Physically Computing Physical Computing: Creative Tools for Building with Physical Materials and Computation

Michael Gubbels[1] and Jon E. Froehlich[2]
Makeability Lab | Human-Computer Interaction Lab (HCIL)
[1]College of Information Studies, [2]Department of Computer Science
University of Maryland, College Park, MD
{mgubbels, jonf}@umd.edu

ABSTRACT
Physical computing refers to the activity of creating physical artifacts and giving them behaviors through a combination of building with physical materials, computer programming, and circuit building (*e.g.,* connecting sensors and actuators). Physical computing is common among artists, engineers, and even children (*e.g.,* in workshops). Recent tools such as the *Arduino* have lowered barriers to physical computing by abstracting away technical complexity (*e.g.,* interfacing with sensors). However, these tools are based on traditional programming paradigms and are often extraneous to the artifacts being built rather than an integral part. They often tether makers to a computer separate from the context of their making. We present *Pixel*, a tangible user interface for physical computing that addresses some of these issues. We describe our current prototype, and the gestural and visual programming environments for building with it.

Categories and Subject Descriptors
D.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*user-centered design*

General Terms
Design, Human Factors, Languages

Keywords
Physical computing; tangible user interface; constructionist learning; programming language; digital manipulatives

1. INTRODUCTION
Physicality plays a central role in shaping human experiences. It affects the way we touch, think, manipulate, and move about and make sense of the world [8]. Nevertheless, as famously portrayed by Igoe *et al.*'s "finger eye" [10], the human body and the physical environment have been long been ignored by computers. Recently, however, computing and sensing technologies are increasingly built into material things, transforming our artifacts and environments to be "smarter" and more aware of themselves and their context. With this shift, the role of physicality, context, and materiality in computing becomes increasingly important. In this paper, we explore a new platform for building physical computing experiences that takes advantage of its own physicality to enable makers to construct physical artifacts and program their behaviors without a conventional computer.

While recent technologies such as the *Arduino* [1], *Spark Core* [16] *MaKey MaKey* [4], and *Scratch4Arduino* [9, 13] have focused on making some aspects of physical computing easier—specifically programming and connecting electronic peripherals—these tools require conventional means for programming (*e.g.,* a keyboard, mouse, and two-dimensional display), limiting their approachability and how they can be used. Moreover, these systems rely solely on traditional textual programming languages on a computer, which are often physically and even logically separate from the artifact being built. By contrast, consider how clay can be shaped directly with one's hands. Recognizing this separation, others have begun to explore more direct means of computing through tangible user interfaces (TUIs) for programming including *FlowBlocks* [19], *AlgoBlock* [18], and *Topobo* [14]. However, their tangible manifestations, logical constructs, and even syntax are often little more than physical manifestations of their counterparts in traditional programming.

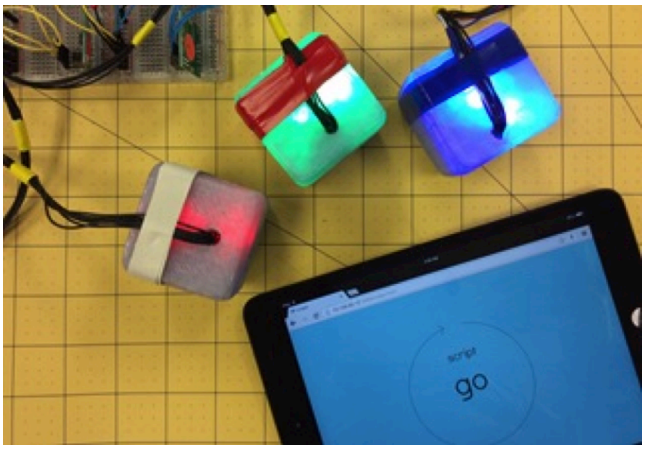

Figure 1: *Pixel* is a new tangible user interface (TUI) for building physical computing experiences. It is comprised of three parts: (i) a set of one or more wireless *Pixel* modules that form the TUI (3 are shown above); (ii) a gestural programming language that can link the modules and specific high-level system behavior; (iii) a visual, touch-based programming language for specifying more sophisticated behaviors (*e.g.*, conditionals, delays)—shown above on the tablet.

