

GesturAR: An Authoring System for Creating Freehand Interactive Augmented Reality Applications

Tianyi Wang*

School of Mechanical Engineering,
Purdue University
West Lafayette, Indiana, USA
wang3259@purdue.edu

Xun Qian*

School of Mechanical Engineering,
Purdue University
West Lafayette, Indiana, USA
qian85@purdue.edu

Fengming He

School of Electrical & Computer
Engineering, Purdue University
West Lafayette, Indiana, USA
he418@purdue.edu

Xiyun Hu

School of Mechanical Engineering,
Purdue University
West Lafayette, Indiana, USA
hu690@purdue.edu

Yuanzhi Cao

School of Mechanical Engineering,
Purdue University
West Lafayette, Indiana, USA
cao158@purdue.edu

Karthik Ramani

School of Mechanical Engineering,
Purdue University
West Lafayette, Indiana, USA
ramani@purdue.edu

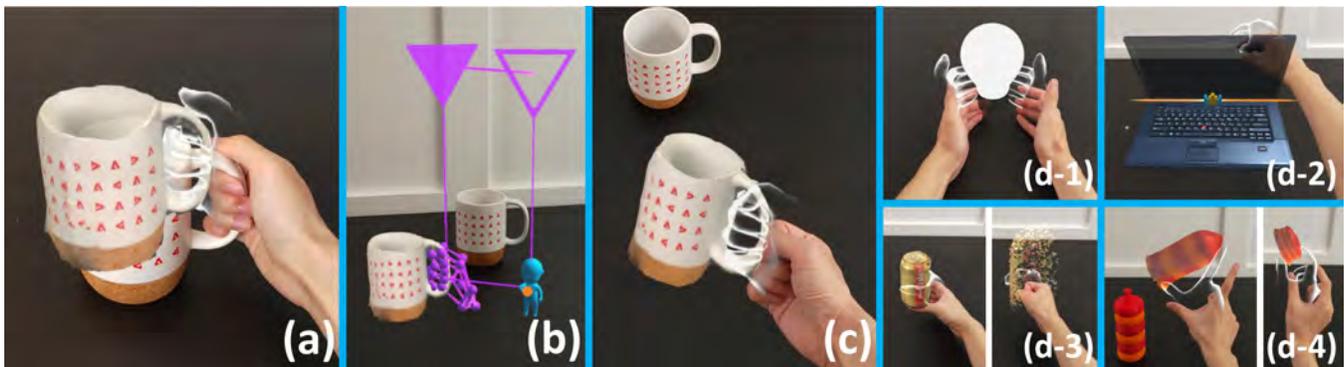


Figure 1: An overview of GesturAR system workflow. (a) A user demonstrates a grabbing gesture next to a virtual cup to define a *trigger* for a freehand AR application. (b) The user then selects the *manipulating action* and connects it with the hand gesture. (c) During testing, the user grabs the virtual cup with the pre-defined gesture. (d) The four freehand interaction scenarios supported by GesturAR: (d-1) A light bulb is lit after the user performs a *static* gesture. (d-2) The user opens the lid of a virtual laptop through a *static* gesture. (d-3) A virtual soda can is broken after a clenching *dynamic* gesture. (d-4) The length of a toy spring changes synchronously with the user's *dynamic* gesture.

ABSTRACT

Freehand gesture is an essential input modality for modern Augmented Reality (AR) user experiences. However, developing AR applications with customized hand interactions remains a challenge for end-users. Therefore, we propose GesturAR, an end-to-end authoring tool that supports users to create in-situ freehand AR applications through embodied demonstration and visual programming. During authoring, users can intuitively demonstrate the customized gesture inputs while referring to the spatial and temporal context. Based on the taxonomy of gestures in AR, we proposed a hand

interaction model which maps the gesture inputs to the reactions of the AR contents. Thus, users can author comprehensive freehand applications using trigger-action visual programming and instantly experience the results in AR. Further, we demonstrate multiple application scenarios enabled by GesturAR, such as interactive virtual objects, robots, and avatars, room-level interactive AR spaces, embodied AR presentations, etc. Finally, we evaluate the performance and usability of GesturAR through a user study.

KEYWORDS

Freehand interactions, immersive authoring, Augmented Reality, embodied demonstration

*Both authors contributed equally to this research.



This work is licensed under a Creative Commons Attribution International 4.0 License.

UIST '21, October 10–14, 2021, Virtual Event, USA
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8635-7/21/10.
<https://doi.org/10.1145/3472749.3474769>

ACM Reference Format:

Tianyi Wang, Xun Qian, Fengming He, Xiyun Hu, Yuanzhi Cao, and Karthik Ramani. 2021. GesturAR: An Authoring System for Creating Freehand Interactive Augmented Reality Applications. In *The 34th Annual ACM Symposium on User Interface Software and Technology (UIST '21)*, October 10–14, 2021, Virtual Event, USA. ACM, New York, NY, USA, 16 pages. <https://doi.org/10.1145/3472749.3474769>

1 INTRODUCTION

Augmented Reality (AR) technology has been broadly adopted in a variety of areas including manufacturing [67, 69], design [71, 101], education [24, 89] and entertainment [55, 72]. Interacting with virtual content plays an essential role in most of these AR experiences. As a dominant approach for manipulating real-world objects, hand gesture has been well-accepted to be an intuitive method for interacting with virtual AR contents as well [12, 73], especially while using hands-free AR head-mounted devices (AR-HMD) [31, 104]. Leveraging the recent advances in hand tracking techniques [18, 62, 86], researchers have facilitated natural gestures such as touching [10, 79], grasping [23, 83, 85], and holding [30] virtual objects without external tracking devices. Such freehand interactions greatly improve the immersiveness of interactions within the AR experiences.

Most of the prior works [12, 23, 31, 83, 85] focus on pre-defined interactions for the virtual manipulation, which are unable to cover the complexity and diversity of the hand-based interactions used in our everyday life [75]. Interacting with different objects usually requires specific hand gestures. For example, VirtualGrasp [96] has shown that different gesture preferences have been found when grasping books, cups, and pens in a virtual environment. Furthermore, one object may have various reactions when encountered with different gestures. For instance, a virtual soda can can be held tightly with one or two hands, placed on the palm, or squeezed. Consequently, it is difficult for an AR application to include all the hand-object interactions in advance. On the other hand, end-users have more in-depth knowledge about their activities and gesture preferences [19]. As a result, we are highly motivated to empower end-users to author personalized hand-related interactions.

Freehand AR application typically detects hand gesture inputs in real-time to invoke the corresponding responses of the virtual contents. However, building algorithms to recognize a certain gesture requires professional expertise. On the other hand, embodied demonstration provides an intuitive way to create gesture-enabled virtual contents [38]. Via the demonstration of a few examples, users can build customized gesture detection applications [8, 27, 53, 54] without looking into the low-level details. This way, even non-expert AR consumers can design the freehand interactions according to their personal preference and specific surrounding contexts.

Popular programming-based authoring tools (Unity3D [87], Unreal [88], ARCore [4], ARKit [5], etc.) have a steep learning curve and are therefore cumbersome for non-professional users to create AR applications [66]. In contrast, the immersive experience supported by AR-HMD fosters the evolution of authoring workflows in an in-situ and ad-hoc fashion [41, 92]. The WYSIWYG (what you see is what you get) metaphor enables users to directly build 3D models [33] and create animations [7, 98] by manipulating virtual objects. Further, users are able to create interactive contents with in-situ visual programming interfaces [29, 45, 105]. Thus, we propose to embrace the embodied demonstration of hand gestures into an immersive AR authoring environment. To this end, users can view, manipulate and edit the demonstrated gestures as elements of the visual programming interface and link them with virtual content behaviors to create customized gesture-enabled AR experiences.

We propose GesturAR, an end-to-end authoring system that supports the real-time creation of AR applications with freehand inputs. We build our authoring environment on a pair of optical see-through AR glasses with a built-in hand tracking module [31]. GesturAR allows intuitive authoring of customized freehand inputs through embodied demonstration while using the surrounding environment as contextual reference (Figure 1a). Users then complete the freehand interactions by matching the hand gestures with reactions of virtual contents using a visual programming interface (Figure 1b), which is designed based on previous elicitation studies [75, 95]. Thus, users can create freehand interactions through simple trigger-action programming logic (Figure 1d). Further, with the support of a real-time hand gesture detection algorithm, users can instantly explore the authored AR experience (Figure 1c).

Following is a list of our contributions:

- A comprehensive in-situ authoring workflow for end-users to create and perform customized freehand interactions through embodied demonstration.
- A freehand interaction model that spatially and temporally maps the hand inputs to responding behaviors of the virtual contents based on a real-time hand gesture detection algorithm.
- An AR interface for generating virtual assets, demonstrating hand gestures, and creating freehand AR applications through visual programming.
- A wide range of example applications demonstrating the potentials of the proposed authoring system. And the user studies for evaluating the system usability.

2 RELATED WORKS

2.1 Freehand interactions in AR

Freehand interactions (or barehand interactions) [90] have long been proposed in the area of HCI as a natural interaction method that relieves users from external tracking devices. Some works used freehand interactions to remotely "pan and zoom" 2D contents on big screens or wall displays [11, 25, 63, 80]. Meanwhile, a similar metaphor was applied to 3D contents in VR/AR environments. FingARTips [12] allows users to pinch and move virtual contents by detecting fiducial markers attached to fingertips. With the development of RGB-D and stereo cameras, 3D geometry of the hands was retrieved for hand-virtual object collision detection [10, 30]. Researchers have also explored manipulating virtual contents by detecting key points on bare hands. Simple gestures such as pinching [32, 59, 69], pointing [48], palming [40] and grabbing [99] can be recognized and mapped to the selection, translation, rotation or scaling of virtual contents. Furthermore, using portable devices, i.e. Leapmotion [86], researchers can achieve full-hand skeleton and perform object manipulation with higher accuracy [79, 85]. However, most previous works map specified hand gestures to limited operations, i.e. selection, translation, or rotation. In other words, hand gesture detection was only used as a mouse click and drag-and-drop metaphor to manipulate 3D contents, while the flexibility and dexterity of hands weren't fully exploited. In contrast,

GesturAR endeavors to explore an interface that can cover a majority of the hand gestures and enrich the usage beyond simple operations.

Meanwhile, instead of using predefined gestures, researchers encourage end-users to select preferred freehand interactions for multiple immersive applications [75]. VirtualGrasp [96] let users demonstrate how they grasp objects and utilize the object-gesture pairs to retrieve corresponding virtual objects. Similarly, Force Push [100] learned a force mapping model from users for remote object manipulation. Recently, MagicalHands [7] selected gestures for authoring particle animation from users' demonstrations. By introducing end-users into the design process, the freehand interaction space is greatly expanded and the resulting AR freehand interactions correspond with the end-user's life experience. Yet, the prior arts above employed an offline process where users could only try out the immersive applications after the researchers had collected, processed, and learned from the user-performed examples. On the other hand, GesturAR proposes a real-time authoring process where users not only design freehand interactions but also try them out immediately in the same AR context.

2.2 Gesture authoring through embodied demonstration

The metaphor of programming by demonstration [52] greatly enhances end-user development by overcoming low-level programming details. Specifically, embodied demonstration refers to the actions performed by the human's body. The body or hand poses can be used as references for creating 3D models [16, 38, 43] and dynamic contents [7, 13, 98]. Meantime, they can become examples to train classifiers for gesture detection. Gesture Coder [53] and Gesture studio [54] enable 2D multi-touch gesture creation through a demonstration and declaration process. Meanwhile, researchers allow users to record sensor outputs of demonstrated mid-air gestures, then use them as examples to train detection algorithms [8, 27, 102] such as Dynamic Time Warping or Hidden Markov Chain. However, these works need a desktop interface to visualize and edit the sensor outputs and may need iterative demonstration to improve accuracy. Thus these workflows are not compatible with immersive AR/VR environments. Recently, GhostAR [14] and CAPtURAR [91] integrate a human action editing interface so that users can visualize, manipulate and edit body gestures in AR. However, the detection algorithms presented in the works above [8, 14, 27, 53, 91, 102] mainly focus on raw sensor output or full-body actions and cannot be directly applied to gesture-centered interactions.

