FlowPut: Environment-Aware Interactivity for Tangible 3D Objects

JAN RIEMANN, MARTIN SCHMITZ, ALEXANDER HENDRICH, and MAX MÜHLHÄUSER, TU Darmstadt, Germany

Tangible interaction has shown to be beneficial in a wide variety of scenarios since it provides more direct manipulation and haptic feedback. Further, inherently three-dimensional information is represented more naturally by a 3D object than by a flat picture on a screen. Yet, today's tangibles have often pre-defined form factors and limited input and output facilities. To overcome this issue, the combination of projection and depth cameras is used as a fast and flexible way of non-intrusively adding input and output to tangibles. However, tangibles are often quite small and hence the space for output and interaction on their surface is limited. Therefore, we propose FlowPut: an environment-aware framework that utilizes the space available on and around a tangible object for projected visual output. By means of an optimization-based layout approach, FlowPut considers the environment of the objects to avoid interference between projection and real-world objects. Moreover, we contribute an occlusion resilient object recognition and tracking for tangible objects based on their 3D model and a point-cloud based multi-touch detection, that allows sensing touches also on the side of a tangible. Flowput is validated through a series of technical experiments, a user study, and two example applications.

CCS Concepts: • **Human-centered computing → Information visualization**; **User interface toolkits**; *Ubiquitous and mobile devices*; *Haptic devices*; Ubiquitous and mobile computing systems and tools;

Additional Key Words and Phrases: Displays, touch, object tracking, projection, layout, optimization

ACM Reference Format:
Jan Riemann, Martin Schmitz, Alexander Hendrich, and Max Mühlhäuser. 2018. FlowPut: Environment-Aware Interactivity for Tangible 3D Objects. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 2, 1, Article 31 (March 2018), 23 pages. https://doi.org/10.1145/3191763

1 INTRODUCTION

Today's screen-based touch interfaces are often criticized to lack haptic experience. To mitigate this, researchers propose tangible user interfaces that allow for a more direct haptic interaction with 3D objects [16]. Naturally, such tangible objects are well suited to carry three-dimensional information, which is often lost on flat screen-based 2D interfaces, especially for non-experts (e.g. interpreting a 2D relief map compared to a 3D tangible of the relief map). However, an interactive system is required to provide input and output for such tangibles.

Research in tangible UIs already investigated in- and output on tangibles. Work has shown that, for instance, if the tangible is used on a tabletop display, optical pipes placed within the object, transferring the light from the screen under the tangible to its surface, enable basic output [55]. However, this approach is not viable for conventional tables. In this case, top-projected augmentation is often used, allowing to project onto the tangible's surface as well. In order to do so, tracking of the tangibles is required. To that end, fiducial markers are often used

Authors' address: Jan Riemann, riemann@tk.tu-darmstadt.de; Martin Schmitz, schmitz@tk.tu-darmstadt.de; Alexander Hendrich, hendrich@tk.tu-darmstadt.de; Max Mühlhäuser, max@tk.tu-darmstadt.de, TU Darmstadt, Hochschulstraße 10, 64289, Darmstadt, Hessen, Germany.

2474-9567/2018/3-ART31 $15.00
https://doi.org/10.1145/3191763

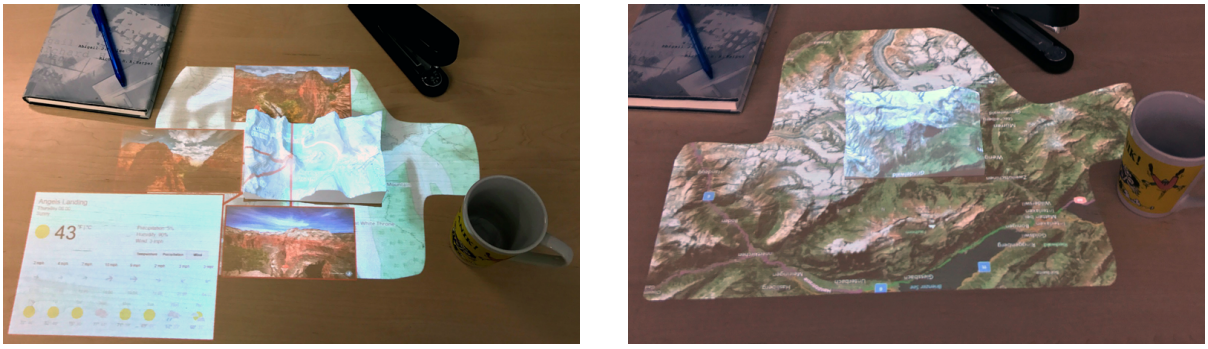

Fig. 1. The space on and around a tangible is augmented with an environment-aware projection which places free floating UI elements like images or information widgets (left) or maximizes the space of an UI element (right) based on the environment.

[23, 26, 33, 34]. However, they require modification of the tangible (which can be a problem if arbitrary objects shall be used) and are visible to the user.

In a different line, 3D printing has evolved to a level where it is possible to embed sensing and basic output into the 3D-printed objects [49, 55]. While this would generally allow for interactive tangibles, the actual possibilities are still very limited (e.g. it is not possible to print the whole surface as a display) and the process is very complex, time-consuming and requires a lot of expertise, thereby limiting the possible use cases. Further, in all variants, touch input on the tangible's surface is difficult to realize, but necessary to interact with the UI on the tangible.

We propose FlowPut, a framework to provide input and output for many 3D objects (see Figure 1). To that end, FlowPut tracks 3D objects based on their digital 3D model, augments their surface and surroundings using a top-mounted projector and provides multi-touch input across the objects' and the table's surfaces. Further, we contribute environment-aware layout functions for placement of UI elements on and around the object that avoid clutter and bad projection (e.g. on text documents that are lying around). The 3D models required for object tracking can either be provided from an existing file or be scanned ad hoc, allowing to use many 3D objects with FlowPut. For developers, FlowPut provides touch and object movement events and automatically generates environment-aware layouts, consisting of any standard UI controls.

Using FlowPut, one could imagine planning a hiking trip: To inform oneself about the available routes, tourist information offices usually provide a large selection of brochures and maps. For the experienced hiker, there is no problem to understand a route's difficulty by looking at a flat relief map. The occasional hiker might have problems, resulting in a wrong assessment whether she's capable of taking the route. Having a tangible of the relief map would make understanding of the route much easier. At the same time, hikers are interested in sight-seeing and places to rest along the way (e.g. restaurants or viewpoints). Using FlowPut, additional information is visualized on and around the tangible relief map.

In summary, we contribute in this paper:

- A point-cloud based object recognition and tracking for 3D objects providing low latency recognition and real-time tracking.
- A point-cloud based touch input detection for 3D objects that provides multi-touch not only on the objects' upside but also on its sides.
- A projection mapping and layout approach allowing environment-aware projected user interfaces by considering optical and physical properties of the projection surface. s

2 RELATED WORK

2.1 Digital Augmentation and Layout

Many efforts have been made to create *augmented paperwork* with digital projection or input, thereby allowing the user to interactively add digital content (e.g. annotations or animations) onto [15, 19, 30, 33, 40, 54] or next to physical documents [26, 31]. Research investigates content-awareness by avoiding projecting onto textual content either automatically [5, 30, 50] or through user intervention [59]. To avoid projection on content, it is common to either analyze the surface texture (e.g. by a RGB camera) to avoid textural interference [30] or to analyze the physical structure of objects [5] to avoid physical interference. Based on these results, it is common to either relocate content to a free area [30] or transform the representation to fit the free space [5] or combinations thereof. These approaches all rely on a binary projection quality assessment, masking-out areas deemed unsuitable and thereby losing the information how unsuitable the areas actually are.

In a similar direction, *occlusion* has been identified as a problem when working with physical objects on interactive tabletops. As a result, user-driven layouts have been explored to circumvent obstacles and occlusion for menus [29]. Also, systems have been developed to automatically avoid occlusion on tabletops for labels [45] or general objects instead of menus [9, 22, 41]. Again, these systems rely on a binary occlusion measure and do not trade off between closeness to the original location and display quality. This can lead to situations in which an object is displayed far away from its intended location as no closer large-enough free-spot is available.

For *tangible user interfaces*, projective augmentation is of particular importance. One common approach is to augment 3D objects by tracking of and projecting onto them using sensors in the environment. Thereby, visual content can be augmented onto a physical environment [18, 56, 57], onto moving objects [21, 51], or next to tangibles [8]. Further work explored the use of the objects' surrounding to change its look [32] or provide additional context [53]. It is noteworthy that in the context of these related works, it is often assumed that the object's surface is uniform and suitable for projection [8, 21, 51] and projection happens directly on or around the tangible in a static manner [8, 53].

In the field of *augmented reality*, the placement and layout of visual overlays is an important issue as well. Grasset et al. [13] propose an image-based approach using an edge detector combined with a saliency map to avoid displaying over important regions. In a similar direction, Orlosky et al. [38] locate dark, billboard-like structures in an image to overlay text together with a hough lines detection to ensure the stability of the projection. Based on predefined or generated 3D models, research has emerged to inform the layout of AR environments. For instance, free and occupied spaces can be modeled by using rectangular areas, which are then used to layout content [2, 3]. Using RGBD-cameras, Ens et al. [7] propose an approach for spatial constancy, using a Kinect fusion [17] based model together with a saliency map. However, both approaches require generating a complete 3D model of the whole scene to identify the locations of the objects in the target workspace area. Gal et al.

Table 1. Properties of a representative selection of digital augmentation systems. ● indicates that a property is supported, ◐ indicates partial support and ○ no support.