We present *Pixel*, a modular tangible user interface designed specifically for creating and shaping physical computing experiences. *Pixel* is programmable through the performance of specific movements with modules, called gestures, and a visual programming language. *Pixel* consists of three parts: *(i)* a set of one or more programmable, cube-shaped modules that together form the modular TUI, *(ii)* a gestural programming language for programming the TUI, used to specify high-level system behavior, and *(iii)* a visual programming environment for specifying detailed system behaviors.

Pixel abstracts some of the complexity of embedding behaviors in artifacts being built, specifically loops and I/O mappings. Our abstractions take inspiration from music sequencers (*e.g.,* Monome 64 from *monomer.org*) and the potter's wheel. Like

IDC 2014, June 17-20 2014, Aarhus, Denmark. http://www.idc2014.org

these, *Pixel* provides a "live" looping context into which modules can be sequenced using gestures *and* for programming individual modules via the visual environment. This allows makers to shift their attention from explicitly specifying loop constructs in an abstract syntax to directly sequencing behaviors themselves within an existing looping infrastructure, fundamentally changing the programming activity. Similar to the way a potter's wheel provides a potter with a platform that continuously rotates to assist them in shaping clay, *Pixel* provides an automatically looping structure into which modules can be sequenced using the gestural language. This enables makers to focus on composing behaviors rather than constructing looping constructs and communication between devices. This is analogous to shifting a traditional potter's focus to directly shaping clay on a potting wheel from creating the potting wheel or other tools themselves or sculpting the clay *as if* it were made on a potting wheel. We believe this approach may lead to a novel paradigm for designing physical computing platforms that support a more seamless exploration across the boundaries between computation and materiality. These ways of programming are described further in Section 4.

Our early-stage work on *Pixel* offers three contributions: ***(i)* a novel interactive loop paradigm** that creates a loop that repeats automatically and continuously. This extends the notion of "event loops" as expressed by *Processing* and *Arduino*, further abstracting unnecessary complexity from the user; ***(ii)* a physically-based gestural programming language** that allows makers to sketch high-level program structure between *Pixel* modules and peripherals (e.g., sensors) and to create mappings between input and output peripherals; ***(iii)* a visual programming language** that allows makers to compose sophisticated module behaviors via a touch-based tablet environment (*e.g.,* changing the speed or direction of a connected motor).

To illustrate how one might use *Pixel*, we develop an example scenario throughout this paper inspired by Alan Kay's demonstration of Sqeak Etoys [7]. We will show how a child named Alanna uses *Pixel* to build a remote control car (Figure 2).

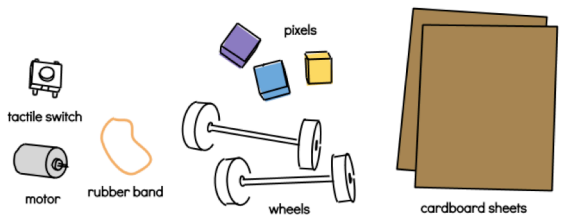

Figure 2: The materials used in our example, in which Alanna builds a remote control car with Pixel.

2. BACKGROUND AND RELATED WORK

Researchers have long appreciated that physical materials can influence how people understand and engage with the world. In 1837, Friedrich Fröbel, the inventor of the first kindergarten in Germany, famously designed a diverse range of physical objects to help children learn and to recognize the common patterns and forms found in nature [3]. These material manipulatives served as "thinking tools." They provided a "medium" for children to explore abstract concepts such as shape, size, and color.

In the same way, researchers today appreciate the computer as a "computational material" that might support children to discover and recognize patterns in the physical world, and to *create new patterns and behaviors* within it. Papert captures this sentiment in his assertion, "*We can give children unprecedented power to invent and carry out exciting projects by providing them with access to computers, with a suitably clear and intelligible programming language and with peripheral devices capable of producing on-line real-time action*" (p. 353, [11]). This vision appears clearer than ever before with the advent of affordable embedded physical computing platforms (*e.g., Arduino*), systems that sense, respond to, and affect the physical world, and even have Internet connectivity [*e.g.,* 13]. Yet, certain aspects of programming (*e.g.,* syntax [17]) as well as connecting peripherals to physical computing platforms (*i.e.,* connecting an accelerometer to an embedded computer) present significant challenges to children.

