SpaceState: Ad-Hoc Definition and Recognition of Hierarchical Room States for Smart Environments

Andreas Fender[1], **Jörg Müller**[2]
[1]Aarhus University, Denmark, andreasfender@cs.au.dk
[2]University of Bayreuth, Germany, joerg.mueller@uni-bayreuth.de

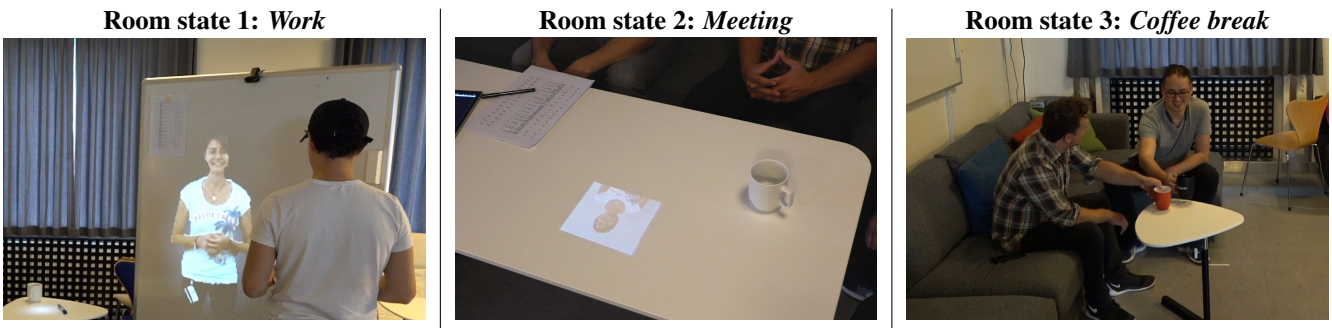

Figure 1. Designers create spatial user interfaces and design variations for each purpose of the same physical space. SpaceState automatically switches to the intended content based on the physical layout of the room. Example application: an adaptive room-scale video conferencing system. (1) *Work* state. If the local participant is working at the whiteboard, then the whole body of the remote participant is displayed on the whiteboard. Depending on whether the local participant is standing left or right, the remote participant is displayed right or left. (2) *Meeting* state. If a large table is in the room, but no whiteboard, then the caller's face is displayed on the table. (3) *Coffee break* state. The remote participant cannot call in during this state.

Abstract

We present SpaceState, a system for designing spatial user interfaces that react to changes of the physical layout of a room. SpaceState uses depth cameras to measure the physical environment and allows designers to interactively define global and local states of the room. After designers defined states, SpaceState can identify the current state of the physical environment in real-time. This allows applications to adapt the content to room states and to react to transitions between states. Other scenarios include analysis and optimizations of work flows in physical environments. We demonstrate SpaceState by showcasing various example states and interactions. Lastly, we implemented an example application: A projection mapping based tele-presence application, which projects a remote user in the local physical space according to the current layout of the space.

Author Keywords

Point clouds; State recognition; Spatial user interfaces; Interface toolkit; Projection mapping

INTRODUCTION

Digital augmentation of physical spaces is one of the longest standing visions in Human-Computer Interaction. The *Office of the Future* [29] proposed to turn the majority of surfaces in offices into displays resulting in a spatial user interface. Since then, this vision has been realized in several implementations. A wealth of research has built on this concept to explore possible applications and solve specific problems. For instance, *LightSpace* [38] is an implementation using off-the-shelf hardware. The goal of this research area is to seamlessly merge physical and virtual worlds. However, almost all such systems require careful design and calibration of the physical environment, in order to enable augmentation with virtual content. This is in contrast to realistic usage of physical spaces, which are frequently rearranged and modified to different degrees. For most of above examples, such rearrangements break the virtual augmentation. This prevents users from rearranging space ad-hoc, like they normally would.

In order to achieve a deeper merging of physical and virtual worlds, the virtual world needs to adapt to the physical. This problem was addressed in *OptiSpace* [8], which automatically adapts the position of virtual content depending on the availability of projection surfaces, occlusions etc. This approach solves the problem on a *syntactic* level using visibility measurements and automatic calibration in order to enable the output of virtual content in different physical environments. However, we argue that the problem is deeper. Physical environments are rarely changed randomly, but the change usually carries meaning. For instance, in an office, meeting room or seminar room, tables and chairs are regularly rearranged to reflect changes of usage of the room. Therefore, a current trend is to make spaces more and more flexible in regards to their physical arrangement. Modern furniture and interior design companies offer solutions for easy physical reconfiguration depending on the situation, e.g., by attaching wheels to furniture or by designing modular furniture pieces [16]. With this, offices can be arranged easily, which is especially relevant,

ISS '19, November 10–13, 2019, Daejeon, Republic of Korea.

ACM ISBN 978-1-4503-6891-9/19/11 ...$15.00.
http://dx.doi.org/10.1145/3343055.3359715

e.g., for workshops, where rearrangements happen frequently [13]. Digital content should then not only *syntactically* adapt to these changes. Ideally, the room layout should change the *semantics* of the digital content. The physical and virtual world should share a common *state*, with a meaning that has been carefully chosen by the designer of the virtual content.

We present SpaceState, a system that identifies such *states* of a physical space from real-time point clouds. With SpaceState, designers of spatial user interfaces can easily define physical states to adapt virtual content accordingly in order to create systems, which are aware of the current state(s) of the room. With our approach, states can be defined on the fly without the need of predefined 3D models. We capture point clouds, so that states can be defined in 3D space. Furthermore, our technique divides the space into volumes, which are strictly either non-overlapping or hierarchical. This allows for defining independent and hierarchical states. To our knowledge, SpaceState is the first system, which:

- Allows designers of spatial user interfaces to create content, which is mapped to the state and sub states of the physical room.
- Identifies the current state of a room and its sub states in real-time using a fast voxel-based algorithm.
- Organizes states hierarchically and allows for defining mutually independent states.

In this paper, we describe our voxel grid based approach and the involved algorithms for defining and identifying hierarchical states.

EXAMPLE APPLICATION

As a proof-of-concept, we implemented an application prototype based on SpaceState. The application draws inspiration from *Room2Room* by Pejsa et al. [27]. Using projectors and multiple Kinect cameras, we illuminate the environment to project a remote participant onto the surfaces of the local physical space. The aforementioned *Room2Room* system assumes a static environment and only adapts to the location of the local participant. In SpaceState, we allow for multiple combinations of furniture arrangement and participant locations to adapt the projection accordingly. Figure 1 shows how the application adapts to different states and situations of the room.

Furthermore, the identified states are utilized on the other end, i.e., at the remote participant. The remote participant is automatically informed about the current state of the room before picking up or making the call. That is, the participant knows beforehand, whether the whole body, only the face or no video at all will be transmitted. Furthermore, during the *Coffee break* state, calling in is automatically disabled.

Walkthrough

In this sub section, we illustrate how the example application was built in SpaceState. Before starting to generate states and projecting content, the designer calibrates the system, i.e., the projectors and Kinect devices (e.g., by using the *RoomAlive Toolkit* [39]). Next, the designer initializes the built-in functionality for streaming video from the remote participant. The goal is to adapt the output of the video stream to the current room state.

