



ProInterAR: A Visual Programming Platform for Creating Immersive AR Interactions

Hui Ye*
School of Creative Media,
City University of Hong Kong
Hong Kong, China
huiye4@cityu.edu.hk

Jiaye Leng*
School of Creative Media,
City University of Hong Kong
Hong Kong, China
jiayeleng2-c@my.cityu.edu.hk

Pengfei Xu
College of Computer Science and
Software Engineering,
Shenzhen University
Shenzhen, Guangdong, China
xupengfei.cg@gmail.com

Karan Singh
Department of Computer Science,
University of Toronto
Toronto, Ontario, Canada
karan@dgp.toronto.edu

Hongbo Fu†
School of Creative Media,
City University of Hong Kong
Hong Kong, China
hongbofu@cityu.edu.hk



Figure 1: *ProInterAR* is an integrated visual programming platform for creating immersive AR interactions with a tablet and an AR-HMD device. (a)-(b) The AR creator specifies physical contents or creates virtual contents (i.e., a physical white box and a virtual tree) from the AR-HMD. (c)-(d) The creator programs the interactive behaviors of the created AR contents from a block-based visual programming interface in the tablet. (e) The creator executes, watches, and controls the programmed AR application in the AR scene. In this example, an AR game “pat and bounce” is created using *ProInterAR*: The player first uses his hands to pat a virtual ball to make it move in the AR space continuously. It will bounce when it collides with the AR contents (i.e., physical walls, ground, a physical box, a physical chair, and a virtual tree). If the player catches and pats the ball once by hand, the score will be added by one.

ABSTRACT

AR applications commonly contain diverse interactions among different AR contents. Creating such applications requires creators to have advanced programming skills for scripting interactive behaviors of AR contents, repeated transferring and adjustment of virtual contents from virtual to physical scenes, testing by traversing between desktop interfaces and target AR scenes, and digitalizing AR contents. Existing immersive tools for prototyping/authoring such interactions are tailored for domain-specific applications. To

support programming general interactive behaviors of real object(s)/environment(s) and virtual object(s)/environment(s) for novice AR creators, we propose *ProInterAR*, an integrated visual programming platform to create immersive AR applications with a tablet and an AR-HMD. Users can construct interaction scenes by creating virtual contents and augmenting real contents from the view of an AR-HMD, script interactive behaviors by stacking blocks from a tablet UI, and then execute and control the interactions in the AR scene. We showcase a wide range of AR application scenarios enabled by *ProInterAR*, including AR game, AR teaching, sequential animation, AR information visualization, etc. Two usability studies validate that novice AR creators can easily program various desired AR applications using *ProInterAR*.

*Both authors contributed equally to this research.

†Corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CHI '24, May 11–16, 2024, Honolulu, HI, USA

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-0330-0/24/05

<https://doi.org/10.1145/3613904.3642527>

CCS CONCEPTS

• Human-centered computing → Ubiquitous and mobile computing systems and tools; Mixed / augmented reality; Graphical user interfaces; User interface toolkits.

KEYWORDS

AR contents, AR interactions, Visual programming

ACM Reference Format:

Hui Ye, Jiaye Leng, Pengfei Xu, Karan Singh, and Hongbo Fu. 2024. *ProInterAR: A Visual Programming Platform for Creating Immersive AR Interactions*. In *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI '24)*, May 11–16, 2024, Honolulu, HI, USA. ACM, New York, NY, USA, 15 pages. <https://doi.org/10.1145/3613904.3642527>

1 INTRODUCTION

Augmented Reality (AR) seamlessly blends the virtual and physical realms, enhancing our perception and providing immersive experiences that were once confined to science fiction. From entertainment and gaming [30, 33] to education [43], healthcare [51], and industry [10], AR applications have the potential to transform numerous sectors and redefine the way we live, work, and play [6].

AR applications involve numerous types of interactions of both real and virtual AR contents in dynamic or static forms. Currently, a typical workflow to create AR applications usually consists of four steps: 1) conceptualize and plan basic ideas; 2) build AR contents by augmenting physical elements with virtual elements; 3) design user interfaces and AR interactions required for users to interact with the contents; 4) script the controlling and reactive behaviors of the AR contents in game engines and frameworks (e.g., Unity, Unreal, ARKit, ARCore). One significantly difficult step for inexperienced creators without strong programming background is to script AR interactions [20] since it requires advanced low-level coding skills and heavy coding tasks. In virtue of visual scripting [26], some desktop-based tools (e.g., Unreal Blueprints and Lens Studio) enable users to script AR experience using node-based programming. Although they relieve the programming burden, they require the AR contents to be placed first in a virtual scene or template and then transferred to a physical scene with repeated adjustment. Besides, they require creators to traverse between programming on desktop interfaces and testing on target AR scenes from mobile devices, limiting flexibility [2]. On the other hand, AR contents and their properties (e.g., motion, appearance) need to be obtained or used during programming. Virtual contents can be accessed easily in virtual representations, while real-world contents need to be digitized for access by desktop-based programming interfaces and scripted behavior transfer in AR scenes. How to smoothly fuse the AR contents in programming remains a question for novice AR creators.

To make the creation of AR applications to be more in situ, researchers have proposed various immersive toolkits for authoring and prototyping AR interactions. Such toolkits allow users to specify reactive behaviors of AR contents via visual programming workflows [39, 40, 48, 52]. However, these works have mainly focused on domain-specific interaction tasks (e.g., gestural interaction [39], toy-based interaction [52], spatially-aware interaction [48], human-centered context-aware interaction [40]). For more general AR interactions, the behaviors of the real and virtual contents might contain many properties and variations to be controlled, so it is hard to achieve them using existing toolkits. Recent works like LearnIoTVR [53] and FlowMatic [50] are proposed for creating general VR applications. However, they focus on scripting interactions

among virtual contents, while we aim to handle AR scenes containing both real and virtual contents and more complex interactive paradigms, especially between the real and virtual contents.

There are still very few tools for supporting novice AR creators to create general AR applications. One of the exceptions is ARcadia [19], which presents a prototyping platform for a real-time tangible interface based on block-based visual programming. However, it employs marker-based AR through a laptop camera with a fixed viewpoint while we want to utilize the advantages of markerless AR based on AR-HMD (Augmented Reality Head Mounted Display) to provide a more flexible and immersive interface for scripting 3D behaviors of AR contents tightly coupled with users' surroundings. In this work, we first discuss the design scope of general AR interactions. Based on it, we propose *ProInterAR*, a block-based visual programming system to create immersive AR interactions, for novice AR creators who have some background in programming but limited experience in creating AR applications.

To achieve in-situ and portable authoring experience and relieve hand fatigue, we provide user interfaces (UIs) with integrated devices: a visual programming UI from a tablet browser, a scene creation, execution, and controlling UI from an AR-HMD. Creators can construct interaction scenes by creating AR contents from the view of the AR-HMD, script the interactive behaviors by stacking blocks from the tablet UI, and then execute and view the scripted interactions by controlling the contents and detecting the interactions in the AR scene. We design the programming blocks based on Scratch [32], which can be dragged, stacked, and grouped to any created/specified AR contents. Various types of interactions can be implemented with a diverse range of blocks. We demonstrate four application scenarios (i.e., AR game, AR teaching, sequential animation, AR information visualization) to show its expressiveness and usability. We conducted two usability studies with AR creators to evaluate the usefulness of *ProInterAR*. From the studies, we find that novice AR creators can easily create diverse AR applications with varying complexity using our system.

In summary, our work makes the following contributions:

- A design scope that describes general AR interactions.
- A visual programming toolkit integrating a tablet browser and an AR-HMD for creating general AR interactions.
- A demonstration of application scenarios.
- Two usability studies to validate the usefulness of the proposed system.

2 RELATED WORK

2.1 Visual Programming Approaches and Interfaces

To program interactive behaviors, developers usually use traditional text-based programming interfaces to express the logic, iteration, and operation. However, it requires a steep learning curve and a solid understanding of syntax and coding structures. Visual programming paradigms [4, 26] are proposed to lower the entry barriers to non-experienced developers, with the forms of flowchart-based programming [5, 15, 49], block-based programming [17, 32, 53], state-machine programming [3, 21], etc. As a

representative type of flowchart-based programming, node-graph-based programming is widely adopted in prototyping complex program logic and data processing pipelines for various applications, e.g., graphical applications [37], AR/VR applications [31, 34, 50], and machine learning tasks [9]. Alternatively, block-based programming [13, 19, 24, 32] is considered to be easy for beginners [1, 41, 53] by utilizing simple and self-contained blocks and drag-and-drop interfaces. Among them, Scratch [32] is one of the popularly used tools for kids and beginners to create interactive 2D graphical applications, e.g., stories, animations, games. We share a similar goal with Scratch – to allow non-professional creators to create interactive applications easily, but differently – to script immersive 3D AR applications. Thus, we carefully design our visual programming interface based on Scratch.

2.2 Toolkits of Developing AR/VR Applications

Currently, professional creators usually resort to 3D game engines such as Unreal and Unity to create AR/VR applications by building scenes and writing scripts using the programming language supported by a chosen engine. Creators must implement object behaviors, user interactions, gameplay mechanics, and any custom functionalities required for the AR/VR experience. However, it requires extensive low-level programming.