	Augmented Paperwork				Occlusion		Tangibles		Augmented Reality			
	[5]	[30]	[33]	[40]	[9]	[45]	[8]	[53]	[3]	[7]	[10]	[13]
Texture Interference Detection	◐	●	○	●	○	○	○	○	●	●	○	●
Physical Interference Detection	●	○	○	○	◐	◐	○	○	○	○	○	○
Relocate Content	○	●	○	●	●	●	○	○	●	●	●	●
Transform Content	●	○	○	○	○	○	○	○	○	○	●	○

proposed a constraint-based optimization approach for AR layout generation [10] that considers the position and scaling. However, they do not consider interferences with physical objects/surfaces and therefore do not take into account the quality of the projection or overlay. While this is sufficient in an ideal AR-world with opaque overlays, current implementations of AR glasses or projection based AR systems are semi-transparent and hence interference should be accounted for. Setting aside the technological possibilities of fully opaque overlays, there are scenarios in which transparency is even desired. For instance, when using AR to guide a worker during a machine repair, it could be fatal if opaque overlays covered dangerous machine parts or limit the worker's free field of sight.

In summary, while suitable for their respective target environments, the existing approaches have drawbacks when projecting on physical objects. To avoid interference, they often search for uniform surfaces in terms of color (e.g. low saliency, no edges or corners, etc.), neglecting their physical structure. For projection, physical edges, optically not visible in the camera image, can cause problems. Also, dark surfaces are often not well-suited for projection while being uniform. Moreover, when adjusting the presentation of the content to adapt to the environment, existing approaches often either relocate or transform content but not both.

In contrast to prior work (see Table 1 for a comparison), FlowPut expands on this and takes into account physical features like surface smoothness or edges as well as the surface lightness to assess projection quality by employing a continuous projectability measure. For layouting, FlowPut combines relocation with the transformation of UI elements to provide automatically optimized layouts.

2.2 Touch Input on Surfaces

Touch input has become a popular input modality, often used for tangible user interfaces. There are several different approaches to provide touch input on and around tangible 3D objects.

Touch functionality can be embedded into or attached to tangibles by utilizing *embedded sensing*. This can be done using capacitive [46] or acoustic sensors [37] delivering information on whether an object is grabbed or not. Also, it is possible to integrate touch-sensitive pipes [4, 47, 55] into objects. More advanced but also more complex is the use of capacitive touch sensor foils [11, 20, 35, 36, 48] that allow detecting touch points of individual fingers. However, these approaches often require to manually assemble additional electronics or are only applicable on simple, developable surfaces.

Another stream of research utilizes *external sensors*, mounted in the environment (such as RGBD cameras), to enable touch input on a large variety of surfaces and objects without the need to alter the tangibles or surfaces itself [24, 26, 54, 56, 58–60]. Yet, most of the approaches [24, 58, 60] rely on the concept of background subtraction which is suitable for static scenes and can be extended to dynamic scenes by continuously updating the background model. However, this approach is often limited, for instance when body parts, present during the

Table 2. Properties of a representative selection of touch detection approaches. Embedded approaches require sensors to be attached to or into the object. External approaches use sensors in the object's environment. ● indicates that a property is supported, ◐ indicates partial support and ○ no support.

	Embedded Sensing					External Sensing				
	[35]	[37]	[46]	[47]	[48]	[24]	[26]	[54]	[58]	[59]
Dynamic Scenes	●	●	●	○	●	●	○	●	○	○
Touch on Upside	●	○	○	●	●	●	◐	●	●	●
Touch on Sides	●	○	○	●	●	○	○	○	○	○
Coordinates	2D	-	-	-	2D	2D	2D	2D	2D	2D

update, need to be removed from the background model. Also, touch capabilities are usually limited to surfaces facing towards the camera, neglecting touch on the object's sides.

In contrast to prior work (see Table 2 for a comparison), we contribute an approach based on point-cloud processing instead of a background model being subtracted to detect touch. The touch points are established by detecting intersections of the objects' point cloud with point clouds identified as body parts. The intersection based approach allows us to expand touch to the sides of objects, which is currently only possible with embedded sensing requiring additional electronics within the tangible. Further, FlowPut can deliver 3D coordinates of a touch relative to an object instead of limited 2D surface coordinates.

2.3 Object Recognition and Tracking

Projection mapping requires an accurately calibrated projector and camera system. The camera's extrinsic and intrinsic parameters are usually known or can be calculated, for instance, using Zhang's method [62]. Calibration parameters for the projector are similarly computed by finding the projected 2D positions of known 3D locations in the physical scene [1, 39].

The object tracking method used for projection mapping must be very precise and fast to avoid misalignments between the tracked object and the projected texture. Previous approaches use magnetic sensors [1, 28, 63] or markers [23, 26, 33, 34] to obtain the object's position and rotation. However, these approaches require specialized hardware, additional sensors or markers that need to be attached to the objects.

To overcome these issues, optical methods have been investigated [5, 30, 40, 54] that use image analysis to recognize objects. Yet, these approaches are affected by projection on objects since the appearance is changed. This can be avoided by using infrared cameras [5].

Depth-based tracking methods are not intrusive and use the object's 3D model and a depth camera [7, 25, 27, 44, 51, 52, 63]. Since depth cameras often use the infrared spectrum, they are immune to interferences by visual projection. The initial pose estimate is often acquired by matching Fast point feature histograms (FPFH) [25, 43, 52], while the pose updates in the tracking phase are computed using the Iterative closest point algorithm (ICP) only [43, 52] or in combination with a particle filter [25, 27]. However, depth-based methods are computationally very expensive and the camera and projector also contribute to an additional delay, which has to be compensated by predicting the object's motion [1, 27, 63].

In contrast to prior work (see Table 3 for a comparison), we contribute a method to detect and track objects using depth measurements from a consumer-grade depth camera that provides real-time performance. Additionally, we apply an inlier-based method of the ICP algorithm, which allows tracking partly occluded objects (e.g. by the user's hands).

Table 3. Properties of the most related object recognition and tracking systems. For object tracking, ● indicates full tracking and ◐ recognition only. For object recognition, ● indicates a specific object, ◐ only the object's category are identified; ○means only presence of objects is detected.

	Marker				Optical				Depth		P. Cloud	
	[1]	[23]	[26]	[33]	[5]	[30]	[40]	[54]	[7]	[51]	[43]	[52]
Object Recognition	●	●	●	●	○	○	●	●	○	○	◐	◐
Object Tracking	●	●	●	●	◐	◐	●	●	◐	●	◐	●
Immune to Texture Changes	●	●	●	●	●	○	○	○	●	●	●	●
Unmodified Objects	○	○	○	○	●	●	●	●	●	●	●	●

3 FLOWPUT OVERVIEW

FlowPut tracks 3D objects, provides touch input on and around them, and projects visual output onto those objects and their surroundings. The projection additionally takes into account the surface structure of the object and the environment to avoid projection on unsuitable surfaces (e.g. strongly textured or dark). This functionality is conveniently provided as an API, thereby making it easy to develop full applications requiring interactive tangibles or quickly prototype interactive systems.

To this end, FlowPut requires a hardware-setup providing a projector, a depth camera and optionally an IR camera. The IR camera allows keeping track of the environment without interference from the projected content. An overview of the setup is shown in Figure 2. We envision this projection-camera system to be combined into a single piece of furniture so that it can be part of any room's standard infrastructure in the form of a non-obtrusive conventionally looking lamp. We identified the following requirements to ensure a broad applicability of FlowPut:

(1) *Real-time performance*
The object recognition, tracking and layout generation should be performed in real-time as required for interactive systems.

(2) *Stable and occlusion resilient object recognition and tracking*
Predefined objects that are recognized once should be tracked in a stable manner to avoid flickering and jumping of visualizations. The framework should be able to track multiple objects simultaneously and tolerate cluttered environments since the target setting are office desks and tables. As the user's hands or other body parts may cover the objects during movement, tracking should be resilient to partially occluded objects.

(3) *High touch accuracy and resolution*
For an interactive system, touch input should be possible. To allow fine-grained interaction on tangibles, a high resolution of the touch detection is required. Further, the number of false and not-recognized touches should be minimal to ensure a positive user experience.

(4) *Occlusion and interference aware UI*
The UI layout of the framework should respect the physical environment of the objects and avoid interference and occlusion by dynamically clipping or relocating the UI elements with respect to a dynamically changing environment.

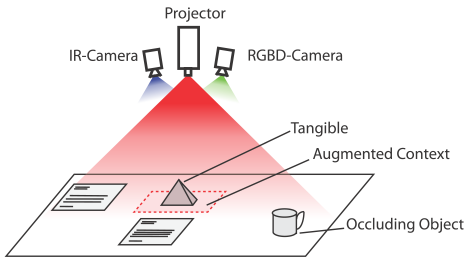

Fig. 2. Overview of the FlowPut environment consisting of a projector, RGBD-camera and IR camera mounted above a conventional table. The framework tracks known tangible objects, detects occluding (or other) objects and projects digital content, e.g. contextual information, on and around the tangibles.

4 OBJECT TRACKING AND TOUCH INPUT DETECTION

FlowPut's object tracking and touch input detection methods only rely on depth data since image-based methods would interfere with the color or texture of the projection. The depth information is processed as a 3D point cloud with each depth pixel resulting in a single point.

4.1 Input and Reference Model Preprocessing

Processing large point clouds is very time consuming as most algorithms (e.g. the iterative closest point algorithm) are at least linearly or quadratically dependent on the number of points. Therefore, reducing the size of the point clouds, while still keeping as much valuable information as possible is one of the key challenges to achieve real-time performance.

4.1.1 Model preprocessing. Since the reference models of the objects to be tracked are static (e.g., data from a 3D-scan), they can be preprocessed before the application startup. During this process, the resolution of the object's point cloud is reduced to 2 mm in order to match the resolution of the depth camera's point cloud data.

4.1.2 Plane removal. For the input point cloud captured by the depth camera we can simply remove all points belonging to the table's surface plane (See Figure 3 left versus center, where the table plane is removed). Those points do not have any value for the following processing steps as the plane can be represented using a simple plane equation ($z = 0$). However, the noise floor of the depth camera causes the table's surface points to be randomly distributed above and below the table's actual surface by up to 1 cm. The majority of these points can be removed by applying a height threshold (e.g. $\text{abs}(z) < 0.25\ cm$). While filtering most of the table's surface, this also loses the lower parts of the objects. However, the remaining object points are sufficient enough for a robust pose estimation. Any further outliers are removed by applying a statistical outlier test, which checks each remaining point if it has at least 5 neighbors within a small neighborhood (0.75 cm), otherwise it is removed as well, since it is most likely a residue from the plane removal process.

