ARcadia: A Rapid Prototyping Platform for Real-time Tangible Interfaces

Annie Kelly
R. Benjamin Shapiro
University of Colorado Boulder
Boulder, CO
annie.kelly@colorado.edu
ben.shapiro@colorado.edu

Jonathan de Halleux
Thomas Ball
Microsoft Research
Redmond, WA
jhalleux@microsoft.com
tball@microsoft.com

ABSTRACT
Paper-based fabrication techniques offer powerful opportunities to prototype new technological interfaces. Typically, paper-based interfaces are either static mockups or require integration with sensors to provide real-time interactivity. The latter can be challenging and expensive, requiring knowledge of electronics, programming, and sensing. But what if computer vision could be combined with prototyping domain-aware programming tools to support the rapid construction of interactive, paper-based tangible interfaces? We designed a toolkit called ARcadia that allows for rapid, low-cost prototyping of TUIs that only requires access to a webcam, a web browser, and paper. ARcadia brings paper prototypes to life through the use of marker based augmented reality (AR). Users create mappings between real-world tangible objects and different UI elements. After a crafting and programming phase, all subsequent interactions take place with the tangible objects. We evaluated ARcadia in a workshop with 120 teenage girls and found that tangible AR technologies can empower novice technology designers to rapidly construct and iterate on their ideas.

ACM Classification Keywords
H.5.2. Information interfaces and presentation: User Interfaces;
H.5.1. Information interfaces and presentation: Multimedia Information Systems – Artificial, augmented, and virtual realities

Author Keywords
Tangible user interfaces; real-time interactivity; augmented reality; paper prototyping; block-based programming.

INTRODUCTION
Paper prototyping is a commonly used method employed by designers and developers in the building of user-centered interfaces and applications. This practice is advantageous for designers because it is a simple, low-cost technique for iterative usability testing. However, paper-based techniques are best suited for prototyping static mockups of 2D Graphical User Interfaces (GUIs); these methods do not easily allow for the prototyping of real-time, tangible interactive systems. Transforming paper prototypes into live, interactive tangibles typically involves integrating electronic sensors with craft or other fabrication technologies, and authoring software to work with those sensors and integrate their data with the rest of a prototype software system. This approach is typically challenging, time-consuming, expensive, and requires working with electronics and programming. While there are available programming and electronics toolkits targeted towards makers and avid DIYers, these products come with their own limitations. These toolkits are often expensive; consumers may need to purchase a microcontroller or single-board computer, and also purchase compatible sensors [17]. There are some less expensive solutions such as Arduino, but they require more prerequisite knowledge of hardware and software such as circuitry, and programming real-time sensor interactivity. The complexity of these tools, and the amount of engineering effort that they demand for productive use, may also make repeated design iteration prohibitively expensive.

Recently, researchers have been adopting AR techniques in order to address some of the shortcomings of available prototyping toolkits [8] [6] [9] [13] [10]. One of the key elements of Augmented Reality is that computer generated graphics and sounds are overlaid onto the physical world, or a representation of the physical world. This can be achieved by using projection mapping, creating composite videos of reality with graphics superimposed, and other techniques. Development of AR applications has become increasingly popular in gaming, entertainment, artistic performances, and education. Although AR is becoming more ubiquitous, the development of these applications still requires the work of experienced programmers and artists. Developers commonly program AR applications using 3D gaming engines such as Unity or Unreal, and then deploy these applications across desktops, mobile phones, and other platforms. Artists working on AR applications work in the realm of 3D graphics which typically requires knowledge of advanced topics such as the graphics/rendering pipeline, 3D modeling, materials, complex mathematical transformations, animation, and more. These prerequisites can make it intimidating and nearly impossible for inexperienced software

CHI'18, Apr 21–26, 2018, Montréal, Canada

ISBN 978-1-4503-5620-6/18/04. . . $15.00
DOI: https://doi.org/10.1145/3173574.3173983

developers to incorporate the affordances of AR into their paper prototyping practices.

Currently, there are limited solutions for beginners or non-programmers who would like to work with AR. A few recent frameworks such as Google's ARCore [7] and Apple's ARKit [2] can make it easier for developers to create their own mobile, AR applications. However, these frameworks are still targeted towards developers that have substantial software development knowledge. These frameworks are not as useful for beginner developers who would like to use AR to incorporate real-time interactivity into their paper-based interfaces. In addition, these toolkits typically do not provide an API for enabling the use of physical and tangible objects to alter the state of the AR application; instead, they are focused on designing applications where users interact with a mobile device to enact changes on overlaid graphics on a real-time video feed, offering little to no interaction with tangible objects in the real world. This is because the available frameworks do not include mechanisms for developers to define event-based interactions with tangible objects. Consequently, it is difficult to use current AR frameworks to create applications that link interactions with tangible objects to changes in augmented graphics and sounds. In addition, developers that explore non-mobile device based AR technologies are presented with bulky and expensive hardware that is difficult to develop for.