To address this issue, researchers have explored the use of different types of visual programming interfaces with lower entry barriers in building AR/VR applications [11, 14, 17, 19, 50]. Commercial tools [34, 38] with rich media integration employ node-based systems and allow users to connect nodes that represent specific actions, operations, or functions to define the logic and interactive behavior. However, these tools require users to first build scenes and script programs on desktop interfaces and then deploy the applications on target AR/VR-enabled devices, and place AR contents first on virtual scenes or templates and then convert and adjust them to physical scenes repeatedly. Meanwhile, users must map the real-world contents and their digital representations in the authoring tools. The gap between the target scene and the developing platform induces in-situ programming interfaces. For example, an early work, Smarter Objects [14], introduces an initial prospect in designing a tablet-based AR interface to program the functionality of physical objects and virtual interactions by connecting lines. We also utilize a tablet-based visual programming interface, but differently, targeting immersive AR scenes for more general 3D interactions. ARcadia [19] presents a block programming interface to define and execute interactive behaviors between tangible objects and UI elements in marker-based AR scenes. It is built on a desktop with a camera with limited FoV. In contrast, our markerless AR-HMD system enable users to author and test AR interactions from an AR-HMD and program on a tablet, thus allowing users to experience unconstrained perspectives while keeping free traverse between two devices. BlocklyXR [17] adopts a block-based programming approach to allow general users to design storytelling. Compared to these works, we aim to support the programming of 3D interactions of various physical/virtual objects/environments controlled from an AR-HMD, especially for the close interactions between the virtual and physical elements. Besides, our system supports hand-based interactions such as hand grasping and collision.

For VR applications, FlowMatic [50] and XRSpotlight [11] utilize functional reactive programming and programming with examples, respectively, to help developers build VR interactions. While they provide much control to event- and example-based interactions for virtual objects, our block-based programming UI introduces more basic programming concepts with flexible combination and event listening for both virtual and physical objects.

2.3 AR Authoring/Prototyping Interfaces for Contents and Interactions

Instead of programming interfaces, recent research has focused more on exploring interactive techniques for authoring and prototyping AR contents and interactions. Various types of interactions can occur between real and virtual contents, such as responsive and interactive visualization and animation between humans, tangible objects, sketches, and animated characters [18, 27, 29, 36, 44, 47], freehand AR interactions with digital contents [8, 39], spatially-aware interactions among physical objects and humans [46, 48], etc. Although most existing works propose novel authoring workflows for creating AR experiences and interactions for end-users/designers, they support creating domain-specific AR applications. More general and customized interactions that contain multiple control (e.g., conditions, iterations, loops, time waiting) are not well supported. For example, CAPturAR [40] provides an AR programming interface to author human-involved context-aware interactions. Instead of human-centered behavior, we want to support more general AR interactions happening with various real (including a user’s hands and head) and virtual contents. From the design goal’s perspective, CAPturAR aims to provide a high-level rule-based authoring tool for building personalized daily applications based on historical context data, while ours offers a grounded programming tool to script various AR interactions from the basic function combination. ProObjAR [48] allows users to design spatial interactions of physical objects in an event-triggering workflow, but it does not support the design of linked interactions that involve multiple event-triggering mechanisms. In AR applications, the interactive behavior of one object might work as the triggering condition of another object’s behavior. The object behaviors might have different levels of variations to be controlled. However, the existing works can only author the pre-defined interaction workflows. To fill in these gaps, we propose a programming toolkit that can 1) offer more control over the behavior properties and variations and 2) handle different levels of control (e.g., iterations, loops) of/between the behaviors of the object(s).

3 DESIGN SCOPE OF GENERAL AR INTERACTIONS

Since most of the existing AR authoring tools are tailored for specific tasks, there is no framework discussing the possibilities and boundaries of general AR interactions. In this section, we develop a design scope that describes general AR interactions from multiple dimensions, and discuss how ours and the closely related works support these dimensions.

Subject. AR contents can be divided into four subject types: real objects (e.g., tangible items, human body parts, animals), virtual objects (e.g., 3D models, visual effects and filters, annotations), real

Table 1: Design scope of AR interactions, and how the closely related works cover the dimensions. RO, VO, RE, and VE refer to real objects, virtual objects, real environments, and virtual environments, respectively.

		ProInterAR	CAPTurAR[40]	ProObjAR[48]	GesturAR[39]	Pronto[22]	MechARSpace[52]	Rapido[21]	Arcadia[19]	RealityCanvas[44]	ARMath[18]	Teachable Reality[25]
Subject	RO	■	■	■	■	■	■				■	■
	VO	■	■		■	■				■	■	
	RO - VO	■	■		■	■	■			■	■	■
	RO - RE	■	■	■	■	■						■
	RO - VE	■	■									
	VO - RE	■	■			■		■				■
	VO - VE	■	■									
Scenario	Domain-specific		■	■	■	■	■	■		■	■	■
	General	■				■			■		■	■
Concurrency	Serial		■	■	■	■	■	■		■	■	■
	Parallel	■				■		■	■			
Logic	Simple			■	■	■	■			■	■	■
	Complex	■	■					■	■		■	■

environments (e.g., walls, grounds), and virtual environments (e.g., virtual surfaces). Environments provide a broader spatial context in which the interaction is situated than objects. Any AR interaction can be decomposed into basic interactions happening between two subjects of the same or different type(s). Removing meaningless combinations of two subjects, we list seven combinations in Table 1 within the subject dimension. Most existing works focus on authoring the interactions of real or virtual objects. In these works, real environments mainly serve as a contextual background [21, 22, 40], which does not involve direct interactions with other AR contents. Our system aims to enable both the environments and objects with direct interaction capacities, such as a human evading moving virtual walls, and a ball falling to the ground and bouncing up.

Scenario. AR interactions are designed to meet the needs of different usage scenarios. Most of the existing works explore AR interactions in certain domains of usage scenarios by adopting a certain type of AR contents, such as context-aware human-centered behavior [40], spatially-aware object interactions [48], freehand interactions [39], toy-based interactions [52], mobile AR prototypes [21], etc. Interactions in more general usage scenarios such as prototyping AR experience [22] and tangible AR [19, 25] involve broader paradigms. Our system aims to support general interactions (i.e., spatial, hand-based, exterior, and geometric interactions) of broader types of AR contents.

Concurrency. From the time perspective, AR interactions can happen serially or parallelly. In the serial mode, one interaction follows the completion of its previous one. This sequential nature ensures that each interaction receives the necessary attention before moving on to the next. On the other hand, in the parallel mode, AR interactions can occur concurrently, enabling multiple interactions to occur simultaneously. This parallelism leads to a more dynamic and interactive AR experience. The works relying on a trigger-action workflow [39, 40, 48, 52] usually support serial interactions. Video-flow-based prototyping [21, 22, 44] can support simultaneous interactions ideally, but they mainly aim to author a series of interaction flows along the video flow. We design our system to program interactive behaviors for individual AR contents so that the interactions can happen independently, enabling both serial and parallel patterns.

Logic. An AR interaction can encompass a range of logic flows, varying from simple to complex. In simpler cases, the logic flow may involve straightforward and linear sequences of steps such as

a single trigger-action interaction [39, 48, 52]. However, AR interactions can also feature complex logic flows incorporating branching, loops, iterations, time control, and multiple conditional statements. For instance, an AR interaction might involve recognizing different human behaviors [40], each triggering a specific set of effects. We want to support programming interactions with complex logic controls.

4 SYSTEM DESIGN

4.1 Overview

To support the programming of the general AR interactions discussed above, we propose *ProInterAR*, a block-based visual programming system implemented in an integrated tablet and AR-HMD interface. An iPad Air 4 and a Microsoft HoloLens 2 are used to deploy the application. The presented ideas can be easily ported to other combinations of tablets (e.g., Microsoft Surface Pro) and AR-HMD devices (e.g., Apple Vision Pro). *ProInterAR* consists of three main components: a scene creation UI (Section 4.2), a visual programming UI (Section 4.3), and an execution and controlling UI (Section 4.4). We design our visual programming UI in the tablet, mainly considering the hand fatigue issue of immersive programming. The system is designed based on the imperative programming concept [12]. We will first introduce these three components and UIs and then use an example (Section 4.5) to illustrate the programming workflow.

4.2 Scene Creation UI

Interactive Content Selection/Creation. To start developing an AR application, the creator first needs to create the contents to be interacted with. Based on the design scope discussed above, we allow users to create AR contents from four categories: real-world objects, real-world environments, virtual objects, and virtual environments.

Real-world Objects. A real-world object needs to be tracked with a 6-DoF pose for interaction. Since the limited computation capacity of HoloLens 2 does not support real-time estimation of a 3D bounding box of arbitrary objects, we adopt an interactive approach for manually initiating an object’s bounding box. The user drags a 3D bounding box to a real-world object of interest and adjusts its volume and direction using pinch gestures (Figure 2(a)). Once finished, a real-world object with its position and orientation is created in the system and passed to the programming UI for access. Since the user can manipulate the real-world object by hand, we

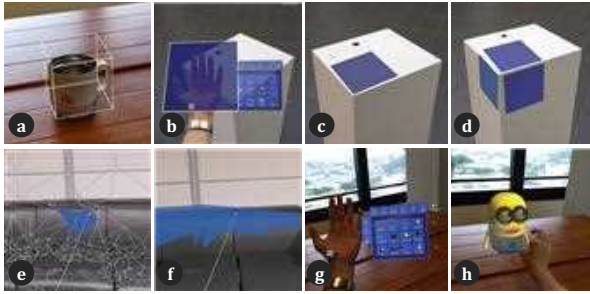


Figure 2: Create or specify AR contents for interaction. (a) Create a bounding box to specify a real-world object. (b)-(c) Create a plane using a palm gesture to specify a planar real-world environment. (d) Create multiple planes and group them together to specify a compound real-world environment. (e)-(f) Select a part of the detected spatial mesh to specify a curved-surfaced or customized real-world environment. (g) The object library contains various commonly used virtual contents. (h) Drag and drop a virtual model from the library and transform it using hand gestures.

track its dynamic 6-DoF pose according to the hand pose due to the relative rest state between the object and the hand. Two specific types of real-world objects, i.e., hands and head, are also in the content lists for users to select in our system. We extract the positions and orientations of hand objects (i.e., left-hand palm, right-hand palm, and 15-finger joints) and the head object from the HoloLens.