4.1.3 Clustering. The filtered point cloud needs to be separated into a single cluster for each individual physical object in the scene in order to decide which processing step is applied next (see Figure 3 center versus right, where the point cloud is split into four separate clusters). For already tracked objects, the object's last position and rotation is known from previous frames. Thus, we can extract appropriate points using the object's dimensions and a small margin to allow for movements in between frames. Those points are subsequently passed to the appropriate tracking method. For the remaining points, we first cluster the filtered point cloud using an Euclidean clustering algorithm with a 5 cm cluster-to-cluster distance. For each resulting cluster, we decide whether it belongs to the user's body or if it is an unknown cluster that could probably represent a object that should be tracked. We consider clusters as body by evaluating if the cluster intersects with the field of view of the camera, because the user's arms always come into the workspace from the outside. Such body clusters are passed to the touch input detection, while all other clusters are passed to the object recognition.

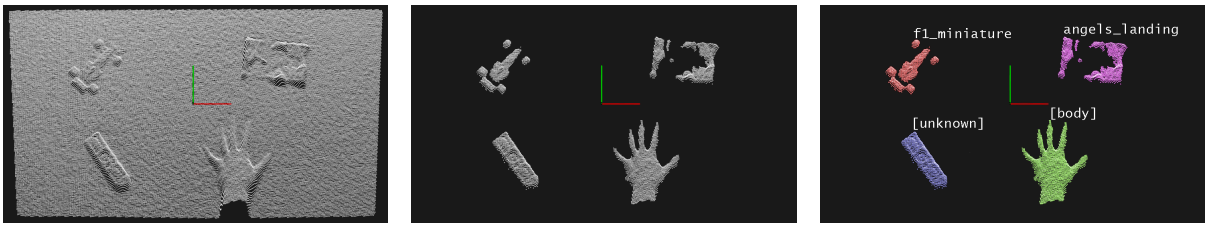

Fig. 3. Processing pipeline of the point cloud data: raw (left), preprocessed (center) and clustered/tracked (right)

4.2 Object Recognition

The object recognition maintains a database of objects to track and their respective point cloud. Its task is then to identify any unknown clusters and estimate the 3D position and rotation of found objects.

In order to compare two point clouds for similarity, they need to be aligned with each other. Previous work [25, 43, 52] use 3D features and matching algorithms to find a good alignment, but these methods are computationally very expensive and did not provide robust and accurate enough results for noisy data. Therefore, we make use of the already isolated clusters and align the unknown clusters with the trackable models 3D point clouds using principal component analysis (PCA). By applying PCA to the clouds, we receive three principal axes in order of their variance. Further, we know which side of the clouds faces up and that the models can be assumed to be laying on a flat surface. Therefore, we project the clouds onto the x-y-plane and only compare the respective largest principal axis of the two clouds to compute a single rotation angle around the z-axis. However, PCA only yields an axis, but no direction. Hence, we further check a second rotation angle, which is just the complement to the first rotation angle (α+180°). The translation between the two clouds is compensated by moving their respective center of mass to the origin.

The initial rotation estimation during object recognition is based on the point cloud's principal axis. For objects with a rectangular bounding box, as used in our applications, this is very easy to compute. For objects with a quadratic bounding box, principal axes may not be distinguishable. Hence, we have to check four possible rotations (instead of two). In case the object has a varying surface structure along one axis, the correct rotation can still be found. However, if the object is symmetric, the rotation is inevitably ambiguous.

After alignment, the quality of the match is evaluated using a fitness score. For each point in the model's cloud, we compute the nearest neighbor in the aligned unknown cluster and accumulate their distances. The average point-to-point distance is then used as the fitness score, which is independent of the model's cloud size. If the fitness score is below a certain threshold (e.g. average point mismatch less than 2 mm), indicating a good alignment, the unknown cluster is identified as a trackable object and the object tracking is initiated using the pose estimate derived from the PCA alignment.

As the fitness score computation is also linearly dependent on the number of points in the model's point cloud, we use a subsampled model point cloud with a cloud density of 1 cm. This results in about 100-200 remaining points depending on the model's size. This reduction strongly decreases the computation time, while still maintaining enough expressiveness to determine the matching quality.

4.3 Object Tracking

Object tracking is performed using the iterative closest point algorithm (ICP) [61] with the last known pose as initialization. ICP minimizes the matching distance of two point clouds by iteratively transforming them based on their respective point to point nearest neighbor matches. However, the ICP algorithm can only cope with small misalignments between both objects clouds or else it might not converge towards the correct transformation. Therefore, we need to minimize the movements between two consecutive runs of the tracking algorithm. This is done by reducing the computation time and therefore being able to process each individual frame. To reduce the computation time we use the subsampled model's point cloud and a filtered and cropped input cloud of the scene.

The ICP algorithm uses a similar score as the previously described fitness score. However, it ignores point-to-point distances larger than 2.5 cm to allow for occlusion, e.g. by the user's hands or fingers. As a consequence, occluded points do not reduce the fitness score, while the remaining points are still expressive enough to determine a good alignment.

The pose returned by the ICP algorithm for each frame can vary slightly, even when the object is not moving. These slight variations also cause the mapped projection to move accordingly, which is very unappealing and disturbing to the user's eye. Therefore, we added a smoothing filter, that averages over the recently computed

poses and thereby achieves a stable pose estimate. In order to still be responsive in case of actual movements and avoid additional latency due to the smoothing operation, we also employ a movement detection which skips the smoothing step if the object is actually moving.

4.4 Touch Input Detection

FlowPut is able to detect touch interaction on the table's plane and on the tracked object's surface. Unlike background-subtraction based approaches, the point-cloud based approach allows detecting touches on the objects' sides as well. To do so, our detection uses the clusters which were classified as body clusters in the preprocessing step. Instead of performing complex finger detection and only detecting touch at the fingertips, we perform touch detection on the whole body cluster and then filter any false touch points depending on their size.

At first, the distance between each point in the body cluster and the table's or object's surface is computed. For the table, this distance is given by using a simple plane equation. For an object, we transform the model's point cloud according to its last known pose and then perform nearest-neighbor searches for each point of the body cluster. Only points within a small distance to the table or object are considered further, as those points indicate the touched areas. Such areas can be caused by one or multiple fingers or even larger parts of the body touching the table or an object. By clustering the touched areas with a 1 cm minimum distance, we obtain a single cluster for each of the touched areas. If a single touched area has a smaller or larger size than a fingertip (e.g. complete hand resting on the table or noise) the touch area is discarded. Considering a resolution of 2 mm to 2.5 mm, the typical fingertip of size 16 mm to 20 mm [6] has a point cloud consisting of 5 to 50 points.

Using this approach for touch detection, each touch point is assigned to a specific body cluster. This enables multi-touch for multiple users that can be easily extended by adding user tracking and identification to allow associating touch points to specific identified users. To offer touch-down, touch-move and touch-up events, the detected touch points are tracked over time. The tracking information is used to smooth the touch event's position using exponential filtering and thereby increase its accuracy. It also improves the touch detection's robustness by filtering single noisy measurements.

Additionally, FlowPut detects if certain predefined areas of a tracked object are touched. These so-called *touch regions* act as a virtual button and can be easily defined using a simple tool and provide more semantical meaning to the applications (e.g. the roof of the house was touched) than just a simple 3D coordinate. The touch regions are internally stored as sub surfaces of the object's point cloud. They are activated if the touch point's closest surface point is within the touch region's surface.

5 ENVIRONMENT-AWARE UI LAYOUT

While people are working, tables are usually cluttered with a wide variety of different objects with different textures (e.g. paper documents, pens, or notebooks). A naive projection system without an awareness of its environment would produce an irritating user interface that projects over such objects in an uninformed manner, resulting in a poor user experience and usability (e.g. digital text projected over a printed text often renders both unreadable).

In order to reflect the presence of different objects and textures, but at the same time provide a sophisticated output on and around a tangible object, layout techniques that consider the environment are required to place UI elements. As illustrated in Figure 4, we propose two techniques that utilize the object's surrounding for visual output (1) by placing UI elements based on environmental constraints, and (2) by projecting as much visual content in the surrounding of the tangible object as possible without jeopardizing projection quality.

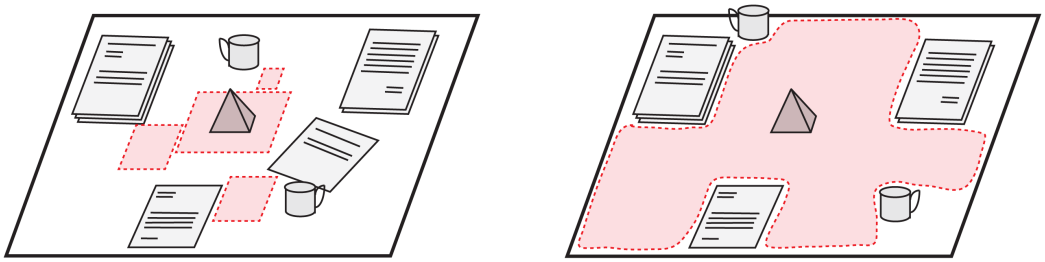

Fig. 4. Layout for UI elements based on environmental constraints (left) and maximum space layout (right). Projection is shown in red.

5.1 Projectability

The environment-aware UI layout techniques described in this paper rely on a per-pixel projectability measure, which we proposed earlier [42]. The projectability measure assesses the suitability of the area covered by the projector regarding projection quality. Instead of computing a binary measure (i.e. projectable/not projectable), we employ a continuous projectability in a range from 0 (very suitable) to 255 (unsuitable).

In general, areas suitable for projection are of light color and ideally smooth without textures or physical surfaces that often vary in depth. These factors are considered and can be flexibly weighted when the projectability map P is calculated:

$$P = w_{\text{smooth}}\, C + w_{\text{light}}\, L + w_{\text{depth}}\, D + w_{\text{user}}\, U$$

The user can specify the individual weights to customize the projection assessment. The maps are computed as follows.