RELATED WORK

The inception of tangible user interface research was (and still is) very much about representing digital information as physical artifacts, and in effect providing users a physical way of controlling said information beyond the classic scenario of a user interacting with a graphical interface on a screen. Hiroshi Ishii and the MIT Tangible Media Group's work has offered a vision of interfaces where "tangible bits" and "radical atoms" unite, illuminating new paradigms for how we think about human computer interaction [11].

Much recent work in the area of tangible user interfaces investigates the development of applications for enabling the mixing of TUIs and GUIs without embedded electronics, i.e. for interfaces where interactions with physical materials shape virtual representations without the need to embed electronic sensors within the tangibles. For example, Smart Objects [8] uses Qualcomm's Vuforia framework to construct mappings between physical objects and their virtual representations on touchscreens. When a mobile device, such as a tablet, is held in front of the TUI, a GUI is overlaid onto different elements of the TUI such as knobs and buttons. Initially, to program the logic behind the TUI, the user must drag and drop certain actions or menu items onto each of the TUI elements using a touch screen. However, once a user has accomplished this they can manipulate the application solely by interacting with the TUI.

Other work investigates adding reactivity to tangible interfaces without the need for attaching sensors to tangible objects through the use of a mounted depth camera, leap motion, and projector to capture the interactions of the TUI and display the resulting application state [6].

Horn et. al created Quetzal and Tern, two tangible programming toolkits for computer science education [9]. Their approach pre-maps specific meanings to a pre-constructed set of composable tangible blocks (e.g. a block to make a robot move forward). Their work is more about using tangible objects to design programs, rather than creating programs to add interactivity to tangible objects.

Prototyping AR experiences with paper materials has been previously studied by [13] and [10]. They argue that standard paper-based prototyping techniques for real-time interactive interfaces often require copious amounts of paper or storyboards, and do not capture the essence of the intended application.

GOALS

Our goals for ARcadia were for it to be low-cost, easy to use, enable abstractions for common TUI elements (i.e. sliders and knobs), enable overlaid graphics and musical output, offer customizable mappings between fiducial markers and their computational abstractions, and enable real-time interactivity with the tangible elements. We decided to support the overlaying of graphics onto the live video feed of the tangible interface to better indicate to users which fiducial markers are "live" (i.e. which markers are currently being tracked by the system), and to help represent the state of controller abstractions like sliders, knobs, and buttons. In addition, we wanted users to program their prototypes in a web browser using a block-based programming language to better support novice programmers.

DESIGN

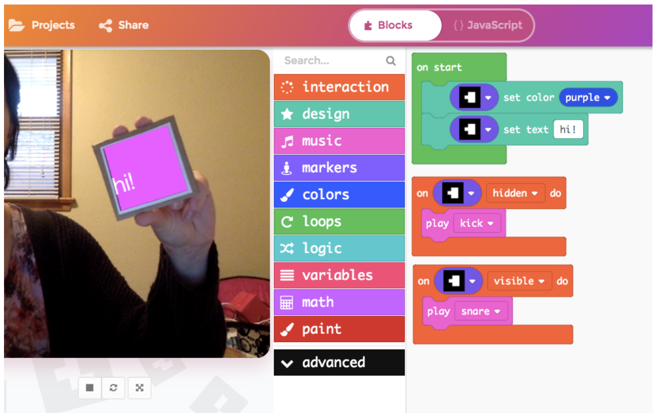
For the initial design of our application we decided to focus on the use case of building real-time musical controllers.

With our goals and use-case in mind we designed ARcadia to support these primary features:

- Customizable declarative mappings of fiducial markers to common user interface elements (e.g. buttons, sliders)
- Real-time interactivity using an event-based programming architecture
- Ability to utilize low-cost construction materials
- Novice-friendly programming
- Built-in programming abstractions of common TUI elements such as sliders and knobs
- No need for specialized hardware
- Real-time music and 3D AR overlays generated in the browser

Consider the following hypothetical scenario of a musician named Carrie who prototypes her own tangible user interface.

Carrie would like to prototype her own disc jockey (DJ) table interface for a musical performance. She wants her DJ table to contain a wheel that she can rotate in order to change the speed of the musical track that is playing. Carrie would also like to have three buttons on her DJ table that she can use to toggle different synthesizer effects on and off: distortion, delay,



```
on start
    set color purple
    set text hi!
on hidden do
    play kick
on visible do
    play snare
```

Figure 1. A screenshot of the ARcadia programming editor [1]. The program on the right side of the editor corresponds to real-time mappings between fiducial markers and different computational abstracts. In this case, when the fiducial marker defined in the program is detected by the camera it will display a purple background with the text "hi!" overlaid on top. In addition, when the marker is "hidden" (i.e. the application loses tracking of the marker) the browser will play a kick drum sample, and when the marker is "visible" (i.e. tracking of the marker is resumed) the browser will play a snare drum sample.