First, the designer arranges the movable furniture in the room to allow for working on a whiteboard. The designer then presses a button to capture the room and to create the first state, which he or she calls *Work*. He or she then places the projection of the remote participant on the whiteboard using the default user interface provided by the *Unity3D* game engine. Next, during a meeting, the designer creates a second room state called *Meeting*, by pressing the capture button again. Now he or she places the rendering of the remote participant on the meeting table as a small window. Lastly, during a coffee break, the designer creates the third room state called *Coffee break*. Except for a small spotlight on the coffee table, nothing is projected during this state.

The system now automatically distinguishes between the three states of the room and displays the remote video stream as defined for each state. However, the defined projections might not be the most appropriate ones for the variants within a room state. For instance, if the room is in a meeting state, then there might still be different chair arrangements. Therefore, designers can define sub states. Within the *Work* state, the designer captures the position of a person in front of a whiteboard. The position of the projection can then be defined for key positions of the person within this room state. For the *Meeting* state, the designer defined two sub states: If there is only a table available, then the projection is displayed as a small patch (see Figure 1.2). If a whiteboard is present, then the projection shows the remote participant in full body size on the whiteboard.

RELATED WORK

Several systems have been developed to make physical spaces interactive. For instance, *RoomAlive* by Jones et al. [18] turns living rooms into immersive displays using projection mapping. *UbiDisplays* by Hardy et al. [15] enables ad-hoc creation of projected displays in physical space. Xiao et al. present *WorldKit* [40], which is a tool to create interactive widgets ad-hoc in physical space.

None of these systems cope with changes of the physical environment and they are not aware of its state. In the examples, projections are calibrated for static environments. We extend these ideas by not only letting designers create content in physical space ad-hoc, but also let them define for which state the content is active. In the remainder of this section, we review previous approaches in related areas.

State identification

Several approaches for identifying states of spatial data can be found in medicine. The use of voxel grids have been proven useful in this area. For instance, with *multi-voxel pattern analysis* [2], voxelized patterns of active regions in a brain are classified to identify stimuli. We base our state identification on voxel grids as well, but applied to physical space. Related previous work based on physical surfaces utilized voxel grids for arranging content in physical space [9, 8] or for visually manipulating it [21]. However, this paper deals with inferring the state of the room, whereas adjusting the arrangement of

content is one of the possible use cases. Another approach is the use of machine learning to identify the current state directly from video feeds. For instance, Bhatia et al. [4] identify the progression of a surgery based on predefined states. Danninger et al. [6] use video streams to identify the activity within an office and to mitigate interruption by displaying contextual information, e.g., to visitors at the office entrance. However, such approaches do not allow for organizing the states in 3D volumes. Instead of one or multiple video streams, we operate on one voxel grid, which inherently resolves redundancies from multiple depth camera viewpoints. This makes defining spatially independent or hierarchical states easier than with purely video based approaches.

Adaptive context-aware systems

The context awareness research area has accelerated when computing moved away from the desktop, and was used in various different situations. Context has been defined as "any information that can be used to characterize the situation of entities (i.e., whether a person, place, or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves. Context is typically the location, identity, and state of people, groups, and computational and physical objects" [7]. Numerous systems that adapted to their contexts have been developed [30, 1]. The *Proximity Toolkit* [22] tracks users including their pose relative to each other and to devices. Similarly, activity recognition identifies actions and intentions of humans [34, 33, 35]. For instance, Mori et al. [24] built a sensing system to accumulate and summarize daily activity data. Activity recognition enables a broad range of applications, e.g., in assistive technology [5], since it allows systems to react to situations and users' intentions.

While our work is related to context-aware systems and activity recognition, we are pursuing a more general approach, which infers the context from physical surfaces and the physical layout of the space.

Spatio-temporal grouping

Spatio-temporal grouping techniques generate a structured representation of independent moving objects and background [23]. With *motion segmentation*, moving objects are identified in dynamic scenes [36, 12]. These segmentation algorithms are often restricted to objects that meet certain criteria, like being rigid [31, 37, 28]. Yan et al. [41] propose a general approach, which is not restricted to rigid bodies. Goh et al. [14] and Fradet et al. [11] cluster motions of objects. Motion segmentation is generally object-centric, whereas we do not focus on separating objects and identifying motions.

3D and 4D reconstruction

Advances in surface reconstruction techniques allow for creating high accuracy virtual representations of physical surfaces [19, 42]. The physical space is scanned using RGB or RGB-D cameras to create an accurate virtual representation, typically as a 3D mesh. A lot of research has been done in the area of static and dynamic surface reconstruction, which we refer to as 3D and 4D reconstruction, respectively. *KinectFusion* by Newcombe et al. [26] creates a detailed SLAM based 3D reconstruction, using a commodity depth camera. A more recent version called *DynamicFusion* [25] uses voxelized warp fields to generate 4D reconstructions. Keller et al. [20] and Süßmuth et al. [32] present 4D reconstruction approaches that do not rely on voxel grids. Herbst et al. [17] propose a SLAM approach that copes with changing object locations.

All of these systems focus on creating very detailed meshes to reconstruct static and dynamic surfaces as accurately as possible. They typically update the background surface whenever it changes and overwrite the previous one without representing the changes [26]. Furthermore, these approaches are often limited to objects that fit certain criteria like not changing their topology [25] or being rigid [31]. We are not focusing on high-precision reconstruction, but instead on representing different states. Geometric reconstruction of each of these states can be seen as complementary to our work.

SYSTEM

SpaceState is tracking the physical environment at all times to define and identify states. During the state creation step, designers can declare the current room layout as a new state and position virtual content accordingly. New states can also be defined as sub states of other states. After a state system has been generated, SpaceState can identify the current state of a physical environment in real-time.

All parts are interleaved, while the application is running. That is, we follow a designer-in-the-loop approach, with which new states are defined on the fly depending on the current physical layout. Once a state is defined, it can immediately be identified and distinguished from pre-existing states and virtual content can be bound to it. Contents can be images, text, browser windows, interactive widgets etc.

User interface and workflow

SpaceState features a unified graphical user interface for defining states as well as for associating content. Figure 2 shows the user interface of SpaceState. The user interface is integrated as a Unity3D plugin. Overall, the system is split into two

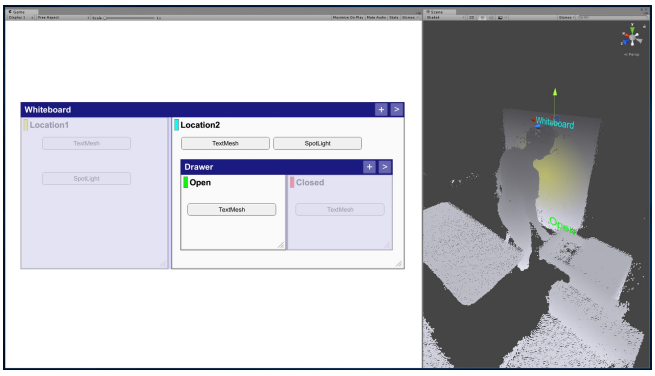

Figure 2. The user interface of SpaceState is integrated into the Unity interface. Designers can navigate over a 2D canvas to create and organize states and content (left). The current point cloud is displayed and content can be manipulated using Unity's widgets (right). Based on the real-time point cloud, designers can create new states at any time by pressing the "+" buttons of the identifiers in the canvas. Virtual content can be created and bound to states using drag-and-drop.