5.1.1 Color Smoothness. To locate obstacles (e.g. printed text or figures) an edge detector is used in order to capture textured areas. This includes text, lines, circles, and other shapes as well as more gradual changes in color of the surface (e.g. the wood-texture of a table) and is encoded in a *color edge map* C. We use a gradient magnitude based approach that allows for a continuous response for edges depending on the difference in color. The gradient is derived using the Sobel operator in x and y direction. The magnitude is defined as the length of the resulting vector.

5.1.2 Lightness. Even though smoothness is able to capture a wide range of possible obstacles, there are (very) dark, yet smooth surfaces that do not contain any edges. Therefore, we consider the surface lightness in order to find suitable spaces in form of a *lightness map* L. It is defined as the inverse of the normalized, thresholded, and grayscaled color image.

5.1.3 Physical Surface. Additionally, the physical surface is factored based on the depth information captured using a depth camera, because users have a different viewpoint than the camera. Hence, the surface might seem flat and continuous (i.e. no visible edges) from the camera's perspective, but not continuous from the user's perspective (e.g. due to tall objects). Analog to smoothness, a gradient-magnitude based approach is applied to the depth image to calculate the *depth edge map* D.

5.1.4 User Mask. Besides projection quality aspects, an application or a user might have additional personalized constraints (e.g. masking a certain area to always avoid projection). These can be taken into account through a *user mask* U which allows specifying biases for certain areas.

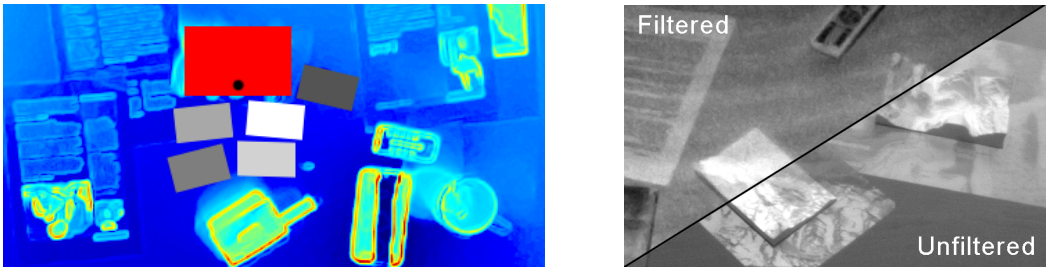

Fig. 5. Visualization of the element [gray] placement on top of a projectability map [blue=good score, red=bad]. The elements target position is the black dot on the physical object [red box] (left). Image captured by the IR camera with and without IR filter. The projected content is invisible in the filtered image (right).

5.1.5 Interference. Obviously, the (visual) projectability measure would be influenced by the projection. For example, when considering a completely white, even surface, the projectability would allow projection anywhere. However, if the framework decides to project at a place P, the next projectability calculation would suggest avoiding projection at P as the camera picks up the projected image. The projection would then be moved to another place and the process would start over. To avoid this, we use an offline computation approach, that computes the projectability of, for instance, an object or the environment before the projection starts and caches the map for future use (e.g. projection of text onto objects). This approach is very suitable for environments where only tracked objects are moved and the textures remain static.

Since we target dynamic scenes as well, this would also be a severe limitation. Therefore, we extend the caching approach as follows: Since the framework knows where it projects, the cached projectability map can be updated regularly in areas where no projection occurs, filling the projection areas with the cached information. To account for texture changes in areas with projection, we use an additional IR-camera equipped with a low-pass filter to filter out visible light, i.e. the projection (see Figure 5 right). In this way, the areas with projection are monitored, and if significant changes in (IR) texture are detected, the projection can be very shortly disabled, the map updated in this area, and then, based on the updated map, re-enabled and, if necessary, moved to another place.

Using this approach, the projection needs to be disabled to update the map. To not completely disable the projection one could synchronize the camera with the projector and generate a short gap within one of the colors-phases of the projector (in case of a DLP projector) as done in [19]. However, this is rather complex and requires modifying the projector. Therefore, we propose another approach: Projectors with laser light sources, that are becoming more and more popular, use three (R,G,B) very specific wavelengths. These wavelengths could be filtered for the camera by using narrow-band notch filters, so that the camera can constantly monitor the area in the visible spectrum but does not pick up the projection at all.

5.2 Placement Based on Environmental Constraints

For the placement of additional related information, the layout process should consider the projection quality of the target surface and a set of other constraints as well. For instance, labels should be as close as possible to the object they label and be oriented towards the users for improving readability while being displayed on a surface with good projection quality. Since these constraints often contradict each other (e.g. a big projectable space far away vs. a less projectable small space nearby), we propose to use an optimization-based approach to generate a UI layout, that computes an optimal place for all UI elements based on a set of environmental constraints (see Figure 4 left and Figure 5 left).

The elements to be placed are characterized by their 2D position, size (width and height), and orientation: $v = (x, y, s, \alpha)$. Width and height are represented by a scaling factor s to maintain the aspect ratio of the element. We formalize the optimization as a minimization problem, since our goal is to minimize the total costs C_{Total} for the placement of all elements $\mathcal{E}$ with each element $e \in \mathcal{E}$ having a set of costs $\mathcal{C}_e$ where each cost $c(v)$ is weighted with an individual factor $w_{e,c}$:

$$\min \; C_{\text{Total}}(v) = \sum_{e \in \mathcal{E}} \sum_{c \in \mathcal{C}_e} w_{e,c} c(v)$$

This formulation allows for an individually weighted set of constraints for each element based on their individual properties (e.g. readability of a text element is more crucial compared to an image element that tolerates more interference). For single elements, it is sufficient to optimize v by only considering the projection quality and the element-local constraints (e.g. minimum size). In case of multiple elements, it is also important to ensure that the elements do not overlap each other. This is fed into the optimizer as an additional constraint. To avoid unwanted flickering, we added a motion filter which only repositions the elements if their new location largely differs from the last position.

5.2.1 Constraints. The optimization currently considers the following constraints but can be easily extended with additional ones.

Projection quality: To ensure a good projection quality, the table's surface is assessed by the framework based on the projectability map P, containing a continuous projection quality value for each pixel. The projection quality for an element is then computed by averaging the individual pixel values $P_{x,y}$ from the map across the area of the element e given its current location, rotation, and scale:

$$c_P = \frac{1}{width \cdot height} \sum_{(x,y) \in e} P_{x,y}$$

Proximity to the desired location: The position cost is modeled as the Euclidean distance between the desired location p_d and the actual location $p = (x, y)$ of the element:

$$c_{pos} = \sqrt{(p_d - p) \cdot (p_d - p)^T}$$

Orientation towards the user: Since the projected content is ideally oriented towards the user who requested the element to be displayed, the angle cost is modeled as the absolute difference between the angle making it face towards the user α_d and the angle α of the element itself:

$$c_{angle} = \text{abs}(\alpha_d - \alpha)$$

Besides the cost for deviations from α_d, a hard constraint on the maximum rotation is enforced (i.e. $\alpha_{\min} \leq \alpha \leq \alpha_{\max}$). This is necessary to avoid rotating elements by 180 degrees and to ensure readability of text elements since too much rotation often renders text unreadable.

Scale: Depending on the content, an element may be scaled to fit into a projectable area. This can be achieved using a scale constraint which is modeled as the absolute difference between the original scale of the element s_d and the actual scale s of the element:

$$c_{scale} = \text{abs}(s_d - s)$$

As for the angle, scaling is constraint on the minimum and maximum scale (i.e. $s_{\min} \leq s \leq s_{\max}$). In most cases, $s_{\max}$ will be equal to s_d with both being one (the element should be displayed at its intended size and not larger).

5.2.2 Normalization. The constraints are not consistent in their value range, i.e. the scaling factor is a value between 0 and 1, the orientation angle falls in a range from 0 to 2π and the projection quality is in a range from 0 to 255. If the cost values would be directly summed up, the projection quality, for instance, would have a much higher impact than a deviation in scale or rotation and therefore would always outweigh them. To consider the different ranges and allow to weight the different metrics for any element individually, we normalize the cost values c to the interval $[0, 1]$ and then weight them using element specific factors. If necessary, this can be extended using more complex (e.g. non-linear) normalization functions. Initially, all weights are 1, so that every factor is weighted identically. However, users may change the weights if desired.

5.3 Maximum Space Layout

Besides the need to place information related to the tangible in its surrounding, there are situations in which the tangible needs to be viewed in its context. For instance, in case of a tangible relief map, the framework could not only texture the 3D tangible itself but also display the geographic map information on the table around the tangible (see Figure 1 right). However, such context information is fixed in its position relative to the tangible and thus cannot be placed with our optimization technique. Also, the information displayed should be as large as possible. While one can display this in a naive way by just projecting everywhere within the projectors reach, this is problematic on cluttered tables, because the projection heavily interferes with other physical objects on the table. This has been shown to be very undesirable and distracting for users [40].

Therefore, we propose to use the projection quality to layout the element around obstacles using a flood fill approach. The area around the tangible is filled as long as the projectability score is below a predefined threshold. In order to close tiny holes and smoothing the contour, two iterations of dilate/erode are performed. To improve the stability of the detected occlusion regions at the border areas and prevent flickering of the contour, we perform additional smoothing and filtering operations. The UI element is then clipped according to a binary map denoting the connected projectable area around the tangible. To visually enhance the contour, we approximate a low complexity polygon of the contour and transform the linear line segments into a continuous Bézier curve.

6 APPLICABILITY

We consider FlowPut as a unique solution for turning tangibles into interactive applications, combining the tangible with touch-enabled user interfaces (i.e. both content and controls) and leveraging the dynamically changing display space surrounding it in an environment-aware manner. We consider two categories of applications as particularly well suited: (1) interactive tangible 3D displays and (2) interactive tangible 3D proxies. We will explain the value of FlowPut for these two categories by means of two interactive sample applications, namely (1) a tangible relief map and (2) a tangible car proxy (see Figure 6).