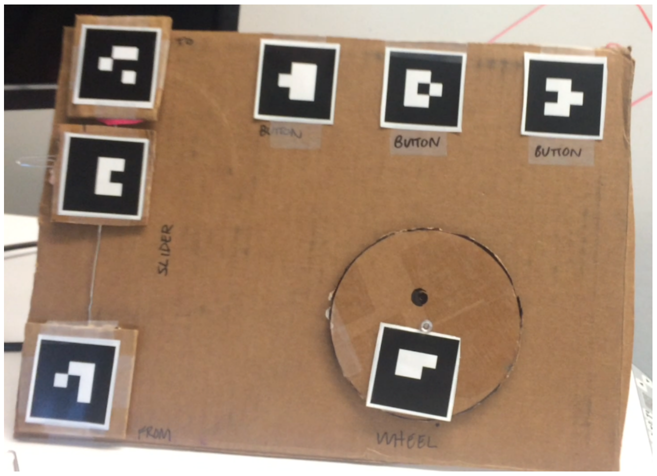


Figure 2. A cardboard prototype of a DJ table. The three fiducial markers in the upper right section correspond to toggle buttons that map to different musical effects, the marker on the wheel maps its rotation to the tempo of the music track that is playing, and the three markers on the left side make up a slider element that controls the volume of the music.

and chorus. In addition, she would like to have a vertical slider that she can interact with to adjust the volume of her music. She needs all of these features to work in real-time since the intended use of her project is a musical performance. Carrie is a beginner programmer and does not want to spend a lot of money to prototype her project as she is worried that she may need to build several prototypes before she finds one that suits her needs. She decides to use ARcadia so that she can simply use her laptop, webcam, paper, and cardboard.

All of ARcadia runs inside a web browser. The fiducial marker tracking system only requires the browser's access to real-time camera data (using the HTML5 features for accessing desktop or mobile devices' cameras). Therefore, users do not require sensors or other types of electronics to build their prototypes; all they need is a computer equipped with at least one camera and some basic office supplies. Designers using ARcadia also program their interfaces within the browser. Once completed, users interact with ARcadia projects in the physical world, through manipulation of the paper-based tangible interface. It is these features of ARcadia that enable Carrie to begin prototyping her interface with low-cost.

Carrie prepares her materials for her project. She prints out a sheet of paper with images of seven unique fiducial markers. She cuts them out in preparation to eventually glue them to her DJ table. She cuts out a roughly 12" x 8" piece of cardboard to use for her table. She then cuts out a cardboard circle to be used for her wheel.

Designers can prepare their ARcadia materials using a few different methods. They can print out several different marker images, cut them out, and then glue them onto cardboard for extra stability. Another option is to hand-draw the markers using a thick black pen (or any color as long as the contrast between the marker and the background is high enough). In this case, Carrie chooses to print out her markers.

Now that Carrie has prepared her construction materials, she begins to assemble her DJ table. She attaches the wheel to the cardboard table using a thumb tack so that it can move freely when she spins it. She then glues one of her fiducial markers onto the wheel. Then, Carrie glues three more markers onto the table that she intends to use as her toggle buttons. Finally, she takes on the more complicated task of building her slider mechanism. She straightens out a paper clip and attaches a marker to it such that it can slide freely up and down along the clip. She then places one marker at the top of the paper clip, and another at the bottom. Carrie has now completed the physical construction of her prototype using paper, cardboard, and other commonly used office supplies. The cardboard and paper prototype of the DJ table can be seen in figure 2.

In our application, we wanted designers to be able to easily construct TUIs and set them up anywhere. In addition, we wanted the tangible prototypes to be as low-cost as possible. In Carrie's case, she already had access to a laptop with a compatible web browser and a camera, which is the most expensive hardware necessary to use ARcadia.

Drawing on Ishii and Ullmer's work we can understand ARcadia in terms of "bits and atoms," where "atoms" are the physical artifacts users interact with and "bits" are the computational abstractions the atoms map to [11]. Fiducial markers in the real world serve as anchors for ARcadia's representational bridge between atoms and bits.

Carrie now needs to set up her camera in order to enable tracking of the fiducial markers via the browser's video capture. Carrie has a tablet equipped with two webcams: a front-facing camera and a rear-facing camera. She decides to use her rear-facing camera so that she can interact with the DJ table with it behind her laptop (figure 6).

In order to enable the image tracking of markers, the designer and/or user must have access to a camera. The camera can be embedded into a laptop, an external camera attached to a computer, a camera in a mobile device, etc. We wanted our system to be able to render 3D graphics atop the fiducial markers so we opted to use cameras and video feeds as opposed to projection like that of Funk and Korn's work [6].

