SnapToReality:
Aligning Augmented Reality to the Real World

Benjamin Nuernberger[1,2]   **Eyal Ofek**[1]   **Hrvoje Benko**[1]   **Andrew D. Wilson**[1]
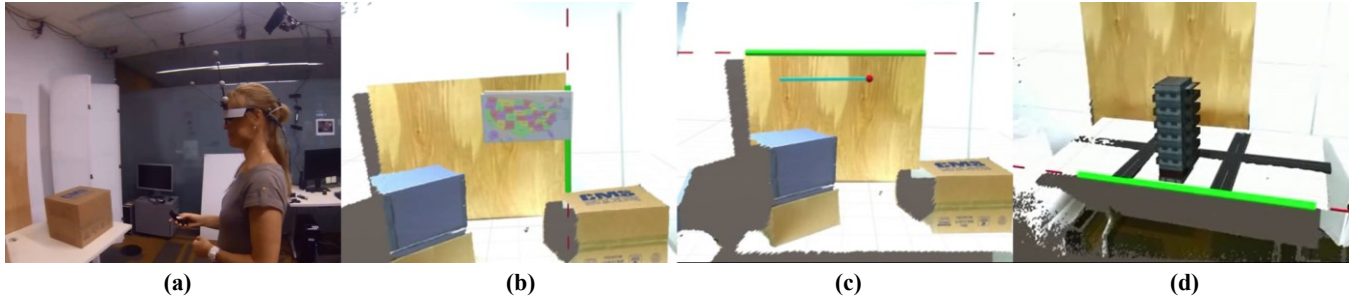
[1]Microsoft Research
Redmond, WA, USA
{eyalofek, benko, awilson}@microsoft.com

[2]University of California, Santa Barbara
Santa Barbara, CA, USA
bnuernberger@cs.ucsb.edu

Figure 1: *SnapToReality* **allows users to easily position, orient, and scale AR virtual content with respect to real world constraints. Our prototype (a) extracts real world planar surfaces and edges as constraints so that users can easily align virtual content to the real world via snapping (b, c).** *SnapToReality* **techniques enable the seamless integration of AR content into the real world (d).**

ABSTRACT
Augmented Reality (AR) applications may require the precise alignment of virtual objects to the real world. We propose automatic alignment of virtual objects to physical constraints calculated from the real world in real time ("snapping to reality"). We demonstrate *SnapToReality* alignment techniques that allow users to position, rotate, and scale virtual content to dynamic, real world scenes. Our proof-of-concept prototype extracts 3D edge and planar surface constraints. We furthermore discuss the unique design challenges of snapping in AR, including the user's limited field of view, noise in constraint extraction, issues with changing the view in AR, visualizing constraints, and more. We also report the results of a user study evaluating *SnapToReality*, confirming that aligning objects to the real world is significantly faster when assisted by snapping to dynamically extracted constraints. Perhaps more importantly, we also found that snapping in AR enables a fresh and expressive form of AR content creation.

Author Keywords
Interaction techniques; snapping; augmented reality; 3D user interaction; user studies.

ACM Classification Keywords
H.5.1. Information interfaces and presentation (e.g., HCI): Multimedia Information Systems—Artificial, augmented, and virtual realities.

INTRODUCTION
In the real world, the positioning of objects is often guided by physical constraints (e.g., stacking objects on the table, or aligning a chair against the wall) or design constraints (e.g., aligning collinear objects, or resizing objects to have the same dimensions).

Performing similar actions in Augmented Reality (AR) may be difficult as virtual objects do not necessarily follow the rules of physics. Meanwhile, positioning virtual content in space around a user is an important task in many AR applications, such as industrial design, home redecoration, games, etc. While it is possible to position objects without regard to the real world surrounding the user, many AR experiences require a precise alignment between virtual objects and the real environment (e.g., Figure 2).

In traditional desktop computing, many sophisticated techniques are available to achieve precise alignment, such as dynamic guides and snapping behaviors [4]. Snapping is a common technique that helps users precisely align content with respect to certain constraints. A typical approach is to automatically align an object with a constraint whenever the user moves the object near to that constraint. For example, Microsoft PowerPoint uses "Smart Guides" to snap content so that it is parallel to other content, directly touching other content, etc. Snapping is also readily used in many 3D modeling programs, such as SketchUp and AutoCAD, making typically difficult 3D positioning tasks easier.

We introduce *SnapToReality*, the concept of snapping *virtual* content to *physical* constraints calculated from the real world in real time. In addition to supporting traditional alignment tasks in AR, SnapToReality enables new forms of interaction due to the world's dynamic nature: A door may be opened, a chair may be moved, and objects may be added or taken away, creating new opportunities for alignment. Furthermore, users may stack objects to generate a new snapping constraint, or even use physical tools (e.g., a measuring tape) to guide the formation of virtual objects.

Snapping to physical constraints is inherently more difficult than snapping to virtual constraints because the physical constraints are not known *a priori* and therefore must be extracted on-the-fly. If such constraints are extracted perfectly, the snapping procedure may be considered identical to snapping to virtual constraints. However, due to sensor noise, complexity of the environment, and AR hardware limitations, it is easily possible to extract the wrong constraints, causing an overall negative impact on performance. To our knowledge, our prototype is the first to demonstrate the extraction of 3D physical constraints to enhance the performance of 3D snapping in AR.

Our main contributions are as follows:

- A proof-of-concept prototype that implements two SnapToReality alignment techniques—snapping to real world edges and planar surfaces in real-time—enabling new expressive forms of interaction by utilizing dynamic real world scene content. Special care is taken to visualize the constraints in the real environment and outside the field of view.

- A detailed analysis of the important design considerations for SnapToReality alignment techniques, including the user's limited field of view, noise in constraint extraction, issues with changing the view in AR, issues concerning dynamic scenes, and visualization of physical constraints.

- A user evaluation demonstrating (a) that positioning and rotating AR objects to be aligned with a real world target is faster with snapping than without snapping; and (b) that snapping enables expressive forms of interaction for creating AR content aligned to the dynamic real world.

RELATED WORK

Snapping to Virtual Constraints
Snapping virtual content to virtual constraints has been investigated in many ways. Eric Bier's work on snap-dragging [3,4] was one of the first. In its 3D version, snap-dragging enables the precise placement of a 3D cursor (the "snap-dragging skitter"), which snaps to points, curves, and surfaces whenever it is close to existing geometry. Snapping between objects using simple affine transformations was also available, thus making precise, relative 3D modeling easier. The AR content creation mode in our prototype draws inspiration from the snap-dragging skitter.

Over the years, snapping has been improved in various ways [1,10,13,17,27]. Today, snapping is also used in many popular desktop and web applications, such as Microsoft PowerPoint, Google Slides, SketchUp, AutoCAD, etc. However, most if not all of these applications apply snapping to virtual constraints only. Snapping to real physical and dynamically changing constraints makes SnapToReality stand apart from previous works.