6.1 Interactive Tangible 3D Displays

Projecting information on everyday surfaces in combination with touch detection allows for many unobtrusive interfaces (cf. [14]). FlowPut adds to this vision by providing the means to not only make flat 2D surfaces interactive but physical 3D tangibles as well. To illustrate this category, we created an interactive map relief as an example application that leverages the physical 3D geographical shape of a 3D-printed object to improve the spatial understanding of a geographical terrain (see Figure 6 left). By augmenting the tangible with a projected map control, users can search for nearby images by touching a point on the tangible map. The map's projection is maximized using the maximum space layout and images are optimized to optimally leverage the available display space surrounding the tangible (see Figure 1 left).

After registering the tangible's 3D model to FlowPut, any existing UI content or control can be easily bound to its position and orientation. Moreover, FlowPut automatically clips the UI elements according to the maximum

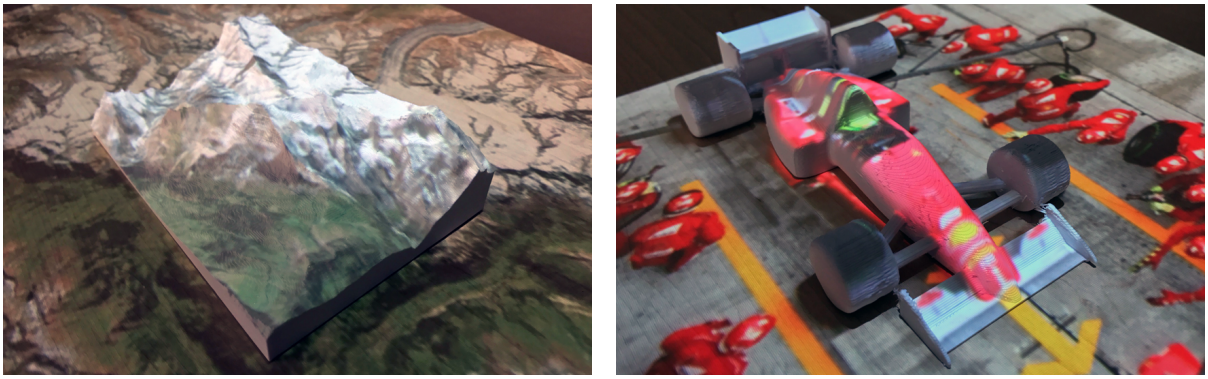

Fig. 6. Two interactive example applications: a tangible relief map (left) used as a tangible 3D display, and a tangible car (right) used as a 3D tangible proxy.

space layout. Detected touches on the tangible and table are automatically redirected onto the respective UI element. In our sample, touches on the tangible are used to retrieve images for the respective geographic location and register them to FlowPut's optimization procedure. The position and rotation of each image are continuously and automatically adjusted to the most adequate space on the table.

In summary, an existing standard map application could be easily transferred to FlowPut by interchanging the layout-controlling components (e.g. grids) with FlowPut.

6.2 Interactive Tangible 3D Proxies

Today, 3D objects are often customized or configured on a computer, for instance using CAD software (e.g. for 3D printing) or product configurators (e.g. for customized cars or furniture). For expert users, these systems are easy to use after a steep learning curve. However, for novice users, FlowPut is well suited to be used to easily explore, adjust or configure such 3D objects by directly interacting with a corresponding tangible proxy. For instance, users may adjust the object's surface properties (e.g. add color or texture directly on the proxy's surface) or perform customizations (e.g. enabling or disabling parts for a 3D print). These changes are instantly visualized on the tangible proxy and can be fed into the configuration software in parallel. Thereby, a more direct and haptic interactive experience can be provided. In addition, FlowPut is not limited to one tangible proxy: FlowPut works with multiple proxies in parallel, thereby supporting scenarios like areal planning as well (e.g. placement of machines in a factory or furniture in a living room). Here, the ability to interact with the individual proxies (e.g. configuring a machine type) as well as with the ensemble (e.g. placing the machines in the hall) is convenient.

To illustrate this category, we implemented an interactive tangible car proxy as an example application (see Figure 6). Using FlowPut, users can specify a car's colors and surface materials, or control the car's interiors via touching the specific part of the tangible proxy. The updated specifications are then displayed in-place, enabling a more direct manipulation than conventional touch- or mouse-based interfaces. Furthermore, by touching a part of the car, additional information is shown at the most suitable place on the table using the constraint-based layout algorithm (see Figure 1 right).

New proxies can be added by registering their 3D models to the framework and binding a dedicated UI element to their position. The named touch regions facilitate binding the new model to the UI elements by naming the corresponding touch regions in the same way (e.g. every car has tires that should invoke the "select tire" dialog).

7 IMPLEMENTATION

Our lab implementation is based on a standard full HD projector (1080p) mounted 2.35 m above the table together with a Kinect V2 depth and color camera for projectability assessment and an additional IR camera (uEye UI-548xSE-M with an IR low-pass filter) to detect texture changes. The cameras are mounted about 95 cm above the table (see Figure 7). Our demo objects are 3D-printed with a standard BCN3D Sigma 3D printer and regular white Verbatim PLA filament. The respective 3D models are preprocessed using a Meshlab filter script, which normalizes the triangle mesh by applying a uniform mesh resampling, re-computing the vertex normals and exports the simplified mesh into a .PLY-format.

Fig. 7. Our lab setup

The FlowPut framework is written in C++ using Qt. Object recognition, object tracking and touch detection process the Kinect's depth data in a point cloud data structure. Therefore, we use the plane removal, noise removal, Euclidean clustering, nearest-neighbor-searches (k-d tree), iterative closest point algorithm and principal component analysis methods provided by the point cloud library (PCL). The calibration from Kinect to projector, computation of the projectability maps and computing the maximum space layouts uses OpenCV. To optimize the 2D UI elements placements according to the projectability maps, we use the non-linear optimization library NLopt and the gradient-free local optimization using the principal-axis method. In our current setup, the user position is fixed to one side of the table for the rotation of the elements. The software implementation, however, allows to define different target orientations for each element to reflect multiple user positions, e.g. provided by a user tracking system. Communication with other applications is based on Google's Protocol Buffers for serialization and TCP as the network protocol.

Applications for FlowPut can be written in any language or run on separate machines, as the FlowPut API is accessible via ProtoBuf messages sent over TCP. Thereby, it is possible to use any desired UI toolkit. For instance, our example applications are built in C# using WPF.

8 EVALUATION

To assess the applicability and ensure to cover a broad range of scenarios, we evaluate and discuss each of the four previously defined requirements in the following.

8.1 Real-time Performance

The user experience heavily relies on the application being responsive enough to directly detect and respond to any user input (e.g. touch, object or occluder movement). Therefore, real-time performance is a key element for the FlowPut framework. Each frame has to be completely processed before the next frame arrives (maximal 33 ms at 30 frames per second).

We measured the computation times for the individual processing steps during a normal use case scenario with up to three tracked objects and three unknown objects at a time, touch interaction and up to 12 individual free flowing UI elements. The measurements were conducted on a desktop computer running Windows 10 with 8 GiB RAM and a 3.6 GHz Intel i5 Processor.

The preprocessing takes about 15.6ms (σ = 6.1 ms), which is mainly caused by the expensive noise filtering and clustering process. After the initial preprocessing, all of the following processing steps can run in parallel and therefore their times do not add up to each other. Object detection requires 1.2 ms (σ = 0.4 ms) and object tracking 4.3ms (σ = 1.9 ms), which is however influenced by the model's size. Touch detection and tracking are only evaluated if the user's hands are present in the scene and take 0.3 ms (σ = 0.6 ms).

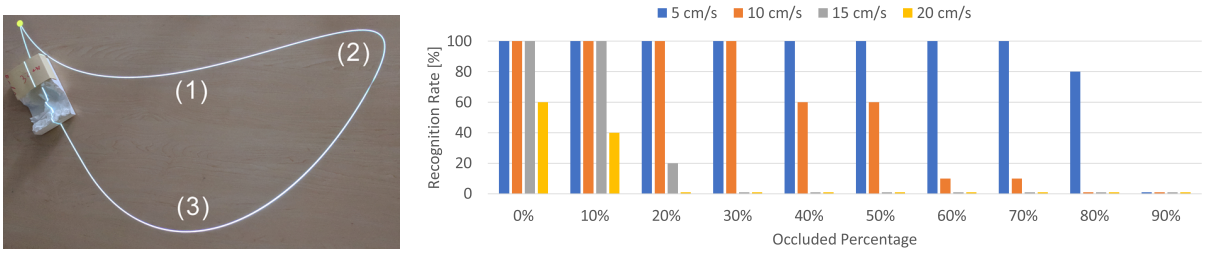

Fig. 8. General setup with 40% occluded object (left) and results (right) of the evaluation of the performance of our tracking approach

Optimizing the UI elements arrangement and computing the maximum space layout is not performed on a frame-by-frame basis and hence does not inflict with the critical path. Instead, they are constantly running and updating in the background with the currently available object positions and projectability maps. The time taken for the layout optimization is mainly dependent on the number of elements. For eight UI elements, the optimization takes 12 ms (σ = 5.4 ms). Computing the maximum space layout for two objects takes 8.4 ms (σ = 4.5 ms).

This leads to a maximum processing time of 19.9 ms (σ = 6.4 ms) per frame, which is below the 33 ms limit required to process all frames. However, the Kinect and projector add about 155 ms of latency, largely outnumbering the framework's latency and leading to a visible delay when moving objects at high speeds.

8.2 Stable and Occlusion Resilient Recognition and Tracking

For our example applications, we utilize models between 15.8 cm × 10.0 cm × 4.8 cm and 14.3 cm × 9 cm × 3 cm in size. However, our tests revealed that models down to 5 cm × 5 cm × 2 cm can be detected. The minimum height of 2 cm is due to the fact that most of the object's point's need to be above the table surface with respect to the Kinect's noise floor of ±5 mm.