Real-world Environments. Different from real-world objects, real-world environments usually stay static. So, we only need to specify their initial states without tracking them later. Three techniques are supported to create a real-world environment for interaction. First, we allow users to create a plane using a hand palm gesture (Figure 2(b)) and transform it using a pinch gesture to overlay it with a physical environment (Figure 2(c)). It can turn a planar physical environment into an interactable object in the system. Second, we allow users to create multiple planes and group them to form a compound environment object (Figure 2(d)). It is applicable to the physical environment with multiple planes (e.g., a chair with vertical and horizontal planes). Third, for curved surfaces or other customized environments, users can use a hand-pointing gesture to select a part of a spatial mesh detected by HoloLens 2 (Figure 2(e)-(f)). Using these three techniques, an environment is specified and passed to the programming UI for selection.

Virtual Objects. We provide an object library that contains multiple commonly used virtual elements in AR applications (Figure 2(g)). This library can be easily expanded based on target applications. The user can drag and drop a virtual object to the AR scene and change its size, position, and orientation using a pinch gesture (Figure 2(h)). In addition, we allow users to import their customized virtual objects by uploading them from the interface of the tablet browser.

Virtual Environments. The user can use a similar method to real-world environment creation to create a virtual environment, or drag-and-drop an argument value like box zone to work as a virtual environment. The creation and deletion of all the contents are updated in real-time in the programming UI and the AR scene.

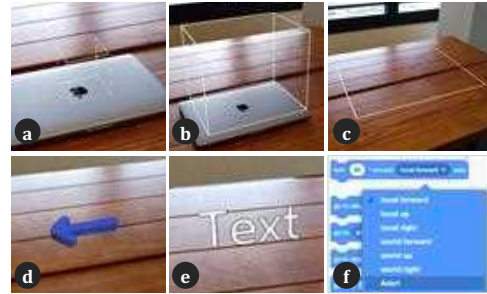


Figure 3: Create five types of argument values: (a) Create a small bounding box to specify a certain position, (b) Create a large bounding box to specify a certain box zone, (c) Create a plane to specify a certain plane zone, (d) Create an arrow to specify a certain axis, and (e) Create a text proxy to specify a certain text. (f) The argument option field of a block in the programming UI is updated when an “Axis1” is created.

Argument Value Creation. Besides the contents to be interacted with, some argument values also need to be created in advance in the AR scene and passed to blocks for further interaction. For example, when the user wants to program the behavior of a car rotating around a customized axis or the effect of a mole to appear at a random location within a customized zone in a whack-a-mole game, the axis and the zone need to be first created in the scene and then shown in the block argument field for specification. Our system supports the creation of five types of argument values (i.e., position, box zone, plane zone, axis, and text), as illustrated in Figure 3. The user can create a small bounding box that indicates a certain position (Figure 3(a)), a large bounding box indicating a certain box zone (Figure 3(b)), a plane representing a certain plane zone (Figure 3(c)), an arrow that represents a certain axis (Figure 3(d)), and a certain text (Figure 3(e)). Once an argument value is created, it will be passed to the argument option field for selection (Figure 3(f)).

4.3 Visual Programming UI

The UI running on the tablet provides a visual programming interface to specify the AR interactions of the created contents. Based on Scratch [32], we design the *ProInterAR* programming interface (Figure 4(a)). During the programming phase, the user can hold and operate on the tablet while wearing the AR-HMD to view the AR scene (Figure 4(c)). The UI consists of three main components: a block list, a block-stack field, and a content-selection list (Figure 4). The content-selection list shows all the created AR contents in the AR scene. Once an AR content is created, the user can rename it in this field.

Block Categories. According to the coding contents, the blocks are divided into the following nine categories, as illustrated in Figure 5 with different colors. We re-design the categories of motion, looks, and sensing to make them applicable for AR interactions in 3D space. We propose a new category “Hand” to enable hand-based interactions. The other categories are directly adopted from Scratch.

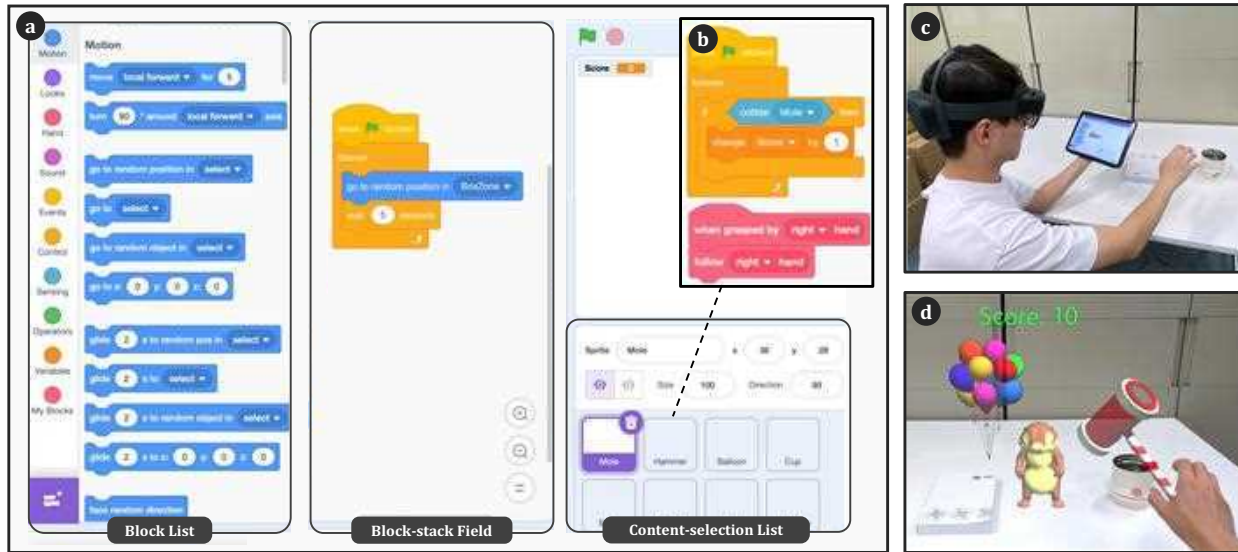


Figure 4: The visual programming UI and walk-through of *ProInterAR*. The programming UI (a) consists of a block list, a block-stack field, and a content-selection list. The user can drag the blocks from the block list and drop and stack them in the block-stack field. The created AR contents are shown in the content-selection list for selection. The user can press the green flag and the red button to execute and stop the program, respectively. The walk-through of a “3D Whack a Mole” example: The user first creates AR contents, which are shown in the content-selection list on the tablet UI (a). The user then stacks the blocks (c) for the corresponding contents: the random occurrence blocks for the mole object in the block-stack field (a), the hand grasp blocks ((b)-bottom) and collision detection blocks ((b)-up) for the hammer object, and the collision condition blocks for the balloon, cup, and book objects. The user views and tests the program by grasping the hammer to hit the mole and avoiding hitting other objects from the AR-HMD view (d).

- **Motion.** We design the motion blocks (Figure 5(a)) to enable spatial interactions: move, rotate, go to, glide to, face, set x/y/z to, if collide and bounce, obtaining content position, direction, distance to another content, and x/y/z position. Some of the blocks have argument fields to be input or selected. These individual motion blocks can be singly used or combined to cover most of the common spatial interactions.
- **Looks.** We provide the size-related (i.e., change or set size), presence-controlling (i.e., show or hide), and text (i.e., set text with color) blocks (Figure 5(b)) to control the appearance change.
- **Hand.** We design the blocks of grasping real or virtual objects, objects going to a certain hand finger joint or following hand, and performing certain hand gestures (Figure 5(c)) for hand-based interactions.
- **Sound.** We design the sound blocks of playing/stopping sounds and control the sound volume (Figure 5(d)).
- **Control.** We keep most of the controlling blocks (Figure 5(e)) from Scratch for achieving repetitions, conditionals, loops, iterations, and time waiting .
- **Events.** We keep the event blocks (Figure 5(f)) from Scratch to trigger the start of the program and enable event listening using broadcasting functions.
- **Sensing.** We design four sensing blocks for detecting the conditions in AR scenes: collision, position equaling, rotation equaling, and relative position layout (Figure 5(g)).

- **Variables.** We also allow users to create their own variables, set or change them, and hide/show their presence in the AR scene (Figure 5(h)).
- **Operators.** We keep most of the operator blocks (Figure 5(i)) from Scratch to deal with mathematical operations.
- **My Blocks.** We also support defining functions with numbers, text, and booleans as parameters by stacking blocks, which can further help users to quickly implement similar functionalities.

Stacking Blocks. For each selected content, the user drags, drops, and stacks the blocks from the block list to program the interaction. According to the block shapes, the blocks can be stacked or filled one by one. Multiple stacked block groups can be created to run independently for each content.

4.4 Execution and Controlling UI

After programming the interactions, the user can execute the program by clicking the green flag in the UI (Figure 4), and then test the program by viewing the effects, manipulating the contents, and performing interactions with AR-HMD. The user can return to the programming UI to update the blocks and view the updated changes at any time.