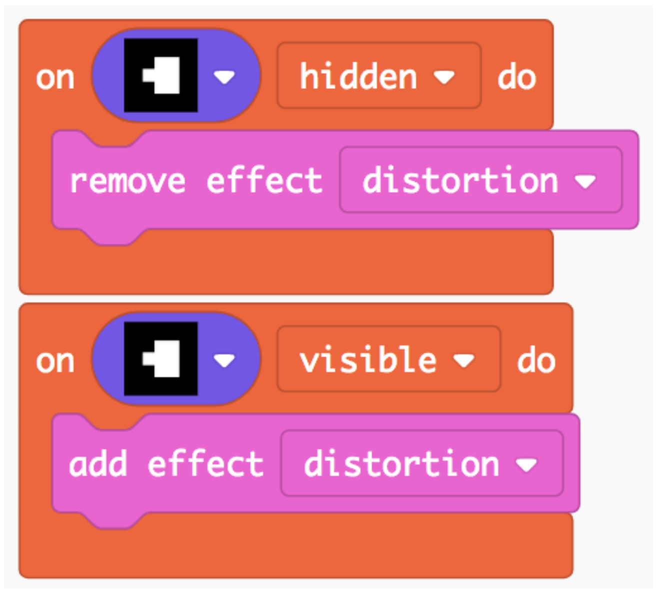

```
on [marker] hidden do
    remove effect distortion
on [marker] visible do
    add effect distortion
```

Figure 3. Code that is mapped to one of the fiducial marker buttons from the DJ table prototype. When the button (i.e. the fiducial marker) is covered by the user's hand and is "hidden" the distortion effect on the music will be removed, when the button is "visible" to the camera again the distortion effect will be re-added to the music.

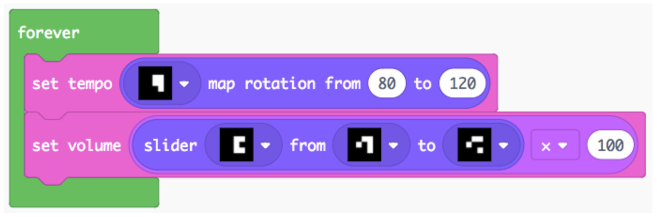

```
forever
    set tempo [marker] map rotation from 80 to 120
    set volume slider [marker] from [marker] to [marker] × 100
```

Figure 4. This code snippet illustrates the wheel and slider elements of the DJ table prototype. The `forever` block defines logic that will run repeatedly. This allows for the real-time interactivity of the DJ table because the values of the slider and wheel will be continuously mapped to the volume and tempo values respectively.

With Carrie's DJ table constructed and her camera rig set up, she now begins to program the interactive elements of her project. First, Carrie starts by programming the mappings for her three toggle buttons. She uses the `on marker hidden do` *and* `on marker visible do` *blocks for each of the three buttons. Inside of each of these blocks she also adds code specifying whether to add or remove a particular musical effect. Figure 3 shows the code Carrie programs for adding and removing distortion based on the visibility of marker 0, i.e. one of the effects toggle buttons for her DJ table. After she finishes programming her buttons, she moves on to program her wheel/knob. As mentioned before, Carrie wants her wheel to control the tempo, or speed, or her music that is playing. To achieve this, Carrie uses a few blocks, mainly the* `map rotation` *and* `set tempo` *blocks. She decides to bound her rotation mapping between the values 80 and 120 because she wants the tempo to be 80 beats per minute (bpm) at minimum, and 120 bpm at maximum. She puts this code inside of a* `forever` *block so that the program continuously maps the wheel's rotation to the music's tempo (figure 4). Finally, Carrie needs to program the functionality of her volume slider. She wants the interactivity to work such that she can control the volume between the values 0 and 100 by moving it up and down. Carrie uses both the* `set volume` *and* `slider` *blocks. Within the slider block Carrie has to map which fiducial markers correspond to which aspects of the slider. She maps marker 5 to serve as the bottom of the slider and marker 7 to represent the top of the slider. She then maps marker 10 to act as the movable element of the slider, meaning that she will move marker 10 up (towards marker 7) when she wants the volume to be higher, and she will move marker 10 down (towards marker 5) when she wants the volume to be lower. She also puts this logic inside of her forever loop (figure 4).*

Our approach allows the user to select the logic and events mappings that different interactions with TUI elements will represent. This is in contrast to Horn's work we discussed earlier where the tangible block objects contain pre-mapped logic for different actions that a robot could perform [9].