To evaluate the tracking's resilience to occlusion, we manually occluded an object with cardboard after the initial recognition. We tested occlusions of 0% to 90% relative to the object's surface in 10% steps. We recruited five participants, all male and aged between 29 and 35. While partly occluded, each object was moved by the participant along a predefined path with a length of 215 cm across the table. Since the tracking is also dependent on the object's speed, participants moved the object at five different speeds (5 to 25 cm/s in 5 cm/s steps). To ensure constant and correct speed, participants had to move the object along a yellow dot, digitally moved along the path (see Figure 8 left for the setup and the path). The path was chosen to contain a nearly straight section (1), a 180-degree turn (2), and a slightly sloped section (3). The object was moved along the path once by each participant for each combination of occlusion and speed. This leads to a total of five recordings for each combination. We counted the number of successful trials. A trial was successful if the tracking was never lost during the trial and the detected pose did not differ more than 3 mm and 5 degrees along any axis from the predefined path.

The results are shown in Figure 8 right. As one would expect, the tracking becomes more unstable with increasing occlusion as well as with increasing speed. At 25 cm/s, tracking was always lost while moving the object even without occluding the object. With 90% occlusion, tracking was lost either directly upon occlusion or after a few centimeters of moving. The loss at 90% (or 15.8 cm² of visible area) occlusion is consistent with the object's minimum size of 25 cm² (5 cm × 5 cm). At 5cm/s, the slowest speed, tracking was only lost once at 80% occlusion. For a more practical speed of 10 cm/s, tracking is stable for up to 50% occlusion. At even higher speeds

the success rate drops further: At 15 cm/s we observed 100% success rate up to 10% occlusion and only 20% at 20% occlusion. At 20 cm/s, we observed 60% success without occlusion and 40% with 10% occlusion.

In practice, this limitation does not pose a severe restriction because the occlusion of an object is usually caused by the user's hands while moving the tangible, which is only temporary. The tracking is immediately restored after the user removes their hand, leading to only short outages of the projection during fast movements. This is especially prominent with small objects where the hand can occlude nearly the whole surface while the object is grabbed. However, this is a general problem of all vision-based approaches since a marker or the object itself needs to be visible for the camera.

In 76.47% of the cases, tracking was lost at the narrow curve during the 180-degree turn (see (2) in Figure 8 left), i.e. when a fast rotation combines with linear translation. Therefore, the results could further improve for non-rotating move-only interactions. In general, we noted that the tracking's occlusion performance is also related to the surface geometry of the object: areas without distinguishing surface features are not as important for the tracking as highly structured areas.

8.3 Touch Accuracy

In order to evaluate the touch detection accuracy, we performed a small-scale evaluation with the same five participants as for the occlusion study. To that end, we selected five different representative surface geometries, namely two vertical planes to evaluate touch on object sides (one parallel to the camera's field of view, one perpendicular), a convex spot standing out of a surface, a concave area and finally, as a reference, a spot on the table's flat surface.

Table 4. Standard deviations of the reported touch coordinates for different surface geometry in mm.

	Touch Plane	σ_x	σ_y	σ_z
Side (Frontal)	x-z	2.95	0.01	4.73
Side (Side)	y-z	0.21	2.83	4.82
Convex	x-y	2.13	1.77	1.39
Concave	x-y	5.25	2.08	1.63
Flat	x-y	2.54	2.08	0.00

The spots were marked on the model and touched repeatedly, each one 3 times by each participant, leading to a total of 15 samples per point. A projected blue dot indicated a successful (i.e. detected by the framework) touch on the target area. Besides two failed touches inside the concave area, all others were successful. The touch coordinates reported by FlowPut were recorded. We computed the standard deviation as a measure of repetition accuracy from the values. The results are shown in Table 4. The table's surface is in the x-y plane, the z-axis is perpendicular to the table plane in upward direction.

Touch on concave surfaces yields the worst results due to the ambiguity of the intersection, as there are multiple possible touch points from the camera perspective, from which the center one is picked. However, a standard deviation of 5.25 mm along the x-axis is still within the size of a fingertip (cf. [6]).

For flat and convex surfaces, the results improved further, with only about 2 mm standard deviation on the x- and y-axis and about 1.5 mm on the z-axis (see Table 4 for the exact values). Given the fact that the camera observes an area of about 1 m in width and 80 cm in height at a resolution of 512×424 pixels, 2 mm standard deviation is equivalent to 1 pixel in the raw depth image, which is an acceptable performance.

For the touch recognition on the side surfaces, there is no huge difference between the two different orientations of the sides. In both cases, the axis perpendicular to the orientation of the side (e.g. the y-axis for the frontal side) has a very low standard deviation since its value is directly derived from the model's tracking data. For the variable axes, we observed a slightly higher variance for the x- (or y respectively) axis than with a flat or convex area. The z-axis has the worst deviation but is still within the typical size of a fingertip.

8.4 Occlusion and interference aware UI

By utilizing projectability maps for our UI layout optimization, FlowPut can adapt the UI to the physical environment. Since this as an automated process which constantly adjusts the UI layout, we investigate how users perceive such a dynamic UI compared to a static one.

We conducted a user study to assess the impact of an optimization-based layout approach on user experience. To that end, we compared two conditions (static and dynamic) for photo browsing using our tangible relief map. In both conditions, participants were asked to find a specific photo from a set of 10 photos bound to landmarks on the map, which were marked with a red dot. By touching the red dot on the map, participants could toggle the visibility of a landmark's photo. In the static condition, we displayed photos in a standard list of photos right to the map. In the dynamic condition, we used FlowPut to automatically layout each photo at a spot close to the landmark. In both cases, the photos were connected to the landmark using a red line. To provide a more realistic environment, we also added common objects to the scene (keyboard, pen, another tangible and a coffee mug).

There were 6 participants (all male, aged between 27 and 34). We chose a within-subject design. For each task, participants were given time to familiarize themselves with the system. Each session lasted about 20 minutes and participants were asked to think-aloud. We conducted a semi-structured interview after condition to collect additional feedback specifically regarding the layout aspects.

In the static condition, all participants complained that after placing the tangible relief map, the projection collided with other physical objects. All but one participant started with moving the physical objects away to have a free working area. While P6 found the presentation with the list structured (since the photos were always to be found at the same place), P2, P3 and P5 complained that the relation between landmarks and photos is often unclear despite the connecting lines (P5: "Lines intersect other landmarks", P3: "The lines were sometimes crossing"). P4 noted that "the many photos are overextending".

In contrast, in the dynamic condition, we observed that all participants opened more photos at the same time and were amazed by the automatic layout. They found it more well-arranged and tidier (P3: "It's good that the photos are always close to their location on the map", P6: "It's nice that the photos automatically avoid obstacles", P5: "Following the lines is much easier now"). However, P1, P2, and P4 also criticized that there was no way to intervene, e.g. by pinning an object to its current position or manually moving it away (P4: "It would be interesting to manually move the photo as well"). Furthermore, participants were occasionally irritated by photos moving due to camera noise. To improve this, the position changes could be filtered even more to ignore movements below a certain threshold, hence avoiding "shaking" (P2) of the projected content.

9 LIMITATIONS

This paper presents FlowPut, an environment-aware touch-enabled projection framework. However, FlowPut has limitations that must be considered before usage.

9.1 Minimum Object Size

Currently, tangibles should be at least 2 cm high above the table to track them correctly. The same holds for the overall size of a tangible that needs to be at least 5 cm by 5 cm. As FlowPut is mainly targeted at tangibles with sufficient space for top-projection, this restriction is usually not an issue. However, there are cases in which users might want to track multiple smaller tangibles (e.g. playing pieces for an augmented board-game). The limitations of the object minimum height and size are not inherent to FlowPut's algorithms, but the result of the depth camera's noise (e.g. for our setup, camera noise is about ±5 mm at a distance of 1 m) and overall resolution. Additionally, the flying-pixel effect is further affecting the precision of the measurements [12]. Future depth cameras are likely to mitigate this issue due to reduced noise and increased resolution.

9.2 Touch Point Accuracy

The touch point's location accuracy depends on the object's surface geometry. Touch points on convex regions of the surface (e.g. the top of a mountain) result in more accurate location estimates compared to concave or flat regions (e.g. a valley in a tangible relief map). As we select the closest surface point to the fingertip as the touch location, there is usually a unique point for convex regions. However, in case of concave regions, there are often multiple closest points and hence multiple possible touch locations. While we use the center of the candidate cluster, a lower noise and higher resolution depth camera would increase performance. Future work could also explore more advanced algorithms to disambiguate touch points (e.g. by taking the user's viewing perspective into account).

9.3 Body Part Classification

To determine body clusters within the point cloud data, FlowPut uses a simple heuristic that defines a cluster as belonging to a body if it hovers above the table surface, starts at the edge of the camera field of view and had not previously been classified as another object. As a result, a new object moved into the tracking area is classified as body part until it is placed on the surface and not touched anymore. Thus, it is not possible to directly interact with a new object without moving the hand away after placing it. Future work should investigate methods to separate the user's hand from the object while being held to mitigate this issue. This would also allow the user to interact with the object while being held in the air above the table.

9.4 Automated Element Placement

Despite the projectability map used by FlowPut as a basis for layout generation is adjustable via a user mask, there is currently no way for users to influence the placement of elements more fine-grained. Based on the user feedback in the evaluation, future work should investigate ways for users to actively specify the desired placement in addition to the location computed by FlowPut. For instance, users may move or fixate an element's location with a three finger gesture.

10 SUMMARY

We presented FlowPut, a framework providing interactivity for tangible objects by visually augmenting them and their surroundings using environment-aware projection and by providing multi-ouch input across the tangibles' and the surrounding table's surface. FlowPut aims at casual use where the tangible objects are used together with office equipment and documents on the same surface. Therefore it provides environment-aware layout mechanisms to minimize interference between the projected digital content and the physical environment. We contributed a real-time detection and tracking approach for partly-occluded 3D objects based on their 3D model, a multi-touch detection for object's sides, as well as a set of environment-aware UI layout techniques. We validated our contributions through user studies and discussed FlowPut's applicability as well as its limitations.

ACKNOWLEDGMENTS

This work was funded by the German Federal Ministry of Education and Research (BMBF) [01IS12054, 01IS17050].