connected parts for positioning 3D content and for defining states. The first part is the Unity scene view, which shows the real-time point cloud of the physical environment. It is used for placing virtual content in 3D (see Figure 2 right). The second part is a 2D canvas, which we call *state canvas*. The state system, its hierarchy and the contents are organized on that canvas (see Figure 2 left). Virtual content is managed in both parts. The state canvas allows for creating content and setting properties, whereas the Unity scene view is used for setting its transformation. The state system is only managed in the state canvas. Designers can navigate on the canvas and insert *state identifiers* at any time. A state identifier contains any number of states, and it controls, which of its states are active. State identifiers are represented as boxes with blue title bars. A button on that title bar triggers the creation of a new state based on the current physical layout. The new state gets integrated into the state identifier after a short processing time while the application keeps running. States are represented as boxes within their identifiers. Content like images or browser windows can then be put into the created states to make that content appear whenever the respective state is active. During runtime, each state identifier calculates, which of its states is active. Inactive states are grayed out in the state canvas. Each state identifier has a second button to configure and investigate the identifier. Within that menu, designers can see, which state was active for which amount of time, e.g., to see in which state the room was mostly in. State identifiers and states can be dragged out of and into other states and identifiers to manipulate the hierarchy at any time. If separate state identifiers are created on the same hierarchical level, then they do not influence each other when identifying their states. That is, the states are independent across identifiers.

Besides this default mode, where content is created live and ad-hoc depending on the current physical layout, there are other ways to create and manipulate content. After creation, states can be activated and deactivated manually to create content for them. The UI then shows the captured point cloud of the state, instead of the current physical point cloud, to base the 3D content on. Furthermore, a real-time point cloud can be recorded and played back, so that designers can navigate through the 3D video and generate states based on certain frames.

State system structure

Figure 3 shows the hierarchical structure of SpaceState. Applications are organized in states and sub states. States can be active and inactive depending on whether the current physical state matches the state that the state represents. Furthermore, states contain content, which is active, if and only if, the associated state is active. Consequently, the content of the application is closely connected to the states and can be organized hierarchically as well. Multiple state identifiers in the same states work independently (see for instance identifiers A and B in Figure 3). For instance, a state identification of a drawer in one part of the room should not influence the state identification of a whiteboard position. These different modes can be mixed arbitrarily and individually for each state when setting up the hierarchical state system.

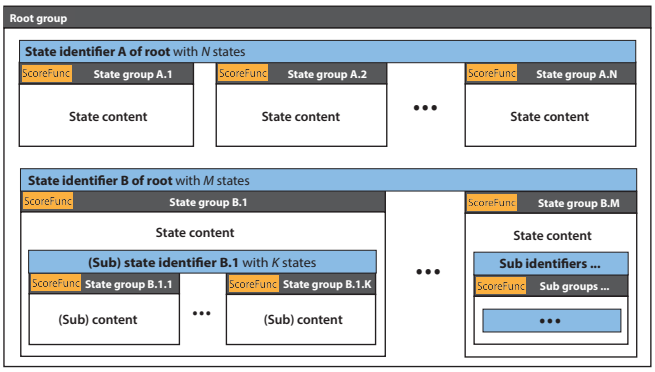

Figure 3. Hierarchical structure of states including content. A state can contain content to be displayed and/or interacted with. The contents are active depending on whether their state is active, i.e., the physical space is in the state or sub state that the defined state represents. Every state has an associated score function to base the identification on. A group can have multiple parallel state identifiers for their child-groups. This implies that the states of one part in the room do not affect the states of another part in the room.

DATA STRUCTURES AND ALGORITHM

This section describes the concepts and data structures for representing the states. Furthermore, it deals with the algorithm for identifying states in real-time. For the sake of simplicity, we first describe the concepts and algorithm for identifying one global state of the room. Afterwards, we discuss, how the same algorithm generalizes to hierarchies of state identifiers and independent states.

Definitions

We start by defining some basic mathematical representations of different objects and concepts to be used to describe the algorithm.

Point cloud P_t

We denote the point cloud that represents the physical space at the current frame t as $P_t \subset \mathbb{R}^3$. The point clouds change in real-time, generating a new P_t at each frame. They are used for both, generating the states and identifying them.

Score functions

Given a point cloud, we define a score function for every point in 3D space:

$$score_t : \mathbb{R}^3 \mapsto [0 \,..\, 1] \tag{1}$$
$$score_t(x,y,z) = \frac{1}{1 + D_t(x,y,z)}$$

whereas D_t is the distance to the closest point of the point cloud:

$$D_t : \mathbb{R}^3 \mapsto [0 \,..\, \infty[\tag{2}$$
$$D_t(x,y,z) = \min_{p \in P_t} \|(x,y,z)^T - p\|$$

Given a point in 3D space, the score outputs a value between 0 and 1 that describes how close the point is to the physical surface. A value of 1 means the point is on a physical surface. Whenever a state is created, the current score function is captured and associated with the state.

Voxel grids

We divide the space into a regular grid of voxels. Each voxel of a voxel grid is a cube with an edge length of s_v, whereas we used $s_v = 10$cm for most examples, as this length has an appropriate level of accuracy. The two corners $g_{min}, g_{max} \in \mathbb{R}^3$, which span the grids are chosen to cover the physical space. The voxel size and the grid corners are the same for all grids of an application. Each voxel grid contains $N_v \in \mathbb{N}$ voxels, whereas N_v depends on the voxel size and the extends.

All voxels are indexed using i:

$$i \in I_{all}, \text{ where } I_{all} = \{1 \,..\, N_v\}$$

The center points of all voxels ($voxelcenter : I_{all} \mapsto R^3$) are constant and independent of the voxel grid, i.e., the same index refers to the same center point across all voxel grids.

Voxel grids are used in three parts of SpaceState, whereas each have different purposes. (1) For regularizing the point cloud (one global voxel grid that changes every frame). (2) For storing information for each state (persistent voxel grid for each state). (3) For storing information for the state identifier (persistent voxel grid for each state identifier). All of these will be described in the following.

Regularized point clouds

As a basis for calculating the score, we first regularize the live point cloud. Regularizing means, that we assign an average position to each voxel based on the points that it contains.

We define the $i-th$ voxel's point set as the set of points, which are located within its boundaries, whereas each point lies in exactly one voxel:

$$P_i = \{p \in P_t \mid p \text{ lies within voxel } i\}$$

A voxel is called currently *occupied*, if it contains at least 5 points from the current P_t. We create a list of voxel indices of the occupied voxels.

$$C_t = \{i \in I_{all} \mid \|P_i\| \geq 5\}$$

We chose 5 as the minimum amount, since it worked well with the voxel size, amount of cameras and level of noise in our setup. The reliability of the algorithm is not very sensitive to the chosen number. All points within the same occupied voxel are averaged:

$$avrgpos : C_t \mapsto \mathbb{R}^3 \tag{3}$$
$$avrgpos(i) = \frac{1}{\|P_i\|} \cdot \sum_{p \in P_i} p$$

As a side effect of averaging the points for each voxel, noise from the depth cameras is reduced. We further reduce noise by smoothing the average positions and ignoring flickering voxels, i.e., voxels that frequently switch between being above and below the minimum point count of 5.

Throughout the application runtime there is always one voxel grid containing the regularized and smoothed point cloud of the current frame. That point cloud serves as input for identifying states and generating new states. The original point cloud P_t is not needed anymore from this point.

State creation

As mentioned above, for the sake of simplicity we assume that there is only one state identifier, when describing the algorithm. The state identifier has N_s states, whereas we use $k \in \{1 \,..\, N_s\}$ to refer to the $k-th$ state of the identifier.