Visual programming languages (VPLs) may simplify some aspects of programming. For example, *Scratch*, which was inspired by the way children play and build with *Lego*, presents programming as a collection of blocks visually designed to snap together in specific ways to create programs [12]. *Scratch* has been found to successfully introduce children to programming concepts in an engaging manner [*e.g.,* 5]. Similarly, *Modkit* (and *Scratch4Arduino*), a VPL heavily influenced by *Scratch*, suggests that it simplifies some *programming* tasks on the Arduino [2]. However, these VPLs are not well integrated with the task of *physically connecting peripheral devices* (*e.g.,* a DC motor) to computing platforms, a crucial part of physical computing. *Pixel*'s VPL provides step-by-step guidance *in situ* for connecting electronic peripherals (such as *Adafruit* does with tutorials).

Recent efforts, such as *littleBits, Snap Circuits,* and, earlier, *Lego Mindstorms* simplify aspects of electronic circuit building for children (*e.g.,* [6]). However, these are kits designed for use with proprietary components, not common, widely available electronic peripherals. Others have simplified interfacing with prototyping platforms to ease connecting physical materials and electronics. For example, *MaKey MaKey* allows children to create electronic switches and improvise input devices using a wide range of conductive materials, including bananas, copper tape, wire, fruit, and skin [4]. *Pixel* takes this further by additionally allowing children to *(i)* connect electronic output devices to modules (in addition to inputs) to create conditional behaviors and *(ii)* create parallel sequences of these conditional behaviors.

While *Scratch* and *MaKey MaKey* provide valuable insight for how to design a programmable physical computing platform, they also have limitations. *MaKey MaKey* is not easily programmable (it uses *Arduino*) and is not, itself, affected by manipulating the materials connected to it. Similarly, manipulating peripherals connected to a computer does not affect the programs built in *Scratch* (or vice versa). This means that when makers change some aspect of the materials connected to a computer, they cannot observe the effect on the program (or vice versa). *Sqeak Etoys* [7] and *Topobo* [15] hint at ways to directly address this limitation through design. These allow programmers to adjust behaviors (*e.g.,* rotation) of built objects through direct rotation of virtual and physical objects themselves. *Pixel* allows the user to specify new behaviors through gesture and the visual programming environment—these behaviors are manifest immediately in the modules themselves.

3. DESIGN AIMS

We have six design aims drawn from the above literature as well as observations from our own experiences as makers and the first author's observations of children while facilitating activities at workshops, museum exhibits, and community events. These making activities focused on creating artifacts from a wide range of physical materials (*e.g.,* paper, copper tape), basic digital

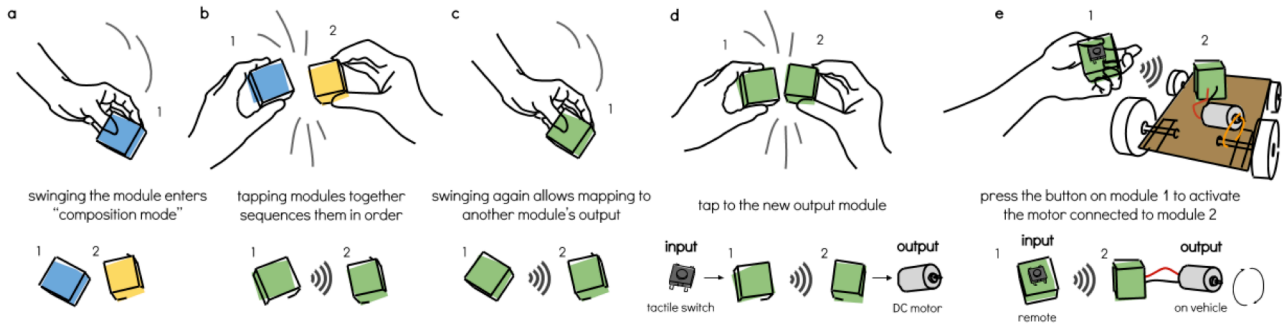

Figure 3: A Pixel module is a self-contained wireless embedded computer that recognizes gestures and can form ad hoc connections with other Pixel modules. Modules also support wired connections for integrating with I/O peripherals. In (e), a tactile button is connected to a Pixel module, which, when pressed, activates a connected motor.

electronics (*e.g., MaKey MaKey*, batteries, LEDs, switches), and programming (in *Scratch* and *Arduino)*.