REFERENCES

[1] Deepak Bandyopadhyay, Ramesh Raskar, and Henry Fuchs. 2001. Dynamic Shader Lamps: Painting on Movable Objects. In *Proceedings of the IEEE and ACM International Symposium on Augmented Reality (ISAR'01) (ISAR '01)*. IEEE Computer Society, Washington, DC, USA, 207–. http://dl.acm.org/citation.cfm?id=582828.881324

[2] Blaine Bell, Steven Feiner, and Tobias Höllerer. 2001. View Management for Virtual and Augmented Reality. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology (UIST '01)*. ACM, New York, NY, USA, 101–110. https://doi.org/10.1145/502348.502363

[3] Blaine A. Bell and Steven K. Feiner. 2000. Dynamic Space Management for User Interfaces. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology (UIST '00)*. ACM, New York, NY, USA, 239–248. https://doi.org/10.1145/354401.354790

[4] Eric Brockmeyer, Ivan Poupyrev, and Scott Hudson. 2013. PAPILLON: Designing Curved Display Surfaces with Printed Optics. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology (UIST '13)*. ACM, New York, NY, USA, 457–462. https://doi.org/10.1145/2501988.2502027

[5] Daniel Cotting and Markus Gross. 2006. Interactive environment-aware display bubbles. In *Proceedings of the 19th annual ACM symposium on User interface software and technology (UIST '06)*. ACM, New York, NY, USA, 245–254. https://doi.org/10.1145/1166253.1166291

[6] Kiran Dandekar, Balasundar I Raju, and Mandayam a Srinivasan. 2003. 3-D Finite-Element Models of Human and Monkey Fingertips to Investigate the Mechanics of Tactile Sense. *Journal of Biomechanical Engineering* 125, 5 (2003), 682. https://doi.org/10.1115/1.1613673

[7] Barrett Ens, Eyal Ofek, Neil Bruce, and Pourang Irani. 2015. Spatial Constancy of Surface-Embedded Layouts Across Multiple Environments. In *Proceedings of the 3rd ACM Symposium on Spatial User Interaction (SUI '15)*. ACM, New York, NY, USA, 65–68. https://doi.org/10.1145/2788940.2788954

[8] Markus Funk, Oliver Korn, and Albrecht Schmidt. 2014. An Augmented Workplace for Enabling User-defined Tangibles. In *CHI '14 Extended Abstracts on Human Factors in Computing Systems (CHI EA '14)*. ACM, New York, NY, USA, 1285–1290. https://doi.org/10.1145/2559206.2581142

[9] Genki Furumi, Daisuke Sakamoto, and Takeo Igarashi. 2012. SnapRail: a tabletop user interface widget for addressing occlusion by physical objects. In *Proceedings of the 2012 ACM international conference on Interactive tabletops and surfaces (ITS '12)*. ACM, New York, NY, USA, 193–196. https://doi.org/10.1145/2396636.2396666

[10] R. Gal, L. Shapira, E. Ofek, and P. Kohli. 2014. FLARE: Fast layout for augmented reality applications. In *2014 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. 207–212. https://doi.org/10.1109/ISMAR.2014.6948429

[11] Nan-Wei Gong, Jürgen Steimle, Simon Olberding, Steve Hodges, Nicholas Edward Gillian, Yoshihiro Kawahara, and Joseph A. Paradiso. 2014. PrintSense: A Versatile Sensing Technique to Support Multimodal Flexible Surface Interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 1407–1410. https://doi.org/10.1145/2556288.2557173

[12] J. M. Gottfried, R. Nair, S. Meister, C. S. Garbe, and D. Kondermann. 2014. Time of flight motion compensation revisited. In *2014 IEEE International Conference on Image Processing (ICIP)*. 5861–5865. https://doi.org/10.1109/ICIP.2014.7026184

[13] Raphael Grasset, Tobias Langlotz, Denis Kalkofen, Markus Tatzgern, and Dieter Schmalstieg. 2012. Image-driven View Management for Augmented Reality Browsers. In *Proceedings of the 2012 IEEE International Symposium on Mixed and Augmented Reality (ISMAR) (ISMAR '12)*. IEEE Computer Society, Washington, DC, USA, 177–186. https://doi.org/10.1109/ISMAR.2012.6402555

[14] John Hardy and Jason Alexander. 2012. Toolkit Support for Interactive Projected Displays. In *Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia (MUM '12)*. ACM, New York, NY, USA, Article 42, 10 pages. https://doi.org/10.1145/2406367.2406419

[15] Björn Hartmann, Meredith Ringel Morris, Hrvoje Benko, and Andrew D. Wilson. 2010. Pictionaire: Supporting Collaborative Design Work by Integrating Physical and Digital Artifacts. In *Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work (CSCW '10)*. ACM, New York, NY, USA, 421–424. https://doi.org/10.1145/1718918.1718989

[16] Hiroshi Ishii and Brygg Ullmer. 1997. Tangible Bits: Towards Seamless Interfaces Between People, Bits and Atoms. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI '97)*. ACM, New York, NY, USA, 234–241. https://doi.org/10.1145/258549.258715

[17] Shahram Izadi, Richard A. Newcombe, David Kim, Otmar Hilliges, David Molyneaux, Steve Hodges, Pushmeet Kohli, Jamie Shotton, Andrew J. Davison, and Andrew Fitzgibbon. 2011. KinectFusion: Real-time Dynamic 3D Surface Reconstruction and Interaction. In *ACM SIGGRAPH 2011 Talks (SIGGRAPH '11)*. ACM, New York, NY, USA, Article 23, 1 pages. https://doi.org/10.1145/2037826.2037857

[18] B.R. Jones, R. Sodhi, R.H. Campbell, G. Garnett, and B.P. Bailey. 2010. Build your world and play in it: Interacting with surface particles on complex objects. In *Mixed and Augmented Reality (ISMAR), 2010 9th IEEE International Symposium on*. IEEE Computer Society, Washington, DC, USA, 165–174. https://doi.org/10.1109/ISMAR.2010.5643566

[19] Sasa Junuzovic, Kori Inkpen, Tom Blank, and Anoop Gupta. 2012. IllumiShare: Sharing Any Surface. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 1919–1928. https://doi.org/10.1145/2207676.

2208333

[20] Yoshihiro Kawahara, Steve Hodges, Benjamin S. Cook, Cheng Zhang, and Gregory D. Abowd. 2013. Instant Inkjet Circuits: Lab-based Inkjet Printing to Support Rapid Prototyping of UbiComp Devices. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp '13)*. ACM, New York, NY, USA, 363–372. https://doi.org/10.1145/2493432.2493486

[21] Mohammadreza Khalilbeigi, Roman Lissermann, Wolfgang Kleine, and Jürgen Steimle. 2012. FoldMe: Interacting with Double-sided Foldable Displays. In *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction (TEI '12)*. ACM, New York, NY, USA, 33–40. https://doi.org/10.1145/2148131.2148142

[22] Mohammadreza Khalilbeigi, Jürgen Steimle, Jan Riemann, Niloofar Dezfuli, Max Mühlhäuser, and James D. Hollan. 2013. ObjecTop: occlusion awareness of physical objects on interactive tabletops. In *Proceedings of the 2013 ACM international conference on Interactive tabletops and surfaces (ITS '13)*. ACM, New York, NY, USA, 255–264. https://doi.org/10.1145/2512349.2512806

[23] Han-Jong Kim, Ju-Whan Kim, and Tek-Jin Nam. 2016. miniStudio: Designers' Tool for Prototyping Ubicomp Space with Interactive Miniature. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 213–224. https://doi.org/10.1145/2858036.2858180

[24] Florian Klompmaker, Karsten Nebe, and Alex Fast. 2012. dSensingNI: A Framework for Advanced Tangible Interaction Using a Depth Camera. In *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction (TEI '12)*. ACM, New York, NY, USA, 217–224. https://doi.org/10.1145/2148131.2148179

[25] Daisuke Kobayashi and Naoki Hashimoto. 2014. Spatial Augmented Reality by Using Depth-based Object Tracking. In *ACM SIGGRAPH 2014 Posters (SIGGRAPH '14)*. ACM, New York, NY, USA, Article 33, 1 pages. https://doi.org/10.1145/2614217.2614226

[26] H. Koike, Y. Sato, and Y. Kobayashi. 2001. Integrating paper and digital information on EnhancedDesk: a method for realtime finger tracking on an augmented desk system. *ACM Trans. Comput.-Hum. Interact.* 8, 4 (Dec. 2001), 307–322. https://doi.org/10.1145/504704.504706

[27] Ryo Koizumi, Daisuke Kobayashi, and Naoki Hashimoto. 2015. Acceleration of Dynamic Spatial Augmented Reality System with a Depth Camera. In *Proceedings of the 2015 International Conference on Cyberworlds (CW) (CW '15)*. IEEE Computer Society, Washington, DC, USA, 50–53. https://doi.org/10.1109/CW.2015.42

[28] Daisuke Kondo, Yuichi Shiwaku, and Ryugo Kijima. 2008. Free Form Projection Display and Application. In *Proceedings of the 5th ACM/IEEE International Workshop on Projector Camera Systems (PROCAMS '08)*. ACM, New York, NY, USA, Article 8, 2 pages. https://doi.org/10.1145/1394622.1394633

[29] D. Leithinger and M. Haller. 2007. Improving Menu Interaction for Cluttered Tabletop Setups with User-Drawn Path Menus. In *Horizontal Interactive Human-Computer Systems, 2007. TABLETOP '07. Second Annual IEEE International Workshop on*. IEEE Computer Society, Washington, DC, USA, 121–128. https://doi.org/10.1109/TABLETOP.2007.24

[30] Chunyuan Liao, Hao Tang, Qiong Liu, Patrick Chiu, and Francine Chen. 2010. FACT: fine-grained cross-media interaction with documents via a portable hybrid paper-laptop interface. In *Proceedings of the international conference on Multimedia (MM '10)*. ACM, New York, NY, USA, 361–370. https://doi.org/10.1145/1873951.1874001

[31] Natan Linder and Pattie Maes. 2010. LuminAR: Portable Robotic Augmented Reality Interface Design and Prototype. In *Adjunct Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology (UIST '10)*. ACM, New York, NY, USA, 395–396. https://doi.org/10.1145/1866218.1866237