Snapping to Real Constraints
Important early works on extracting real-world constraints from 2D images for snapping were Gleicher's image-snapping [16] and Mortensen & Barrett's intelligent scissors [23]. More recently, Lee et al. [20,21] applied the idea of image-snapping to helping users annotate objects in 2D images in augmented reality. Nóbrega and Correia [26] also applied the idea of augmenting 2D images by extracting vanishing points and coarse depth from pairs of images. In SnapToReality, we take the idea of extracting real world constraints a step further, going from 2D to 3D.

To the best of our knowledge, no previous work exists on extracting 3D physical constraints in real-time specifically for snapping in augmented reality. With our prototype, we focused on snapping to real 3D edges and planar surfaces. Edge detection has traditionally been approached by using variants of the Hough Transform [8,9,18], or other 3D methods if such data is available [7,15]. Planar surface detection has also used variants of the Hough Transform [5,8,18,28] and we follow this approach.

Constraint Optimization Approaches
An alternative to snapping is constraint optimization [2,14,22,30,31]. Here, rather than allowing a user to manipulate an object to snap to a set of constraints, objects and constraints are jointly optimized to automatically achieve an optimal configuration of the objects with respect to the constraints. FLARE [14] is a recent example of such a system, in which a rule-based framework is used to lay out AR content with respect to real world planar surfaces. One shortcoming of constraint optimization approaches is that opportunities for user input are limited as the system attempts to achieve optimal configurations automatically. Perhaps the ideal system would first suggest automatic placement of objects via constraint optimization and then allow users to precisely manipulate individual objects thereafter. SnapToReality focuses on this latter part.

SNAP-TO-REALITY CONCEPT
SnapToReality allows snapping virtual objects to physical ones using constraints extracted in real-time. This can be used to position, rotate, and scale AR virtual objects to align with the physical world. Such snapping may be as simple as mimicking physical behaviors of real objects not penetrating each other, or as sophisticated as snapping to lie at evenly spaced distances, or to fit harmoniously with the environment color palette.

Motivating Scenarios

We envision alignments to real-world constraints as one of the core interactive requirements of most AR scenarios that deal with 3D modeling or 3D positioning of virtual content. Imagine the AR task of setting up virtual toys in a child's room. The user builds a train bridge from the bed, stretching virtual columns from the floor up to the height of the bed, and then installs tracks on the bed and a bridge that stretches along the wall. Or maybe a person would like to redecorate her room in AR, hanging virtual pictures on the wall, parallel to the existing artwork, walls, and the floor. Lastly, imagine being able to position a variety of 3D virtual content at different available spaces in the room (e.g., above the fireplace, on top of the kitchen cabinet, at the corner of the tabletop) as envisioned in Figure 2. These are just a few of the interaction scenarios made possible by SnapToReality.

Figure 2: An artistic vision of an AR desktop scenario as depicted by Microsoft's HoloLens[1]. Notice the precise positioning of AR apps in various locations in the kitchen—the "Recipes" app is centered over the cabinet doors; the sports game app over the fireplace; etc.

EXTRACTING PHYSICAL CONSTRAINTS

The core capability of our proof-of-concept SnapToReality alignment techniques is the real-time extraction of real world constraints—specifically linear edges and planar surfaces in the environment. These features are common in man-made scenes and often come from semantically important structures such as floors, supporting horizontal surfaces, walls that separates spaces, and more.

While there are potentially many other geometric and non-geometric constraints that can be deduced from the environment, we believe that linear edges and surfaces represent a good starting point in our exploration of SnapToReality techniques. The principles of our work may be extended to other types of constraints, which we leave as future work (cf. Semantic Snapping). We now describe the details of our extraction algorithms.

Extracting 3D Linear Edges

Typical man-made environments contain many linear features. Some of those edges distinguish objects from

---

other objects behind them, others represent a change in an object's surface normal, and some come from a visible change of the object's color. We would like to recover a representation of such features in the scene so that virtual objects may be aligned to them.

We use a Microsoft Kinect 2.0 camera to capture the scene continuously in several modalities. First the depth channel is used to detect depth edges. The noisy depth data is temporally smoothed using an exponential filter:

$$filteredDepth_t = \alpha \times depth_t + (1 - \alpha) \times filteredDepth_{t-1}$$

where $\alpha = 0.3$ (found by trial and error) is a midpoint between good noise reduction (low α) and reaction to dynamic scenes (high α). Surface normals are computed at each point in the depth image [19] and are used to detect surface normal edges. Finally, the color channel is used to detect color edges. We now describe the pipeline, shown in Figure 3, for the edge extraction in each of these modalities.
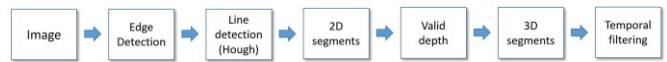
Figure 3: Edge extraction pipeline.

The most dominant lines are extracted from each depth, normals component, or color image by running the Hough Transform [9] on edge points detected by Canny edge detection [6]. A variant of RANSAC [11] is used to extract inlier edge points along each Hough line and to generate a more accurate line equation. Finally each 2D line is divided into segments having a density of edge points greater than some threshold (each line must have at least 35px in our implementation). Special care is given to depth edges, whose location may need to be shifted slightly to lie on the nearby occluding object that originated the edge.

2D line segments, extracted from all modalities, are back projected to 3D space if there are enough valid depth values along their length. RANSAC is used to fit 3D linear segments through the 3D edge points.

Special care is given to ensure the consistency of the recovered constraints over successive frames. We reduce temporal noise in positioning, orientation, and extent, by using a Kalman filter. Extracted segments are associated with existing edges based on proximity and similarity of their orientation. In addition, we require edges to have been seen in at least five frames prior to allowing the edge to be used for snapping; this minimizes the chance of the user relying on false positives from the edge extraction. Once detected, constraints will be maintained and updated unless they are not seen for one second, at which point they are removed; this helps overcome short periods of occlusion of constraints that may occur as a result of the user's motion or the scene object moving. Our edge extraction algorithm runs around 15 Hz in our prototype implementation. See Figure 4 and the supplementary video for typical results.
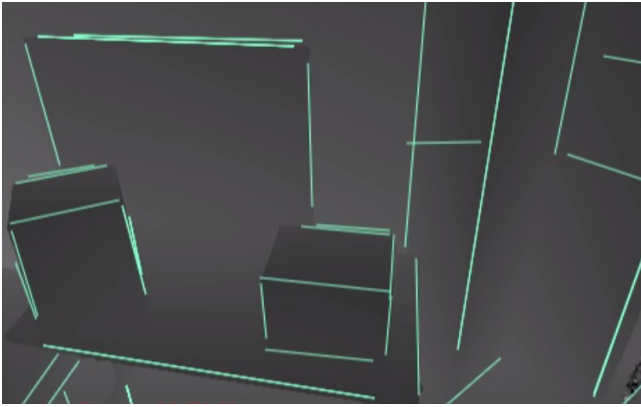
Figure 4: An example of Kalman filtered edges; please see the supplemental video for more examples.