4.5 Programming Walk-through

Here, we use an example to show the *ProInterAR* workflow (Figure 4). A user, Fred, wants to create an AR game, “3D Whack a Mole”:

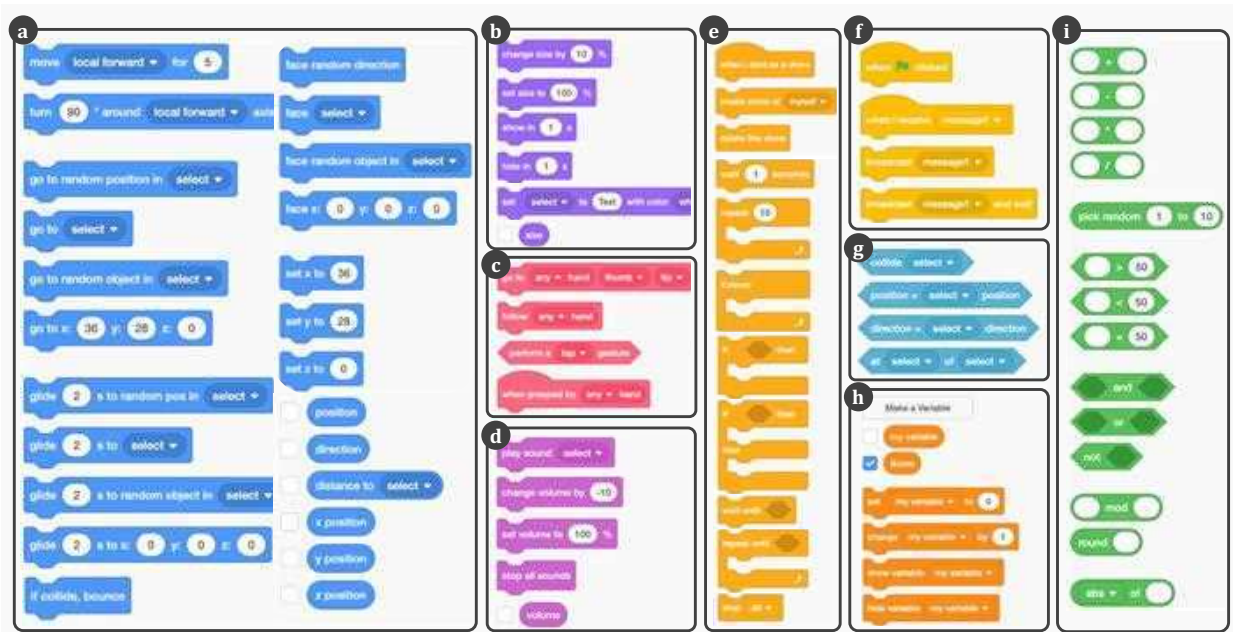


Figure 5: The nine block categories of ProInterAR: (a) Motion, (b) Looks, (c) Hand, (d) Sound, (e) Control, (f) Events, (g) Sensing, (h) Variables, and (i) Operators.

The mole will appear at a random location for a while and disappear in a 3D zone, repeatedly. The player must grasp a hammer and hold it to hit the mole. Every time he hits the mole, the score will be added by one. During the process, three objects are in the scene: a physical cup and book on the desk, and a bunch of virtual balloons in mid-air. If the hammer hits these objects, the score will be deducted by five. So, the player should try to hit the mole but avoid hitting the objects.

To start, Fred creates five objects in the AR scene: a virtual mole, a virtual hammer, a bunch of virtual balloons, a physical cup, and a physical book. For virtual objects, Fred directly drags them from the object library. For the physical cup and book, Fred presses the bounding box creation button and drags the boxes to the positions of the cup and book, and then adjusts their volumes, respectively. The objects are shown in the content-selection list in the programming UI correspondingly. Besides the objects, a box zone needs to be created to work as an active area of the mole. So, Fred presses the “add box zone” button and drags the box, and adjusts its volume above the desktop.

Switching to the programming UI (Figure 4(a)(c)), Fred first adds a new variable named “Score” to calculate the game score. It will show in the AR scene. Then Fred stacks a group of blocks to the mole object: controlling the mole to appear at a random location in the created box zone (Figure 4(a)). For the hammer object, Fred stacks the hand-grasping blocks to indicate that when the player grasps the hammer, it will follow the player’s hand (Figure 4(b)-bottom), and the blocks of collision detection with the mole to increase the score (Figure 4(b)-top). Finally, for the cup, book, and balloon, Fred builds the same block groups that control the scores to

be deducted by five when the hammer hits them. It can be achieved by copying and pasting the block group from one object to another.

After the scene creation and programming phases, Fred can execute the program by clicking the green flag in the programming UI, and then a mole starts to appear at random locations repeatedly. Fred grasps the hammer from the desktop, and the hammer follows his hand (Figure 4(d)). To increase the score, he hits the mole with the hammer and avoids hitting the objects. When the hammer touches the other objects, the score decreases by five.

5 IMPLEMENTATION

We build ProInterAR in an integrated tablet and AR-HMD interface. The programming UI running on the tablet browser is implemented using JavaScript based on the React framework [35]. The AR interface running on the AR-HMD is developed in Unity with MRTK [23] and deployed to the HoloLens 2. To connect the programming web UI in the tablet and the AR interface in HoloLens 2, we build a Node.js server and make the two clients communicate with each other via the server based on WebSocket protocol [7]. Specifically, due to the limited computation capacity of HoloLens 2, we run the server on a PC instead of the HoloLens. Then, we open the two clients on the tablet browser and the HoloLens, respectively, both of which are wirelessly connected to the server in the same Local Area Network (LAN) to achieve real-time communication. The server works as a transfer station that sends a message from one client to another. The message could be a block of data in the programming UI or information of objects and argument values in the AR scene. For the block data, we construct 11-dimension data consisting of a unique thread ID, task name (i.e., action, condition, get value), block name, source object, target object, axis, value, duration, and

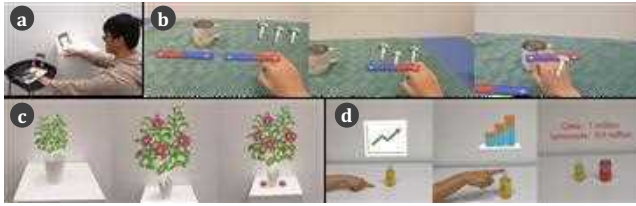


Figure 6: Applications of *ProInterAR*: (a) AR game, (b) AR teaching, (c) Sequential animation, (d) AR information visualization.

relative position. When any block in the web UI is executed, the tablet client will send a message in the format of the constructed data to the HoloLens client via the server forward. After receiving the message, the HoloLens will directly execute the corresponding action of the object if the task name in the message is “action”. If the task is “condition” or “get value”, the HoloLens must send a detection result or obtained value back to the tablet. Besides the feedback data for a block, when the objects and argument values are newly created in the AR scene, it will also send such data to the tablet client to update the object and argument lists in the web UI.

For the condition detection among objects, we use the collision detection method in Unity. For hand/head and hand grasp detection, we track the hand/head pose in real-time, and detect the grasping gesture by checking if the hand pose remains unchanged over a duration. When the hand pose starts to change, the grasp is detected as ending. The other hand gestures are detected using similar methods. For physical object tracking, we use the relative pose between the specified 3D bounding box and the hand to update the 3D object pose with respect to the hand pose in real time.

6 APPLICATION SCENARIOS

6.1 AR Game

AR game is one of the most popular AR applications to the public community. Using *ProInterAR*, creators can design and develop their own interactive AR games, which range from simple and casual experiences to more complex and immersive gameplay. Here, we use an instrument-playing game in AR to show one application scenario of *ProInterAR*. A creator wants to build AR instruments for a personal band using multiple physical objects in daily life and patting these instruments by hand to play different sounds (Figure 6(a)). To achieve it, the creator can create bounding boxes for the physical contents (e.g., chairs and table planes). Then, in the programming UI, the creator stacks the blocks for these objects so that when they collide with hands, they will play certain instrument sounds. To play it, the user pats the objects by hand and form a music sequence. For more complex interactions, the creator can create a virtual keyboard on the desk plane for collision with different fingers.

6.2 AR Teaching

AR technology in educational process can enhance and transform the way students learn and engage with educational contents [18]. Teachers and students can build their own AR learning and experimental contents using *ProInterAR*. For example, when teaching the

properties of the magnetic force in a physics class, a teacher wants to demonstrate the different movements between bar magnets, a metal cup, and iron screws, but only has a cup at hand. He/she imports the virtual bar magnets and iron screws to the AR scene, and programs the motions of these objects when the hand grasping one bar magnet and approaching other objects: the other bar magnet with the same pole will move away, the screws will be attracted closely and follow the magnet, and the bar magnet will be attracted to the cup surface (Figure 6(b)).

6.3 Sequential Animation

Animation is another popularly used component in general AR applications. *ProInterAR* is capable of creating animations with various DoFs. Here, we show the creation of a sequential animation built by *ProInterAR*, which is not easy to create using existing AR authoring tools [39, 48] due to the sequential triggering workflow. For example, a creator aims to create a plant growing animation (Figure 6(c)): a virtual plant is planted in a physical flower pot, and it will grow with an increasing size. Once it changes to a large size, it will turn to blossom and yield fruit, and the fruit will fall to the ground. Using *ProInterAR*, the creator first drags a virtual plant to the pot and a flower and fruit to the plant and sets them hidden at the beginning. Then, in the programming interface, the creator sets the plant to change size and a condition that when the size reaches a threshold, the flower will show. The flowers will also change their sizes. When their sizes reach a threshold, the fruit will show up for a while, and then fall to the ground.

6.4 AR Information Visualization

AR information visualization provides a possibility to present and display data, information, and visualization in a spatial and interactive manner. It combines virtual and real-world environments to enhance the understanding and exploration of data. Our system can also benefit from easy creation of AR information visualization applications. For example, a sales department in a company is exploring the sales volumes of several types of drinks. When one drink is placed on the table and the user clicks on the table or the drink, it will show different sales charts. When another drink is placed close to it, it will show the comparison sales chart (Figure 6(d)).

7 USABILITY STUDIES

To evaluate the usefulness and expressiveness of *ProInterAR*, we conducted two separate usability studies, including an individual evaluation (Section 7.1) for evaluating the performance of individual users in a single session and a long-session evaluation (Section 7.2) for evaluating the continuous performance of two users across five days.