When the designer captures a new state, the current regularized point cloud is not stored directly. Instead, we generate a new voxel grid and create a discretized score function s_k for the state k. For each voxel, we store the score of its center.

$$s_k : I_{all} \mapsto [0 \,..\, 1] \tag{4}$$
$$s_k(i) = score(voxelcenter(i))$$

Overall, s_i can be seen as a discrete pre-computed approximation of the *score* function (see Figure 4). Furthermore, for each voxel we store the id of the closest occupied voxel.

From here, we will denote the point cloud and voxels that change at every frame as *live* and preprocessed static state data as *pre-captured*.

State identifier preprocessing

Some of the calculations for state identification can be preprocessed to speed up the real-time identification. The preprocessed data needs to be updated, whenever a new state is added or changed, since this data depends on all states.

First, we iterate over all states of the state identifier and calculate the *relevance* of each voxel. The relevance describes how much the score function differs among all states:

$$\omega : I_{all} \mapsto [0 \,..\, 1]$$
$$\omega(i) = \max_{k \in N_s} \{s_k(i)\} - \min_{k \in N_s} \{s_k(i)\}$$

The result is a number between 0 and 1, which is the difference between the maximum and minimum score across all states for the voxel i. In principle, this value describes how much weight this voxel has for the identification. Afterwards, we create a

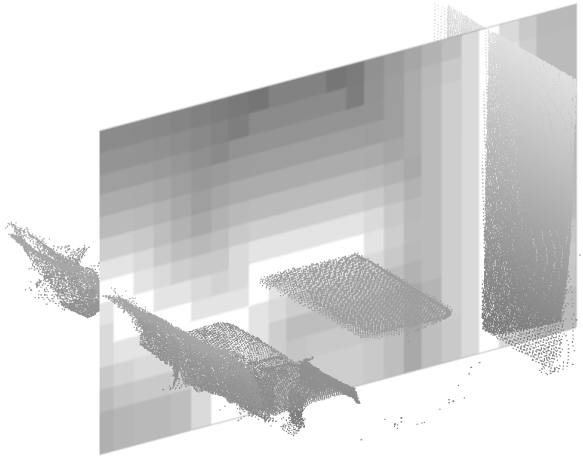

Figure 4. Example score function. The picture shows a transverse section, which visualizes samples of the score for the depicted point cloud. The brighter a square is on the transverse section the closer it is to the closest point on the point cloud.

set of relevant voxels, which have a minimum relevance:

$$I_{rel} = \{i \in I_{all} \mid \omega(i) \geq 0.1\}$$

This basically discards all voxels, which do not differ much across states, e.g., voxels that are close to completely static surfaces. Figure 5 shows relevant voxels of two example identifiers. There are in fact more conditions for a voxel to be relevant for the state identifier. We will discuss this in *Independent identifiers*.

Real-time state identification

Given the preprocessed data and the live point cloud, the score for a state can be computed. There are two main goals for the algorithm: Firstly, it has to detect states even if the physical objects do not perfectly match the pre-captured state, e.g., a whiteboard like in Figure 6. Secondly, it has to be robust with respect to points in the live point cloud that should not influence the state, e.g., moving objects or people in the room.

The general idea for the score calculation is to check for each point of the pre-captured surface, how far away the closest point in the current live point cloud is. In other words, the score is the sum of each pre-captured surface point's distance to the current point cloud over the whole pre-captured surface. The solution is similar to finding point correspondences within the *Iterative Closest Point* algorithm [3].

One direct approach would be to iterate over all occupied voxels of the pre-captured state and find the current closest point in the live point cloud. However, due to the preprocessing, it is a lot faster to find the closest point to the pre-captured state, given a live point, i.e., the other way around. For this reason, we instead iterate over each live point once and use the pre-computed closest voxel index at that position in the grid. That index is the index of the closest voxel in the pre-captured data. We denote this as a live point being *projected* onto a pre-captured voxel. For the closest voxel, we then store the maximum score, i.e., if subsequent live points are projected onto the same pre-captured voxel, only the highest score is stored. Iterating over every point in the regularized live point cloud to calculate the maximum score for the projected voxels runs in linear time. Next, we simply sum up all the maximum scores of the voxels in the pre-captured surface. This step also runs in linear time, whereas in worst case the amount of iterations is the number of the pre-captured occupied voxels of the state.

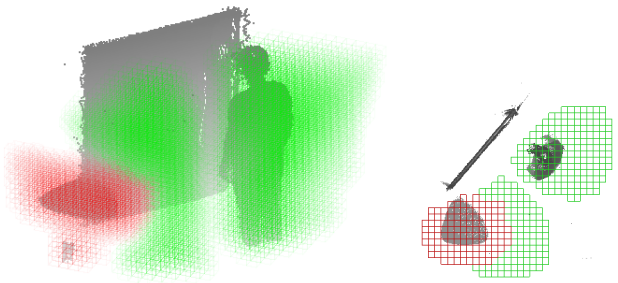

Figure 5. Relevant voxels of two independent state identifiers with two states each. The first identifier distinguishes between the user position relative to the whiteboard (green) and the second one tracks whether or not a table is present (red). The opacity indicates the relevance. Note that the relevant voxels of the table identifier (red) and the person location (green) do not overlap so that they do not interfere. Highly relevant voxels of the table remove less relevant voxels for the location identification (see part of green volume, which is carved-out by red voxels). The part between the two person locations is irrelevant, because these voxels could not distinguish between the person being left or right. A third state would be needed (PersonMiddle) to fill out this part.

This algorithm is tolerant towards slightly misplaced objects, while ignoring objects of different shape. For instance, a flat whiteboard, which is slightly shifted compared to its pre-captured state will still receive a high score, since many live points are projected to different voxels of the pre-captured whiteboard. In contrary, if a person instead of a whiteboard stands in that region, then that person will only contribute little score to the whiteboard state. We will elaborate on the robustness of the algorithm in *Robustness test*.

Independent identifiers

Previously, we described the algorithm for one state identifier, i.e., for identifying one global state. A state group can have more than one sub state identifier, which are then independent of each other. For instance, state identifiers *A* and *B* in Figure 3 are independent. The relevant voxels of independent identifiers of the same group influence each other. To make sure the state identifications do not interfere, voxels are removed from the identifiers so that voxel grids do not overlap across identifiers. A concrete example is depicted in Figure 5. If a voxel is relevant to multiple independent state identifiers, then it is removed from all identifiers, except for the one where the voxel has the highest relevance. This implicitly shapes volumes in the physical space, within which states are identified independently, e.g., the person and the table in Figure 5.

Performance

The state identification is the most critical part for performance, since it needs to run at every frame. The voxel grids with pre-calculated scores, which are generated for each state, allow for very fast access during runtime. With our hierarchical approach, only sub states of active super-states have to be processed. Furthermore, independent state identifiers can be executed in parallel.

Within each state score calculation, the biggest performance bottleneck is the iteration over the large number of voxels. Therefore, the data structures and the algorithm are designed to minimize the amount of iteration steps. Firstly, the amount of points is reduced during runtime by regularizing the live point cloud. Secondly, it is never necessary to iterate over all voxels in one loop during runtime. For instance, for every new frame, we only reset and clear voxels, which have been occupied in the previous frame by maintaining an array of occupied voxel indices. This is particularly important, because generally most voxels are unoccupied and do not need to be reset. This heavily reduces the number of iterations.