Extracting Planar Surfaces

Planar surfaces are detected in a fashion similar to that of recent plane and scene analysis methods [28]. Depth image normals vote for a plane equation parameterized by its azimuth, elevation, and distance from the origin. A Hough transform on 3D depth points is used to detect major plane equations in the scene.

Next, a greedy strategy is used to associate scene points with those planes. Unassigned 3D points that lie in the vicinity of each candidate plane (up to ~10 cm), with compatible normal directions, are associated with the plane (~10° angle, and may increase to 20° as depth noise grows for farther away planes; see Figure 5). This surface extraction algorithm runs around 4 Hz on our prototype implementation.
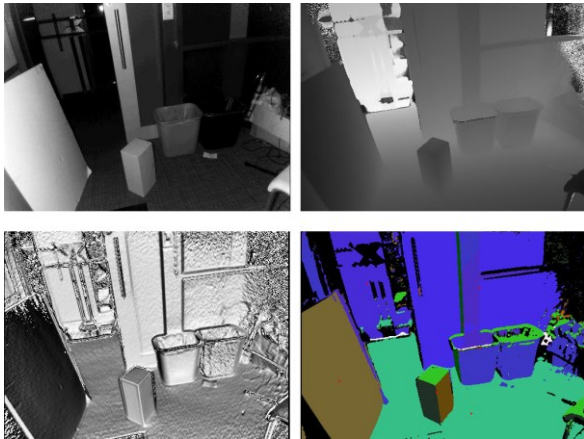

Figure 5: Plane extraction stages: IR image (top left), depth image (top right), shaded mesh indicating surface normals (bottom left), and recovered major planes (bottom right).

SNAP-TO-REALITY TECHNIQUES

We now discuss how the extracted constraints are used in precise alignment and snapping. In our prototype, a virtual object snaps to real edges and planar surfaces that are found to be compatible with the object's own edges and planes. The object's own edges and planes are referred to as "internal constraints" since they originate from the virtual object, while the physical constraints are external to the

---

Algorithm 1

```
SnapToRealConstraint:
  BestRealConstraint ← null
  MinCost ← ∞
  for each Internal {Edge, Planar Surface}
Constraint I:
    for each Real {Edge, Planar Surface}
Constraint R:
      if IsCompatible(I, R):
        cost ← Cost(I, R)
        if cost < MinCost:
          BestRealConstraint ← R
          MinCost ← cost
        endif
      endif
    endfor
  endfor
  return BestRealConstraint

D ← {edge, plane} distance threshold
A ← {edge, normal} angle threshold
IsCompatible(I, R):
  if distance(centroid of I, R) > D
    return false
  if angle_between(I, R) > A
    return false
  return true

Cost(I, R):
  c ← α × distance(centroid of I, R)
  c ← c + β × distance(centroid of I, centroid
of R)
  if R was previously BestRealConstraint
    c ← γ × c
  endif
  return c
```

object. Our approach is designed to support the interactive selection of compatible physical constraints. For example, an AR designer may decide that a virtual toy soldier's internal constraints should simply be its bounding box, thus allowing snapping to physical planes and edges.

We now describe the two SnapToReality alignment techniques implemented in our prototype—*edge snapping* and *planar surface snapping*.

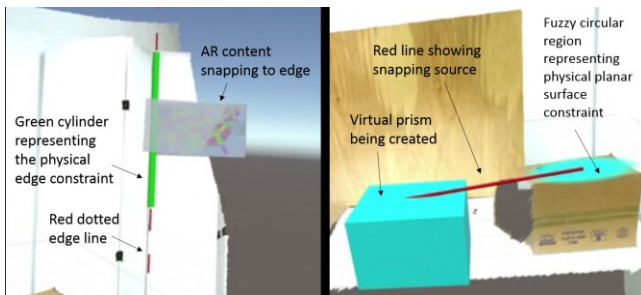### Edge & Planar Surface Snapping

Our edge and plane snapping algorithm is summarized in Algorithm 1 and described as follows. For each internal edge (plane) constraint, we search through all physical edge (plane) constraints for *compatible* edges (planes).

An edge (plane) is compatible if the distance between the internal edge midpoint (plane centroid) and its projection onto the physical edge (plane) is less than some threshold (e.g., 20 cm), and the internal and physical edges are approximately parallel (e.g., within 20° for edges and 60° for planes). We keep the physical edge (plane) with the lowest cost, where the cost is a linear combination of (a) the distance between the internal edge midpoint (plane centroid) and its projection onto the physical edge (plane) and (b) the distance between the internal edge midpoint (plane centroid) and the physical edge midpoint (plane

centroid); in our experiments, $\alpha = 1, \beta = 1$. The second component of this linear combination (b) aids in favoring edges (planes) that are nearby. For temporal consistency, previously selected edges and planes are favored ($\gamma = 0.1$). If a compatible physical edge (plane) is found, we rotate and translate the AR virtual content so that its corresponding internal edge (plane) is parallel to and overlapping the physical edge (plane).

Visualization of Compatible Physical Constraints
To minimize user distraction, compatible edges and planes are visualized only when snapping to them. Compatible edges are visualized via a green cylinder and a dotted red cylinder edge extending outward from the edge (see Figure 6, left). Compatible planar surfaces are visualized via a fading circular region on the surface's centroid and a red line connecting the snapping virtual content to the surface's centroid (see Figure 6). This red line visualization is important for small field of view AR displays, as the user may otherwise not know what the virtual content is snapping to (cf. User's Limited Field of View).

Figure 6: Visualizations for compatible physical edges (left) and planar surfaces (right).

PROTOTYPE SNAP-TO-REALITY IMPLEMENTATION
In our AR system, the user wears an optical see-through head-mounted near-eye display (Lumus DK-32[2] 1280x720) calibrated with a method similar to the single point active alignment method [29]. Figure 7 shows the view through the Lumus display. Our glasses are equipped with 6 retro-reflective spheres that are tracked using a ceiling-mounted motion capture system, Natural Point's OptiTrack Flex 3 system[3] (12 cameras). Our glasses are also augmented with a 9 degrees-of-freedom inertial measurement unit (SparkFun's Razor SEN-10736[4]) which provides a smoother estimate of the unit's orientation. The orientation data from the IMU (~160Hz) is integrated with the orientation and position data from the OptiTrack system (~100Hz) for a relatively smooth, low latency operation. We render at 60Hz, matching the refresh rate of the glasses. Figure 8 shows the hardware components.