As Carrie is programming the mappings of her interface, we see the parallels between the UI elements of her prototype and the common UI elements of buttons, sliders, and knobs. For example, her use of altering a marker's state between "hidden" and "visible" in order to turn on or off a particular musical effect parallels a typical interaction with a touch button on a user interface. Physical buttons normally have binary positions they can be in, whether it is a push button or a toggle button. Furthermore, the functionality of her wheel (or knob) and slider that use markers' positions and rotations to map to different output values mimics how a user would typically interact with those similar elements on a standard UI (figure 5).

Now, Carrie can begin interacting with her prototype.

When Carrie is finished building and programming her prototype, she is then able to interact with it. In this scenario, she is both the *designer* and the *user* (figure 6).

A real-time feed of camera data is displayed to the left of users' code, and the effects of users' programs (i.e. mapping 3D objects to fiducial markers detected in the camera feed) are overlaid on this video. As users make changes to their code, the code executed by the browser updates in real-time, enabling users to see and/or hear the effects of their program modifications immediately. This allows for a very rapid design and development loop.

IMPLEMENTATION DETAILS

ARcadia is built on Microsoft MakeCode's open-source framework for developing in-browser programming editors for beginners where their code can be executed directly within the browser [14]. MakeCode has already been used to create editors for the BBC Micro:bit, the Adafruit Circuit Playground Express, Minecraft, and several other products [15].

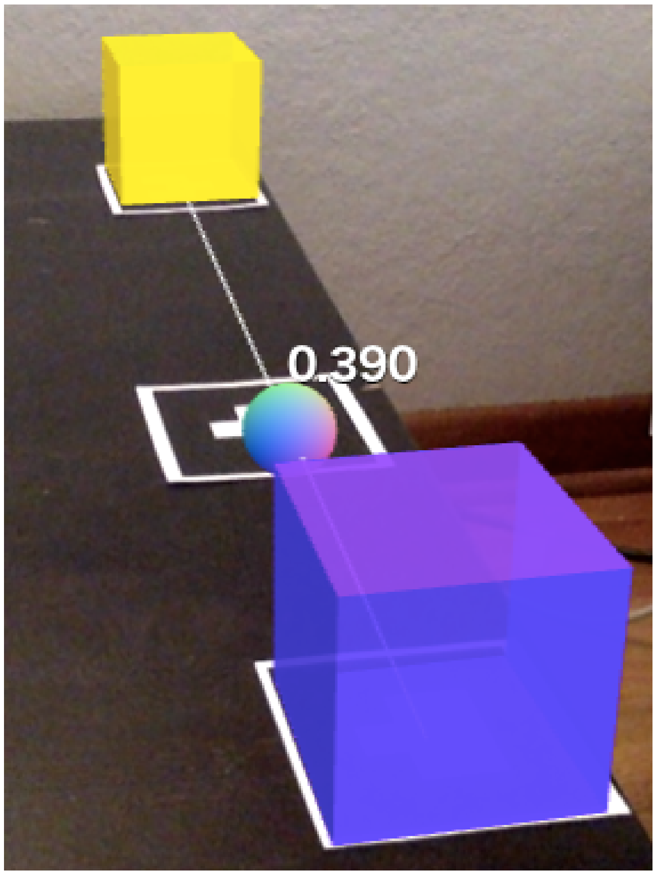


Figure 5. This is a screenshot of the video feed view when a designer declares a slider construct. The fiducial marker at the top of the photo (marker 2) represents the maximum output that the slider can go to. The fiducial marker at the bottom of the photo (marker 0) represents the minimum output the slider can go to. The marker in the middle of the photo (marker 1) is the one the user would typically interact with; she/he would move it closer towards the bottom marker to create a smaller output value, and in contrast would move it towards the top marker to create a larger output value. The middle part of the slider gets mapped to a value between 0.0 and 1.0 based on its distance between the two outer markers.

ARcadia also uses the AR.js [5] and three.js [4] libraries for rendering 3D graphics in the browser, and the jsARToolkit [3] for image-recognition and tracking of environments and several kinds of fiducial markers. AR.js provides a simplified API for initializing different kinds of fiducial markers for tracking from a user‘s webcam or mobile phone camera. In addition, ARcadia uses the Tone.js [18] library for generating real-time audio in the browser.