CONTENT AND SUB STATE ALIGNMENT

When users switch between different physical arrangements, i.e., when they switch between captured states, then the arrangements might not exactly match the recorded states. For

instance, if designers capture a state of a whiteboard, then the state will be identified, even if the whiteboard is not exactly at the capture position. However, the content of the captured state was specified relative to the captured whiteboard and it might therefore be misaligned at runtime. Furthermore, sub states might rely on the exact position of the whiteboard. For instance, in Figure 5, the green relevant voxels always need to be aligned with the whiteboard to reliably identify whether the user is standing left or right relative to the whiteboard. To address this, we apply the *Iterative Closest Point* (ICP) algorithm [3]. In general, ICP takes two point clouds and finds a translation-rotation-scale transformation between these. Different variants exist for different use cases. In our case, we only need rigid transformations (translation-rotation), but we need to handle point clouds, which do not fully overlap, since there are redundant or missing points in the live point cloud. With this, we can align the contents of the currently active state with the live point cloud as depicted in Figure 6. Furthermore, sub state identifiers utilize the alignment to identify sub states more reliably. SpaceState supports ICP refinement for states with rigid objects.

The identified state together with the data, which is already processed in SpaceState form a useful input for the ICP algorithm for the following reasons:

- The identified state can be seen as initial guess for ICP, i.e., the algorithm is only executed for active states and therefore it is likely that a rigid transformation can be found and that only few iteration steps are necessary.
- The regularized real-time point cloud is reused for ICP to heavily reduce the number of points. Only few locally averaged representative points need to be aligned.
- Instead of aligning complete point clouds, only points within relevant voxels need to be aligned. Instead of the whole point cloud, only the relevant objects are mapped to the captured data with a rigid transformation.

The last point is particularly important to increase speed and robustness. For instance, if a whiteboard needs to be aligned, then all points of other furniture like a couch have to be ignored in the live data. In Figure 5, if the small table is misaligned, then only the live points from within the red volume are considered for finding the transformation.

We run the ICP refinement at every frame to find the rigid transformation from the live point cloud to the static captured points of the currently active state(s). The resulting transformation is then the inverse. Using the live point cloud as source and the captured point cloud as static target followed by inverting the transformation is much faster than the other way around, since the captured point cloud contains a static preprocessed voxelized distance function. We can reuse that distance function for ICP to quickly find corresponding points during iterative alignment.

We apply the resulting transformation in two different ways. Firstly, all contents associated with the state are undergoing the transformation to align them with slightly misplaced objects. For instance, if a whiteboard contains projected content or interactive widgets, then these will be aligned to the physical whiteboard. Secondly, the transformation is applied hierarchically, that is, all sub state identifiers within the currently active state apply the resulting transformation to the point cloud before identifying their states. For instance, if a table can be elevated to switch between sitting and standing, then ICP would first transform to the exact current elevation of the table before states of objects on the table are identified.

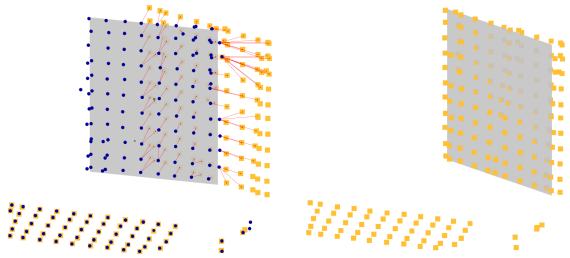

Figure 6. Example refinement based on ICP. The state for the whiteboard has been captured as depicted with the blue circles (left). The orange squares depict the live point cloud during runtime. The state is still identified, even if the whiteboard is slightly off compared to its captured pose. Using ICP, the virtual whiteboard contents (gray surface) and the sub states can be transformed to match the whiteboard's pose (right). Based on the concept of relevant voxels, the table (lower part) is ignored during refinement.

IMPLEMENTATION

This section provides an overview of the hardware, software and frameworks we used.

Hardware and software

In our example setup, we use three Kinect devices, each connected to one PC to process RGB-D streams and transmit them to the main PC, which runs the application. In addition, we use a 4K projector for projecting high-definition content, such as browser windows and text. SpaceState is implemented in *Unity3D* using *CSharp*. The architecture of SpaceState is based on the *Velt* framework [10]. The framework handles the various streams of multi-RGB-D camera networks and generates point clouds in real-time. The SpaceState components and algorithms are implemented as data flow nodes to follow the pipes-and-filters architecture of Velt. RGB-D streams from multiple devices are combined into the real-time regularized point cloud and passed on to the main measurement node of SpaceState. Furthermore, we implemented specialized nodes and node group types to represent states and identifiers.

Example application

Figure 7 shows the structure and state system of the application that we discussed in *Example application*. The contents of the states are based on the built-in datatypes of Velt. These include spotlights and renderers for RGB-D streams. A remote machine streams RGB-D data from a single Kinect. The communication with the remote machine is handled by Velt. SpaceState directs the stream to the states, which are currently active. States either generate an RGB-D-mesh to render a full size version or a small textured quad mesh that simply shows a region within the video stream.

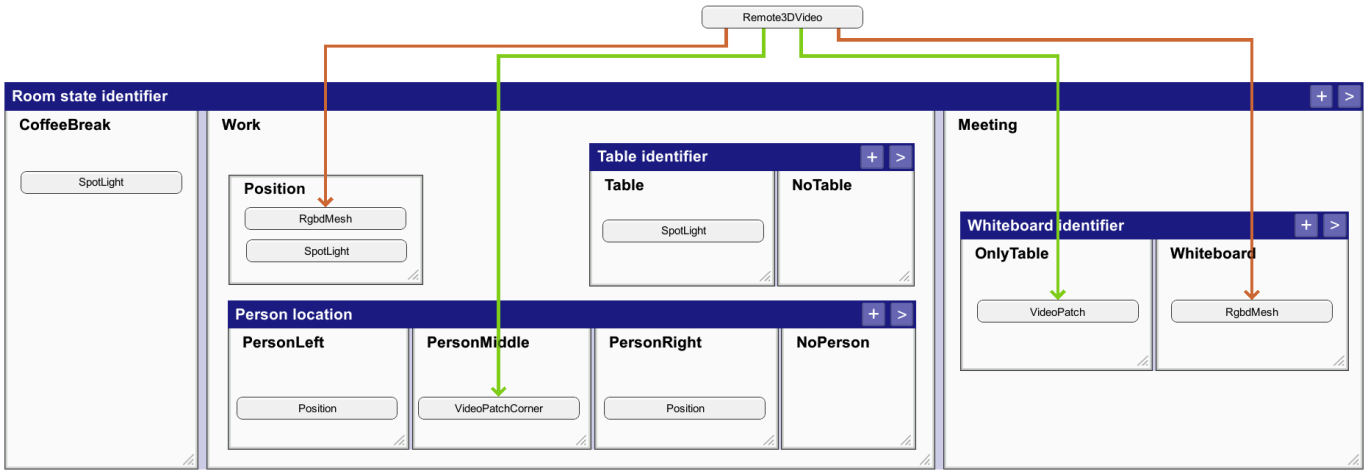

Figure 7. The state system of our example application as rendered in the SpaceState user interface. The RGB-D data stream is received from a remote machine. Depending on the active state, the stream is displayed at different locations and in different ways.

