PIB: A Physical Interface Builder for Authoring Interactive Environments in the Arts

Eitan Mendelowitz
Department of Computer Science,
Smith College
Northampton, MA 01063

Abstract

The Microsoft Kinect camera and similar technologies have enabled countless interactive artworks that react to body position and gesture. In creating these works, artist use available *application programming interfaces* (APIs) and software libraries. Because these APIs are designed for screen based interaction, they make it difficult to create site specific installations for which interactions are situated in the installation space. This paper introduces PIB, a physical interface builder that addresses the APIs' shortcomings by allowing artists and programmers to define interactive volumes *in-situ* using there bodies in the installation space.

Motivation

With the widespread access to Microsoft Kinect and similar technologies the art world has seen an explosion of visual and auditory works that are able to "see" and react to users' body positions, gestures, and movements. In my own work, I frequently collaborate on public art installations that use stereo vision systems or time-of-flight range cameras to create environments that respond to people's locations in a space through computer controlled lighting, projection, or sound. These types of work are enabled by the availability of computer vision *application programming interfaces* (APIs) coupled with relatively affordable imagers.

The popular APIs typically provide programmers with the locations of hands, body positions, and centers of mass. Some also provide for the detection of specific gestures (like hand waving). One limitation of these APIs is that they are primarily designed to facilitate screen based interaction. As as result, the APIs assume a camera placed directly above or below a screen [MSoft 2014] and define a coordinate system relative to the camera's position.

Due to their focus on screen based interaction, these API's do not provide facilities for interpreting a user's interaction with their environment. Because hand and body locations are reported relative to the camera's location and not relative to the physical space in which the camera is situated the existing API's make it difficult to design interactions grounded in the built environment. As a result, most artworks using depth cameras have been designed to be interacted with while facing a

screen so that the users can see live representations of themselves interacting with onscreen graphics (e.g. [HYBE 2012],[Britz 2012], and [Milk 2012]). Those works which are not designed to be used while facing a screen are almost always site specific. When being installed in a new space, these installations typically require detailed measurements of the installation space coupled with prolonged periods of calibration and customization. For each new installation space, the camera needs to be painstakingly registered within the physical environment.

This paper presents a software system that allows artists and designers to define interactive volumes using their bodies in-situ. This approach extends existing APIs, freeing them from their screen-centric assumptions thus allowing site specific installations to be created without tedious time-consuming calibration processes.

Related Work

For traditional, screen-based applications, most modern *integrated development environments* (IDEs) have a *interface builders*. Popular since the early 90s for decreasing the coding required to create complex *graphical user interfaces* (GUIs), interface builders make it easy for software designers to set the location and appearance of graphical elements such as buttons and menus. The elements can be place on the screen using a mouse rather than requiring programmers to specify locations and appearance parameters in code. [Myers 1992] The interface builder aids in the rapid development of attractive and professional looking screen based software. Unfortunately, no analogous set of tools exist to allow programmers to create non-screen based interactive elements. To answer that need, I have developed PIB, a *physical interface builder*, a software system which allows the designers to define interactive volumes in the physical world using their bodies. Just as traditional interface builders allow software designers to interactively place GUI elements on the screen using a mouse; PIB allows designers to place interactive elements in a space using their hands.

In the same way that GUI builders serve a foundation for screen based interaction, this work is intended to serve as a foundation for physically interactive artworks. [Lee 2011] and [Lee 2004] have attempted to solve the challenges of situating interactions in an environment. Both projects approach the problem of bridging the camera based measurements returned by standard APIs with the physical world through the use of physical markers that are easily detected by computer systems – ARToolKit's fiducial markers and RFID tags respectively. Unfortunately, instrumenting the environment with physical objects may often violate an artwork's visual aesthetic and is not always practical or even possible for every installation space or application.

The same conveniences achieved by physical markers can be achieved using hand gestures and software. PIB enables designers to quickly define interactive 3D volumes in a physical space. These interactive volumes can be made reactive to the touch of a hand, a body part, or to the occupancy of a person. For example, to make an area above a table surface interactive, a designer can user their hands to create a volume just above the surface. Just like interface builders typically make it easy for application developers to receive user interface events, so to does PIB facilitate receiving user events from the physical space. Once the volume is defined, the installation or artwork can then receive notifications that the table has been touched with little or no additional coding.