---

[2] http://www.lumus-optical.com/
[3] http://www.optitrack.com/products/flex-3/
[4] https://www.sparkfun.com/products/10736

We used a Microsoft Kinect 2.0 sensor to obtain RGB+Depth images in real-time for extracting the real world 3D edges and planar surfaces. Virtual content was created using the Unity 5 game engine and was shown to the user via the Lumus glasses. In our current prototype configuration, the Kinect camera is mounted above the user to observe the physical scene (see Figure 8).
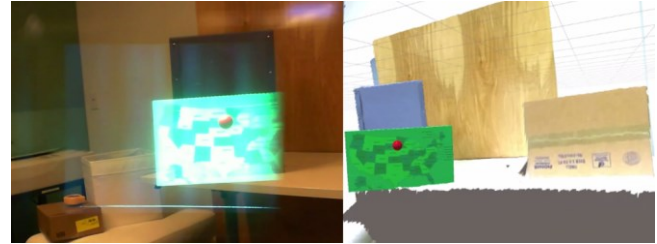
Figure 7: View through Lumus display (left); note that the virtual content does not appear as bright to the human eye. Screenshot of Unity player view (right). The red dot indicates the user's gaze direction.
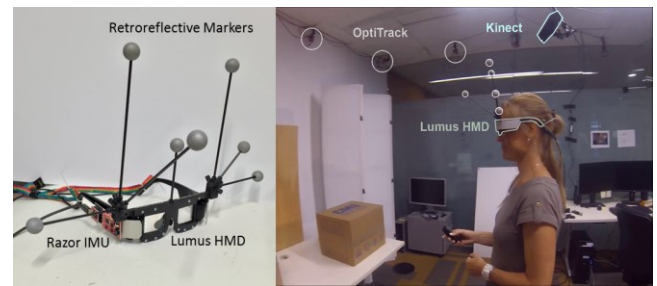
Figure 8: Hardware used in our prototype.

User input was obtained through a Kensington Wireless Presenter remote controller. All processing was done on a single PC with an Intel Xeon E5-1620 processor with 16 GB of RAM and an NVIDIA GeForce GTX 980 GPU.

To correctly render the occlusion of virtual objects by real objects, the depth image acquired by Kinect camera is rendered as a black (invisible in the glasses) mesh. AR virtual objects are rendered offset by about 5cm towards the head-worn glasses to help overcome any noise in the depth image and system calibration.

Demo Applications
To showcase our SnapToReality alignment techniques, we developed two prototype AR scenarios—*content placement* and *content creation*. In each, the user interacts with a virtual object in the following way: A ray is cast from the center of the tracked glasses, and its point of intersection with the scene geometry is regarded as the object or cursor position. The user can then interact with objects that intersect the ray using the buttons on the remote controller.

We considered other interaction methods such as using a ray cast from the handheld remote. After preliminary usability evaluations we settled on head gaze-based interaction, which was simpler to implement as display and interaction share the same location. We leave finding the

most ergonomic interaction method for snapping in AR as future work.

AR Content Placement

A virtual AR app window is positioned in space and is highlighted whenever the user gazes at it (see Figure 7). The user can "pick up" the highlighted window by pressing a button on the remote controller. The window can then be positioned, oriented, and scaled by the user's head gaze direction, until the user drops the object by pressing the same button again. While picked up, the AR app window is positioned at the intersection point of the user's head direction and the geometry, and oriented towards the user's head gaze. To scale an app window, the user drops the window in place, and then gazes at a corner of the window. Pressing a button toggles scaling mode, resizing from that corner while the opposite corner is fixed. Scale snapping works by finding the lowest cost compatible external constraint for each internal constraint used in scaling (using Algorithm 1).
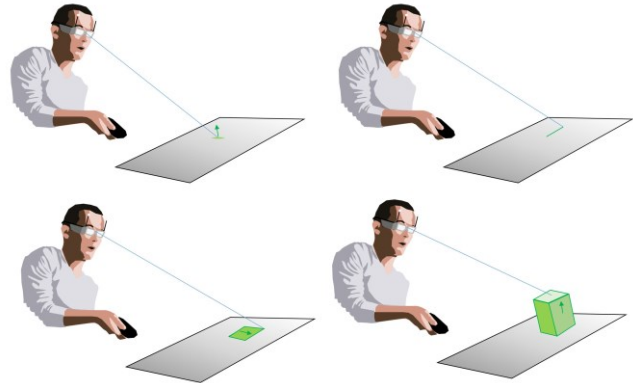
AR Content Creation

This demo application is inspired by SketchUp's line, rectangle, and push/pull drawing tools[5]. As the user gazes at a planar surface, the detected surface is visualized by a fuzzy circle and a surface normal (see Figure 9, top left). Pressing the remote's right button begins drawing a 3D edge on the plane starting at the intersection point of the user's gaze direction and the plane. The edge's end point is controlled by the user's gaze direction (see Figure 9, top right) and can snap to nearby physical edges. Furthermore, the edge can snap to be made parallel or perpendicular to nearby edges. Pressing the right button a second time fixes the edge's end point and begins extending a rectangular area away from the fixed edge. Again, the user's current gaze direction determines the size of the rectangular plane, and a third right button press fixes the plane (see Figure 9, bottom left). Now a 3D box can be extruded upwards away from the plane. The height of the 3D box is determined by finding the closest intersection between the user's head direction and the ray in direction of the 3D box extrusion starting from the 3D box's center. This height can snap to compatible planar surfaces and also to nearby edge endpoints (see Figure 6, right). A final right button press finishes the box creation process (see Figure 9, bottom right). In each step of the process, the left button of the remote performs an 'undo' operation. Figure 9 illustrates this entire process.

Users can optionally scale a face of a completed box by directing their gaze toward the face and pressing the right button. In a similar fashion, the height of the 3D prism with respect to that face is determined by the closest intersection between the user's gaze direction and the ray in direction of the extrusion, starting from the 3D box's center.

---

[5] http://www.sketchup.com/learn

Prototype Limitations

One limitation of our prototype is that the Kinect camera is statically mounted above the user. However, we envision similar interactions would be possible using a depth camera worn by the user or mounted on the glasses.

Another limitation is that the accuracy of the Kinect depth map is limited, contributing a significant source of noise. More sophisticated surface reconstruction methods such as Kinect Fusion [25] or DynamicFusion [24] might help.

Figure 9: Flow of AR content creation. Top left: The user's head direction defines a cursor on a physical surface. Top right: Clicking a button starts drawing a line on the surface. Bottom left: Clicking again extrudes the drawn line to a rectangle. Bottom right: Extruding the rectangle to a 3D box.