EVALUATION

SpaceState does not directly aim at end-users, but designers of spatial user interfaces. Therefore, instead of a user study, we evaluated our system on two separate levels. First, we conducted an expert review to evaluate SpaceState from a spatial user interface design perspective. Second, we conducted a robustness test by letting participants rearrange a physical space multiple times to evaluate the reliability of the underlying algorithms and data structures.

Expert review

We conducted an expert review to discuss the concept and implementation of our approach. Overall, we wanted to find out, whether the workflow is comprehensible.

We invited an external expert with more than ten years of experience in 3D interaction design and projection mapping systems. His primary tasks consist of content generation for public installations. He took part in projects, in which he designed, implemented and deployed interactive projection mapping exhibitions.

The expert was greeted and introduced to the setup, concepts and usage of the system. He was then asked to try out the main functionalities by defining different states, sub states and independent states. The expert found it generally easy to define states and generate content for these states:

> *Once you've seen it, it's quite easy to understand. [...] I understand the concept, also about the nested identifiers.*

We discussed, how it would differ, if the goal was to make the prototype usable for end-users and not only designers. He noted, that the system's full capabilities are not accessible to end-users in its current form. The functionalities would need to be narrowed down and more visual feedback for users is required. From that, we concluded that future extensions would need to trade expressiveness for specialized functionalities to make it end-user-friendly (we discuss this further in *Discussion and future work*). The interviewer asked the expert, whether he could imagine other future extensions.

> *[In our lab] we have been focusing on making the projections extremely precise. I think the combination of these two ideas would be interesting.*

Partially in response to that statement, we incorporated ICP in our system for automatic alignment without markers (see *Content and sub state alignment*).

The interviewer and the expert briefly discussed technologies other than projectors:

> *You can [for instance] use the position of objects to change the contents of your phone [...]. So it doesn't have to be projection mapping. [...] You can also use it to control spatial audio. [For example] you can rearrange the office to turn down the music.*

Overall, the expert appreciated that the system works without markers and without technology that is attached to the furniture or people. He further stated:

> *I like the way that you can freely just arrange furniture and then create states for it. I think that's a new way of thinking about it. You setup your environment and then you just press a button and that's your new system.*

The interview with the expert makes us confident, that the concept of SpaceState is comprehensible and can be used for designing state-aware spatial user interfaces.

Robustness test

With this part of the evaluation, we wanted to investigate the robustness of SpaceState based on physical layouts arranged by external participants.

Data acquisition

The goal of the data acquisition was to gather point cloud recordings of various participant-generated physical layouts. The participants were informed that they were recorded. However, the experimenter did not inform them about the purpose and functionality of the system, so as to not change participants' way of arranging the room, i.e, the participants did not know that very distinct setups are easier to handle for

1) Annotated (ground truth)

2) Identified and correct (green) versus incorrect (red)

Only furniture pre-captured in state: 89% correctness

Furniture and people pre-captured in state: 96% correctness

Figure 8. Timelines of one example session in our performance evaluation. Within the whole duration of the timelines, the space was rearranged multiple times and the usage was acted out for each. Each color in the top parts of the plots represents one of the four states. Gray time spans are transitioning times (rearrangements) and not part of the measurement. The first timeline contain manually annotated states (1). We created a tool to create timelines that compare the annotations with the states that SpaceState identified (2).

Figure 9. During our evaluation, participants did not arrange the furniture exactly in the same way after recreating layouts repeatedly. This example shows a state with a table, a chair and a whiteboard. The participant unknowingly put the whiteboard about 30 centimeters away (orange) from where it was captured (blue). SpaceState still successfully identifies the state, also while people move within it.

the system. No state identification was running at this stage as we only recorded depth streams. Eight participants (three female) were invited subsequently to our test setup, which initially only contained a couch and approximately 3x2 meters of empty space. One experimenter was present to guide the participants and annotate important events for the recordings. Outside the test space, the experimenter provided a selection of movable furniture like chairs, tables with wheels, a movable whiteboard etc. In the first part, participants were asked to arrange the physical space according to three different predefined purposes: *Meeting*, *Working alone* and *Lunch with invited guest*. Afterwards, they were asked to come up with their own purpose and arrange the space accordingly (*Custom*). For the *Custom* state, participants came up with arrangements for *small seminars with multiple chairs*, *watching movies as a group*, *small social events including a bar counter* and more. In the second part, the participants were repeatedly asked to reconstruct the four previously defined layouts. The experimenter randomly selected from the generated layouts that the participant generated and subsequently asked the participant to arrange the layout again. The participants were not informed beforehand that they would need to arrange the same layouts repeatedly. The experimenter referred to the layouts by name and did not support the participants with memorizing the exact layout. This led to some variations when arranging the same layout. Figure 9 shows an example for a physical arrangement that differs from the exact arrangement when it was arranged the second time. Additionally, participants were asked to act out the usage of each space, e.g., pretend they were working in the *Working alone* setup. Overall, each layout was arranged 3 times. That is, the space was rearranged 12 times per session. A session took about 30 minutes. With eight sessions, the space was rearranged 96 times in total. After all participants completed, we used the generated point cloud recordings for subsequent steps in the performance evaluation.

Annotation and analysis

Based on the data acquisition, we first manually annotated the time spans for each state in the recordings to form a baseline for the test. We excluded transition times or other situations where none of the intended arrangements were in place, e.g., while participants were instructed. That is, the annotated time spans include all times where the state was active, including times in which people were moving around in the arrangement. We then generated state systems based on each participant's arrangements, i.e., one state system per participant. Lastly, we played back the annotated time spans of the recordings while SpaceState continuously identified the states as if the point clouds were live. While SpaceState was running, a separate part of our system compared the identified states with the manually annotated baseline and measured the amount of time they were equal (see Figure 8).

Results and observations

The evaluation revealed that SpaceState was mostly able to successfully identify states (approx. 93% over all frames, see Table 1 for details), even if the arrangements were not perfectly aligned to the pre-captured data. There were, however, some problematic state systems. States, which do not differ much in terms of furniture do get mixed up occasionally. For instance, one participant made *Lunch* and *Work* only differ in terms of one additional chair, which was only partially visible to the depth cameras. If the arrangement is then misaligned, these states could not be identified reliably. Capturing users in addition to furniture can help addressing this: Capturing one person in the *Work* state and two in the *Lunch* state would make the states differ a lot more and very easy to distinguish.

	Meeting	Work	Lunch	Custom
Meeting	22.91%	0.06%	2.47%	0%
Work	0%	27.14%	0.54%	0%
Lunch	1.9%	1.59%	21.23%	0%
Custom	0.31%	0%	0%	21.81%

Table 1. Confusion matrix of the states added up over all participants. The rows contain the manually annotated states and the columns contain the identified states. The percentages describe how long each entry was active relative to the total duration. The lunch states were particularly problematic as participants often made them resemble work or meeting setups.

Figure 8 shows how the accuracy increases when capturing users as well. There are, however, still a few wrongly identified frames in the timeline for the times when the pre-captured users were not present. In that case, capturing a second variation of a state can be beneficial. Even though states can benefit from capturing users, we only pre-captured furniture for our final performance evaluation across all states in order to be consistent. This means, e.g., we used the 89% from the example participant in Figure 8 and achieved an overall accuracy of 93% (correct frames divided by total frames of all participants). A designer would need to decide on a state-per-state basis, whether the state is also defined by the location of users and include or exclude them when capturing the state.