The system uses ARToolkit's 3x3 barcode matrices to distinguish between different fiducial markers, and ARcadia itself has built-in support for recognizing 16 of these unique markers. Users can distinguish between the different fiducial markers solely by the shape, or they can distinguish the markers by numbers 0 to 15. In the augmented workplace system, users can define any physical object as part of the TUI [6], however since our entire system runs in the web-browser we wanted to limit the amount of processing power needed for real-time

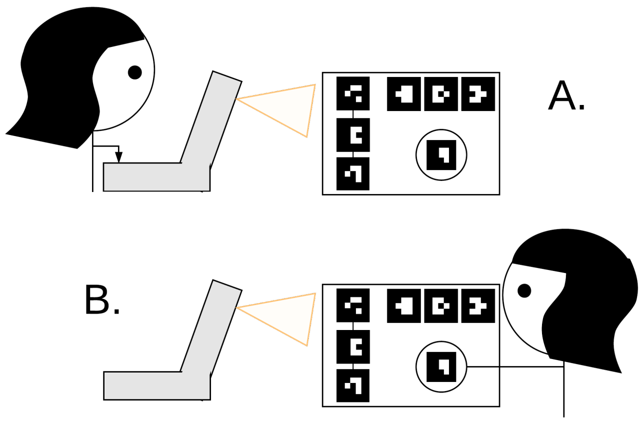


Figure 6. This diagram demonstrates the two roles an ARcadia user might perform: A. *designer* and B. *user*. The designer is the one who constructs the physical prototype and programs the different mappings, the user is the one who ultimately interacts with the prototype after it is built.

interaction by limiting users to 16 different barcode fiducial markers.

The framework is event-driven and is based on relationships between atoms (i.e. fiducial markers attached to craft construction materials); whether it is the relationship of one atom to another, or the relationship of an atom to the camera. In addition, ARcadia has built-in functions for transforming physical relationships between atoms to classical UI elements like sliders and knobs.

EVALUATION

We ran two 90 minute ARcadia workshops with 60 female middle school and high school students in each session. The students were enrolled in a summer Microsoft workshop called DigiGirlz whose goal is to provide opportunities to young women to learn about careers in technology and participate in some hands-on programming workshops. Overall, the participants had limited programming experience, and for some this workshop was the first time they had experienced programming.

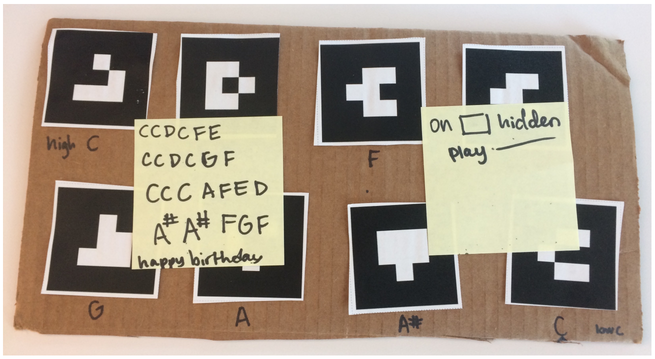


Figure 7. An AR piano for playing "Happy Birthday" developed by two high school students who attended the DigiGirlz workshop. The different notes of the song can be played by covering a specific marker and then uncovering it (i.e. triggering an `on visible` event), offering button-like semantics. The different markers map to the notes labeled beneath them.

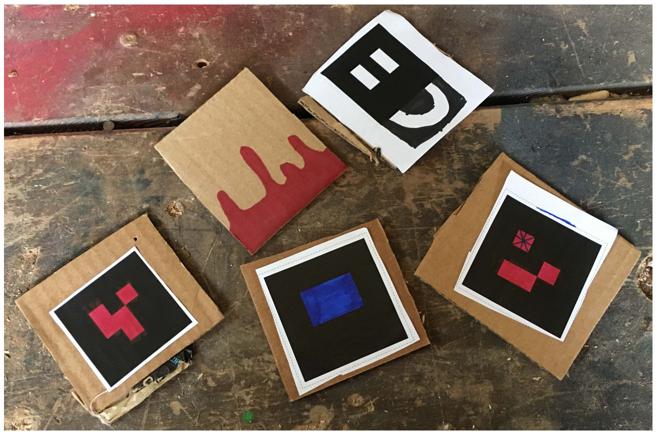


Figure 8. An example of participants incorporating their own aesthetic choices into their fiducial markers.

The workshops

Before each workshop began we placed an assortment of fiducial markers, cardboard, pens, glue, scissors, and other craft materials in the center of each table. Around 10 students sat at each table and each one received a laptop equipped with two cameras (both front and rear-facing webcams). The structure of the groups varied from participants working individually to participants working together in larger groups.

At the start of each session we gave a simple demonstration of how to set up a basic interaction with a fiducial marker. We demonstrated this by writing a short program that let us trigger different drum samples by alternating between eclipsing and revealing the marker to the camera; essentially, when a user covers the marker with her hand the browser plays a kick drum sample, and when the user removes her hand (thus revealing the marker to the camera) the browser plays a snare drum sample. We chose to do this as our demo because it demonstrates the event-driven nature of interacting with the markers in a small amount of code. Because we chose to start the sessions off with this demo, most of the groups created simple, button-based music controller prototypes — a limitation we would like to address in future workshops. In the future we plan to offer participants time to explore the ARcadia editor before we provide examples, and we plan to have more complex examples to show the students (i.e. examples that incorporate sliders and knobs instead of only buttons).