To this end, GesturAR is designed to follow the metaphor of embodied demonstration while fully considering the characteristics of freehand interactions in AR. The elicitation studies in previous works [7, 75, 96] reveal that a typical gesture in AR contains two kinds of properties, (1) local properties, which refer to the poses of hands, i.e. relative positions of joints, and (2) global properties, which refer to the positions, directions and movement of the palms. Inspired by these findings, GesturAR analyzes the key properties of users' gesture demonstration and embeds a real-time hand gesture recognition technique for a smooth gesture authoring experience.

2.3 Immersive authoring tools for AR applications

AR applications enable users to interact with a mixed virtual-physical environment. Authoring an AR application usually involves two steps: 1) creating virtual contents as well as their behaviors and 2) defining the interactions between users and virtual contents [44]. While popular AR application authoring platforms such as Unity [87] and Unreal [88] are powerful but obscure for non-professional users [66], alternative tools and workflows are proposed for designers and end-users to create AR applications [9]. Some works focus on rapid prototyping of AR experience through Wizard-of-Oz (WOz) [2, 64] or video prototyping [49, 50] rather than functional AR applications. Meanwhile, in-situ authoring was proposed [45, 92, 103] to blend the authoring process within the AR interaction space and allow for intuitive 3D manipulation instead of 2D programming. Window Shaping [33] and SceneCtrl [101] empowered the creation of static 3D models and virtual scenes by leveraging the spatial perception of AR devices. Furthermore, animations of virtual contents can be created using direct in-situ manipulation [7, 13, 14, 98]. Moreover, the visual programming capability [21, 29, 91, 105] is integrated into immersive interfaces to pair user interactions with virtual contents behaviors. For instance, iaTAR[45] connected user inputs with object properties to build tangible virtual models, and let users test the interaction immediately after the connection. Besides, Ng et al. [68] created situated games using trigger-action links. However, most previous tools accept limited user input modality such as spatial location [26, 68], or fiducial markers [36, 45, 70, 81]. While some works encourage users to use their hands during the authoring process [82, 92], the system that enables end-users to create AR applications with customized freehand input remains unexplored.

To fill in this critical gap, GesturAR combines embodied demonstration of hand gesture and real-time gesture detection with an in-situ authoring interface. In addition, to enable a fluent authoring experience, GesturAR interface integrates well studied interaction techniques that are presented in previous works such as scanning [33] and sketching [50] for virtual content creation, animation creation through direct manipulation [98], as well as visual programming using icons and connections [29, 105]. With GesturAR, users can create an interactive AR application from scratch, test it in real-time, and share it with other users.

3 GESTURAR SYSTEM DESIGN

3.1 AR freehand interaction model

The input-output model has been widely adopted by previous interaction prototyping systems including Exemplar [27], Trigger-Action-Circuits[3], Astral [42] and Kitty [35]. In such model, an interaction involves two components, an *input* that is initiated by a subject, and an *output* that is generated by an object in response to the *input*. An AR freehand interaction adopts the similar pattern where the *input* is a hand gesture and the *output* is the behavior of relevant virtual content.

Previous elicitation studies on AR hand interactions [75, 95] categorized the AR gestures in to six dimensions: *Nature*, *Symmetry*, *Binding*, *Locale*, *Form*, and *Flow*. Among these dimensions, *Form*,

which describes the status of the hands, and *Flow*, which describes the response of the virtual contents, corresponding to the *input* and *output* respectively. Following the categories of *Form* and *Flow*, we derive two types of *input* and two types of *output*:

- *Static input*, which is a specific status of the hands, including hand pose, position, direction and handedness,
- *Dynamic input*, which is a time series of hand status,
- *Discrete output*, which responds right after the gesture completion,
- *Continuous output*, which responds during the gesture.

Therefore, we would like to explore the resulting AR freehand interactions created by combining different types of *inputs* and *outputs* and fill in the 2 by 2 table presented in table 1 to complete the interaction model in GesturAR.

Table 1: AR freehand interaction model

<i>input</i> \ <i>output</i>	Discrete	Continuous
Static	<i>static-provoking</i>	<i>manipulating</i>
Dynamic	<i>dynamic-provoking</i>	<i>synchronous</i>

Static + Discrete: This category represents the interactions in which virtual contents respond right after a *static* gesture is detected. Thus, we name it *static-provoking* type. Touching a surface or pressing a button are *static-provoking* interactions. We show an example in Figure 1d-1, where a light bulb glows after a user performs a two-handed holding gesture beneath it.

Static + Continuous: This category represents the interactions in which virtual contents keep reacting to the user’s *static* gesture. The most common scenario, in this case, is *manipulating*, where an object follows the transform of the user’s hands. Figure 1c shows a virtual cup is manipulated by a user while another example is shown in Figure 1d-2 where the virtual laptop lid follows the user’s hand while being constrained and rotating along a hinge.

Dynamic + Discrete: This category represents the interactions in which virtual contents respond right after a *dynamic* gesture is detected. Similar to *static-provoking* type, we name it *dynamic-provoking* type. Waving hands, clapping hands and punching are *dynamic-provoking* interactions in daily life. An example of *dynamic-provoking* interaction in AR can be found in Figure 1d-3. A user breaks the soda can by clenching the fist.

Dynamic + Continuous: This category represents the interactions in which a virtual content responds synchronously to the movement of the hands during a *dynamic* gesture, like resizing an object with the distance between the hands. Therefore, we name it *synchronous* interaction. Figure 1d-4 shows a *synchronous* interaction where a user can stretch a spring using a pinching gesture.

3.2 Programming AR freehand interactions through trigger-action connections

To enable an intuitive authoring experience and a minimized learning curve, we adopt a trigger-action programming model in GesturAR reflecting the *inputs* and *outputs* of the freehand interactions. We provide users with two conceptual primitives, namely *trigger*

(represented as a solid triangle) and *action* (represented as a hollow triangle) (Figure 1b). This way, users can create the four types of AR freehand interactions by connecting different *triggers* and *actions*.

While running an AR application in GesturAR, a *trigger* monitors the user’s behavior and sends information to an *action* through the connection between them. Typically, there are two kinds of information, i.e. signal (for *discrete* interaction) and value (for *continuous* interaction). In GesturAR authoring interface, users can use the following kinds of *triggers*:

- *static* gesture that can emit both signals (when a *static* gesture is detected) and a value (the transform of the hands).
- *dynamic* gesture that can emit both signals (when a *dynamic* gesture is completed) and a value (the progress of the dynamic gesture).

and *actions*:

- *following action* (the behavior to follow another object) that can receive a value (i.e. the transform of the other object).
- *animation action* that can receive either a signal to start playing the animation, or a value that controls the progress of the animation.
- multiple predefined *actions*, such as *appear/disappear*, *mesh deformation*, *mesh explosion*, etc. that can receive signals from *triggers* (Figure 6).

A freehand interaction is valid once the *trigger* and *action* can send and receive the same type of information and the type of the freehand interaction is decided by the combination. for instance,

- *static* gesture *trigger* + *following action* leads to a *manipulating* interaction.
- *static* gesture *trigger* + *animation action* or predefined actions leads to a *static-provoking* interaction.
- *dynamic* gesture *trigger* + predefined *actions* leads to a *dynamic-provoking* interaction.
- *dynamic* gesture *trigger* + *animation action* can result in either a *dynamic-provoking* interaction or a *synchronous* interaction. GesturAR generates a *synchronous* interaction if the *dynamic* gesture *trigger* is created after the *animation action*, or a *dynamic-provoking* interaction otherwise.

Besides, users can connect multiple *actions* to one *trigger* to activate them together or connect multiple *triggers* to one *action* so that every *trigger* can activate the same *action*. Users are encouraged to use multiple *trigger-action* connections to build interesting AR applications. To ensure valid authoring, GesturAR rejects the connection between the mismatching *triggers* and *actions*. In addition, to enlarge the interaction space of the AR applications, GesturAR also provides extra *event triggers* (Figure 6) that are beyond the scope of freehand interactions, such as *gaze event* (user looks an object), *approaching event* (user approaches an object), *position event* (a virtual object arrives at a location) and *collision event* (two objects collide with each other), etc. These *triggers* can emit signals and make *discrete* interactions when connected to *animation action* or other predefined *actions*.

3.3 GesturAR Authoring Interface

In this section, we introduce the interface of GesturAR, an authoring system for creating freehand interactive AR applications. Basically,

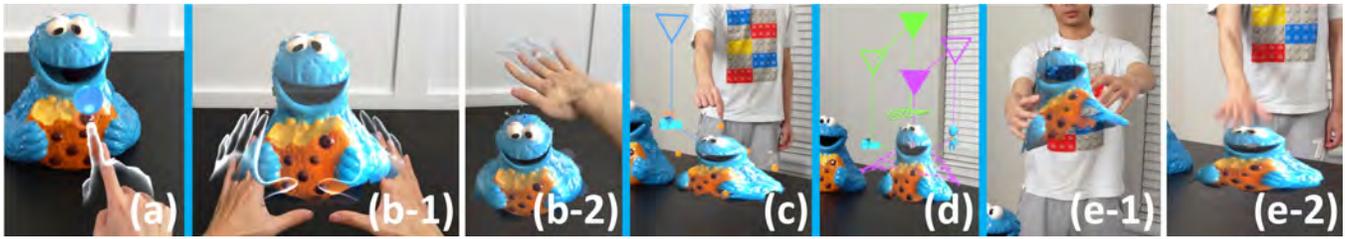


Figure 2: GesturAR system walk-through. (a) The user first creates a virtual cookie monster by scanning around the real object in *Creation Mode*. (b) In the *Authoring Mode*, the user demonstrates a *static* double-handed holding gesture (b-1) and a *dynamic* patting gesture (b-2) spatially referring to the virtual object. (c) The user demonstrates a scale change of the virtual object by dragging the scale handle. (d) The user connects a *manipulating* with the holding gesture, and the scale change with the patting gesture through visual programming. (e) In *Play Mode*, the user can perform the same holding gesture to manipulate the cookie monster (e-1), or squashes it by the patting gesture (e-2).

GesturAR interface is separated into three modes, a *Creation Mode* for users to create virtual assets, an *Authoring Mode* for users to design and edit freehand interactions, and a *Play Mode* for users to try out the authored AR application. Users can choose the modes using a menu floating next to the left hand (Figure 3a). To better understand the GesturAR workflow, we take a look at a user who wants to create an interactive virtual cookie monster using GesturAR (Figure 2). Specifically, the user prefers to hold the cookie monster using two hands (Figure 2e-1) or squashes it by patting its head (Figure 2e-2). First, the user needs to **create the virtual asset** of the cookie monster in *Creation Mode* (Figure 2a). Then the user **demonstrates the gestures** of holding it and patting its head in *Authoring Mode* (Figure 2b-1,2). After that, the user **edits the behaviors** of the virtual model. The user selects a *following action*, and demonstrates an *animation action* that "the cookie monster is squashed" (Figure 2c). After finishing authoring the hand gestures and the object behaviors, the user connects them through **visual programming** (Figure 2d). Now the user authors two freehand interactions: a *manipulating* interaction and a *dynamic-provoking* interaction. Finally, the user enters the *Play Mode* to play with the interactive cookie monster (Figure 2e-1,2). In the rest of this section, we will describe the operations of each step in detail.

3.3.1 Create virtual objects. As an all-in-one AR authoring interface, GesturAR incorporates a *Creation Mode* for users to create virtual assets in-situ. Users can make a virtual object by importing 3D models, mid-air sketching or 3D scanning. To import a 3D model, a user can turn on a menu with models of everyday objects using the right-hand menu. While doing mid-air sketching, a brush tip is floating in front of his/her right hand index finger. The user can change the color and width of the brush (Figure 3b), or leave a trace while moving the right-hand (Figure 3c). To perform a 3D scanning, the user touches the surrounding surface with a spherical scanning tip (Figure 2a). The mesh pieces of the touched surface then appear. Note that the current implementation of GesturAR is not able to perform 3D scanning using the native hardware on Hololens2. The scanning experience is simulated with a pre-scanned mesh of the surrounding surfaces.