While impressed by how quickly children understood some facets of prototyping tools like *MaKey MaKey* such as how to connect a custom video game controller using alligator clips, we were dissatisfied with how these tools constrained children's making activities. For example, because both *Scratch4Arduino* and the *MaKey MaKey* require physically tethering to a computer, children seemed confined in space and did not leverage the extent of material possibilities and the physical space around them. Moreover, the reliance on a traditional computer seemed to separate the maker from the experience of working in the physical world and with physical materials. With *Pixel,* we wanted to address these limitations. Thus, *Pixel* modules can be distributed and programmed throughout the maker's environment (supported both by the gestural and visual programming languages), freeing them from working at a fixed location (*i.e.,* the computer) and allowing greater exploration of the materials themselves.

While our work is still preliminary, we are guided by the following six design aims:

- **Design to emphasize materiality:** Allow the makers to stay focused on using materials and the affordances of the physical world. For example, eliminate the need to tether to a computer.
- **Allow programming everywhere:** Relatedly, eliminate the need to write programs away from the physical context of a maker's work (such as when using a traditional computer for programming). Allow makers to do programming tasks among the materials.
- **Provide responsive feedback:** Provide immediate responsive feedback for actions to reveal the system state and to support thinking, playing, and confidence in the system, and to support working with materials.
- **Hide system complexity:** Similar to popular programming languages, we aim to abstract unnecessary system complexity. For example, we aim to free makers from having to create loop structures by creating them by default.
- **Allow easy connection:** Support making intuitive links between physical materials, electronic peripherals, and *Pixel* modules. For example, connecting a tactile switch as an input to a module.
- **Allow easy sequencing:** Provide easy support for sequencing events. For example, easily creating a sequenced light show.

4. PROTOTYPE OVERVIEW

Our overarching goal for *Pixel* is to design a physical computing platform that makes the combined experience of computer programming, circuit building, and using materials more coherent. As noted in the introduction, *Pixel* is comprised of three parts: *(i)* "smart" physical modules that form the TUI; *(ii)* a gestural programming language that defines the set of gestures recognized by modules, and *(iii)* a visual programming environment for specifying detailed system behaviors. We cover each below.

4.1 Modules and Gestural Programming

Each *Pixel* module (*e.g.,* Figure 3) is a self-contained wireless computer that can automatically recognize gestures, form *ad hoc* wireless connections with other *Pixel* modules, and has physical ports for connecting electronic input and output peripherals. Our current prototype modules (viewable in Figure 1) are plastic enclosures fabricated using a 3D printer containing an *Arduino*-compatible microcontroller, ZigBee mesh and Wi-Fi networking breakout boards for communication, an inertial measurement unit for gesture recognition, a circuit to connect I/O peripherals, and an RGB LED. The embedded LEDs give each module a "personality" and deliver immediate visual feedback about recognized gestures and program state.

Module behavior: By default, each module detects the state of its electronic input switch (either "open" or "closed"), and, if "closed", powers a connected electronic output device (if any). This succession of behavior from input to output is called the module's *activation routine*. For example, a tactile switch could be connected to the input port and a DC motor could be connected to its output port with alligator clips. When the tactile switch is pressed, the motor activates.

Modules operate in one of two modes. In "performance" (or "execution") mode, each module simply performs its activation routine. To use gestures, however, modules must be in a second mode called "composition" mode. In this mode, gestures are used to specify the behaviors that make up the high-level structure of a *Pixel* program, primarily creating sequential behavior between multiple modules and specifying conditional behaviors.

Each *Pixel* module recognizes the following gestures: "swinging," being shaken, tapped to another module, and whether a module is being tilted left or right. *Pixel* responds to a gesture by providing direct visual feedback and changing *Pixel*'s behavior as defined by the gestural language. For example, "swinging" (raising a module, then quickly swinging it down again) causes a module in performance mode to enter composition mode and "shaking" reverses the effect of the gesture, returning to performance mode (*i.e.,* similar to common "undo" functions).