A popular project that students created was an AR piano. These prototypes typically only used button abstractions where events were triggered by hiding and revealing fiducial markers with their hands. One pair created an AR piano that contained only the notes needed to play Happy Birthday transposed to the key of C (figure 7). They placed sticky notes on their piano that explained how to use it, and they wrote the corresponding notes for each marker. The code they wrote worked similar to the code we provided in our demo, i.e. `on marker hidden play C note` and so on for the rest of the notes on their piano.

A problem that the participants often had was deciding whether to use their front or rear-facing camera. Those who used front-facing cameras typically closed their laptops at a sharp angle so that the camera could see the piano on their table, which prevented them from being able to look at the screen while they played it. In future workshops we plan to provide external webcams and camera stands to participants to make it easier to prototype table-top interfaces.

During the workshops we found that participants cared about the aesthetics of the fiducial markers they worked with (figure 8) Some girls colored in different parts of their fiducial markers. For example, one group decided to color in the white part of their marker dark blue, but upon realizing that there was not enough contrast for the marker to be tracked by the camera they decided to use red to color in their other markers. In addition, another group used a marker that looked like an equals sign and added a mouth to it to make it look like a smiley face (see top right in figure 8). One participant wrote in her feedback, “I wish you could put blocks ON YOUR FACE!” We did in fact see one group experimenting with taping fiducial markers on their foreheads, and we have recently been experimenting with temporary tattoos to determine fun and feasible ways to use wearable fiducial markers to control ARcadia.

The lack of physical working space provided to participants led many participants to build basic projects where they would hold up different fiducial markers to the front-facing webcam that would make different sounds, show different shapes, and show different colors. This is in contrast to the table-top style interfaces we thought more participants would construct. One participant created a project where she could use a fiducial marker as a paintbrush, tracing lines on the screen as she moved her "brush" relative to the camera. She used the rear-facing camera on her laptop and attached a fiducial marker to a paper clip so that she could augment the world behind her laptop by drawing pictures onto it. Many of these kinds of projects were fun and creative, but did not exactly capture the types of musical, table-top interfaces we thought participants would focus on.

Participant feedback

Throughout the workshops, we asked the students to anonymously write down things that they liked about the editor, and things that they wished the editor could do. Most of the positive responses had to do with the ease of use, musical functionality, and overall creativity that the system offers. Many of the critical responses, i.e. the “I wish” responses, mentioned problems with the software itself. For example, some girls complained about the image recognition being “spotty”, meaning that ARcadia would occasionally flicker and lose and regain tracking repeatedly. This is a possible limitation of ARToolkit, or can be attributed to users accidentally eclipsing their markers with their fingers, poor room lighting, and/or issues with performance in the browser. Rendering 3D graphics and scheduling musical events requires a lot of CPU and can be quite demanding on the browser.

In addition, many of the students expressed a desire to use ARcadia for more than just making music. One respondent said she would like to see “the possibilities of augmented reality in regular office settings,” while another mentioned she wishes "to design my own personal game in the future." One respondent's opinion that encapsulates the prior two responses is simply, "I wish we could see other things we can do." Future Directions describes the work that we are doing to expand ARcadia to be used across domains beyond simple musical controllers, in order to "widen the walls" of the toolkit [16].

Summary of Design & Findings

We have presented the design of a low-cost, toolkit for the rapid prototyping of real-time, interactive and tangible interfaces. ARcadia enables this through its built-in support for common interface elements and semantics – buttons, sliders, and knobs – and event-based control of system behavior and adjustment of continuous parameters.

We have shown through our initial evaluation that ARcadia can be used productively by novice designers and programmers to create interactive musical prototypes in a short amount of time. We found that the workshop participants generally had ease in iteratively modifying their prototypes. In general, the resulting projects that the girls built would typically take hours and possibly days for even an experienced programmer to create. For example, the "Happy Birthday" AR piano, which was made over the course of 90 minutes by two first-time programmers, would require the work of a much more experienced developer and would still likely take much longer to build. Although the workshops demonstrated ARcadia's ease of use and ability for novices to construct interactive projects, due to space constraints and decisions we made at the beginning of the workshops the projects remained pretty limited in their functionality. Projects only incorporated the use of on or off buttons and no sliders or knobs. In the future we would like to provide an easier way for participants to construct more complex table-top interfaces.

In addition, many of the workshop participants expressed a desire to use ARcadia for other projects beyond simple musical controllers.

CONTRIBUTION

The primary contribution of this work is to demonstrate how AR technologies make it possible for designers to prototype and construct tangible interfaces that require no embedded electronics. Additionally, the fusion of AR technologies and paper prototyping techniques allows for the existence of tools that enable novice programmers and/or engineers to construct said tangible interfaces and incorporate real-time interactivity into their interface designs.