7.1 Study 1: Individual Evaluation

Participants. We recruited 12 participants (8 males and 4 females, aged 10-34, $M = 24$, $SD = 14$). Ten were university students with backgrounds of design (U7-8), human-computer interaction (U1, U3, U6), computer graphics and vision (U2, U4, U5), electronic engineering (U9), and industrial engineering (U10). One was a child (U11) in the fifth grade of primary school, and one was a teenager

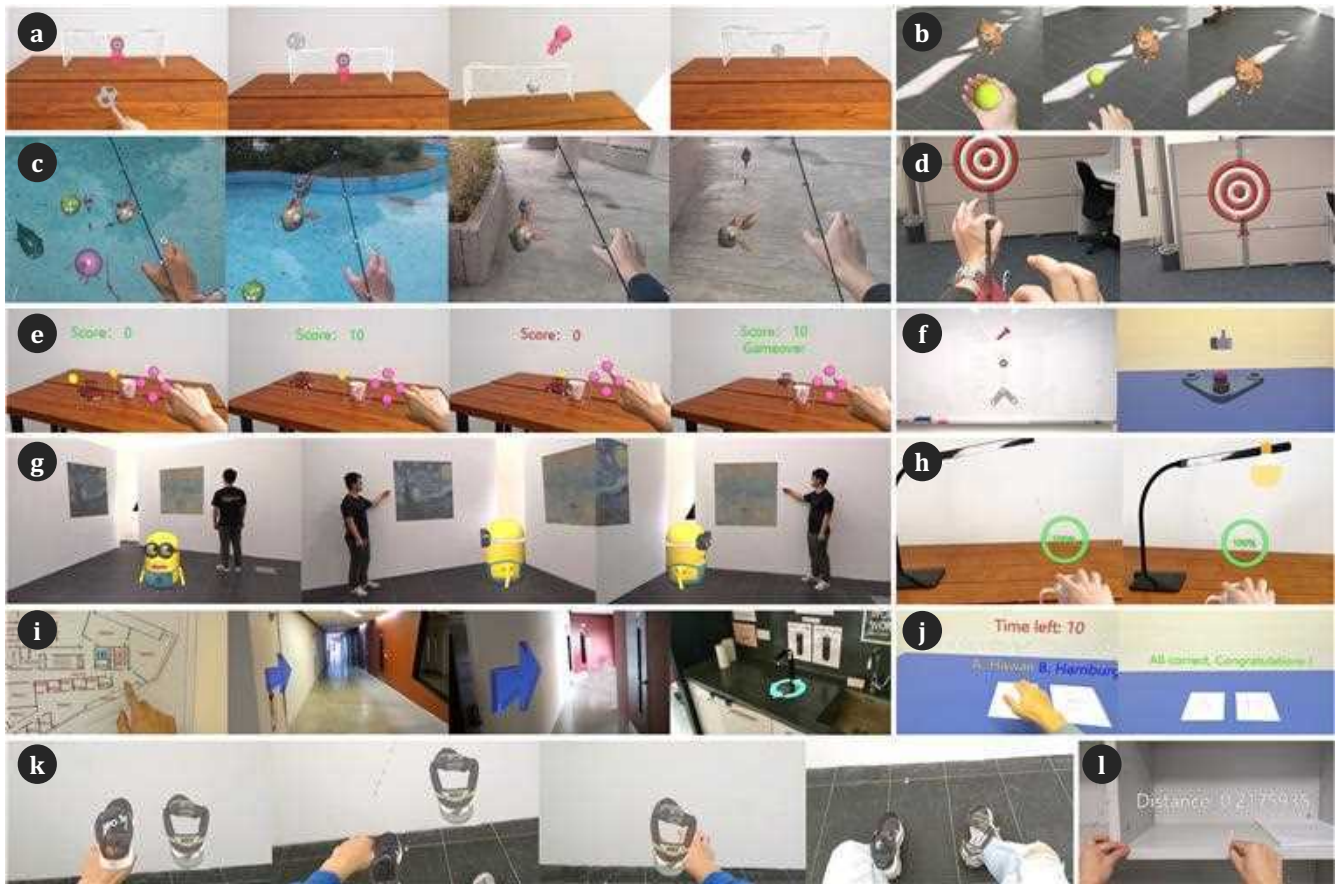


Figure 7: A gallery of the selected programmed applications from Study 1. (a) AR game “flick football”: tap to shoot the ball and hit the goalkeeper. **(b)** AR animation: throw a ball to attract a dog coming close. **(c)** AR fishing game: hold a rod to catch a fish from the pool and leave it on the ground. **(d)** AR archery game: use a pinch gesture to release an arrow: the closer hitting the target, the higher the score gets. **(e)** AR game: press buttons to control the car. When hitting gold coins or the cup, the score will increase or game over. **(f)** AR instruction: instruct to assemble machine parts in a specified order. **(g)** Interaction prototyping for AR tourism: touch the wall to call up a virtual narrator to introduce different paintings. **(h)** IoT function design: move a cup to a certain position to light up the lamp. **(i)** AR navigation: press a target on a map and follow the arrow to navigate there. **(j)** AR Q&A game: press left or right to answer questions for level up. **(k)** AR try-on: pat virtual shoes and try on them. **(l)** AR measurement: use hand distance to measure edge length. Please refer to the supplemental materials for detailed descriptions and corresponding programs.

(U12) in the second grade of secondary school. Four (U1, U7, U8, U11) had used AR applications (e.g., Pokémon GO [28], AR Measure, Just A Line, AR filters, IKEA AR). Half of the participants had strong programming experience (U1-6 with self-reported 4-5 points on a 5-point scale, from 1 = no experience to 5 = strong experience), and the remaining had limited programming experience (U7-12 with self-reported 1-3 points). Four participants (U1, U6, U7, U11, U12) had used visual programming tools (e.g., Scratch, Unreal Blueprints) before. All participants had no AR application creation experience except U1 and U3, who had utilized Unity to create AR teleconferencing and gaming apps. They mainly were novice AR creators, which were our target users. Please note that our current in-lab studies suffer from several limitations such as the selected pool of participants and the small participant samples. Despite our

efforts to diversify the participants’ backgrounds, the majority of them consist of college students, which may introduce potential biases. We would like to emphasize that our studies primarily aim to provide a preliminary evaluation on the user experience of our system. In the future, we will strive to test our system with a more diverse range of participants in field studies.

Tasks and Procedure. At the beginning of the study, we showed the participants the UIs and workflow of *ProInterAR*. The participants learned how to use our system through a simple interactive example as a tutorial: “uses two buttons to control the movement of a virtual cartoon character, which triggers a text box display when it hits a real cup on the table.” We started the introduction of *ProInterAR* with simple conditional and motion statements to guide the participants to realize the required functionality in the

tutorial, and we answered their questions in detail when they did not understand. We took a step-by-step explanation approach for the participants with little programming experience to familiarize them with *ProInterAR*. After the tutorial, we asked each participant to complete the creation of two AR applications, one game-related and the other non-game-related, and encouraged them to use real objects/environments and virtual objects/environments as much as possible. The participants were asked to think about their target interaction results roughly in advance. During the study, we discussed with them to further refine their target interactions by informing them about what our current implementation of *ProInterAR* could and could not achieve. There was a 5-minute break between designing the two applications. After completing each program, each participant experienced and tested the AR applications they designed and developed. At the end of the study, we asked the participants to fill in a System Usability Scale (SUS) questionnaire on a 5-point scale (1 = strongly disagree to 5 = strongly agree), as well as a NASA-TLX questionnaire on a 5-point scale (the lower, the better). In addition, we conducted a small-scale interview with them to discuss their user experience. The interviews and interaction process were audio-recorded and later transcribed with their agreement by filling out an informed consent form. We encouraged them to express their positive and negative opinions. Each study took about 50-70 minutes. Each participant was compensated with a 13-USD gifted card.

Results. Each application took about 25-40 minutes for the participants to complete. They spent most of the time (i.e., about 20-30 minutes) iteratively building AR scenes, programming the behaviors, and testing the programs, and then about 5-10 minutes playing or interacting with each application. A total of 22 AR applications were created by the participants (Figure 7) since the child and teenager participants each only created one application. These results covered extensive application scenarios, including AR games, AR animation, AR instruments and tutorials, AR navigation, AR shopping, interaction prototyping, IoT function design, AR measurement, AR fitness, etc. For game-related applications, some participants transformed classic 2D games into 3D AR versions by replacing traditional keyboard and mouse controls with hand interaction (U4, U9, U10). Some enjoyed designing games that involved virtual and real-world interactions by incorporating the surrounding environment or objects (U1-3, U5, U7-9, U11). Some preferred utilizing expansive real spaces to design hide-and-seek games (U11). For non-game-related applications, most participants tended to realize the convenient experiences that AR could bring based on their own imagination (U1-2, U5-7, U9). A few participants wanted to reenact certain real-life moments, even if they involved somewhat unrealistic animations (U4). Some utilized our system as a tool for prototyping purposes (U3, U8, U10). The applications include all the types of contents supported in our system: real objects (e.g., cup, hand, hand finger), real environments (e.g., wall plane, table surface), virtual objects (e.g., fishing rod, bow and arrow, virtual dog and character), and virtual environment (e.g., virtual wall). The interactions include spatial movements of a single object (e.g., ball and character moving), collisions between two objects or objects and environments (e.g., hand-table, car-obstacle), appearance change (e.g., arrow show and hide, text color change), gestural interactions (e.g., hand grasping, pinch gesture), context-aware events (e.g., state

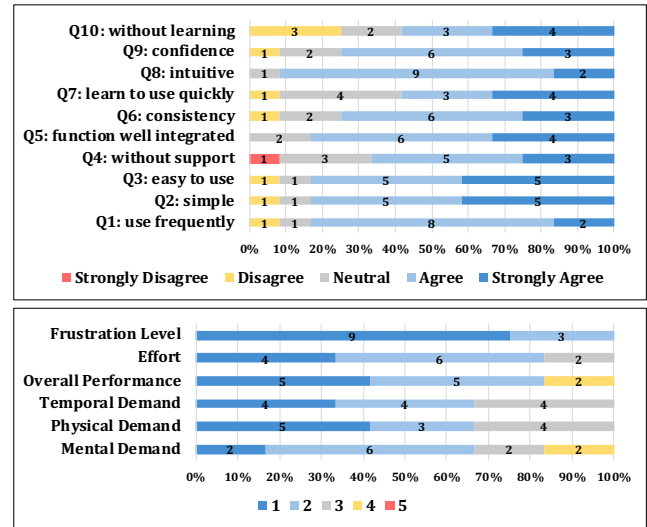


Figure 8: SUS (Top) and NASA-TLX (Bottom) score distributions. The question description in SUS is the key points from the full SUS questions. For the scores of SUS, the higher, the better. For the scores of NASA-TLX, the lower, the better.

detection to trigger IoT functions), and audio effects (e.g., playing music). In terms of programming, we found that hand-object collision detection was one of the most popular triggering conditions, and the participants were very comfortable engaging in interactions with hand touches regardless of whether the target objects were virtual or real. Moreover, they preferred collision detection even for conditional judgments without involving their hands, such as determining whether two objects reached the same location, and the participants usually used collision detection rather than distance or position detection. One possible explanation is that collisions are more intuitive and easier to understand. Variables were frequently used because they could serve as both scoreboards in games and state controllers in programming and also helped achieve communication across objects similar to broadcasting. Multiple arguments (e.g., position and box zones, user-defined axes) were created and passed to the block field. Various controls like loops, repetition, conditional judgments, broadcasting, and time waiting were used to script complex interaction paradigms.