DESIGN CONSIDERATIONS FOR SNAP-TO-REALITY

In this section, we discuss practical design considerations, limitations, and insights gained while developing SnapToReality techniques. Because these observations are not limited to our current prototype, we hope that this discussion will help developers and researchers extend and apply the SnapToReality concept. We note that in addition to what we discuss here, traditional snapping design issues (not specific to SnapToReality) are also important when designing any snapping system (e.g., snapping hierarchies, the amount of constraints to show to the user, etc.).

The main considerations and challenges in enabling SnapToReality systems are: limited field of view, noise in constraint extraction, changing the view in AR, issues concerning dynamic scenes, and visualization of physical constraints. We now discuss each in detail.

Limited Field of View

User's Limited Field of View

Today's AR glasses typically have a very limited field of view (e.g., the Lumus glasses have about a 40° diagonal FOV). Thus, many constraints that an object may snap to are outside the user's field of view. In addition, even if AR glasses had an unlimited field of view, a particular constraint could still be behind the user, out of sight. There are at least two ways to address this problem.

First, the system can visualize available constraints that are outside the field of view. This is the approach taken in our prototype. When virtual content begins to snap to a compatible constraint, we visualize those constraints in case they are out of sight: for edges, by extending them with red dotted edges; for planes, by drawing a red edge from the virtual content to the plane's centroid (see Figure 6).

Another possible way to overcome this problem is to let the user actively preselect a constraint by bringing it into the field of view and then returning the view to manipulate the virtual content. For example, the user may gaze directly at a constraint and then press a button to select it. A positive consequence of this approach is that the user's current field of view can aid in the physical constraints extraction process. For example, the system may simply follow the user's gaze to limit its extraction there.

Camera's Limited Field of View

The camera's limited field of view also means that extracting many physical constraints is challenging. If the user wants to snap virtual content to many different constraints, the camera must first be able to see all those constraints. Thus, unless snapping is limited to the camera's field of view, a global analysis of the scene is required.

Noise in Constraint Extraction

In contrast to typical virtual scenes, the complexity of real world geometry, texture, and sensor noise may lead to a very dense and noisy set of potential constraints. As with any snapping technique, high density of constraints will negatively affect the utility and usability of snapping.

Active selection of constraints by the user is again one way to overcome this problem. By putting the user "in the loop" of the constraint extraction process, the effects of sensor noise can be diminished. The user can either select the constraint directly or explicitly discard it.

We can also substantially minimize the effects of sensor noise by limiting the types of compatible constraints we extract. This can be achieved by adjusting the virtual content's snapping abilities (called "internal constraints" in our prototype). For example, a cube has 6 planar surfaces; each of the cube's planar surfaces may only snap to physical planar surfaces, not corners or edges. Therefore, under this assumption, we can limit the types of physical constraints to extract to those that are compatible with the virtual content. Used by our prototype, this approach mitigates problems with sensor noise.

Changing the View in AR

In AR, users see virtual objects interacting with real objects as if they were life size. Users may move themselves closer to the real and virtual objects to inspect them more closely, or move to encourage the extraction of a physical constraint. However, users are limited by the physics of the real world in how they may change their view. This limits the ability to see more detail, perform more precise manipulation, zoom out to see an overview, or look behind

objects that are occluded. In addition, physically moving around the environment may introduce user fatigue. Future SnapToReality systems need to account for these considerations, including snapping at far distances and helping the user know about available snapping constraints outside the field of view (cf. User's Limited Field of View).

Interacting with Dynamic Constraints

When constraints are obtained in real-time from the dynamic physical world, new types of snapping interaction are possible. Physical constraints can be altered on the fly to enable virtual content to be translated, rotated, or scaled more appropriately. For example, virtual content can be scaled to different physical extents by simply moving physical objects in the scene. The possibilities of actively manipulating reality to alter virtual content in precise ways via snapping could be explored further.

One interesting question that arises with dynamic constraints is how to deal with a constraint that is currently being used, but then suddenly disappears in the real world during the interaction (e.g., a real object moves away). In our prototype, we chose to simply keep the constraint alive until the end of the interaction. However, the constraint does not move with the real object as it moves away. Such capability may be desired and we leave it as future work.

Visualizing Physical Constraints in AR

Choosing how to visualize extracted physical constraints is another important matter of design. Should the system visualize all available constraints, none, or only those currently being snapping to by virtual content?

In some simple cases it may be obvious which physical object a virtual object is snapping to without rendering the available constraints. In others, it may be necessary to render all available constraints. Visualizations are particularly important when snapping to physical constraints whose source is far away or outside the user's field of view (e.g., snapping virtual content to be aligned with the top of a table that is behind the user). In this case, clear guides should let the user be aware of the source. This is especially important when using today's available AR displays which have a narrow field of view. Figure 6 shows how our prototype visualizes edge constraints and planar surface constraints, drawing inspiration from common snapping applications using dotted linear edges to represent linear constraints.

USER EVALUATIONS

We conducted three preliminary evaluations of various aspects of our prototype.

### Evaluation of Constraint Extraction Algorithms

To evaluate our constraint extraction algorithms, we recorded eleven scenes showing a large range of physical environments. We asked four participants (all male, 24 to 33 years old, avg. 28 years old) to indicate up to ten edges and ten surfaces they would envision aligning AR content

to for each of the scenes. Participants indicated this by drawing on the color image of each scene.

We compared the edges and planar surfaces drawn by users to those extracted from our algorithm by aggregating the responses of the four participants and visually comparing them to the extracted constraints (see Figure 10 and the supplementary video). Our algorithm was able to detect 51.95% of user drawn edges (min. 30.43%, max 77.78% across all scenes) and 60% of user drawn planar surfaces (min. 41.67%, max 81.82% across all scenes). We note that for each scene, we did not modify any of the edge or plane detection algorithm parameters. Had we done so for each scene, we believe the percentage of detected user drawn constraints would only increase.
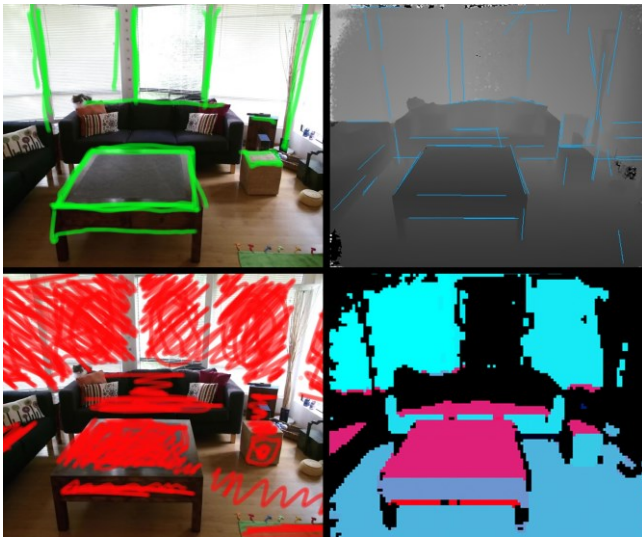
Figure 10: Living room scene overlaid with user drawn edges (top left) and planar surfaces (bottom left) compared to the edges and planar surfaces detected by our algorithm (top right and bottom right, respectively).