Besides simple rigid 3D models, GesturAR enables users to build complex assemblies using **mechanical constraints** (Figure 4a). Figure 4b shows a user who builds a virtual chest using a *hinge joint*.

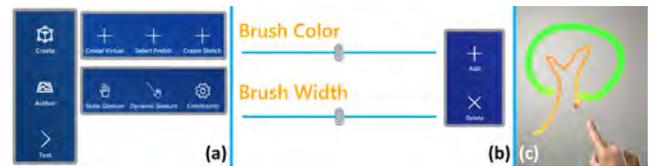


Figure 3: (a) Left-hand menu. The left column is the main menu for the three modes. The icon of the chosen mode will be highlighted. The right two rows are the sub-menus corresponding to the *Creation* and *Authoring Mode*. The sub-menu that does not belong to the current mode will be hidden. (b) Right-hand menu to switch between "Add" and "Delete", and change brush color and width for mid-air sketching. Users can do all regular operations by choosing "Add". While choosing "Delete", any virtual content that is touched by users' hands becomes red and will be deleted when the user performs a 'pinch' gesture. (c) A user is creating 3D sketches with the brush tip.

Users can also bind virtual objects to the environment to constrain the movements. A user can create a door movement using a *hinge joint* or a virtual drawer using a *sliding joint*.

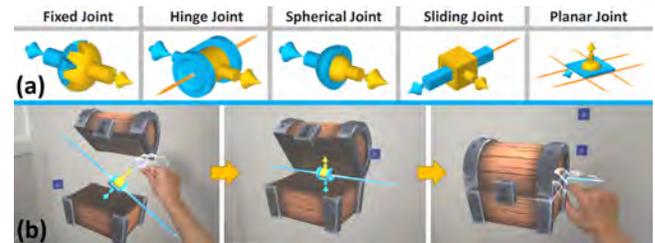


Figure 4: (a) Mechanical constraints supported by GesturAR. (b) Procedure: The user first connects the parent and child objects of a *hinge joint* with lines, then arranges the spatial relationship among the three elements. The movement of the chest lid is constrained with the base.

3.3.2 Demonstrate hand gestures. GesturAR enables users to define a desired freehand interaction by directly performing it while using

real objects or virtual contents as spatial and temporal reference. As shown in Figure 2b-1,2, the user performs gestures against the scanned model. Figure 5 shows the detailed process of demonstrating a hand gesture. Once the button for creating a gesture is pressed, the user has five seconds to prepare. After the preparation, GesturAR starts to record the user's hand gestures. For *static* gestures (Figure 5a), the user holds the hand still for 2 seconds to allow for multiple sampling of the gesture. For *dynamic* gestures (Figure 5b), the user moves hands to perform the gesture, then holds hands still after finishing the gesture. When GesturAR detects that the user has held the same gesture for 2 seconds, it stops recording the gesture and removes the last 2 seconds from the *dynamic* gesture. On completion of the demonstration, a skeleton model of the hand gesture is displayed as the *trigger* object of the gesture. The skeleton model also represents the position and detection range of the gesture. The user can move it or resize its bounding box to modify the gesturing space (Figure 5d). The gesture will only be detected when the hand enters the bounding box to prevent false positive detection. Besides, users can set the bounding box to be bound to a relevant object, stick to world coordinate, or follow the user.

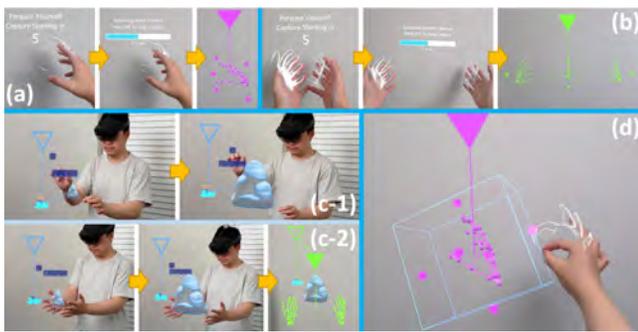


Figure 5: (a) Static, (b) dynamic-provoking, and (c) synchronous gestures authoring procedure. (d) Change volume and position of the gesturing space by manipulating the bounding box. Users can further bind the gesturing space to an object or the world coordinate.

Furthermore, users create *synchronous* interactions by creating *dynamic* gestures after the *animation* has been recorded (Figure 5c). The user first creates a scaling animation of a rock (Figure 5c-1). Then he/she performs a *dynamic* gesture by aligning the hands with the size of the rock (Figure 5c-2). Thus, GesturAR pairs the *key value* of the gesture, which is the distance between the hands, to the progress of the rock scaling animation, and enables the user to synchronously control the size of the rock with both hands.

3.3.3 Edit virtual object behaviors. For each virtual object, GesturAR displays all its possible behaviors (*following*, *animation*, etc.) in a nearby menu. A user can create corresponding *actions* objects by selecting from the menu. Especially, for the *animation* behavior, GesturAR lets users create an animation by directly move, rotate or scale the object with default hand manipulation. In Figure 2c, the user creates a squashing animation by scaling along the y axis of

the virtual object. Additionally, the pre-mentioned *event triggers* (*gaze event*, *approaching event*, *position event*, *collision event*, etc.) are also listed in the same menu. *Triggers* and *actions* supported by GesturAR are listed in Figure 6.

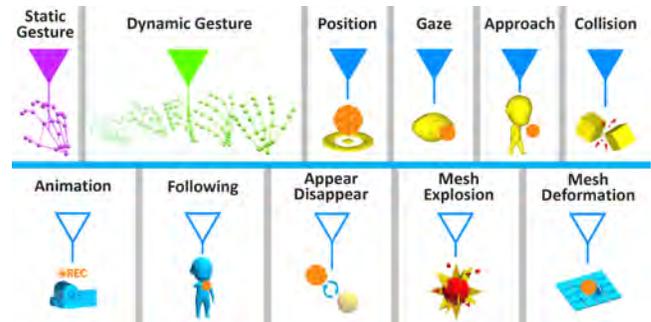


Figure 6: Triggers (top) and Actions (bottom) supported by GesturAR.

3.3.4 Program freehand interactions. The last step of the authoring process is to connect the *triggers* and *actions*, which can be done simply through a drag-and-drop manner. To make the connection, the user first pinches the solid triangle icon of a *trigger* and picks up a line from it. Then he/she approaches the hollow triangle icon of an *action* and releases the fingers to drop the line on the hollow triangle. Once the user finishes connecting, GesturAR color-codes the connection to better visualize the logic. Figure 2d shows two *trigger-action* connections: 1) a green one, which connects a *dynamic* gesture and an *animation*, refers to the "squashing the cookie monster by patting its head" interaction, and 2) a purple one, which connects a *static* gesture and a *following* action, refers to the *manipulation* interaction using a two-hand holding gesture.

3.3.5 Test the AR applications. *Play Mode* supports users to try out the interactive contents on-the-fly. In this mode, the system keeps tracking all the *triggers* authored by the user. An *action* is activated when any connected *trigger* is activated (Figure 2e-1,2). Specifically, the detection of the gesture *triggers* will be elaborated in the next section. Moreover, in *Play Mode*, all the *trigger* and *action* icons are hidden, while the skeleton hand models are left as visual hints.

3.4 Hand Gesture Detection

We describe how GesturAR detects a user's hand gesture for *static gesture trigger* and *dynamic gesture trigger*, as well as extracts *key values* from *dynamic* gestures in *synchronous* interactions in *Play Mode*. Essentially, we rely on the hand joints data provided by Hololens2 hand tracking API ¹.

3.4.1 Static hand gesture recognition. GesturAR detects a *static* gesture based on the position and pose of the hand. For instance, a user can only grab a cup when the hand approaches the handle and performs the grabbing pose (Figure 1c). While the hand position is easy to track, the hand pose is flexible and versatile. Therefore, rather than a classification algorithm that can only detect limited hand poses, we adopt the method of one-shot learning [39]. Basically, we

¹<https://docs.microsoft.com/en-us/windows/mixed-reality/mrtrk-unity/features/input/hand-tracking>

train a Siamese neural network (Figure 7b) that indicates if the two input hand data belong to the same gesture or not. Thus, we are able to detect hand gestures using only one demonstration example by comparing the real-time hand data with it. Since the hand poses are commonly performed by bending different fingers, we directly use the bending angles of the 10 hand finger joints (Figure 7a) as the input to the Siamese network. Due to the symmetry of hands, the Siamese neural network can be applied to either hand. As for the two hand gestures, the left- and right-hand poses are processed separately. And a two-hand gesture is detected only when both hand poses are matched. To reduce the computational cost, we design a small neural network with fully connected layers (Figure 7b). With this model, Hololens2 is able to process the hand pose data at 60 frames per second.

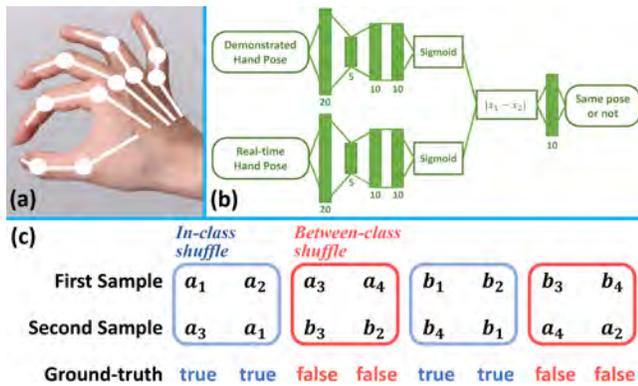


Figure 7: Hand pose detection algorithm used by GesturAR. (a) We focus on the 10 hand joint angles for hand pose detection. (b) Structure of the Siamese neural network. The top and bottom branches of the network share same structure and parameters. This network returns a true or false label that indicates whether the two inputs belong to the same gesture. (c) Our data augmentation method for training the Siamese network. a_i ($i = 1 \sim 4$) and b_i ($i = 1 \sim 4$) represent hand poses that belong to the two different classes. In each epoch, we perform a *in-class* shuffle on half of the data to produce true pairs (blue) and a *cross-class* shuffle on the other half to produce false pairs (red).

We collected a customized hand pose dataset with 18 classes of the hand poses (Figure 8 (top-left)) referring to previous studies [75, 96]. We invited 12 volunteers (7 males, 5 females) to collect approximately 2000 samples for each class. We randomly select 2 volunteers whose samples are used as the validation dataset, and the rest samples are used as training dataset. To train the Siamese neural network, each time we randomly select a pair of samples from the training dataset as inputs. We set the ground truth label as *true* if the two samples belong to the same class, and vice versa. To make sure the training data is balanced, i.e. the numbers of *true* pairs and *false* pairs in each training epoch are same, we perform an *in-class* shuffle and a *cross class* shuffle on the dataset as shown in Figure 7(c). We achieved an accuracy of 98.56% on the validation dataset, as shown in the confusion matrix in Figure 8.

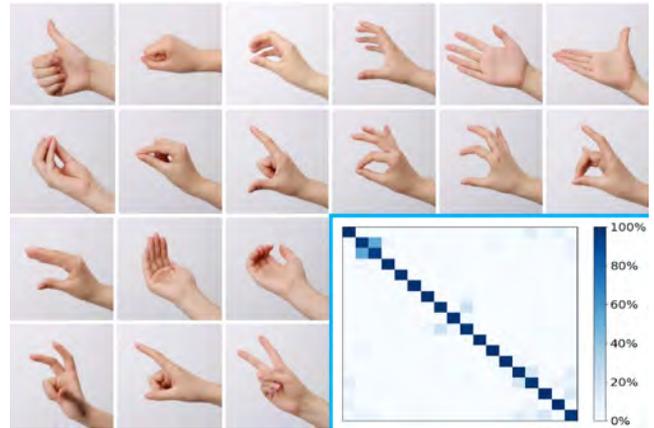


Figure 8: The 18 gestures included in our customized dataset. Roughly, the first 9 gestures involve palm and whole hand, while the rest gestures focus on finger-based manipulation (top-left). The confusion matrix achieved on the validation dataset (bottom-right). The color represents the percentage of true labels among all predicted labels. The overall accuracy is 98.56%.