While not being a major problem, we noted that occasionally important surfaces were occluded which led to a wrong state identification in rare cases. We used three depth cameras in our test. More cameras can be added to make the point clouds less prone to occlusion and noise.

Overall, the results make us confident that SpaceState can successfully identify states, even if arrangements do not exactly match the captured states and even if moving users or objects are present.

DISCUSSION AND FUTURE WORK

In this work we contribute to the area of spatial user interfaces, with the objective of seamlessly blending virtual and physical spaces. Our work starts from two fundamental assumptions. First, we assume that physical spaces are frequently rearranged. Second, we assume that rearrangement is not random, but carries meaning. Therefore, we conclude that, in order to truly blend virtual and physical, virtual content needs to adapt to the "meaning" of the physical arrangement. Extracting and representing the true meaning of a physical arrangement is a very hard problem for artificial intelligence. We do not assume that it can be solved in the near future. Thus, we decide to circumvent the "meaning recognition" problem and leave the interpretation of the scene to the human content designer. The designer interprets what a certain arrangement means, and designs the virtual content for it including its arrangement. Our system then supports the designer by recognizing the current physical layout and adapting contents, while keeping the designer's ideas about functionalities and aesthetics for each of the purposes of the room. The concept can be applied to many environments that get rearranged frequently, but it has its limitations where it needs to be combined with more specialized solutions. For instance, SpaceState can identify states, e.g., to switch modes in a spatial user interfaces, but fine-grained gestural interaction techniques would then be needed for accurately manipulating virtual content.

Currently, the target group of SpaceState consists of designers and not end-users. This means, that designers are creating the application in the environment, where they want to deploy the system and content can be programmed and extended. As discussed in *Expert review*, a different direction would be to build a more specialized system for end-users. A 3D user interface with a set of standard widgets could allow users to create and position content in physical space similar to *WorldKit* [40]. We can extend this concept by letting users position the content in different room states. The system would then fully automatically infer, which physical layout is associated with which combination of contents.

The algorithm works only based on 3D points. In the future, we would like to experiment with additionally incorporating surface normals or surface color for even more accurate state identification. Furthermore, we would like to also explore scales smaller than room-scale. The increasing availability of close-range depth cameras like the Intel RealSense possibly allow for identifying states of smaller objects, e.g., in an assembly line.

Use cases other than content placement can be explored in the future. SpaceState can support analyzing workflows and provide information about whether processes are executed properly. An inspirational use case is the work by Bhatia et al. [4], who track the state of a surgery. Furthermore, working with states can be useful for analyzing and annotating 3D videos.

Lastly, based on the technical foundation provided in this work, we plan to conduct long-term experiments with designers or, with the above described extensions, with end-users. Short-term user sessions will not be sufficient, as furniture is normally not rearranged within minutes, but over longer time spans. A deployment would therefore need to be implemented over the time span of one week or more in a place with flexible furniture, e.g., furniture with wheels or consisting of modular furniture pieces. With such a deployment in a context where the space is frequently reconfigured, we will be able to test, to which degree SpaceState can successfully map the current arrangement and hence meaning of the physical space to the intended digital contents.

CONCLUSION

In this paper, we presented SpaceState, a system for designing spatial user interfaces that can react to changes in the physical environment. Designers develop virtual content and states together with a drag-and-drop user interface. There is no need for large data sets or 3D models to generate and identify environmental states. Our approach is a step towards the virtual world adapting to the meaning of the physical spatial arrangement. Instead of adapting room arrangements to the digital enhancements, our vision is to enable the design of user interfaces that encourage users to articulate the physical space around them according to the situation. Furthermore, if new uses for a room emerge, then our approach only requires to add new states ad-hoc instead of reprogramming the application. Instead of constraining the physical environment to be static to maintain alignment with virtual content, we utilize the arrangement as input to synchronize the meaning of real and digital. We hope that our approach contributes to a future where virtual and physical worlds can be connected in a more meaningful way than before.

ACKNOWLEDGEMENTS

The authors would like to thank the participants of the data acquisition for the robustness test. This work has been supported by IFD grant no. 6151-00006B for the project entitled: MADE Digital.

REFERENCES

[1] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. 1999. Towards a Better Understanding of Context and Context-Awareness. In *Handheld and Ubiquitous Computing*, Hans-W. Gellersen (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 304–307.

[2] Michael Anderson and Tim Oates. 2010. A critique of multi-voxel pattern analysis. In *Proceedings of the Cognitive Science Society*, Vol. 32.

[3] Paul J. Besl and Neil D. McKay. 1992. Method for registration of 3-D shapes, Vol. 1611.

[4] Beenish Bhatia, Tim Oates, Yan Xiao, and Peter Hu. 2007. Real-time identification of operating room state from video. In *AAAI*, Vol. 2. 1761–1766.

[5] L. Chen, C. D. Nugent, and H. Wang. 2012. A Knowledge-Driven Approach to Activity Recognition in Smart Homes. *IEEE Transactions on Knowledge and Data Engineering* 24, 6 (June 2012), 961–974. DOI: http://dx.doi.org/10.1109/TKDE.2011.51

[6] Maria Danninger and Rainer Stiefelhagen. 2008. A context-aware virtual secretary in a smart office environment. In *Proceedings of the 16th ACM international conference on Multimedia*. Citeseer, 529–538.

[7] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. 2001. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human–Computer Interaction* 16, 2-4 (2001), 97–166. DOI: http://dx.doi.org/10.1207/S15327051HCI16234_02

[8] Andreas Fender, Philipp Herholz, Marc Alexa, and Jörg Müller. 2018. OptiSpace: Automated Placement of Interactive 3D Projection Mapping Content. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA, Article 269, 11 pages. DOI: http://dx.doi.org/10.1145/3173574.3173843

[9] Andreas Fender, David Lindlbauer, Philipp Herholz, Marc Alexa, and Jörg Müller. 2017. HeatSpace: Automatic Placement of Displays by Empirical Analysis of User Behavior. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. ACM, 611–621.

[10] Andreas Fender and Jörg Müller. 2018. Velt: A Framework for Multi RGB-D Camera Systems. In *Proceedings of the 2018 ACM International Conference on Interactive Surfaces and Spaces (ISS '18)*. ACM, New York, NY, USA, 73–83. DOI: http://dx.doi.org/10.1145/3279778.3279794

[11] M. Fradet, P. Robert, and P. Pérez. 2009. Clustering Point Trajectories with Various Life-Spans. In *2009 Conference for Visual Media Production*. 7–14. DOI: http://dx.doi.org/10.1109/CVMP.2009.24

[12] K. Fragkiadaki, G. Zhang, and J. Shi. 2012. Video segmentation by tracing discontinuities in a trajectory embedding. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 1846–1853. DOI: http://dx.doi.org/10.1109/CVPR.2012.6247883

[13] Bene GmbH. 2019. Bene Pixel - Ideas workshop. https://bene.com/en/office-furniture-concepts/office-furniture/en/pixel-ideas-workshop. (2019). Accessed: 2019-11-05.

[14] A. Goh and R. Vidal. 2007. Segmenting Motions of Different Types by Unsupervised Manifold Clustering. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*. 1–6. DOI: http://dx.doi.org/10.1109/CVPR.2007.383235

[15] John Hardy, Carl Ellis, Jason Alexander, and Nigel Davies. 2013. Ubi displays: A toolkit for the rapid creation of interactive projected displays. In *The International Symposium on Pervasive Displays*.