FUTURE DIRECTIONS

We plan to address participants' feedback and pursue additional design ideas we have for ARcadia through additional functionality and user-centered research.

Networked Tangible Interfaces

In order to extend ARcadia to be used across a variety of domains we have been working on adding networking capabilities to allow users to send data in and out of ARcadia between external programs. We have completed two prototypes of networked versions of ARcadia, one that utilizes Open Sound Control (OSC) protocol to send information from ARcadia to programs like Processing, Max/MSP, Unity, etc. For example, we have used ARcadia to build a prototype where fiducial markers can be used to control parameters of a synthesizer. In addition to OSC, we have also been working on a version of ARcadia that uses WebRTC to share data peer-to-peer with other Microsoft MakeCode applications. For instance, we can use ARcadia to send data to a browser-based MakeCode editor for a YouTube scrubber where different interactions with fiducial markers control different parameters of the video like playback rate and volume (an example of this can be seen at [12]).

Studies

In the future we plan to conduct further evaluations of users developing prototypes with ARcadia. In addition, we plan to make alterations to our application based on participant feedback from the pilot workshop with a particular focus on improving system performance around image tracking and audio synthesis, and also increase the creative functionality to increase the potential scope of prototypes that can be created with ARcadia. We also plan to improve the workshop based on limitations we ran into in our pilot study, such as providing external webcams, camera stands, and more working space to participants.

ACKNOWLEDGEMENTS

We would like to thank Microsoft Research and the Microsoft MakeCode team for their tremendous support of this project. Thank you Richard, Sam, and Guillaume for development advice and help navigating the MakeCode framework, and thank you Jacqueline and Abhijith for your feedback throughout the early stages of ARcadia's development. In addition, we thank DigiGirlz for letting us lead an ARcadia workshop for their group. We would also like to thank the reviewers for their constructive feedback.

REFERENCES

1. 2017. ARcadia. `https://laboratoryforplayfulcomputation.github.io/arcadia/`. (2017).

2. Apple. 2017. ARKit. `https://developer.apple.com/arkit/`. (2017). [Online; accessed 5-August-2017].

3. ARToolKit. 2017. Javascript ARToolKit v5.x. `https://github.com/artoolkit/jsartoolkit5`. (2017). [Online; accessed 1-September-2017].

4. Ricardo Cabello. 2017. three.js. `https://threejs.org/`. (2017). [Online; accessed 21-June-2017].

5. Jerome Etienne. 2017. AR.js. `https://github.com/jeromeetienne/AR.js/blob/master/README.md`. (2017). [Online; accessed 21-June-2017].

6. Markus Funk and Oliver Korn. 2014. An Augmented Workplace for Enabling User-Defined Tangibles. *Proceedings of the 2014 CHI Conference on Human Factors in Computing Systems* (2014), 1285–1290.

7. Google. 2017. ARCore. https://developers.google.com/ar/. (2017). [Online; accessed 5-August-2017].

8. Valentin Heun, Shunichi Kasahara, and Pattie Maes. 2013. Smarter Objects: Using AR Technology to Program Physical Objects and their Interactions. *Proceedings of the 2013 CHI Conference on Human Factors in Computing Systems* (2013), 2939–2942.

9. Michael S Horn and Robert J K Jacob. 2007. Designing Tangible Programming Languages for Classroom Use. January 2007 (2007). DOI: http://dx.doi.org/10.1145/1226969.1227003

10. Andrew J Hunsucker and Jennifer Wang. 2017. Augmented Reality Prototyping For Interaction Design Students. *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (2017), 1018–1023.

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

12. Annie Kelly. 2017. Augmented Reality Tangible YouTube Scrubber. https://youtu.be/pbuMtTkT1F4. (2017).

13. Felix Lauber, Claudius Böttcher, and Andreas Butz. 2014. PapAR: Paper Prototyping for Augmented Reality. *Automotive UI'14* (2014).

14. Microsoft. 2017a. Microsoft MakeCode. https://github.com/Microsoft/pxt. (2017). [Online; accessed 5-June-2017].

15. Microsoft. 2017b. Microsoft MakeCode. https://makecode.com/about. (2017). [Online; accessed 11-June-2017].

16. Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. 2009. Scratch: Programming for All. *Commun. ACM* 52, 11 (Nov. 2009), 60–67. DOI: http://dx.doi.org/10.1145/1592761.1592779

17. R Benjamin Shapiro, Annie Kelly, Matthew Ahrens, Benjamin Johnson, and Heather Politi. 2017. BlockyTalky: Tangible Distributed Computer Music for Youth. *The Computer Music Journal. MIT Press.* (2017).

18. Tonejs. 2017. Tone.js. https://tonejs.github.io/. (2017). [Online; accessed 2-July-2017].