3.4.2 *Dynamic gesture recognition.* Traditional ways to handle the temporal and spatial information contained in a *dynamic* gesture are dynamic time warping (DTW) [14, 51], hidden markov model (HMM) [20, 60] or neural networks [17, 56]. Yet these methods need a large amount of training data and massive computational power. Instead, we propose a *changing-state* method for a fluent run on Hololens2. We record a *dynamic* gesture as a time series of hand status data $[f_1, f_2, \dots, f_N]$. Each frame f_i contains hand information including joint positions, palm position, moving speed, etc. It is time-consuming to directly analyze the entire time series. To distill key features from the time series, we apply a *state* to describe the status of a hand at each frame. A *state* contains three attributes, namely the hand pose (*Pose*), moving direction (*mDir*) and palm rotation (*pRot*). The latter two attributes are evaluated with respect to the user’s local coordinate systems. As shown in Figure 9a-1,2, we use verbal labels rather than numerical values to note the moving direction and palm rotation. Further, we implement the Siamese neural network described in the previous section to tell whether the hand poses in two different frames are in the same class. This way, we can combine the adjacent frames with same *state* and encode the *dynamic* gesture into a shortlist of *states* (Figure 9b). We present the detailed algorithm (Algorithm 1) below. Typically, a *dynamic* gesture can be transferred into a list with two to six *states*. In *Play Mode*, we apply a similar *state* encoding algorithm to the real-time hand tracking data (just change the f_i in the for loop to the real-time data) and save the *states* in a queue. To detect a *dynamic gesture*, we compare the latest elements of the queue with its *state* list. If all the *states* are sequentially matched, we assume the user is performing that gesture. For the gestures that involve both hands, we handle the left and right hand tracking data separately. Such gestures are detected when the *state* lists of both hands are matched. Additionally, to avoid false-positive detection, we only

detect a gesture when the user's hands enter the gesturing space of the authored gesture (Figure 5d).

Algorithm 1 Calculate *state* series from a *dynamic* gesture

```

1: procedure CALCULATESTATES( $[f_1, f_2, \dots, f_N]$ )
2:   States  $\leftarrow$  array[]
3:   get state  $s_1$  from  $f_1$ 
4:   add  $s_1$  to States
5:   for  $i \leftarrow 2, N$  do
6:     get state  $s_i$  from  $f_i$ 
7:      $s_d \leftarrow$  the last element of States
8:     if SiameseNetwork( $s_i.Pose, s_d.Pose$ ) is true then
9:       if  $s_i.mDir \neq s_d.mDir$  or  $s_i.pRot \neq s_d.pRot$  then
10:         $s_{new} \leftarrow (s_d.Pose, s_i.mDir, s_i.pRot)$ 
11:        add  $s_{new}$  to States
12:     else
13:       add  $s_i$  to States
14:   return States
    
```

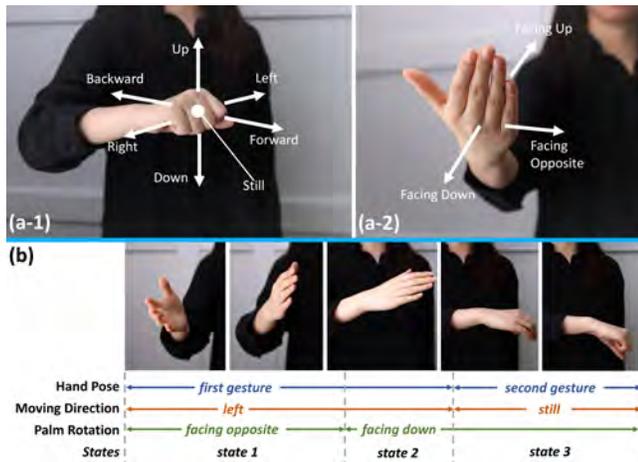


Figure 9: (a) The labels used to describe the hand movement and palm rotation. (a-1) The hand movement is characterized by 6 moving directions (*Up*, *Down*, *Left*, *Right*, *Forward* and *Backward*) or *Still* (not moving). (a-2) Three labels are used to describe the rotation of the palm. Respectively, *Facing Up*: the palm faces toward the user's head; *Facing Down*: the back of the hand faces toward the user's head; *Facing Opposite*: the palm faces the other side of the body. (b) The process of converting a *dynamic* gesture into a list of *states*. The hand changes to a new *state* when any of the three attributes changes.

3.4.3 Temporal and spatial mapping in synchronous interaction. To achieve the synchronization between a *dynamic* gesture and a virtual content *animation* in a *synchronous* interaction, GesturAR first extracts a numeric *key value*, like the distance between hands or angle between fingers, from the *dynamic* gesture (Figure 10). Then

GesturAR maps the *key value* to the progress of the *animation* to achieve temporal and spatial correlation between the hands and the virtual contents. Considering the type of hand gestures, we achieve the numeric *key value* in different ways. For instance, the pinching and clamping gestures are mapped to fingertip distances (Figure 10a,b); the holding and grasping gesture are mapped to a circle formed by the hand (Figure 10c); some gestures that involve finger bending are mapped to the bending angles (Figure 10d,e); the gestures that involve hand movement are mapped to the moving distance (Figure 10f); further, the two hand gestures are mapped to the distance or angle between the hands (Figure 10g,h). In *Authoring mode*, after the user has demonstrated a *dynamic* gesture in a *synchronous* interaction, the gesture is classified into one of the types mentioned above using the *changing-state* method, and the corresponding *key value* is extracted.

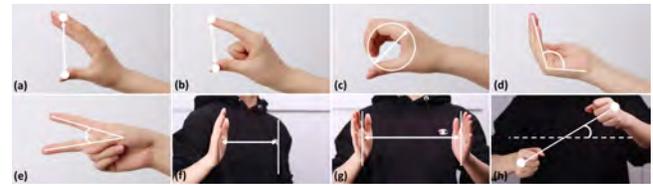


Figure 10: The mapping between hand gestures and the numeric *key values* in *synchronous* interactions.

4 APPLICATION SCENARIOS

4.1 Realistic object manipulation with multiple ways

Many objects in real-life respond differently when hands interact with them in different ways. For instance, a basketball can be held with two hands, spin on one finger or bounce when the user pat on it. Programming such various types of behaviors need vast effort [37]. With GesturAR, users can simply demonstrate the hand gestures and connect them to different behaviors respectively. As shown in Figure 11a-1, a user connects the two-hand 'holding' gesture with a *manipulating action* of a virtual basketball, a pointing gesture with a 'spinning' *animation* and a 'patting' *dynamic* gesture with a 'bouncing' *animation*. Then, the user can interact with the virtual basketball in a similar way as real basketballs (Figure 11a-2).

Furthermore, the purpose of a gesture can be inferred through its spatial property. For example, holding the head of a pencil implies 'writing' while holding the tail implies 'erasing'. Benefited from the in-situ visualization and spatial awareness of GesturAR, the user can demonstrate two 'holding' gestures at the head and the tail of the pencil and connect them to the 'writing' and 'erasing' *actions* respectively (Figure 11b-1). The users also connect both gestures to a *following action* to let the pencil follow the movement of the hand. Now, the user can either write or erase by grabbing different parts of the virtual pencil (Figure 11b-2).

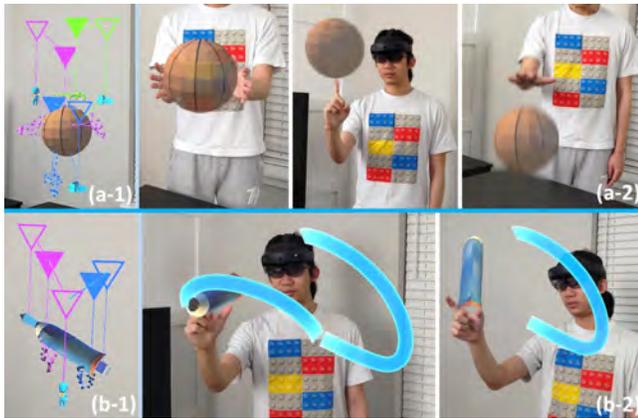


Figure 11: Virtual basketball: (a-1) A 'holding' *static* gesture and a 'pointing' *static* gesture are connected to a *following* *action*; the 'pointing' gesture is also connected to a 'spin' *animation*; a 'pat' *dynamic* gesture is connected to a *dribble* *animation*. (a-2) The user can hold the basketball using both hands, spin the basketball on the index finger, and dribble the basketball. Virtual pencil: (b-1) Two 'holding' *static* gestures are placed at the tip and the bottom of the pencil, connecting to a *following* *action*; the tip-side gesture is connected to a 'writing' *action*, while the other one is connected to an 'erasing' *action*. (b-2) The user can hold the tip of the pencil to write, or erase a stroke when holding the bottom of the pencil.

4.2 Interactive virtual agents and robots

Assistant agents and robots, such as robotic pets [84], and virtual assistant [57], largely enriches our daily life with their interactive behaviors. Recently, embodied gestures have been embraced as a popular modality for humans interacting with robots [97] and virtual avatars. Meanwhile, in-situ prototyping of such experience greatly accelerates the process of interaction design for these agents [76]. Figure 12a illustrates the process of prototyping a virtual mobile robot using GesturAR. The user first scans a mock-up robot as the virtual model. Then, to better simulate the moving behavior of the robot, the user adds a *planar joint* to constrain the robot to the ground. Next, the user creates two *animations* of the robot, namely 'move forward' and 'rotate', and maps them with the 'come' gesture and 'moving hand' gesture. This way, the user can either call the robot to come or let it dance by following the hand movement.

In another scenario, the user plans to prototype a digital avatar. The user can first scan one of his friends into the system. Next, by setting two *spherical joints* at the agent's elbow and wrist, the agent's arm can move realistically. As a result, the user creates a humanoid agent that can wave the hands after the user waves the hands, shake hands with the user, and pops up a sketched "Hi" after the user pats on the agent's shoulder respectively (Figure 12b).

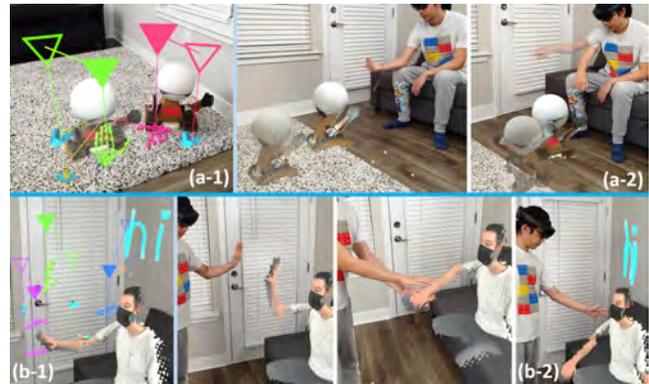


Figure 12: (a) An interactive toy robot: (a-1) A *planar joint* is connected to the robot. A 'come' *dynamic* gesture is connected to a 'moving' *animation* *action* as a *dynamic-provoking* *interaction*, while a 'pointing' *dynamic* gesture is demonstrated referring to a 'rotation' *animation* as a *synchronous* *interaction*. (a-2) The toy robot can move towards the user after the user does a 'come' gesture and rotates synchronously with the user's 'pointing' hand command. (b) A humanoid agent: (b-1) Two *spherical joints* link the agent's body, upper, and lower arm. A double-handed *static* gesture is created next to the agent's hand and connected with a *following* *action*. A 'waving hand' *dynamic* gesture is connected with a similar 'waving hand' *animation* *action* of the agent. A sketched "Hi" is linked with a 'patting' *static* gesture. (b-2) The virtual agent can wave the hand, shake hands with the user, and pop up a "Hi" after the user pat on the agent's shoulder.