[16] Heartwork. 2019. Square System. http://heartwork.com/product/square-system. (2019). Accessed: 2019-11-05.

[17] E. Herbst, P. Henry, X. Ren, and D. Fox. 2011. Toward object discovery and modeling via 3-D scene comparison. In *2011 IEEE International Conference on Robotics and Automation*. 2623–2629. DOI: http://dx.doi.org/10.1109/ICRA.2011.5980542

[18] Brett Jones, Rajinder Sodhi, Michael Murdock, Ravish Mehra, Hrvoje Benko, Andrew Wilson, Eyal Ofek, Blair MacIntyre, Nikunj Raghuvanshi, and Lior Shapira. 2014. RoomAlive: magical experiences enabled by scalable, adaptive projector-camera units. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*. ACM, 637–644.

[19] Michael Kazhdan and Hugues Hoppe. 2013. Screened Poisson Surface Reconstruction. *ACM Trans. Graph.* 32, 3, Article 29 (July 2013), 13 pages. DOI: http://dx.doi.org/10.1145/2487228.2487237

[20] M. Keller, D. Lefloch, M. Lambers, S. Izadi, T. Weyrich, and A. Kolb. 2013. Real-Time 3D Reconstruction in Dynamic Scenes Using Point-Based Fusion. In *2013 International Conference on 3D Vision - 3DV 2013*. 1–8. DOI:http://dx.doi.org/10.1109/3DV.2013.9

[21] David Lindlbauer and Andrew D. Wilson. 2018. Remixed Reality: Manipulating Space and Time in Augmented Reality. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA. DOI: http://dx.doi.org/10.1145/3173574.3173703

[22] Nicolai Marquardt, Robert Diaz-Marino, Sebastian Boring, and Saul Greenberg. 2011. The proximity toolkit: prototyping proxemic interactions in ubiquitous computing ecologies. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, 315–326.

[23] Remi Megret and Daniel DeMenthon. 2002. *A survey of spatio-temporal grouping techniques*. Technical Report. MARYLAND UNIV COLLEGE PARK LANGUAGE AND MEDIA PROCESSING LAB.

[24] Taketoshi Mori, Katsutoshi Asaki, Hiroshi Noguchi, and Tomomasa Sato. 2001. Accumulation and summarization of human daily action data in one-room-type sensing system. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, Vol. 4. IEEE, 2349–2354.

[25] Richard A. Newcombe, Dieter Fox, and Steven M. Seitz. 2015. DynamicFusion: Reconstruction and Tracking of Non-Rigid Scenes in Real-Time. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[26] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. 2011. KinectFusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*. 127–136. DOI: http://dx.doi.org/10.1109/ISMAR.2011.6092378

[27] Tomislav Pejsa, Julian Kantor, Hrvoje Benko, Eyal Ofek, and Andrew Wilson. 2016. Room2Room: Enabling life-size telepresence in a projected augmented reality environment. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*. ACM, 1716–1725.

[28] Samunda Perera and Nick Barnes. 2013. *Maximal Cliques Based Rigid Body Motion Segmentation with a RGB-D Camera*. Springer Berlin Heidelberg, Berlin, Heidelberg, 120–133. DOI: http://dx.doi.org/10.1007/978-3-642-37444-9_10

[29] Ramesh Raskar, Greg Welch, Matt Cutts, Adam Lake, Lev Stesin, and Henry Fuchs. 1998. The Office of the Future: A Unified Approach to Image-based Modeling and Spatially Immersive Displays. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98)*. ACM, New York, NY, USA, 179–188. DOI: http://dx.doi.org/10.1145/280814.280861

[30] Albrecht Schmidt, Michael Beigl, and Hans-W Gellersen. 1999. There is more to context than location. *Computers & Graphics* 23, 6 (1999), 893 – 901. DOI: http://dx.doi.org/https://doi.org/10.1016/S0097-8493(99)00120-X

[31] Young Min Shin, Minsu Cho, and Kyoung Mu Lee. 2013. Multi-object reconstruction from dynamic scenes: An object-centered approach. *Computer Vision and Image Understanding* 117, 11 (2013), 1575 – 1588. DOI: http://dx.doi.org/https://doi.org/10.1016/j.cviu.2013.06.008

[32] Jochen Süßmuth, Marco Winter, and Günther Greiner. 2008. Reconstructing Animated Meshes from Time-Varying Point Clouds. *Computer Graphics Forum* 27, 5 (2008), 1469–1476. DOI: http://dx.doi.org/10.1111/j.1467-8659.2008.01287.x

[33] Jaeyong Sung, C. Ponce, B. Selman, and A. Saxena. 2012. Unstructured human activity detection from RGBD images. In *2012 IEEE International Conference on Robotics and Automation*. 842–849. DOI: http://dx.doi.org/10.1109/ICRA.2012.6224591

[34] P. Turaga, R. Chellappa, V. S. Subrahmanian, and O. Udrea. 2008. Machine Recognition of Human Activities: A Survey. *IEEE Transactions on Circuits and Systems for Video Technology* 18, 11 (Nov 2008), 1473–1488. DOI:http://dx.doi.org/10.1109/TCSVT.2008.2005594

[35] Tim van Kasteren, Athanasios Noulas, Gwenn Englebienne, and Ben Kröse. 2008. Accurate Activity Recognition in a Home Setting. In *Proceedings of the 10th International Conference on Ubiquitous Computing (UbiComp '08)*. ACM, New York, NY, USA, 1–9. DOI: http://dx.doi.org/10.1145/1409635.1409637

[36] René Vidal, Roberto Tron, and Richard Hartley. 2008. Multiframe Motion Segmentation with Missing Data Using PowerFactorization and GPCA. *International Journal of Computer Vision* 79, 1 (01 Aug 2008), 85–105. DOI: http://dx.doi.org/10.1007/s11263-007-0099-z

[37] C. Vogel, K. Schindler, and S. Roth. 2011. 3D scene flow estimation with a rigid motion prior. In *2011 International Conference on Computer Vision*. 1291–1298. DOI: http://dx.doi.org/10.1109/ICCV.2011.6126381

[38] Andrew D. Wilson and Hrvoje Benko. 2010. Combining Multiple Depth Cameras and Projectors for Interactions on, Above and Between Surfaces. In *Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology (UIST '10)*. ACM, New York, NY, USA, 273–282. DOI: http://dx.doi.org/10.1145/1866029.1866073

[39] Andrew D Wilson and Hrvoje Benko. 2016. Projected Augmented Reality with the RoomAlive Toolkit. In *Proceedings of the 2016 ACM on Interactive Surfaces and Spaces*. ACM, 517–520.

[40] 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*. ACM, 879–888.

[41] Jingyu Yan and Marc Pollefeys. 2006. *A General Framework for Motion Segmentation: Independent, Articulated, Rigid, Non-rigid, Degenerate and Non-degenerate*. Springer Berlin Heidelberg, Berlin, Heidelberg, 94–106. DOI: http://dx.doi.org/10.1007/11744085_8

[42] Michael Zollhöfer, Patrick Stotko, Andreas Görlitz, Christian Theobalt, Matthias Nießner, Reinhard Klein, and Andreas Kolb. 2018. State of the Art on 3D Reconstruction with RGB-D Cameras. In *Computer graphics forum*, Vol. 37. Wiley Online Library, 625–652.