, S., and Popović, J. 6d hands: markerless hand-tracking for computer aided design. In *UIST'11*, ACM (2011), 549–558.
39. Wellner, P. Interacting with paper on the digitaldesk. *CACM 36*, 7 (1993), 87–96.
40. Wesche, G., and Seidel, H.-P. Freedrawer: a free-form sketching system on the responsive workbench. In *VRST*, ACM (2001), 167–174.
41. Wilson, A. D. Playanywhere: a compact interactive tabletop projection-vision system. In *UIST'05*, ACM (2005), 83–92.
42. Wilson, A. D. Robust computer vision-based detection of pinching for one and two-handed gesture input. In *UIST'06*, ACM (2006), 255–258.
43. Wilson, A. D. Depth-sensing video cameras for 3d tangible tabletop interaction. In *Tabletop'07*, IEEE (2007), 201–204.
44. Wilson, A. D. Using a depth camera as a touch sensor. In *ITS'10*, ACM (2010), 69–72.