4.3 Room-level interactive AR game

Besides focusing on the interactions with one single object, a broader application scenario is to make an augmented and interactive living space. For instance, the interactive AR gaming area witness a flourishing development in recent years [55, 72]. With GesturAR, the capability of in-situ authoring and object scanning allows end-users to create in-door AR games that exploit the blend of the physical and virtual contexts. Here, the user plans to create a treasure hunt game for the guests using some pirate-themed objects he has (Figure 13a). The user first scans a treasure map into two parts together with a skull. He then places three locked treasure boxes with only one with treasure on the table and hides the key under the physical skull. Then, the user places the *position triggers* within a sketched frame and sketches an arrow physically pointing to the real skull as a visual hint for the keys hidden beneath the skull. Meanwhile, the user places the scanned skull above the correct treasure box and uses it as the object to attach an 'open' gesture connected with a 'sparkle' *animation*. Throughout the entire authoring process, GesturAR empowers the user to seamlessly interweave the virtual and physical world to rapidly create an immersive AR game. When the guests come, one would enter the *Play Mode*, and follow the visual hint scattered in the room to find the treasure (Figure 13a-2).

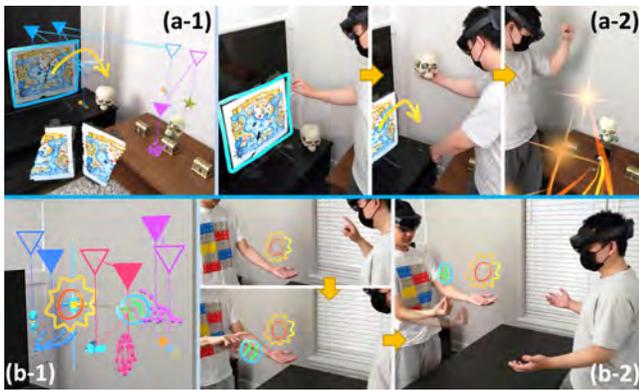


Figure 13: (a) AR treasure hunt game: (a-1) Two *positional triggers* of the scanned map pieces link to two *appear actions* of a virtual arrow and a scanned skull respectively. A 'open' *static* gesture is connected to a *sparkle action*. (a-2) The player first restores the broken treasure map; then finds the hidden key along the virtual arrow; he opens the correct treasure box, and a sparkle animation shows up. (b) In-situ AR presentation: (b-1) A *hinge joint* is connected between the sketched sun and the sketched earth. A *manipulating* interaction is implemented on the sun model. An *appear action* is connected with a *static* gesture located next to the earth. A 'rotation' *animation action* of the earth is linked with a 'pinch rotate' *dynamic* gesture. (b-2) The presenter first holds a sketched sun. Then, the earth appears next to the presenter's right hand. Finally, the earth starts to rotate about the sun after the presenter symbolically rotates his hand.

4.4 Embodied AR presentation

Performing hand gestures during conversations is a common practice. Yet, plain hand gestures only convey limited information due to the lack of expressiveness. GesturAR is useful to improve the efficiency in thought delivery by incorporating embodied hand gestures with in-situ visual representations of dynamic 3D contents and animations in AR. Further, the sketching function enables an immediate and seamless creation of the desired 3D contents for the conversation. Embodied AR presentation is one typical application of GesturAR. Here, a presenter authors an earth-sun animation to explain the Heliocentric theory (Figure 13b). The user first sketches the simple models of the sun and the earth, then constrains the earth to rotate around the sun using a *hinge joint*. During the AR presentation, the presenter first talks about the sun while holding it in front of the listener, then reveals the earth with a 'touching' gesture. Finally, by performing a 'pinch rotation' *dynamic* gesture, the earth model starts to rotate about the sun to illustrate the theory vividly. From the listener's perspective, the augmented hand gestures clearly explain the ambiguous concept with the help of the in-situ visual representations (Figure 13b-2). Similarly, GesturAR can also be implemented in scenarios such as investor pitches, educational lectures, storytelling, etc.

4.5 Entertaining daily life with embodied hand gestures

Our system supports the ubiquitous perception of users' spatial and hand gesture information so that end-users can entertain the daily life with their imagination. Leveraging real-time hand gesture detection, users can perform various pre-defined gestures to trigger entertaining virtual elements or fancy visual effects. For instance, the user can break the room's wall virtually anytime when he feels angry by performing a punching gesture towards the scanned wall. to trigger a pre-authored *mesh deform action* (Figure 14a). Meanwhile, the user can create a portable pet shark with a *synchronous* interaction with a 'grabbing' gesture and a 'scaling' *animation*. So, the user can open and close his hand to control the size of the shark. Further, the user authors a 'swim' *animation*, so that the toy shark can be shown above the user's hand and starts to swim (Figure 14b). Last but not least, the user author a light sword with an adjustable blade. Specifically, the handle of the sword can be held by the user with *manipulating* interaction. Meanwhile, the user authors a *synchronous* interaction to the sketched blade to change the length using a double-handed gesture. Therefore, when the user holds the handle and starts to move the other hand away, the blade is elongated. Afterward, the user can withdraw the sword by performing the gesture reversely 14c).

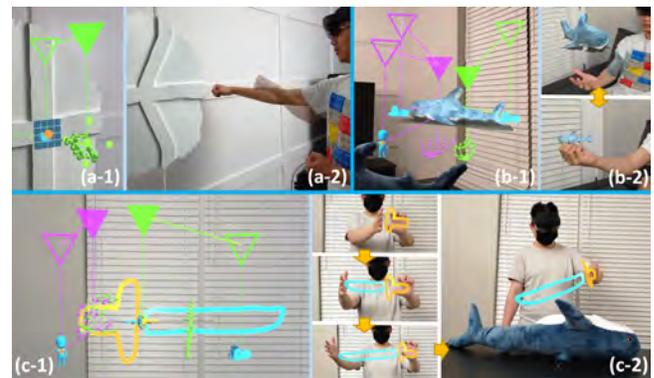


Figure 14: (a) Forceful punch: (a-1) A 'punch' *dynamic* gesture is connected to a *mesh deform action* of the scanned wall. (a-2) The scanned wall is deformed as the user punches. (b) Portable pet shark: (b-1) A 'supporting' *dynamic* gesture connects to a *following action* and a 'swimming' *animation action* of the scanned shark. A 'grabbing' *dynamic* gesture is created from a 'scaling' *animation* for a *synchronous* interaction. (b-2) A scanned shark swims above the user's hand and can be shrunk to the palm. (c) Light sword: (c-1) A *following action* is connected to a 'holding' *static* gesture. A double-handed 'drawing' *dynamic* gesture is created together with the *animation* of the elongation of the blade. (c-2) The user first holds the handle of the sword with the left hand and draws the blade by moving the right hand away. Then, he can hold and strike the sword. After that, the user can withdraw the sword by performing the 'drawing' gesture reversely.

5 IMPLEMENTATION

We build our system on Hololens2 [31] using Unity3D (2019.4.16f1) [87]. The GesturAR user interface is implemented with the support of Microsoft Mixed Reality Toolkit (MRTK) [61], FinalIK [22] and mesh effect libraries^{2 3 4}. The Siamese neural network for one-shot hand gesture classification is trained on a local PC (Intel Core i7-9700K, 3.6GHz CPU, 32GB RAM, NVIDIA RTX2080 GPU) using PyTorch [78] and runs on Hololens2 through Unity Barracuda⁵. Additionally, to enable users to make virtual contents through scanning, we preload the mesh model of the surrounding environment, which is created with an iPad 3D scanner [1], onto the Hololens2. We expose the mesh triangles touched by the user’s pen tip to simulate the scanning experience.

6 USER STUDY

We conducted a three-session user study to evaluate the hand detection model accuracy, immersive hand-object interaction authoring feasibility, and the overall system usability. 12 users (9 males and 3 females, aging from 21 to 30) were recruited. 11 of the users had experienced AR/VR applications or games on cell phones, tablets or head mounted devices. The rest one had the basic understanding of AR/VR concepts. We did not invite any AR/VR designers or programmers since GesturAR is designed for the customization experience of non-expert AR consumers. None of the users had experience with our system before the user study. Note that none of the users provided the hand gesture data for the training process in Section 3.4. To better verify the shareability of our system, each time we invited two users to do the study at the same time. The entire study took 2 hours, and each user was paid 20 dollars. The study was taken in a 5mx5m indoor area and was screen and video-recorded for post-analysis. We first requested the users to experience the Hololens2 built-in tutorial to get familiar with the general freehand AR interaction. Then, for each session, both users first completed the authoring process. Considering counterbalancing, while testing the authoring correctness, 6 users first tested their own applications (self-authored), then their partners’ (other-authored). The other 6 tested in a reversed order. After each session, the users completed a survey with Likert-type (scaled 1-5) questions regarding the usage experience of the system features. After all the sessions, each user took a conversation-type interview to provide subjective feedback and finished a standard System Usability Scale (SUS) questionnaire.

6.1 Session 1: Gesture Recognition Accuracy

To assess the performance of the one-shot learning neural network for hand gesture detection, we selected 8 *static* gestures and 7 *dynamic* gestures (Figure 15). First, each user authored the 8 *static* gestures using the procedure illustrated in Figure 5a. Then, while testing each gesture, the user was asked to perform 1) two gestures that were the same as the targeting gesture (one with right hand, and one with left hand), and 2) two gestures that were distinct, in his/her opinion. Note that each user tested both their own and partners’ gestures. We recorded the data with 2x2 confusion matrices (TP, FP,

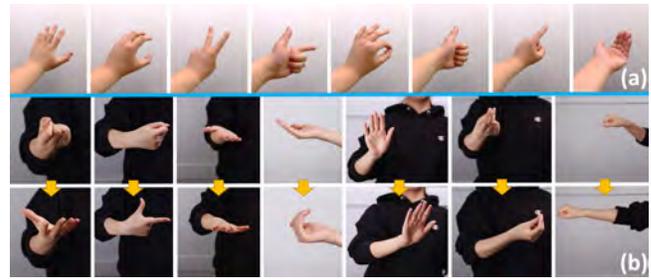


Figure 15: User study session 1 setup: (a) eight *static* gestures, and (b) seven *dynamic* gestures: open-hand, shoot, flip hand, come, wave-hand, pinch-rotate, and punch.

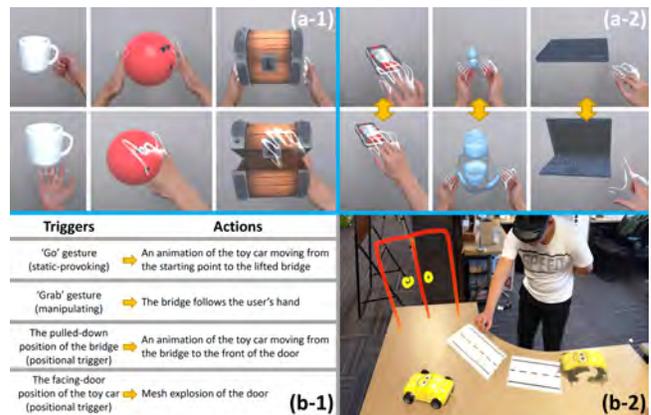


Figure 16: User study session 2 and 3 setup. Session 2: (a-1) 6 *manipulating* interactions: hold the cup handle; support the cup; hold the bowling ball with both hands; hold the bowling ball by holes; hold the chest with both hands; open the chest lid, (a-2) 3 *synchronous* interactions: push/pull a toy car; scaling a rock with both hands; open/close the laptop remotely with thumb and index finger, and (b) Session 3: the table-top interactive AR application.

TN, FN) for all gestures, and calculated the $F_1 = \frac{TP}{TP + \frac{1}{2}(FP+FN)}$ to measure the model performance⁶. Process for the *dynamic* gestures was similar, except that the users only performed each gesture once. And we recorded whether the system successfully detected them.

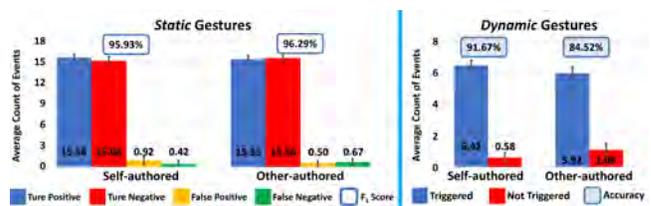


Figure 17: Results of user study session 1.