The SUS scores rated by the participants were overall good, and Figure 8(Top) shows the rating distribution of every SUS question. Most participants thought our system to be easy to use (Q3) with workflows and programming interfaces designed to be simple and easy to understand (Q2), and also intuitive to operate (Q8). The system is also very consistent (Q6), and the integration of features is great (Q5). They thought that in most scenarios where they wanted to easily and quickly realize and experience the target effect in situ, our system could help a lot. They also believed that they would frequently use our system both indoors and outdoors (Q1). All the participants were confident in using our system (Q9), except one participant (U4), who was affected by her unfamiliarity with operating the HoloLens device, and they were satisfied with the results. At the same time, the participants with rich programming experience

also raised some doubts. They thought that perhaps their programming experience allowed them to familiarize themselves with our system very quickly, but they were conservative and thought there was still a learning curve for other users (Q7). Some participants believed that our system had its own rules, requiring additional learning regardless of the user's programming experience (Q10). One participant (U8) with little programming experience felt that it was necessary to have a technical person to guide them through the system initially. Two participants (U11-12) with prior experience using Scratch found it easy to learn (Q4). Figure 8(Bottom) shows the NASA-TLX scores. We found that most of the participants, except for two participants (U6, U8) with little AR-HMD experience, perceived the workload of our system to be low.

To analyze the qualitative feedback during the study and the interviews, we conducted a thematic analysis. Two of the authors coded the transcribed documents by systematically identifying and labeling them and then made a codebook. Then, the authors grouped related codes together to form themes, and reviewed and refined them by revisiting the coded data. Six main themes emerged as follows.

T1: Overall User Experience (All). The participants felt that *ProInterAR* was overall easy to learn or/and use, friendly to beginners, and provided a great understanding of coding concepts. Some participants (U1, U5-6, U9) mentioned the convenience of instant testing after programming since they could “*find problems immediately and adjust the blocks to see the updated results in real-time (U1).*” A participant with an interaction design background (U8) said, “*although my programming experience is limited, I think your system still facilitates me by including a wide variety of interactions between real and virtual contents.*” His programmed application in Figure 7(c) contains a variety of programming elements (e.g., loop, condition) that he had not coded with before. U7 commented that “*it is simple and easy, which is beyond my expectation.*” They also highlighted areas for improvement, such as providing sound effects during operations.

T2: Advantages of Combining Tablet Interface and AR-HMD (U2, U4-5, U7-9, U11). The participants mentioned that the tablet interface was similar to daily usage, less distracting, and allowed for precise and familiar operation habits. The AR-HMD provided an immersive and flexible environment for in-situ authoring. The participants also appreciated the convenience and portability of such an integrated interface, as “*the mobile setup lets me program anywhere without relying on any desktop computer (U4).*” U7 with the design background mentioned that “*it is applicable to mobile scenes where I can design the concepts based on a surrounding environment. Programming on the tablet also streamlines the design procedure with convenient and rapid operations.*” The child participant with Scratch experience (U11) liked the AR authoring feature [42] since it can “*play 3D interactions and turn the 2D elements in Scratch to the 3D elements in the physical world.*” The markerless object tracking was also appreciated by the participants, as U9 said “*it supports tracking objects with different volumes.*”

T3: Similarities and Differences with Traditional Coding (U1-2, U4-6, U9). This theme mainly comes from the participants with a programming background. They noted that *ProInterAR* shared some similarities in terms of logic with traditional coding. However, they

also emphasized the differences, such as the graphical representation of coding logic, the use of pre-defined functions, and the ease of getting started with *ProInterAR*. U1 said our system “*turning coding logic into graphical elements is easy to remember. Graphical memory helps quickly construct the whole game logic instead of suffering from coding format issues. With self-contained blocks, users just need to stack the blocks.*” As a programming beginner, U9 commented that our interface was direct and intuitive; especially with object-centric programming, she can “*quickly create connections between the physical objects and environments to the programming UI.*” She especially liked the block enclosing design (e.g., the space within the if-else blocks) to learn the coding functions.

T4: Application Scenarios (U1-2, U5, U9-12). The participants believed *ProInterAR* had great generalizability to various application scenarios. They mentioned its usefulness in designing games, prototyping, and implementing complex logic in any scene. Specific scenarios mentioned include large-scale playground design, educational purposes for kids, and enabling real objects with IoT functions. U10 expected that “*it can help kids learn coding concepts, how to begin programming, and they will be interesting to start from the objects around them.*” The kid participant (U11) found the tool interesting and useful to allow her to understand complex coding logic intuitively. One participant (U5) with some experience in AR prototyping stated, “*ProInterAR supports the implementation of a wide range of common interaction functions. I believe it can cover the functions of multiple toolkits I have used before for wider usage.*”

T5: Learning Curve (U2-U3, U9, U11). Some participants mentioned that it became easier for them to use *ProInterAR* when they were familiar with it, which usually came with the creation of their second application. The kid participant (U11) needed instruction at the beginning, mainly for the interactions from the AR-HMD, and later she could explore features on her own. U2 and U7 also mentioned that it became very easy when they learned all the system's functions, including various block functions. It indicates that there is a slight learning curve, especially for the participants with limited programming backgrounds.

T6: Limitations and Challenges (U1-3, U4, U11). The participants also mentioned certain limitations and challenges. These include limited debugging functions, confusion when dealing with a large number of objects, difficulty in searching for blocks, and the heavy weight of AR-HMD. Some limitations were inherited from Scratch, like U1 with rich coding experience said, “*since most of the blocks can only control selected objects, it is necessary to use the broadcast function to make cascading effects between different objects, which is a bit cumbersome and can be very error-prone when the program becomes complex, and the system's debugging capabilities are very limited.*” We will improve our block copy-and-paste function to change block arguments more intelligently and include the debugging features in the future. U3, who had designed VR/AR projects, said, “*the creation of a character's smooth motion path requires me to specify multiple individual points, which is a little cumbersome.*” This might be improved by including a user-defined curve for argument creation.




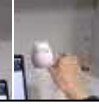



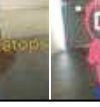
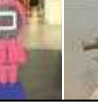




P1: Office Assistant Application					P2: Adventure Game								
Day													
1	(a) Familiarize with coding concept, programming UI, and HoloLens interactions.					(a) Where he grasps and moves the virtual key to the metallic square, the instruction shows above left hand leading to the position of the treasure.							
2	(b) When she sits down, show a water drop above the cup to remind drinking water.					(b) When he arrives near the tree, a monster appears and follows him upon discovering the treasure.							
3	(c) When she sits down and watches screen, show a text and character to remind taking a break. (d) When she leaves, the program resets to the initial state.					(c) A trap to eliminate the monster: he guides the monster to the trap's location and physically activates it by touching the real environment.							
4	(e) Set timewait to (b) and (c). (f) Enhance a physical trash bin, drag the water drops and time reminders to bin to delete and reset timers. (g) When she comes/leaves, play a sound to remind clocking in/out.					(d) He obtains the treasure on the treetop and acquires a light saber by grasping it. (e) Two new monsters appear. (f) Three life points are display above his right hand.							
5	(h) Set an interactive virtual pet on the desktop: when pat the pet, it speaks encouraging words; when perform a pinch gesture, show heart moving from the finger to the pet. When the pet hits the heart, it shakes.					(g) Endow the monsters in (e) one with close-range attacks and the other with long-range attacks. (h) He uses acquired light saber to attack each monster twice and avoids their attacks. Hit by them will lose one life point. (i) When eliminate both monsters, game over.							

Figure 9: The top and bottom rows respectively show intermediate result snapshots of two long-session applications and descriptions for each day.

7.2 Study 2: Long-session Evaluation

From the first study, we found that *ProInterAR* could help users create high-quality AR applications for various usages. However, from the emerging theme T5 (learning curve), we found that it might bring a more natural experience when users continuously use *ProInterAR*, especially for users without coding experience. To investigate the learning effects further, we conducted a long-session evaluation. To understand the learning effects comprehensively and figure out the ceiling of our system, we consider both novice and professional AR creators in this study.

Participants. Two university students (P1: female aged 27, P2: male aged 28) were invited for a five-day evaluation. They did not participate in Study 1 or use our system before. P1 had no AR creation experience before, while P2 had created several AR applications, including an AR-guided training system, an AR-guided robot control system, and an AR game using Unity and Unreal ITK. P1 had no programming experience but P2 had an extensive coding background. Both of them had not used any visual programming tools before.

Tasks and Procedure. Each participant was asked to create an AR application using *ProInterAR* with a bottom-up approach, starting from an initial idea and incrementing on it gradually to form a complete application. The session took about 30 minutes each day. Before starting the tasks, the participants were instructed to use *ProInterAR* through the same example in Study 1. After each day's task, we had small-scale interviews with them to gather feedback.

Results. P1 and P2 designed an office assistant application and an adventure game, respectively. Figure 9 shows their designed applications in each day. Both participants expressed high satisfaction with the final outcome, and their applications included all the types of contents supported by our system. Below we will show some details and findings from their interviews.