Evaluation of SnapToReality Techniques

We invited 8 different participants (all male, 23 to 49 years old, avg. 30 years old) to evaluate our SnapToReality techniques. Participants received $10 for their participation and each session lasted approximately one hour. The physical scene used for interaction included a table, several boxes, a wooden board, a white panel folding screen, and a wall in the background (see Figure 11).

Participants filled out pre-study and post-study questionnaires and separately evaluated both AR Content Placement and AR Content Creation demos. They were given a few minutes to familiarize themselves with the interface for each demo.

Evaluation of AR Content Placement

To evaluate the potential benefits of snapping, we asked the participants to place and size a box representing a mock AR application window at various locations around the room. After verbally confirming that the person understood the interface, they were given a set of five training trials. Next,

each participant completed twenty trials, each composed of a positioning and orienting stage followed by a resizing stage. The position, orientation, and size for each trial were chosen beforehand to align with various physical objects, and the order of the trials was chosen randomly. Ten trials were completed with snapping and ten trials without snapping. The order of snapping and no-snapping conditions was counterbalanced over the eight participants. Interaction in the no-snapping condition was identical to that of the snapping condition except that snapping was disabled (physical constraints were extracted but not used).

Figure 11: View from the Kinect camera for the user evaluation of SnapToReality techniques.

In the positioning and orienting stage, participants were shown an AR app window placed randomly near a target window, which at first was invisible. They then picked up the window by gazing at it and pressing the top button on the wireless remote. At this point, a timer began and the target window location was now shown as a white transparent box. Participants were asked to place the AR app window at the target location as quickly as possible, and end the trial by pressing the top button again. If the window was placed too far away from the target (distance between the center points of the windows greater than 10 cm) or not close enough in orientation (angle between the surface normals of the windows greater than 10°), text appeared notifying the user that the window is not close enough to the target. Participants then had to pick up the window and try again. Once the window was positioned and oriented closely enough to the target, the timer stopped and the user saw the text, "Correct!"

Next, during the resizing stage, a target transparent box appeared, showing the desired dimensions of the window and a second timer began. Participants then had to scale the window to match the target (measured in terms of both position and scale, within 10 cm). To make this second stage similar between snapping and no-snapping conditions, we adjusted the starting position and orientation of the window to be the same for all trials.

In all trials, a one minute timeout was used and treated as outliers in the data, removed before analysis. With snapping enabled, participants took an average of 4.45s (median 2.74s, std. dev. 5.34) to position and orient AR app

windows correctly. With snapping disabled, participants took an average of 11.36s (median 9.89s, std. dev. 6.75). A paired t-test indicated a statistical significance ($p\text{-value} \ll 0.001$). For scaling, participants took an average of 8.18s (median 7.89s, std. dev. 3.97) with snapping and 8.71s (7.43s, std. dev. 5.12) without snapping. A paired t-test did not indicate any statistical significance.

In the post-study questionnaire, participants were asked to compare their experience with and without snapping. Here we list some of their feedback:

"With snapping, I trusted the system to take me there and it allowed me to be a bit careless and [speed] things up. I would just go to the nearest corner and trust the system to pick the best spot up for me."

"Snapping objects [...] was a more subtle interaction. Small head movements were enough to get the job done. Whereas without snaps I had to use my whole body to get the object where I wanted."

Evaluation of AR Content Creation

In the second part of the study, we sought to assess qualitatively how well participants can create simple virtual objects in AR. After explaining the interface, we allowed participants to play with this mode freely as long they wished. For this task, snapping was always enabled.

Participants created various structures of 3D geometry using the simple prism creation tool. Figure 12 shows an example of such a structure created by a participant. Participants were made aware of the ability to place virtual content aligned to dynamic elements in the scene, such as boxes that had been moved into place, and a measuring tape used to extrude prisms to certain heights. Please see the supplemental video for more examples.
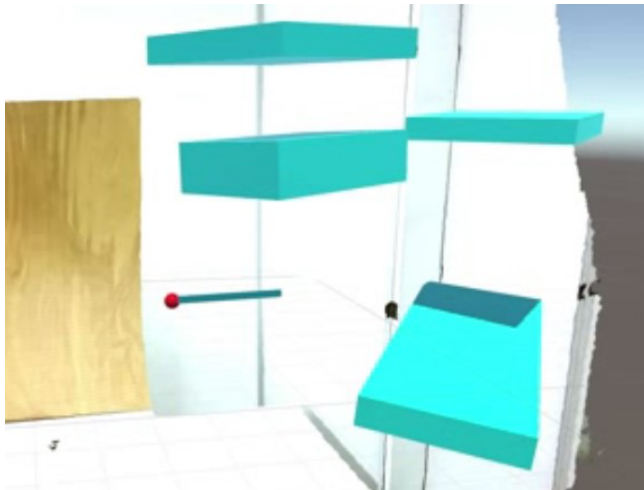
Figure 12: Example of virtual content created by a participant using AR Content Creation. Here, physical walls (planar surfaces) were used as constraints to create "virtual shelves."
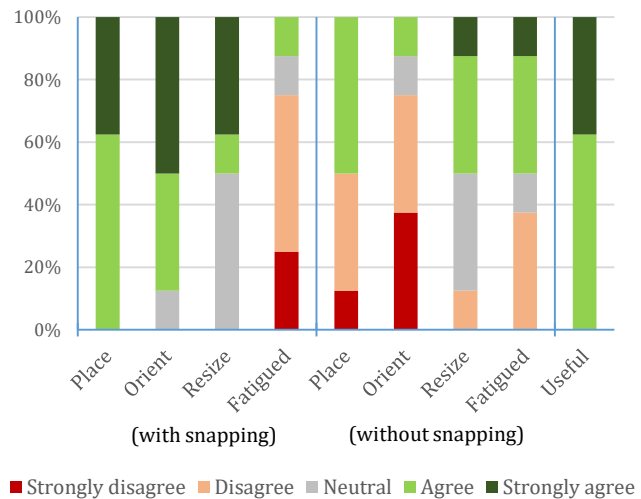
In the post-study questionnaire, participants were asked to describe their approach of creating 3D boxes with snapping. Here we list some of their feedback:

"I started by positioning the cursor near the edge of another object, like the table or box, and then moved to another edge to set the area. Then setting the volume was fairly easy once the other two were in place."

"I started by choosing the 'right plane'. I moved until I saw the angle of the plane I wanted. Then I drew one face on that plane, not caring too much about the snapping. Finally, I adjusted the height, mostly trying to match the height of the surrounding objects using snapping. I resized it sometimes to match different heights."