⁶<https://en.wikipedia.org/wiki/F-score>

²<https://assetstore.unity.com/packages/vfx/particles/spells/mesh-effects-67803>

³<https://assetstore.unity.com/packages/tools/particles-effects/mesh-explosion-5471>

⁴<https://assetstore.unity.com/packages/tools/utilities/gm-mesh-deformer-136461>

⁵<https://docs.unity3d.com/Packages/com.unity.barracuda@0.3/manual/index.html>

Result and discussion. The evaluation result is illustrated in Figure 17. For *static* gestures, our system could successfully distinguish both the correct and wrong gestures performed by the users (F_1 scores of the test of self-authored and other-authored cases were 95.93% (SD=0.04), and 96.29% (SD=0.036)). This result is comparable with the validation accuracy (Figure 8 (bottom-right)). Moreover, we conducted a one-way ANOVA test towards the F_1 scores between the two conditions after the data passed the Kolmogorov–Smirnov normality test ($D(12)=.255$, $p=.35>.05$ and $D(12)=.225$, $p=.51>.05$). No significant difference was revealed between the two conditions ($F(1,22)=0.056$, $p=.82>.05$), which indicated that our hand detection model could achieve a high quality regardless of the user who performed the gesture.

As for *dynamic* gestures, the system also received competent accuracy (91.67% (SD=0.10) for self-authored gestures, 84.52% (SD=0.11) for other-authored gestures). The decrease of the accuracy when testing partners' gestures were mainly attributed to the different preferences when performing similar gestures. For instance, when acting the "come" gesture, some users only bend their fingers while others also moved their lower arms. We will discuss the variation in the later section. Furthermore, the Kolmogorov–Smirnov normality test indicated the normality of the accuracy data ($D(12)=.319$, $p=.138>.05$ and $D(12)=.22$, $p=.54>.05$), and the one-way ANOVA test showed no significant difference between the two scenarios ($F(1,22)=2.79$, $p=.11>.05$), which disclosed that although the accuracy was slightly different, our system could still detect the *dynamic* gestures performed by different users.

6.2 Session 2: Hand-object Interaction Evaluation

In this session, we evaluated the performance of the GesturAR embodied authoring interface for freehand interaction creation. Here, the users mainly experienced the *manipulating* and *synchronous* situations with 6 pre-created virtual models. Specifically, the users authored 2 different *manipulating* interactions for each of the first 3 virtual models (Figure 16a-1), and 1 *synchronous* interaction for each of the remaining 3 (Figure 16a-2). We recorded whether the users successfully interacted with their own and partners' authoring results on the first try during the test.

Result and discussion. All 12 users authored 108 valid tasks in total. The overall test success rate of both the user's own and the partner's applications was 94.44% (SD=0.08), indicating that most users could fluently manipulate virtual objects using the freehand interaction provided by our system.

The freehand interaction-related Likert-type question ratings are shown in Figure 18 (top). In general, the users agreed with the necessity of customizing hand gestures for freehand interactions (Q2: AVG=4.75, SD=0.59). "I really like that I can define my own way to manipulate those virtual objects. The only pinch is way less enough for me (P3)". And the embodied demonstration approach was receptive (Q1: AVG=4.67, SD=0.47). "I think I can only record a gesture correctly when I do it right next to that object" (P3). Meanwhile, most users were content with the capability of the hand-object interactions in our system (Q3: AVG=4.58, SD=0.64). "[Static] gestures are definitely necessary. And after I used your system, I realized [dynamic] gestures are also very important. I'm glad you bring this up

(P7)". Additionally, the *synchronous* interaction was welcomed by the users (Q6: AVG=4.41, SD=0.64). "It's great that your system has that synchronization of the animation. And I like the idea that I can do my gesture while following the animation (P4)". For using the application, most users were confident that they could successfully interact with the virtual objects using the gestures they created (Q4: AVG=4.59, SD=0.49). "I was impressed when I could hold a virtual bowling ball like what I do with a real one (P5)". The survey result also showed positive feedback when trying out others' applications (Q5: AVG=4.25, SD=0.64). "When I tried my partner's app, the visible skeleton really helped me to figure out what to do. And I was surprised that I could open the chest he created that fluently (P11)".

6.3 Session 3: Overall System Usability

Finally, the users were asked to create a table-top AR puzzle game from scratch with all features supported by our system (Figure 16b). After the user did a self-defined "go" gesture, a toy car moved from the starting point and stopped in front of a lifted bridge. Then, the player had to grab the bridge to pull it down so that the car could move forward. Finally, the door was broken into pieces after the car reached the finishing point. Specifically, the users had to scan a toy car, a bridge, sketch a door, and set the hinge joint of the bridge. The required triggers and actions are shown in Figure 16b-1. We recorded whether the users successfully authored the game, and completed their own and partners' games.

Result and discussion. All 12 users successfully completed the authoring processes, tested their own applications, and tested their partners' applications. The overall Likert-type results collected from this session are shown in Figure 18 (bottom). Overall, the users acknowledged that the trigger-action metaphor was suitable for AR application creation (Q3, AVG=4.5, SD=0.67). "It was easy to follow when I first define a trigger, then an action (P10)". Most users felt confident in using the applications created by themselves or others (Q6, AVG=4.25, SD=0.96). "I felt super cool when I could successfully pull down the bridge he created and the car broke the door at the end (P12)". We also asked users about the authoring interface. The scanning and sketching features for virtual content creation received complimentary remarks (Q1: AVG=4.67, SD=0.49). "In my opinion, it's very useful that I can scan something around me and add funny interactions to it" (P1). Meanwhile, the *mechanical constraints* feature was well-received by users (Q2: AVG=4.58, SD=0.66). "It's awesome that I can fluently rotate the virtual bridge. It's definitely necessary to help me create more realistic object behaviors (P6)". Further, the users complimented using lines to build logic connections between triggers and actions (Q5: AVG=4.58, SD=0.66) and were satisfied with the clarity of the UI design during the authoring process (Q4: AVG=4.75, SD=0.62). "I like the idea of using solid and hollow triangles for triggers and actions. It helps me easily find what to connect, of course, using lines is super straightforward (P9)". Finally, the standard SUS survey result for the entire study received 86 out of 100 with a standard deviation of 11.18, which indicated high usability of the entire system.

7 LIMITATIONS AND FUTURE WORK

Haptic feedback in freehand AR interactions. Through the user study, some users brought up that "I'd be more confident to demo

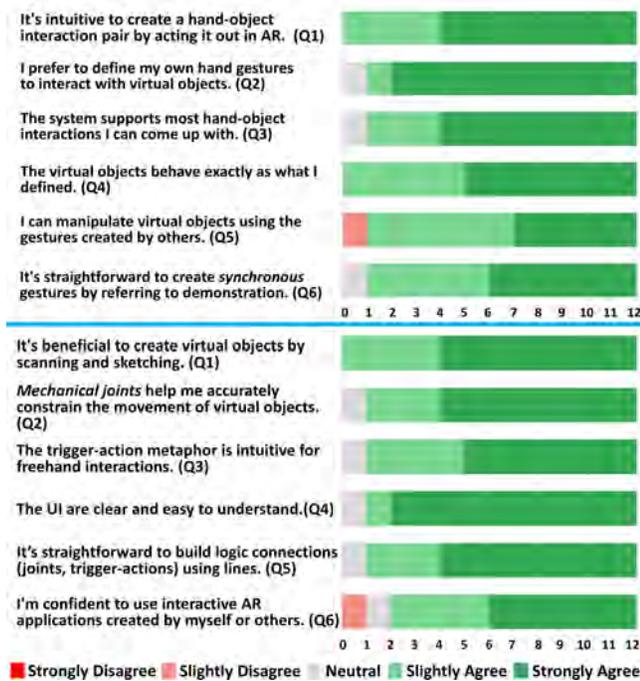


Figure 18: Likert-type questionnaire results of session 2 (top) and session 3 (bottom).

accurate gestures and operate those [virtual] object if I could feel what I'm touching (P2)". The lack of haptic feedback has been identified and discussed in prior freehand AR interactive systems [10, 30]. To improve the immersive experience, haptic gloves [12, 46] would be one solution but they usually are bulky and require external setups. Meanwhile, ultrasound-based devices have been exploited to provide unencumbered tactile sensation for freehand interactions [58, 86, 94], but limit users' gesturing space. Therefore, it would be fruitful to explore the incorporation of haptic feedback in user-defined freehand AR interactions as future work.

Ambiguity and variation of user-defined hand gestures. With GesturAR, enabling end-users to customize hand-object interactions improves the scalability of freehand interactive AR applications. Yet, some users raised that "When I use an object, I'd like to use it in a way I feel comfortable. But when I tried his objects, I was not clear what did he want me to do (P8)". As a shareable application, adapting to different users' tendencies in object interactions and dealing with the ambiguity of hand gestures substantially affect the system usability. To address this issue, prior studies [75, 95] elicited intuitive hand gestures for AR interactions. Chen et al. [15] proposed multi-modal approaches to further elucidate users' operations. From our observation, the user's hand size, handedness, and object size could be key factors. Thus, how to create freehand AR interactions that can be fluently used by others requires deeper research and analysis.

Blend the physical and virtual space. GesturAR enables users to virtualize their surroundings with a scanning technique. *Mechanical constraints* make the movement of the virtual contents more realistic. In the user study, some users suggested that "It would be

more realistic if these virtual items had gravity (P4)". Also, "Why I can't use real objects as triggers (P9)". Prior works have shown that interweaving physical contexts and virtual interfaces largely expands the capability of AR applications [29, 91]. Similarly, when authoring freehand AR applications, it would be beneficial to empower users with the environmental context perception by leveraging state-of-the-art object tracking algorithms and physical engines.

Overlapping of virtual contents. Interacting in the AR domain with bare hands requires a precise selection of the target virtual contents. In current implementation of *Authoring Mode*, each UI element has a cuboid bounding box as its interactive region. However, for different elements that occupy the same space, only one of them can be selected at one time. Thus, the clustering of UI elements and virtual objects frequently caused unnecessary difficulties. Usually, 3-4 Triggers/Action icons can congest the space around a virtual object and make the authoring process cumbersome. "When I created more gestures onto one single object, those bounding boxes were annoying. I had to first move them away. But it broke my authoring process (P11)". One solution could be the multi-modal interface, which has been widely explored in freehand AR applications [47, 74, 93], and has shown benefits regarding virtual object selection and basic object manipulation. Therefore, one improvement of our system would be integrating additional input modalities for a more fluent authoring experience.

Limited ways of mapping between triggers and actions. Currently, GesturAR only supports simple and direct mapping between *triggers* and *actions*. Namely, an *action* reacts immediately when a corresponding *trigger* is detected. Some complex mappings, such as **condition** (an *action* only reacts to a *trigger* given some preconditions, such as location, time or other *triggers*), **delay** (an *action* reacts after a *trigger* has been detected for a period of time) or **chain** (several *actions* react sequentially after a *trigger* is detected) are not supported. One possible solution can be introducing a more comprehensive spatial programming interface that is similar to the ones in CAPturAR [91], Ivy [21] and FlowMatic [105].

Hardware and software constraints. Although virtualizing surrounding objects through scanning was welcomed by the users, the current process is isolated from the main system workflow, which limits the scalability of the system. However, by embedding the state-of-the-art 3D reconstruction and fusion algorithms [34, 77], we believe that our system could provide a seamless and flexible experience of virtual asset creation. Meanwhile, the quality of the Hololens2 hand recognition is significantly reduced as hand occlusion happens, which limits the tangible interactions where users interact with virtual objects while holding real objects. We envision more robust hand-tracking algorithms that can improve the performance of GesturAR and expand the design space of freehand AR interactions in the future.