The Physical Interface Builder

PIB, the physical interface builder, is a Java base tool that uses the SimpleOpenNI [SimpleOpenNI 2014] API to allow artists and designers to create volumes interactively using gestures. To create a volume the designer holds both their hands still, away from their body, for a few seconds. A volume is created between their hands which can be viewed on the computer screen for verification (see Figure 1). Depending on the tool selected the volume can be a sphere or a box. Once created, the volume remains "held" in the designers hands. In this way the volume can be moved or scaled. When the designer wishes to "place" the volume (fix its location and size) the designer simply holds there hands still for another few seconds. The designer is free to create as many volumes as desired.

Figure 1:
Interactive volume creation in PIB.

Created volumes can be edited. All volumes can be grabbed by placing a hand in the object and holding the hand still. Once grabbed the volume can be moved and placed (again by holding the hand still). Spheres can be resized by "grabbing" the surfaces of the sphere with one hand or both moved and resized by grabbing the surface of the sphere with two hands. In a similar manner the dimensions of a box can be adjusted by "grabbing" individual surfaces with one hand. Boxes can be rotated and scaled by grabbing the box with two hands. Using a keyboard, the designer can select and name volumes. The volume names are used to identify events in the artwork.

Once the designer is satisfied with the created volumes their names, locations, orientations, and dimensions can be exported to a human readable text file.

Using Defined Volumes

When running in an artwork or application, volumes previously defined in PIB act as sensors automatically detecting when they are *touched* by a hand or another body part. A body part is said to be *touching* a volume when the part is within the volume. The designer can access each volume by name and poll the volume to see if it is being touched. Alternatively, designers can also choose to implement a listener and receive callbacks when a volume is touched. This design pattern mimics the standard Java approach to receiving mouse and keyboard user interface events.

Designers can also access a volume's location and dimensions. The volumes can be used to define space appropriate coordinate spaces. For example, during the building stage, the designer can place volumes in each corner of the installation space. The location of these volumes can be used to define the bounds of the room.

Designers are also free to edit existing volumes without changing existing code. In this way, a site specific installation can be moved to a new space without any coding. The designer simply needs to move the previously defined volumes to new locations appropriate for the new installation space.

Future Work

While functional, PIB is in early stages of development. We hope to allow new types of volumes (and surfaces) to be defined by extending PIB's gestural authoring language. Additionally, we plan to integrate the tool into popular authoring environments in the arts such as Processing, Max/MSP, and TouchDesigner. This integration may take the form of libraries for those environments or the use of networking protocols like OSC. Once PIB is better documented, the software will be released under an OpenSource license.

Conclusion

In the same way that interface builders allow designers to see and directly manipulate their screen based interfaces, PIB allows installation artists and designers to create and modify interactive elements while situated in a physical space. Artworks can thus be given site specificity without a difficult calibration process.

Works Cited

[Britz 2012] Britzpetermann. 2012. *Schau!* OpenFrameworks, Kinect. http://www.britzpetermann.com/portfolio/schau (accessed January 31, 2014).

[HYBE 2012] HYBE. 2012. *IRIS.* Kinect, Arduino, Porcessing, LCD. http://www.creativeapplications.net/processing/iris-by-hybe-new-kind-of-monochrome-lcd/ (accessed January 31, 2014).

[MSoft 2014] *Kinect for Windows Human Interface Guidelines v1.8.0.* 2013. Kinect for Windows Human Interface Guidelines v1.8.0 Microsoft Corporation.

[Lee 2004] Lee, Gun A, Claudia Nelles, Mark Billinghurst, and Gerard Jounghyun Kim. 2004. Immersive authoring of tangible augmented reality applications. In *Proceedings of the 3rd IEEE/ACM International Symposium on Mixed and Augmented Reality.*

[Lee 2011] Lee, Jae Yeol, Dong Woo Seo, and Gue Won Rhee. 2011. Tangible authoring of 3D virtual scenes in dynamic augmented reality environment. *Computers in Industry* 62 (1): 107-119.

[Milk 2012] Milk, Chris. 2012. *The Treachery of Sanctuary.* openFrameworks, Unity3D, Microsoft Kinect SDK.

[Myers 1992] Myers, Brad A and Mary Beth Rosson. 1992. Survey on user interface programming. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.*

[SimpleOpenNI 2014] "SimpleOpenNI.". SimpleOpenNI. https://code.google.com/p/simple-openni/ (accessed January 31, 2014).