[32] David Lindlbauer, Jörg Mueller, and Marc Alexa. 2017. Changing the Appearance of Real-World Objects By Modifying Their Surroundings. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 3954–3965. https://doi.org/10.1145/3025453.3025795

[33] H. Mitsuhara, Y. Yano, and T. Moriyama. 2010. Paper-top interface for supporting note-taking and its preliminary experiment. In *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*. IEEE Computer Society, Washington, DC, USA, 3456–3462. https://doi.org/10.1109/ICSMC.2010.5642448

[34] Gaku Narita, Yoshihiro Watanabe, and Masatoshi Ishikawa. 2017. Dynamic Projection Mapping Onto Deforming Non-Rigid Surface Using Deformable Dot Cluster Marker. *IEEE Transactions on Visualization and Computer Graphics* 23, 3 (March 2017), 1235–1248. https://doi.org/10.1109/TVCG.2016.2592910

[35] Simon Olberding, Nan-Wei Gong, John Tiab, Joseph A. Paradiso, and Jürgen Steimle. 2013. A Cuttable Multi-touch Sensor. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology (UIST '13)*. ACM, New York, NY, USA, 245–254. https://doi.org/10.1145/2501988.2502048

[36] Simon Olberding, Michael Wessely, and Jürgen Steimle. 2014. PrintScreen: Fabricating Highly Customizable Thin-film Touch-displays. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology (UIST '14)*. ACM, New York, NY, USA, 281–290. https://doi.org/10.1145/2642918.2647413

[37] Makoto Ono, Buntarou Shizuki, and Jiro Tanaka. 2013. Touch & Activate: Adding Interactivity to Existing Objects Using Active Acoustic Sensing. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology (UIST '13)*. ACM, New York, NY, USA, 31–40. https://doi.org/10.1145/2501988.2501989

[38] Jason Orlosky, Kiyoshi Kiyokawa, and Haruo Takemura. 2013. Dynamic Text Management for See-through Wearable and Heads-up Display Systems. In *Proceedings of the 2013 International Conference on Intelligent User Interfaces (IUI '13)*. ACM, New York, NY, USA, 363–370. https://doi.org/10.1145/2449396.2449443

[39] Ramesh Raskar, Greg Welch, Kok-Lim Low, and Deepak Bandyopadhyay. 2001. Shader Lamps: Animating Real Objects With Image-Based Illumination. In *Rendering Techniques 2001*, StevenJ. Gortler and Karol Myszkowski (Eds.). Springer Vienna, Vienna, Austria, 89–102. https://doi.org/10.1007/978-3-7091-6242-2_9

[40] Jan Riemann, Mohammadreza Khalilbeigi, Niloofar Dezfuli, and Max Mühlhäuser. 2015. StackTop: Hybrid Physical-Digital Stacking on Interactive Tabletops. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA '15)*. ACM, New York, NY, USA, 1127–1132. https://doi.org/10.1145/2702613.2732742

[41] Jan Riemann, Mohammadreza Khalilbeigi, and Max Mühlhäuser. 2015. In-Situ Occlusion Resolution for Hybrid Tabletop Environments. In *Human-Computer Interaction – INTERACT 2015*, Julio Abascal, Simone Barbosa, Mirko Fetter, Tom Gross, Philippe Palanque, and Marco Winckler (Eds.). Lecture Notes in Computer Science, Vol. 9298. Springer International Publishing, Cham, Switzerland, 278–295. https://doi.org/10.1007/978-3-319-22698-9_18

[42] Jan Riemann, Mohammadreza Khalilbeigi, Martin Schmitz, Sebastian Doeweling, Florian Müller, and Max Mühlhäuser. 2016. FreeTop: Finding Free Spots for Projective Augmentation. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA '16)*. ACM, New York, NY, USA, 1598–1606. https://doi.org/10.1145/2851581.2892321

[43] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. 2009. Fast Point Feature Histograms (FPFH) for 3D Registration. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation (ICRA'09)*. IEEE Press, Piscataway, NJ, USA, 1848–1853. http://dl.acm.org/citation.cfm?id=1703435.1703733

[44] M. Sano, K. Matsumoto, B. H. Thomas, and H. Saito. 2015. Rubix: Dynamic Spatial Augmented Reality by Extraction of Plane Regions with a RGB-D Camera. In *2015 IEEE International Symposium on Mixed and Augmented Reality*. 148–151. https://doi.org/10.1109/ISMAR.2015.43

[45] Makoto Sato and Kaori Fujinami. 2014. Nonoverlapped View Management for Augmented Reality by Tabletop Projection. *J. Vis. Lang. Comput.* 25, 6 (Dec. 2014), 891–902. https://doi.org/10.1016/j.jvlc.2014.10.030

[46] Munehiko Sato, Ivan Poupyrev, and Chris Harrison. 2012. Touché: Enhancing Touch Interaction on Humans, Screens, Liquids, and Everyday Objects. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 483–492. https://doi.org/10.1145/2207676.2207743

[47] Valkyrie Savage, Ryan Schmidt, Tovi Grossman, George Fitzmaurice, and Björn Hartmann. 2014. A Series of Tubes: Adding Interactivity to 3D Prints Using Internal Pipes. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology (UIST '14)*. ACM, New York, NY, USA, 3–12. https://doi.org/10.1145/2642918.2647374

[48] Valkyrie Savage, Xiaohan Zhang, and Björn Hartmann. 2012. Midas: Fabricating Custom Capacitive Touch Sensors to Prototype Interactive Objects. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology (UIST '12)*. ACM, New York, NY, USA, 579–588. https://doi.org/10.1145/2380116.2380189

[49] Martin Schmitz, Mohammadreza Khalilbeigi, Matthias Balwierz, Roman Lissermann, Max Mühlhäuser, and Jürgen Steimle. 2015. Capricate: A Fabrication Pipeline to Design and 3D Print Capacitive Touch Sensors for Interactive Objects. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology (UIST '15)*. ACM, New York, NY, USA, 253–258. https://doi.org/10.1145/2807442.2807503

[50] Thitirat Siriborvornratanakul and Masanori Sugimoto. 2008. Clutter-aware Adaptive Projection Inside a Dynamic Environment. In *Proceedings of the 2008 ACM Symposium on Virtual Reality Software and Technology (VRST '08)*. ACM, New York, NY, USA, 241–242. https://doi.org/10.1145/1450579.1450633

[51] Jürgen Steimle, Andreas Jordt, and Pattie Maes. 2013. Flexpad: Highly Flexible Bending Interactions for Projected Handheld Displays. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 237–246. https://doi.org/10.1145/2470654.2470688

[52] Kazuna Tsuboi, Yuji Oyamada, Maki Sugimoto, and Hideo Saito. 2013. 3D Object Surface Tracking Using Partial Shape Templates Trained from a Depth Camera for Spatial Augmented Reality Environments. In *Proceedings of the Fourteenth Australasian User Interface Conference - Volume 139 (AUIC '13)*. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 125–126. http://dl.acm.org/citation.cfm?id=2525493.2525508

[53] John Underkoffler and Hiroshi Ishii. 1999. Urp: A Luminous-tangible Workbench for Urban Planning and Design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '99)*. ACM, New York, NY, USA, 386–393. https://doi.org/10.1145/302979.303114

[54] Pierre Wellner. 1993. Interacting with Paper on the DigitalDesk. *Commun. ACM* 36, 7 (July 1993), 87–96. https://doi.org/10.1145/159544.159630

[55] Karl Willis, Eric Brockmeyer, Scott Hudson, and Ivan Poupyrev. 2012. Printed Optics: 3D Printing of Embedded Optical Elements for Interactive Devices. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology (UIST '12)*. ACM, New York, NY, USA, 589–598. https://doi.org/10.1145/2380116.2380190

[56] Andrew D. Wilson. 2005. PlayAnywhere: A Compact Interactive Tabletop Projection-vision System. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology (UIST '05)*. ACM, New York, NY, USA, 83–92. https://doi.org/10.1145/1095034.1095047

[57] A. D. Wilson. 2007. Depth-Sensing Video Cameras for 3D Tangible Tabletop Interaction. In *Horizontal Interactive Human-Computer Systems, 2007. TABLETOP '07. Second Annual IEEE International Workshop on*. 201–204. https://doi.org/10.1109/TABLETOP.2007.35

[58] Andrew D. Wilson. 2010. Using a Depth Camera As a Touch Sensor. In *ACM International Conference on Interactive Tabletops and Surfaces (ITS '10)*. ACM, New York, NY, USA, 69–72. https://doi.org/10.1145/1936652.1936665

[59] Robert Xiao, Chris Harrison, and Scott E. Hudson. 2013. WorldKit: Rapid and Easy Creation of Ad-hoc Interactive Applications on Everyday Surfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 879–888. https://doi.org/10.1145/2470654.2466113

[60] Robert Xiao, Scott Hudson, and Chris Harrison. 2016. DIRECT: Making Touch Tracking on Ordinary Surfaces Practical with Hybrid Depth-Infrared Sensing. In *Proceedings of the 2016 ACM on Interactive Surfaces and Spaces (ISS '16)*. ACM, New York, NY, USA, 85–94. https://doi.org/10.1145/2992154.2992173

[61] Zhengyou Zhang. 1994. Iterative point matching for registration of free-form curves and surfaces. *International journal of computer vision* 13 (October 1994), 119–152. https://www.microsoft.com/en-us/research/publication/iterative-point-matching-registration-free-form-curves-surfaces/

[62] Zhengyou Zhang. 2000. A Flexible New Technique for Camera Calibration. *IEEE Trans. Pattern Anal. Mach. Intell.* 22, 11 (Nov. 2000), 1330–1334. https://doi.org/10.1109/34.888718

[63] Yi Zhou, Shuangjiu Xiao, Ning Tang, Zhiyong Wei, and Xu Chen. 2016. Pmomo: Projection Mapping on Movable 3D Object. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 781–790. https://doi.org/10.1145/2858036.2858329

Received May 2017; revised November 2017; accepted January 2018