Additional supports for professional designers. While GesturAR aims at common AR consumers in their daily life, it can also be used as a prototyping tool for designers and programmers to rapidly validate their ideas. We envision the following improvements of GesturAR to better assist the professional users in the early design stage: 1) integrating with multiple prototyping methods such as physical prototyping [28, 64, 65] and sketching [6]. 2) allowing for customized *triggers* (e.g. IoT sensors) and *actions* (e.g. animations and mesh effects) in addition to the provided ones and

3) using external devices (e.g. tablets or phones) to handle complex spatial programming [29] and record augmented videos [50].

8 CONCLUSION

In this work, we present GesturAR, an all-in-one authoring tool that enables end-users to create AR applications with customized freehand inputs. GesturAR allows users to generate virtual contents in-situ and design their personalized hand gestures by embodied demonstration while using relevant context as spatial and temporal reference. We start from the taxonomy of hand gestures in AR and propose a hand interaction model that maps various types of hand inputs to the responsive behaviors of the virtual contents. Following the interaction model, we design our visual programming interface so that users can author multiple interaction modalities through simple trigger-action programming. Further, we develop a real-time gesture detection algorithm based on one-shot learning and time series analysis to support an instant experience of the authored AR applications. To explore the capability of GesturAR, we demonstrate five groups of application scenarios: creating interactive objects, humanoid and robotic agents, augmenting in-door environment with tangible AR games, making immersive AR presentations, and interacting with entertaining virtual contents. Through a user study, we first evaluate the accuracy of our hand detection network, then prove our system's usability in interactive application authoring from the positive study results and user feedback. Therefore, we believe that GesturAR reveals a novel perspective involving hand gestures with AR interactive applications and inspires future bare-hand immersive environment development.

ACKNOWLEDGMENTS

We wish to give a special thanks to the reviewers for their invaluable feedback. This work is partially supported by the NSF under grants Future of Work at the Human Technology Frontier (FW-HTF) 1839971. We also acknowledge the Feddersen Chair Funds. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agency.

REFERENCES

- [1] 3D Scanner App 2021. 3D Scanner App: Capture Anything in 3D. <https://www.3dscannerapp.com/>.
- [2] Günter Alce, Mattias Wallergård, and Klas Hermodsson. 2015. WozARd: a wizard of Oz method for wearable augmented reality interaction—a pilot study. *Advances in human-computer interaction 2015* (2015).
- [3] Fraser Anderson, Tovi Grossman, and George Fitzmaurice. 2017. Trigger-action-circuits: Leveraging generative design to enable novices to design and build circuitry. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. 331–342.
- [4] ARCore [n.d.]. ARCore. <https://developers.google.com/ar>.
- [5] ARKit [n.d.]. ARKit Overview. <https://developer.apple.com/augmented-reality/arkit/>.
- [6] Rahul Arora, Rubaiat Habib Kazi, Tovi Grossman, George Fitzmaurice, and Karan Singh. 2018. Symbiosisketch: Combining 2d & 3d sketching for designing detailed 3d objects in situ. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–15.
- [7] Rahul Arora, Rubaiat Habib Kazi, Danny M Kaufman, Wilmot Li, and Karan Singh. 2019. Magicalhands: Mid-air hand gestures for animating in vr. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*. 463–477.
- [8] Daniel Ashbrook and Thad Starner. 2010. MAGIC: a motion gesture design tool. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2159–2168.
- [9] Narges Ashtari, Andrea Bunt, Joanna McGrenere, Michael Nebeling, and Parmit K Chilana. 2020. Creating augmented and virtual reality applications: Current practices, challenges, and opportunities. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–13.
- [10] Hrvoje Benko, Ricardo Jota, and Andrew Wilson. 2012. Miragetable: freehand interaction on a projected augmented reality tabletop. In *Proceedings of the SIGCHI conference on human factors in computing systems*. 199–208.
- [11] Fabio Bettio, Andrea Giachetti, Enrico Gobbetti, Fabio Marton, and Giovanni Pintore. 2007. A Practical Vision Based Approach to Unencumbered Direct Spatial Manipulation in Virtual Worlds.. In *Eurographics Italian Chapter Conference*. 145–150.
- [12] Volkert Buchmann, Stephen Violich, Mark Billinghurst, and Andy Cockburn. 2004. FingARtips: gesture based direct manipulation in Augmented Reality. In *Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*. 212–221.
- [13] Yuanzhi Cao, Xun Qian, Tianyi Wang, Rachel Lee, Ke Huo, and Karthik Ramani. 2020. An Exploratory Study of Augmented Reality Presence for Tutoring Machine Tasks. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–13.
- [14] Yuanzhi Cao, Tianyi Wang, Xun Qian, Pawan S Rao, Manav Wadhawan, Ke Huo, and Karthik Ramani. 2019. GhostAR: A Time-space Editor for Embodied Authoring of Human-Robot Collaborative Task with Augmented Reality. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*. 521–534.
- [15] Di Laura Chen, Ravin Balakrishnan, and Tovi Grossman. 2020. Disambiguation techniques for freehand object manipulations in virtual reality. In *2020 IEEE conference on virtual reality and 3D user interfaces (VR)*. IEEE, 285–292.
- [16] Jiawen Chen, Shahram Izadi, and Andrew Fitzgibbon. 2012. KinÊtre: animating the world with the human body. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*. 435–444.
- [17] Xinghao Chen, Hengkai Guo, Guijin Wang, and Li Zhang. 2017. Motion feature augmented recurrent neural network for skeleton-based dynamic hand gesture recognition. In *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2881–2885.
- [18] Chiho Choi, Sang Ho Yoon, Chin-Ning Chen, and Karthik Ramani. 2017. Robust hand pose estimation during the interaction with an unknown object. In *Proceedings of the IEEE International Conference on Computer Vision*. 3123–3132.
- [19] Anind K Dey, Raffay Hamid, Chris Beckmann, Ian Li, and Daniel Hsu. 2004. a CAPpella: programming by demonstration of context-aware applications. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 33–40.
- [20] Mahmoud Elmezzain, Ayoub Al-Hamadi, Jorg Appenrodt, and Bernd Michaelis. 2008. A hidden markov model-based continuous gesture recognition system for hand motion trajectory. In *2008 19th International Conference on Pattern Recognition*. IEEE, 1–4.
- [21] Barrett Ens, Fraser Anderson, Tovi Grossman, Michelle Annett, Pourang Irani, and George Fitzmaurice. 2017. Ivy: Exploring spatially situated visual programming for authoring and understanding intelligent environments. In *Proceedings of the 43rd Graphics Interface Conference*. 156–162.
- [22] FinalIK 2021. FinalIK: Final IK - RootMotion. <http://www.root-motion.com/final-ik.html>.
- [23] Markus Funk, Mareike Kritzler, and Florian Michahelles. 2017. HoloLens is more than air Tap: natural and intuitive interaction with holograms. In *Proceedings of the seventh international conference on the internet of things*. 1–2.
- [24] Terrell Glenn, Ananya Ipsita, Caleb Carithers, Kylie Peppler, and Karthik Ramani. 2020. StoryMakAR: Bringing Stories to Life With An Augmented Reality & Physical Prototyping Toolkit for Youth. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–14.
- [25] Celeste Groenewald, Craig Anslow, Junayed Islam, Chris Rooney, Peter J Passmore, and BL Wong. 2016. Understanding 3D mid-air hand gestures with interactive surfaces and displays: a systematic literature review. (2016).
- [26] Sinem Güven and Steven Feiner. 2003. Authoring 3D hypermedia for wearable augmented and virtual reality. In *Proceedings of IEEE International Symposium on Wearable Computers (ISWC'03)*. 21–23.
- [27] Björn Hartmann, Leith Abdulla, Manas Mittal, and Scott R Klemmer. 2007. Authoring sensor-based interactions by demonstration with direct manipulation and pattern recognition. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 145–154.
- [28] Robert Held, Ankit Gupta, Brian Curless, and Maneesh Agrawala. 2012. 3D puppetry: a kinect-based interface for 3D animation.. In *UIST*, Vol. 12. Citeseer, 423–434.
- [29] Valentin Heun, James Hobin, and Pattie Maes. 2013. Reality editor: programming smarter objects. In *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*. 307–310.
- [30] Otmar Hilliges, David Kim, Shahram Izadi, Malte Weiss, and Andrew Wilson. 2012. HoloDesk: direct 3d interactions with a situated see-through display. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*.