Additional Questionnaire Results

In the post-study questionnaire, participants were asked to rate several statements using a Likert scale. The statements were as follows: "With{out} snapping, I can easily {place, orient, resize} objects {where, how} I want," "While interacting with snapping {enabled, disabled}, I felt fatigued," and "Snapping in augmented reality is useful." Responses are shown in Figure 13. The Wilcoxon signed-rank test found that for the placing, orienting, and feeling fatigued statements, participants rated snapping significantly better than without snapping at a 0.05 level of significance. As with the timing results, no statistical significance was found comparing resizing with and without snapping. Finally, all participants agreed or strongly agreed that snapping in AR is useful.

Figure 13: Chart visualizing post-study questionnaire responses (best viewed in color).

We also asked participants several open-ended questions. Some representative quotes with their corresponding questions are as follows:

Q: When did you find snapping most helpful? Least helpful?
"Snapping was most helpful when moving something quickly across the scene to a general target area. Movements requiring more finesse or accuracy on behalf of

the user are more frustrating when snaps suddenly make drastic shift."

"When snapping worked it was very helpful because I didn't have to move very much to achieve success…"

(Referring to the Likert scale statements): Please briefly describe why you chose the ratings in the previous question.
"Placing objects is quite easy even without snapping. Certainly snapping helped a bit, just did not affect the overall experience too much. Orienting and scaling things, on the other hand, when done imprecisely, can often be quite obvious and seem bizarre, so snapping helps a lot more in these cases. [...] snapping reduces the time needed, hence less fatigue."

"Overall snapping is a useful feature. However, just as with other design programs like Photoshop, it can become obtrusive in certain situations. Easily being able to toggle between snapping and non-snapping behavior would be ideal. In this specific study, it seemed like when the tracking found the surfaces correctly and the world locking was working that snapping was fairly helpful. When either of those two breakdown, snapping becomes more of a hassle."

Q: What snapping features would you like to have for augmented reality?
"It would be great if I could put my hand or finger in the field of view and point where I want the object to snap to, or where I want the edge to scale to. Manual pointing/direction would be nice. I envision it as sort of a physical mouse pointer for the virtual objects."

"Snap to planes and edges of various virtual and physical objects. It'd be nice to also be able to filter what you snap to (kind of like AutoCAD's [...]) at any given time."

Discussion

In general, we observed that participants physically walked around the room much more without snapping than with snapping. This may suggest that 3D snapping is even more important in AR than in traditional 2D or 3D desktop environments where all objects are virtual. In purely virtual environments, "magical" navigation techniques (e.g., flying and teleporting) enable quick and easy viewpoint positioning for easier interaction with virtual content. In AR, however, users are limited to physical navigation and thus are not always able to quickly and easily assume different viewpoints.

The lack of a statistically significant difference in scaling times in our evaluation of AR Content Placement is most likely due to the fact that only two degrees of freedom were being manipulated, and that we did not allow snapping to position content arbitrarily within the snapping threshold (by deactivating snapping or by automatic methods [1,10]). Adding such features and also comparing scaling in three dimensions may change results for future experiments.

In addition to comments that may apply to any snapping system, participants' open-ended responses also verified certain unique points about snapping in AR. For example, one participant noted needing to move their whole body to achieve object positioning when not using snapping; while with snapping, he only needed small head movements (see comment in Evaluation of AR Content Placement).

FUTURE WORK

Selecting Constraints via Other Modalities
We imagine many other ways in which users can actively select constraints. For example, users should be able to physically touch a constraint to select it. Natural language processing along with computer vision scene understanding is also an interesting avenue for future work (e.g., "place the app parallel to and to the right of the door").

Using Snapping for Physical Objects
An interesting idea that has previously been proposed by Forster and Tozzi [12] but still remains an open area of research is the use of snapping in AR to align *physical* objects to *physical* constraints. An application for this might be interior design.

Semantic Snapping
Usually when we refer to snapping, we imply a geometric relationship of snapping. This may include geometric proximity in translation or rotation; objects being perpendicular or parallel to one another; aligning to dominant directions/axes (e.g., gravity); aligning to mid-points; etc. However, using *semantic* relationships is another possible way to achieve snapping. For example, an AR app related to cooking may favor snapping to relevant physical areas of a kitchen (cf. Figure 2). This, however, would require a detailed semantic understanding of the physical scene. In our SnapToReality prototype, we chose to focus on simple geometric relationships.

CONCLUSION
SnapToReality alignment techniques allow users to position, rotate, and scale virtual content to dynamic, real world scenes. Our prototype system extracts physical linear edges and planar surfaces in real-time and allows users to manipulate and create AR content precisely aligned to the real world. We contribute a set of design considerations and insights for future designers of AR snapping systems. In particular, we note the issues of a small field of view in AR, sensor noise in the constraint extraction process, changing the view in AR being limited to physical movements (thus making SnapToReality an important interaction paradigm to quickly achieve precise positioning), interacting with dynamic constraints, and considerations on how to visualize physical constraints. Finally, our user evaluations show that aligning virtual content to physical constraints in our prototype system is significantly faster with snapping than without snapping. Snapping in AR was also shown to enable novel forms of AR content creation interaction.

REFERENCES

1. Patrick Baudisch, Edward Cutrell, Ken Hinckley, and Adam Eversole. 2005. Snap-and-go: helping users align objects without the modality of traditional snapping. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '05), 301-310. http://doi.org/10.1145/1054972.1055014

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), 101-110. http://doi.org/10.1145/502348.502363

3. Eric A. Bier and Maureen C. Stone. 1986. Snap-dragging. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques* (SIGGRAPH '86), 233-240. http://dx.doi.org/10.1145/15922.15912

4. Eric A. Bier. 1990. Snap-dragging in three dimensions. In *Proceedings of the 1990 Symposium on Interactive 3D Graphics* (I3D '90), 193-204. http://doi.org/10.1145/91385.91446

5. Dorit Borrmann, Jan Elseberg, Kai Lingemann, and Andreas Nüchter. 2011. The 3D Hough Transform for plane detection in point clouds: A review and a new accumulator design. *3D Research*. 2, 2, Article 32 (June 2011): 1-13. http://dx.doi.org/10.1007/3DRes.02(2011)3

6. John Canny. 1986. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (PAMI). 8, 6 (Nov. 1986): 679-698. http://doi.org/10.1109/TPAMI.1986.4767851

7. Changhyun Choi, Alexander J. B. Trevor, and Henrik I. Christensen. 2013. RGB-D Edge Detection and Edge-based Registration. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (IROS '13), 1568-1575. http://doi.org/10.1109/IROS.2013.6696558