P1 thought the block-based visual programming UI to be very user-friendly, "Although I don't fully understand the specific details

within each code block, I find it easy to understand the approach where different functions have distinct shapes, and programming is done by matching these shapes together." And P1 commented that "I find it very useful to copy and paste code by directly dragging it onto different objects." Furthermore, P1 mentioned that AR added emotional significance to real-world objects, "A water cup, for example, doesn't have much life to it, but by adding a cute virtual water droplet on it, it creates a stronger emotional connection with me. It might even make me more inclined to drink water on time every day." Overall, P1 believed *ProInterAR* would be helpful for her everyday creativity, "I feel like I can use this tool to create anytime, anywhere. Sometimes I cannot bring my inspiration home, but with this tool, I can immediately implement and experience it when I have a sudden burst of inspiration outdoors." During the study, we observed some phenomena that indicated P1's learning curve. Initially, she complained about our system's inability to provide more information about the code blocks. She desired concise textual explanations when hovering her finger over a code block, which could be added as a future feature, especially for beginners. After implementing some simple triggering effects, she became more proficient in using commonly used code blocks. However, when attempting to combine them into a system and continue adding functionality while ensuring robustness, she found that programming logic became the primary limiting factor for her, "Although I am not going to use more complex code blocks, I now need to consider the logical relationships between multiple variables and objects, which confuses me." After the study, P1 was thrilled with the AR application she designed and expressed, "I am extremely proud to have created such an application. I have realized that the principles behind seemingly simple tools in everyday life are actually quite complex, requiring consideration of various unexpected situations. Now I am filled with confidence in learning programming."

We were surprised that P2 quickly became proficient in using *ProInterAR* almost without our assistance. He commented that the

programming mechanism for each object in the system was very similar to scripting in Unity, thus allowing him to quickly adapt. He also felt ProInterAR to be very practical, *“When designing AR applications before, I had to place the virtual assets in the real world and record their position and rotation information first. Then, I would develop applications using Unity. Finally, I would deploy the application onto the AR-HMD and return to my target scene for testing. During the development process, I could not truly observe how these virtual assets interacted in the real world, thus hindering my creativity. Additionally, whenever I needed to add something new, I had to repeat the above process, which was quite cumbersome. In contrast, ProInterAR enables me to program directly in the scene, add virtual assets whenever I want, and test them on the spot. It’s truly fantastic.”* He believed that the combination of a tablet and AR-HMD was a great compromise, *“Directly programming within an AR-HMD would be a disaster. The controls and clarity would be major issues. Desktop-based development for AR applications separates the creation and development process, which cannot get real-time feedback. A portable device like the tablet provides a good balance of precise control, display clarity, and support for mobile usage.”* Throughout the study, P2 consistently expressed how powerful our ProInterAR was, *“I notice that the system has built-in code blocks like ‘collide,’ ‘face,’ ‘move,’ ‘show,’ ‘hide,’ etc. Implementing such functionalities with real code is not easy and time-consuming. Now, I can conveniently use them and focus on designing the applications I want to create. This significantly reduces time overhead and, in my opinion, makes it more user-friendly for those who are inexperienced.”* After the study, we discussed further insights about ProInterAR with P2. He mentioned, *“I believe this block-based visual programming is highly suitable for beginners. The stacking of blocks and the different shapes and text descriptions on them make it easier for the inexperienced to understand.”* Furthermore, he also mentioned some limitations of ProInterAR. He pointed out that the current ProInterAR had limited capabilities for inter-object communication. It could only be achieved through features like Broadcast and Variable, which resulted in a significant amount of communication-related code in complex AR applications. He stated, *“ProInterAR only supports controls within the current object. However, in my experience, I often need to simultaneously control multiple objects, which requires the ability to control other objects beyond the current object. What I am hoping for is the ability similar to designing scripts on an empty object in Unity to control other objects within the scene. This could contribute to cleaner code and help reduce potential errors and debugging burden when designing large-scale AR applications.”*

8 DISCUSSIONS AND FUTURE WORK

8.1 Scalability to More Complex Applications

Although ProInterAR allows users to create various AR interactions applicable to different scenarios, the current scope has some limitations in meeting all the expectations of our target users.

Physical Simulation. As a powerful tool to enhance realistic behaviors and interactions of virtual contents, physical simulation has not been supported in our current system. Participants U4, who created an effect of throwing a ball (Figure 7(b)) and U10, who created a game of shooting with an arrow (Figure 7(d)), both expected to endow the ball and the arrow with gravity. We can easily extend

our system with a force system by introducing a new category of blocks, where each block simulates a kind of force with different variations. The user can simply drag such blocks to the block stacks of the contents for adding force simulation effects.

Access to More Information. Our system allows users to script spatial interactions by obtaining the pose information of AR contents, including hand joints and head. To support more complex interactions (e.g., context-aware interactions (U3) in CAPturAR [40], other-body-part-based interactions (U5)), obtain context information (e.g., the current time, weather, and traffic condition), and the poses of other body parts, we can include system information and online APIs and enable body part tracking by attaching external sensors.

8.2 Limitations of Content-centric Programming

ProInterAR requires users to program behaviors for each content individually. The blocks of each content are run independently. It is fussy to obtain the state of one content in the blocks of another since it requires setting a broadcasting block to communicate between two contents. Such a content-centric programming manner is commonly used in professional software (e.g., Unity), but they allow to obtain the information of other contents from scripts. By displaying the information of one content in the blocks of another, it might be confusing to distinguish which content the block will work on. One possible solution is to introduce a simple logic of node-based programming, which is good at transmitting information.

8.3 2D Programming versus Immersive Programming

Researchers have discussed the benefits of immersive programming over 2D programming in [53]. We admit that immersive programming provides more spatial information and instant programming feedback. Although our programming interface is set up in a tablet browser, it is not a traditional 2D programming interface built on a desktop, which is isolated from the AR scenarios. Instead, the user can bring the tablet and wear the AR-HMD at the same time. So the user can program the interactive behaviors while keeping an eye on the AR scene. The programmed results can be executed, watched, and controlled instantly. In addition, we think our current programming blocks are not necessarily to be designed in a 3D format, since directly using it in a 2D UI benefits users to have familiar and quick hand interaction *“with less hand fatigue and distraction, and increasing accuracy”*, as reflected from P1 in Study 2, who was experienced in immersive authoring.

8.4 DoFs of Programming Blocks

In our current system, we design the programming blocks as fundamental concepts, allowing users to stack and combine them freely. It is the reason that we allow for more flexible programming for general AR interactions. While it offers much freedom, it sometimes requires creators to stack the blocks for similar actions repetitively. For example, to make a virtual car move around physical objects, creators might need to create several buttons to control the moving direction. Similar block stacks with different argument values (i.e.,

moving direction) and conditions (i.e., if pressed by hand, triggering the movement) must be created for each button. To facilitate these procedures, the wrapped block components with pre-defined functions can be introduced. For example, a virtual joystick and a virtual direction button group can be predefined as block components. Hand-related block components, such as pre-defined relative hand positions and hand directions, can also be wrapped to achieve convenient function specifications.

8.5 Tracking and Detection of Physical Contents

We provide a manual assignment of a 3D bounding box to an arbitrary physical object (e.g., a phone) and a manual plane creation/mesh selection for physical environments. The 6-DoF pose of the physical objects is tracked according to the user's hand pose. However, for automatically moving physical objects (e.g., a moving car and a running human), our approach does not work. Besides, the bounding box can only approximate the rough volumes of the objects. For more detailed and fine shapes and surfaces, it is difficult to detect unless the creator carefully creates and combines multiple planner surfaces. So, in the future, we will integrate more robust and flexible object detection, tracking, and shape estimation algorithms [16] into our system. We will also explore using hand motions to create surfaced planes [45].

9 CONCLUSION

This paper has presented *ProInterAR*, a visual programming tool that enables novice AR creators to create general AR interactions. We first discussed the design scope of general AR interactions from four dimensions: subject, scenario, concurrency, and logic. We analyzed how closely related works on AR prototyping, authoring, and programming cover the aspects of the dimensions, and the scope of our proposed system that would support. Compared to the existing works, we allow users to create general AR interactions directly and closely happening among both real and virtual objects and environments, in a parallel and independent manner, with complex logic flows. Based on these considerations, we designed and developed our system, consisting of three components: a scene creation UI from the AR-HMD, a block-based visual programming UI from the tablet, and an execution and controlling UI from the AR-HMD. Users can create different types of AR contents and behavior variations using the corresponding approaches. The programming UI contains nine categories of blocks to enable motion, looks, hand interaction, sound effects, and multiple logic controls (e.g., event sensing and listening, repetition, condition statements, operators, and variables). We demonstrated the system workflow using an example of creating a game "3D Whack a Mole". We showcased the four potential application scenarios: AR game, AR teaching, sequential animation, and AR information visualization. One individual study and one long-session study were conducted to evaluate the usability, expressiveness, and learning effects of *ProInterAR*. They verified that our system can help creators easily create AR applications with varying complexity.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive comments and the user study participants for their time. This work was supported by grants from the Innovation and Technology Commission (No. ITS/106/22), the City University of Hong Kong (Project No. 9667260, 7005729, 9667234), NSFC (62072316), and NSF of Guangdong Province (2023A1515011297).