- 2421–2430.
- [31] Hololens 2 2021. Hololens 2: Mixed Reality Technology for Business. <https://www.microsoft.com/en-us/hololens>.
- [32] Zhanpeng Huang, Weikai Li, and Pan Hui. 2015. Ubii: Towards seamless interaction between digital and physical worlds. In *Proceedings of the 23rd ACM international conference on Multimedia*. 341–350.
- [33] Ke Huo and Karthik Ramani. 2016. Window-Shaping: 3D Design Ideation in Mixed Reality. In *Proceedings of the 2016 Symposium on Spatial User Interaction*. 189–189.
- [34] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, et al. 2011. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. 559–568.
- [35] Rubaiat Habib Kazi, Fanny Chevalier, Tovi Grossman, and George Fitzmaurice. 2014. Kitty: sketching dynamic and interactive illustrations. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*. 395–405.
- [36] Annie Kelly, R Benjamin Shapiro, Jonathan de Halleux, and Thomas Ball. 2018. ARcadia: A rapid prototyping platform for real-time tangible interfaces. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–8.
- [37] Jun-Sik Kim, MyungHwan Jeon, and Jung-Min Park. 2019. Multi-Hand Direct Manipulation of Complex Constrained Virtual Objects. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 3235–3240.
- [38] Yongkwan Kim and Seok-Hyung Bae. 2016. SketchingWithHands: 3D sketching handheld products with first-person hand posture. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*. 797–808.
- [39] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. 2015. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, Vol. 2. Lille.
- [40] Sinisa Kolaric, Alberto Raposo, and Marcelo Gattass. 2008. Direct 3D manipulation using vision-based recognition of uninstrumented hands. In *X Symposium on Virtual and Augmented Reality*. Citeseer, 212–220.
- [41] Tobias Langlotz, Stefan Mooslechner, Stefanie Zollmann, Claus Degendorfer, Gerhard Reitmayr, and Dieter Schmalstieg. 2012. Sketching up the world: in situ authoring for mobile augmented reality. *Personal and ubiquitous computing* 16, 6 (2012), 623–630.
- [42] David Ledo, Jo Vermeulen, Sheelagh Carpendale, Saul Greenberg, Lora Oehlberg, and Sebastian Boring. 2019. Astral: Prototyping Mobile and Smart Object Interactive Behaviours Using Familiar Applications. In *Proceedings of the 2019 on Designing Interactive Systems Conference*. 711–724.
- [43] Bokyoung Lee, Minjoo Cho, Joonhee Min, and Daniel Saakes. 2016. Posing and acting as input for personalizing furniture. In *Proceedings of the 9th Nordic Conference on Human-Computer Interaction*. 1–10.
- [44] Gun A Lee, Gerard J Kim, and Mark Billinghurst. 2005. Immersive authoring: What you experience is what you get (wxyxiwyg). *Commun. ACM* 48, 7 (2005), 76–81.
- [45] Gun A Lee, Claudia Nelles, Mark Billinghurst, and Gerard Jounghyun Kim. 2004. Immersive authoring of tangible augmented reality applications. In *Third IEEE and ACM International Symposium on Mixed and Augmented Reality*. IEEE, 172–181.
- [46] Jae Yeol Lee, Gue Won Rhee, and Dong Woo Seo. 2010. Hand gesture-based tangible interactions for manipulating virtual objects in a mixed reality environment. *The International Journal of Advanced Manufacturing Technology* 51, 9–12 (2010), 1069–1082.
- [47] Minkyung Lee, Mark Billinghurst, Woonhyuk Baek, Richard Green, and Woon-tack Woo. 2013. A usability study of multimodal input in an augmented reality environment. *Virtual Reality* 17, 4 (2013), 293–305.
- [48] Minkyung Lee, Richard Green, and Mark Billinghurst. 2008. 3D natural hand interaction for AR applications. In *2008 23rd International Conference Image and Vision Computing New Zealand*. IEEE, 1–6.
- [49] Germán Leiva and Michel Beaudouin-Lafon. 2018. Montage: a video prototyping system to reduce re-shooting and increase re-usability. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. 675–682.
- [50] Germán Leiva, Cuong Nguyen, Rubaiat Habib Kazi, and Paul Asente. 2020. Pronto: Rapid Augmented Reality Video Prototyping Using Sketches and Enaction. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–13.
- [51] Yingjiang Li, Jianhong Sun, and Rui Li. 2016. Human Action Recognition Based on Dynamic Time Warping and Movement Trajectory. *International Journal of Simulation-Systems, Science & Technology* 17, 46 (2016).
- [52] Henry Lieberman, Fabio Paternò, Markus Klann, and Volker Wulf. 2006. End-user development: An emerging paradigm. In *End user development*. Springer, 1–8.
- [53] Hao Lü and Yang Li. 2012. Gesture coder: a tool for programming multi-touch gestures by demonstration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2875–2884.
- [54] Hao Lü and Yang Li. 2013. Gesture studio: Authoring multi-touch interactions through demonstration and declaration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 257–266.
- [55] Zhihan Lv, Alaa Halawani, Shengzhong Feng, Shafiq Ur Rehman, and Haibo Li. 2015. Touch-less interactive augmented reality game on vision-based wearable device. *Personal and Ubiquitous Computing* 19, 3 (2015), 551–567.
- [56] Chunyong Ma, Shengsheng Zhang, Anni Wang, Yongyang Qi, and Ge Chen. 2020. Skeleton-based dynamic hand gesture recognition using an enhanced network with one-shot learning. *Applied Sciences* 10, 11 (2020), 3680.
- [57] MagicLeapMica 2021. Magic Leap: I am mica. <https://www.magicleap.com/en-us/news/op-ed/i-am-mica>.
- [58] Atsushi Matsubayashi, Yasutoshi Makino, and Hiroyuki Shinoda. 2019. Direct finger manipulation of 3d object image with ultrasound haptic feedback. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–11.
- [59] Daniel Mendes, Fernando Fonseca, Bruno Araujo, Alfredo Ferreira, and Joaquim Jorge. 2014. Mid-air interactions above stereoscopic interactive tables. In *2014 IEEE Symposium on 3D User Interfaces (3DUI)*. IEEE, 3–10.
- [60] Byung-Woo Min, Ho-Sub Yoon, Jung Soh, Yun-Mo Yang, and Toshiaki Ejima. 1997. Hand gesture recognition using hidden Markov models. In *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, Vol. 5. IEEE, 4232–4235.
- [61] MRTK 2021. MRTK: MRTK-Unity Developer Documentation. <https://docs.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/>.
- [62] Franziska Mueller, Dushyant Mehta, Oleksandr Sotnychenko, Srinath Sridhar, Dan Casas, and Christian Theobalt. 2017. Real-time hand tracking under occlusion from an egocentric rgb-d sensor. In *Proceedings of the IEEE International Conference on Computer Vision*. 1154–1163.
- [63] Mathieu Nancel, Julie Wagner, Emmanuel Pietriga, Olivier Chapuis, and Wendy Mackay. 2011. Mid-air pan-and-zoom on wall-sized displays. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 177–186.
- [64] Michael Nebeling and Katy Madier. 2019. 360Pro: Making interactive virtual reality & augmented reality prototypes from paper. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–13.
- [65] Michael Nebeling, Janet Nebeling, Ao Yu, and Rob Rumble. 2018. Protoar: Rapid physical-digital prototyping of mobile augmented reality applications. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–12.
- [66] Michael Nebeling and Maximilian Speicher. 2018. The trouble with augmented reality/virtual reality authoring tools. In *2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*. IEEE, 333–337.
- [67] Andrew YC Nee, SK Ong, George Chryssolouris, and Dimitris Mourtzis. 2012. Augmented reality applications in design and manufacturing. *CIRP annals* 61, 2 (2012), 657–679.
- [68] Gary Ng, Joon Gi Shin, Alexander Plopski, Christian Sandor, and Daniel Saakes. 2018. Situated game level editing in augmented reality. In *Proceedings of the Twelfth International Conference on Tangible, Embedded, and Embodied Interaction*. 409–418.
- [69] SK Ong and ZB Wang. 2011. Augmented assembly technologies based on 3D bare-hand interaction. *CIRP annals* 60, 1 (2011), 1–4.
- [70] Jong-Seung Park. 2011. AR-Room: a rapid prototyping framework for augmented reality applications. *Multimedia tools and applications* 55, 3 (2011), 725–746.
- [71] Viet Toan Phan and Seung Yeon Choo. 2010. Interior design in augmented reality environment. *International Journal of Computer Applications* 5, 5 (2010), 16–21.
- [72] Wayne Piekarski and Bruce Thomas. 2002. ARQuake: the outdoor augmented reality gaming system. *Commun. ACM* 45, 1 (2002), 36–38.
- [73] Wayne Piekarski and Bruce H Thomas. 2002. Using ARToolKit for 3D hand position tracking in mobile outdoor environments. In *The First IEEE International Workshop Augmented Reality Toolkit*. IEEE, 2–pp.
- [74] Thammathip Piumsomboon, David Altimira, Hyungon Kim, Adrian Clark, Gun Lee, and Mark Billinghurst. 2014. Grasp-Shell vs gesture-speech: A comparison of direct and indirect natural interaction techniques in augmented reality. In *2014 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE, 73–82.
- [75] Thammathip Piumsomboon, Adrian Clark, Mark Billinghurst, and Andy Cockburn. 2013. User-defined gestures for augmented reality. In *IFIP Conference on Human-Computer Interaction*. Springer, 282–299.
- [76] David Porfirio, Allison Sauppé, Aws Albarghouthi, and Bilge Mutlu. 2018. Authoring and verifying human-robot interactions. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. 75–86.
- [77] Victor Adrian Prisacariu, Olaf Kähler, Stuart Golodetz, Michael Sapienza, Tommaso Cavallari, Philip HS Torr, and David W Murray. 2017. Infinitam v3: A framework for large-scale 3d reconstruction with loop closure. *arXiv preprint arXiv:1708.00783* (2017).
- [78] PyTorch 2021. PyTorch. <https://pytorch.org/>.

- [79] Jing Qian, Jiaju Ma, Xiangyu Li, Benjamin Attal, Haoming Lai, James Tompkin, John F Hughes, and Jeff Huang. 2019. Portal-ble: Intuitive free-hand manipulation in unbounded smartphone-based augmented reality. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*. 133–145.
- [80] Matthias Schwaller, Simon Brunner, and Denis Lalanne. 2013. Two handed mid-air gestural hci: Point+ command. In *International Conference on Human-Computer Interaction*. Springer, 388–397.
- [81] Hartmut Seichter, Julian Looser, and Mark Billinghurst. 2008. ComposAR: An intuitive tool for authoring AR applications. In *2008 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*. IEEE, 177–178.
- [82] Jinwook Shim, Yoonsik Yang, Nahyung Kang, Jonghoon Seo, and Tack-Don Han. 2016. Gesture-based interactive augmented reality content authoring system using HMD. *Virtual Reality* 20, 1 (2016), 57–69.
- [83] Deepanjali Shrestha, Hyungwoo Lee, and Junchul Chun. 2018. Computer-vision-based bare-hand augmented reality interface for controlling an AR object. *International Journal of Computer Aided Engineering and Technology* 10, 3 (2018), 257–265.
- [84] SonyAibo 2021. Sony: aibo. <https://us.aibo.com/>.
- [85] Nur SyafiqahSafiee and Ajune Wanis Ismail. 2018. Ar home deco: virtual object manipulation technique using hand gesture in augmented reality. *Innovations in Computing Technology and Applications* 3 (2018).
- [86] ultraleap 2021. Tracking: Leaping Motion Controller. <https://www.ultraleap.com/product/leap-motion-controller/>.
- [87] Unity 2021. Unity: Real-Time Development Platform. <https://www.unity.com>.
- [88] UnrealEngine 2021. UnrealEngine: The most powerful real-time 3D creation platform. <https://www.unrealengine.com/en-US/>.
- [89] Ana Villanueva, Zhengzhe Zhu, Ziyi Liu, Kylie Pepler, Thomas Redick, and Karthik Ramani. 2020. Meta-AR-app: an authoring platform for collaborative augmented reality in STEM classrooms. In *Proceedings of the 2020 CHI conference on human factors in computing systems*. 1–14.
- [90] Christian Von Hardenberg and François Bérard. 2001. Bare-hand human-computer interaction. In *Proceedings of the 2001 workshop on Perceptive user interfaces*. 1–8.
- [91] Tianyi Wang, Xun Qian, Fengming He, Xiyun Hu, Ke Huo, Yuanzhi Cao, and Karthik Ramani. 2020. CAPtURAR: An Augmented Reality Tool for Authoring Human-Involved Context-Aware Applications. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 328–341.
- [92] Matt Whitlock, Jake Mitchell, Nick Pfeufer, Brad Arnot, Ryan Craig, Bryce Wilson, Brian Chung, and Danielle Albers Szafr. 2020. MRCAT: In Situ Prototyping of Interactive AR Environments. In *International Conference on Human-Computer Interaction*. Springer, 235–255.
- [93] Adam S Williams, Jason Garcia, and Francisco Ortega. 2020. Understanding Multimodal User Gesture and Speech Behavior for Object Manipulation in Augmented Reality Using Elicitation. *IEEE Transactions on Visualization and Computer Graphics* 26, 12 (2020), 3479–3489.
- [94] Graham Wilson, Thomas Carter, Sriram Subramanian, and Stephen A Brewster. 2014. Perception of ultrasonic haptic feedback on the hand: localisation and apparent motion. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1133–1142.
- [95] Jacob O Wobbrock, Meredith Ringel Morris, and Andrew D Wilson. 2009. User-defined gestures for surface computing. In *Proceedings of the SIGCHI conference on human factors in computing systems*. 1083–1092.
- [96] Yukang Yan, Chun Yu, Xiaojuan Ma, Xin Yi, Ke Sun, and Yuanchun Shi. 2018. Virtualgrasp: Leveraging experience of interacting with physical objects to facilitate digital object retrieval. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–13.
- [97] Geng Yang, Honghao Lv, Feiyu Chen, Zhibo Pang, Jin Wang, Huayong Yang, and Junhui Zhang. 2018. A novel gesture recognition system for intelligent interaction with a nursing-care assistant robot. *Applied Sciences* 8, 12 (2018), 2349.
- [98] Hui Ye, Kin Chung Kwan, Wanchao Su, and Hongbo Fu. 2020. ARAnimator: in-situ character animation in mobile AR with user-defined motion gestures. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 83–1.
- [99] Shahrouz Yousefi, Mhretab Kidane, Yeray Delgado, Julio Chana, and Nico Reski. 2016. 3D gesture-based interaction for immersive experience in mobile VR. In *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2121–2126.
- [100] Run Yu and Doug A Bowman. 2018. Force push: Exploring expressive gesture-to-force mappings for remote object manipulation in virtual reality. *Frontiers in ICT* 5 (2018), 25.
- [101] Ya-Ting Yue, Yong-Liang Yang, Gang Ren, and Wenping Wang. 2017. SceneCtrl: Mixed reality enhancement via efficient scene editing. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. 427–436.
- [102] Bruno Zamborlin, Frederic Bevilacqua, Marco Gillies, and Mark D’Inverno. 2014. Fluid gesture interaction design: Applications of continuous recognition for the design of modern gestural interfaces. *ACM Transactions on Interactive Intelligent Systems (TiiS)* 3, 4 (2014), 1–30.
- [103] Jürgen Zauner, Michael Haller, Alexander Brandl, and Werner Hartman. 2003. Authoring of a mixed reality assembly instructor for hierarchical structures. In *The Second IEEE and ACM International Symposium on Mixed and Augmented Reality, 2003. Proceedings*. IEEE, 237–246.
- [104] ZED Mini 2021. ZED Mini: Mixed Reality Camera. <https://www.unity.com>.
- [105] Lei Zhang and Steve Oney. 2020. FlowMatic: An Immersive Authoring Tool for Creating Interactive Scenes in Virtual Reality. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 342–353.