8. Richard O. Duda and Peter E. Hart. 1972. Use of the Hough transformation to detect lines and curves in pictures. *Communications of the ACM*. 15, 1 (Jan. 1972): 11-15. http://doi.org/10.1145/361237.361242

9. Leandro A. F. Fernandes and Manuel M. Oliveira. 2008. Real-time line detection through an improved Hough transform voting scheme. *Pattern Recognition*. 41, 1 (Jan. 2008): 299-314. http://dx.doi.org/10.1016/j.patcog.2007.04.003

10. Jennifer Fernquist, Garth Shoemaker, and Kellogg S. Booth. 2011. "Oh Snap" – Helping Users Align Digital Objects on Touch Interfaces. In *Proceedings of the 13th IFIP TC 13 International Conference on Human-Computer Interaction* (INTERACT 2011), 338-355. http://dx.doi.org/10.1007/978-3-642-23765-2_24

11. Martin A. Fischler and Robert C. Bolles. 1981. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*. 24, 6 (June 1981): 381-395. http://doi.org/10.1145/358669.358692

12. Carlos H. Q. Forster and Clésio L. Tozzi. 2001. An architecture based on constraints for augmented shared workspaces. In *Proceedings of the 14th Brazilian Symposium on Computer Graphics and Image Processing* (SIBGRAPI '01), 328-335. http://doi.org/10.1109/SIBGRAPI.2001.963073

13. Mathias Frisch, Ricardo Langner, and Raimund Dachselt. 2011. Neat: a set of flexible tools and gestures for layout tasks on interactive displays. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces* (ITS '11), 1-10. http://doi.org/10.1145/2076354.2076356

14. Ran Gal, Lior Shapira, Eyal Ofek, and Pushmeet Kohli. 2014. FLARE: Fast layout for augmented reality applications. In *Proceedings of the 13th IEEE International Symposium on Mixed and Augmented Reality* (ISMAR '14), 207-212. http://doi.org/10.1109/ISMAR.2014.6948429

15. Natasha Gelfand and Leonidas J. Guibas. 2004. Shape segmentation using local slippage analysis. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing* (SGP '04), 214-223. http://doi.org/10.1145/1057432.1057461

16. Michael Gleicher. 1995. Image snapping. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques* (SIGGRAPH '95), 183-190. http://doi.org/10.1145/218380.218441

17. Seongkook Heo, Yong-Ki Lee, Jiho Yeom, and Geehyuk Lee. 2012. Design of a shape dependent snapping algorithm. In *CHI '12 Extended Abstracts on Human Factors in Computing Systems* (CHI EA '12), 2207-2212. http://doi.org/10.1145/2212776.2223777

18. Paul V. C. Hough. 1962. Method and means for recognizing complex patterns. (Dec. 1962). *US Patent 3069654 A,* Filed Mar. 25, 1960, Issued Dec. 18, 1962.

19. Klaas Klasing, Daniel Althoff, Dirk Wollherr, and Martin Buss. 2009. Comparison of surface normal estimation methods for range sensing applications. In *Proceedings of the 26th IEEE International Conference on Robotics and Automation* (ICRA '09), 3206-3211. http://doi.org/10.1109/ROBOT.2009.5152493

20. Gun A. Lee, Ungyeon Yang, Yongwan Kim, Dongsik Jo, and Ki-Hong Kim. Snap-to-feature interface for annotation in mobile augmented reality. In *Augmented Reality Super Models Workshop at the 9th IEEE International Symposium on Mixed and Augmented Reality* (ISMAR '10), Retrieved September 14, 2015

from
http://www.icg.tugraz.at/Members/arth/arsupermodels/04_lee.pdf

21. Gun A. Lee and Mark Billinghurst. 2012. Assistive techniques for precise touch interaction in handheld augmented reality environments. In *Proceedings of the 11th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and its Applications in Industry* (VRCAI '12), 279-285. http://doi.org/10.1145/2407516.2407582

22. Paul Merrell, Eric Schkufza, Zeyang Li, Maneesh Agrawala, and Vladlen Koltun. 2011. Interactive furniture layout using interior design guidelines. *ACM Transactions on Graphics* (TOG). 30, 4, Article 87 (July 2011): 1-10. http://dx.doi.org/10.1145/2010324.1964982

23. Eric N. Mortensen and William A. Barrett. 1998. Interactive segmentation with intelligent scissors. *Graphical Models and Image Processing*. 60, 5 (Sept. 1998): 349-384. http://dx.doi.org/10.1006/gmip.1998.0480

24. Richard A. Newcombe, Dieter Fox, and Steven M. Seitz. 2015. DynamicFusion: Reconstruction and tracking of non-rigid scenes in real-time. In *Proceedings of the 28th IEEE Conference on Computer Vision and Pattern Recognition* (CVPR '15), 343-352. http://dx.doi.org/10.1109/CVPR.2015.7298631

25. Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. 2011. KinectFusion: Real-time dense surface mapping and tracking. In *Proceedings of the 10th IEEE International Symposium on Mixed and Augmented Reality* (ISMAR '11), 127-136. http://doi.org/10.1109/ISMAR.2011.6092378

26. Rui Nóbrega and Nuno Correia. 2012. Magnetic augmented reality. In *Proceedings of the International Working Conference on Advanced Visual Interfaces* (AVI '12), 332-335. http://doi.org/10.1145/2254556.2254620

27. Ji-Young Oh, Wolfgang Stuerzlinger, and John Danahy. 2006. SESAME: towards better 3D conceptual design systems. In *Proceedings of the 6th Conference on Designing Interactive Systems* (DIS '06), 80-89. http://doi.org/10.1145/1142405.1142419

28. Nathan Silberman, Lior Shapira, Ran Gal, and Pushmeet Kohli. 2014. A contour completion model for augmenting surface reconstructions. In *Proceedings of the 14th European Conference on Computer Vision* (ECCV '14), 488-503. http://dx.doi.org/10.1007/978-3-319-10578-9_32

29. Mihran Tuceryan and Nassir Navab. 2000. Single point active alignment method (SPAAM) for optical see-through HMD calibration for AR. In *Proceedings of the IEEE and ACM International Symposium on Augmented Reality* (ISAR '00), 149-158. http://doi.org/10.1109/ISAR.2000.880938

30. Ken Xu, James Stewart, and Eugene Fiume. 2002. Constraint-based Automatic Placement for Scene Composition. In *Proceedings of the Graphics Interface Conference* (GI '02), 25-34. Retrieved September 14, 2015 from http://graphicsinterface.org/wp-content/uploads/gi2002-4.pdf

31. Pengfei Xu, Hongbo Fu, Chiew-Lan Tai, and Takeo Igarashi. 2015. GACA: Group-Aware Command-based Arrangement of Graphic Elements. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (CHI '15), 2787-2795. http://doi.org/10.1145/2702123.2702198