REFERENCES

- [1] Hussein Alrubaye, Stephanie Ludi, and Mohamed Wiem Mkaouer. 2019. Comparison of Block-Based and Hybrid-Based Environments in Transferring Programming Skills to Text-Based Environments. *arXiv preprint arXiv:1906.03060* (2019).
- [2] 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.
- [3] Renaud Blanch and Michel Beaudouin-Lafon. 2006. Programming Rich Interactions Using the Hierarchical State Machine Toolkit. In *Proceedings of the Working Conference on Advanced Visual Interfaces*. 51–58.
- [4] Margaret M Burnett and David W McIntyre. 1995. Visual Programming. *COMputer-Los Alamitos-28* (1995), 14–14.
- [5] Mengyu Chen, Marko Peljhan, and Misha Sra. 2021. EntangleVR: A Visual Programming Interface for Virtual Reality Interactive Scene Generation. In *Proceedings of the 27th ACM Symposium on Virtual Reality Software and Technology*. 1–6.
- [6] Pietro Cipresso, Irene Alice Chicchi Giglioli, Mariano Alcañiz Raya, and Giuseppe Riva. 2018. The Past, Present, and Future of Virtual and Augmented Reality Research: A Network and Cluster Analysis of the Literature. *Frontiers in Psychology* (2018), 2086.
- [7] MDN contributors. 2023. *The WebSocket API (WebSockets)*. https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API
- [8] Dragos Datcu and Stephan Lukosch. 2013. Free-Hands Interaction in Augmented Reality. In *Proceedings of the 1st Symposium on Spatial User Interaction*. 33–40.
- [9] Ruofei Du, Na Li, Jing Jin, Michelle Carney, Scott Miles, Maria Kleiner, Xiuxiu Yuan, Yinda Zhang, Anuva Kulkarni, Xingyu Liu, et al. 2023. Rapsai: Accelerating Machine Learning Prototyping of Multimedia Applications through Visual Programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–23.
- [10] Paula Fraga-Lamas, Tiago M Fernandez-Carames, Oscar Blanco-Novoa, and Miguel A Vilar-Montesinos. 2018. A Review on Industrial Augmented Reality Systems for the Industry 4.0 Shipyard. *IEEE Access* 6 (2018), 13358–13375.
- [11] Vittoria Frau, Lucio Davide Spano, Valentino Artizzu, and Michael Nebeling. 2023. XRSpotlight: Example-based Programming of XR Interactions using a Rule-based Approach. *Proceedings of the ACM on Human-Computer Interaction* 7, EICS (2023), 1–28.
- [12] David K Gifford and John M Lucassen. 1986. Integrating Functional and Imperative Programming. In *Proceedings of the 1986 ACM Conference on LISP and Functional Programming*. 28–38.
- [13] Google. 2023. *Blockly*. <https://developers.google.com/blockly>
- [14] Valentin Heun, Shunichi Kasahara, and Pattie Maes. 2013. Smarter Objects: Using AR Technology to Program Physical Objects and Their Interactions. In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*. 961–966.
- [15] Danial Hooshyar, Rodina Binti Ahmad, Moslem Yousefi, Farrah Dina Yusop, and S-J Horng. 2015. A Flowchart-Based Intelligent Tutoring System for Improving Problem-solving Skills of Novice Programmers. *Journal of Computer Assisted Learning* 31, 4 (2015), 345–361.
- [16] Rachel Huang, Jonathan Padoem, and Cuixian Chen. 2018. YOLO-LITE: A Real-Time Object Detection Algorithm Optimized for Non-GPU Computers. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2503–2510.
- [17] Kwanghee Jung, Vinh T Nguyen, and Jaehoon Lee. 2021. BlocklyXR: An Interactive Extended Reality Toolkit for Digital Storytelling. *Applied Sciences* 11, 3 (2021), 1073.
- [18] Seokbin Kang, Ekta Shokeen, Virginia L Byrne, Leyla Norooz, Elizabeth Bon Signore, Caro Williams-Pierce, and Jon E Froehlich. 2020. ARMath: Augmenting Everyday Life With Math Learning. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–15.
- [19] 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.
- [20] Veronika Krauß, Alexander Boden, Leif Oppermann, and René Reinert. 2021. Current Practices, Challenges, and Design Implications for Collaborative AR/VR Application Development. In *Proceedings of the 2021 CHI Conference on Human*

- Factors in Computing Systems*. 1–15.
- [21] Germán Leiva, Jens Emil Grønbaek, Clemens Nylandsted Klokmose, Cuong Nguyen, Rubaiat Habib Kazi, and Paul Asente. 2021. Rapido: Prototyping Interactive AR Experiences through Programming by Demonstration. In *The 34th Annual ACM Symposium on User Interface Software and Technology*. 626–637.
- [22] 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.
- [23] Microsoft. 2022. MRTK. <https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/?view=mrtkunity-2022-05>
- [24] MIT. 2023. *App Inventor*. <https://appinventor.mit.edu/>
- [25] Kyzyl Monteiro, Ritik Vatsal, Neil Chulpongatorn, Aman Parnami, and Ryo Suzuki. 2023. Teachable Reality: Prototyping Tangible Augmented Reality with Everyday Objects by Leveraging Interactive Machine Teaching. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–15.
- [26] Brad A Myers. 1986. Visual Programming, Programming by Example, and Program Visualization: A Taxonomy. *ACM SIGCHI Bulletin* 17, 4 (1986), 59–66.
- [27] Michael Nebeling and Katy Madier. 2019. 360proto: Making Interactive Virtual Reality & Augmented Reality Prototypes From Paper. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–13.
- [28] Janne Paavilainen, Hannu Korhonen, Kati Alha, Jaakko Stenros, Elina Koskinen, and Frans Mayra. 2017. The Pokémon GO Experience: A Location-Based Augmented Reality Mobile Game Goes Mainstream. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 2493–2498.
- [29] Ken Perlin, Zhenyi He, and Fengyuan Zhu. 2018. Chalktalk VR/AR. *International SERIES on Information Systems and Management in Creative eMedia (CreMedia)* 2017/2 (2018), 30–31.
- [30] Wayne Piekarski and Bruce Thomas. 2002. ARQuake: The Outdoor Augmented Reality Gaming System. *Commun. ACM* 45, 1 (2002), 36–38.
- [31] Henning Pohl, Tor-Salve Dalsgaard, Vesa Krasniqi, and Kasper Hornbæk. 2020. Body LayARs: A Toolkit for Body-Based Augmented Reality. In *Proceedings of the 26th ACM Symposium on Virtual Reality Software and Technology*. 1–11.
- [32] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. 2009. Scratch: Programming for All. *Commun. ACM* 52, 11 (2009), 60–67.
- [33] Maeve Serino, Kyla Cordrey, Laura McLaughlin, and Ruth L Milanaik. 2016. Pokémon Go and Augmented Virtual Reality Games: A Cautionary Commentary for Parents and Pediatricians. *Current Opinion in Pediatrics* 28, 5 (2016), 673–677.
- [34] Snap. 2023. *Lens Studio*. <https://ar.snap.com/en-US/lens-studio>
- [35] Meta Open Source. 2023. *React*. <https://react.dev/>
- [36] Ryo Suzuki, Rubaiat Habib Kazi, Li-Yi Wei, Stephen DiVerdi, Wilmot Li, and Daniel Leithinger. 2020. RealitySketch: Embedding Responsive Graphics and Visualizations in AR Through Dynamic Sketching. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 166–181.
- [37] Unity. 2023. *XNode*. <https://assetstore.unity.com/packages/tools/visual-scripting/xnode-104276>
- [38] Unreal. 2023. *Introduction to Blueprints*. <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/GettingStarted/>
- [39] 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*. 552–567.
- [40] 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.
- [41] David Weintrop. 2019. Block-Based Programming in Computer Science Education. *Commun. ACM* 62, 8 (2019), 22–25.
- [42] Julia Woodward, Feben Alemu, Natalia E. López Adames, Lisa Anthony, Jason C. Yip, and Jaime Ruiz. 2022. “It Would Be Cool to Get Stamped by Dinosaurs”: Analyzing Children’s Conceptual Model of AR Headsets Through Co-Design. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–13.
- [43] Hsin-Kai Wu, Silvia Wen-Yu Lee, Hsin-Yi Chang, and Jyh-Chong Liang. 2013. Current Status, Opportunities and Challenges of Augmented Reality in Education. *Computers & Education* 62 (2013), 41–49.
- [44] Zhijie Xia, Kyzyl Monteiro, Kevin Van, and Ryo Suzuki. 2023. RealityCanvas: Augmented Reality Sketching for Embedded and Responsive Scribble Animation Effects. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. 1–14.
- [45] Xinchu Xu, Yang Zhou, Bingchan Shao, Guihuan Feng, and Chun Yu. 2023. GestureSurface: VR Sketching through Assembling Scaffold Surface with Non-Dominant Hand. *IEEE Transactions on Visualization and Computer Graphics* 29, 5 (2023), 2499–2507.
- [46] Hui Ye and Hongbo Fu. 2022. ProGesAR: Mobile AR Prototyping for Proxemic and Gestural Interactions with Real-world IoT Enhanced Spaces. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–14.
- [47] 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.
- [48] Hui Ye, Jiaye Leng, Chufeng Xiao, Lili Wang, and Hongbo Fu. 2023. ProObjAR: Prototyping Spatially-aware Interactions of Smart Objects with AR-HMD. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–15.
- [49] Enes Yigitbas, Jonas Klauke, Sebastian Gottschalk, and Gregor Engels. 2023. End-User Development for Interactive Web-Based Virtual Reality Scenes. *Journal of Computer Languages* 74 (2023), 101187.
- [50] 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.
- [51] Egui Zhu, Arash Hadadgar, Italo Masiello, and Nabil Zary. 2014. Augmented Reality in Healthcare Education: An Integrative Review. *PeerJ* 2 (2014), e469.
- [52] Zhengzhe Zhu, Ziyi Liu, Tianyi Wang, Youyou Zhang, Xun Qian, Pashin Farsak Raja, Ana Villanueva, and Karthik Ramani. 2022. MechARspace: An Authoring System Enabling Bidirectional Binding of Augmented Reality with Toys in Real-time. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*. 1–16.
- [53] Zhengzhe Zhu, Ziyi Liu, Youyou Zhang, Lijun Zhu, Joey Huang, Ana M Villanueva, Xun Qian, Kylie Peppler, and Karthik Ramani. 2023. LearnIoTVR: An End-to-End Virtual Reality Environment Providing Authentic Learning Experiences for Internet of Things. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–17.