Connecting modules: Modules can be wirelessly connected to one another with gestures. For example, tapping two modules together puts the modules in sequence, forming looping patterns of behavior. The loop is created and started automatically by *Pixel* and runs *ad infinitum*. One by one, each module in a sequence checks the state of its input switch (if connected) and supplies power to a connected output device (if any). *Pixel* visualizes

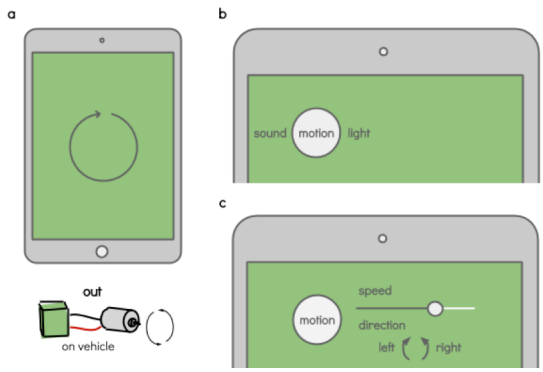

Figure 4: While gestures are used for connecting modules, establishing sequences, and specifying I/O, the visual composer allows makers to fine-tune module behaviors. Above, Alanna is using the visual composer to specify the speed of her vehicle's motor.

looping patterns of behavior by softly lighting the modules in a "sequence color" (*e.g.,* green in Figure 3) and lighting the "active" module in the sequence brightly. These looping patterns form the high-level structure of a program for *Pixel*.

Loops: Programs in *Pixel* are looping behavior sequences. *Pixel* has two types of loops: one at the system level (across module sequences, described above) and another in each module (described in *Visual Programming Environment*).

Example scenario, continued: Figure 3 shows the steps Alanna takes to create a remote control vehicle using *Pixel* and basic materials (Figure 2). She creates a simple cardboard vehicle with wheels, gesturally sequences and remaps the "remote" module's input to the "vehicle" module's output. She connects a tactile switch to the "remote" module, and connects a motor to the "vehicle" module's output. Finally, she tapes the "motor" module and the motor itself onto the cardboard vehicle's platform. With that, she can press the tactile button on the first module to wirelessly control the motor on the vehicle—her very own remote control vehicle! In the next section, we describe how Alanna changes the motor's speed using the visual environment on her tablet.

4.2 Visual Programming Environment

To finely tune module behavior, *Pixel* includes a touch-based visual programming environment, accessible from a mobile device (Figure 4). The visual environment offers an interface to configure each module's behavior loop wirelessly and in real-time. For example, with the visual environment, one could configure a module's output to gradually increase the brightness of an attached LED. With gestures alone, the LED can only be made to switch on or off.

When first opened, the visual environment shows the behavior loop for the first module in the sequence. One can swipe left or right to select the behavior loops for other modules. With the module of interest selected, one can add behaviors represented as nodes to the module's behavior loop.

Example scenario, continued: Figure 4 shows how Alanna adjusts the speed of her car's motor using the visual environment. First, she opens the visual environment on her tablet, creates a "motor" behavior node, and then adjusts the motor's speed by dragging a "slider", monitoring the motor speed as she changes it.

5. CONCLUSION AND FUTURE WORK

Pixel is an ongoing effort to build a physical computing platform that offers more complete support for its constituent activities, specifically programming, circuit building, and working with physical materials. We will continue developing *Pixel*'s gestural and visual environments, and explore collaborative behavior composition with both environments. We will also refine the physical design of modules. (*e.g.,* explore enclosure materials). We plan to conduct evaluations of *Pixel* in museums and workshops. Ultimately, we hope that our work will inspire further development on physical computing tools that provide integrated support for making with physical materials, computer programming, and electronic circuit building.

6. REFERENCES

[1] Arduino: *http://www.arduino.cc/*.

[2] Booth, T. and Stumpf, S. 2013. End-user experiences of visual and textual programming environments for Arduino. *End-User Development*. (2013), 25–39.

[3] Brosterman, N. 1997. *Inventing Kindergarten*. Harry N. Abrams.

[4] Collective, B.M. and Shaw, D. 2012. Makey Makey. *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction - TEI '12* (New York, New York, USA, Feb. 2012), 367.

[5] Franklin, D. et al. 2013. Assessment of computer science learning in a scratch-based outreach program. *Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE '13*. (2013), 371.

[6] Johnson, S. and Thomas, A.P. 2010. Squishy circuits. *Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems - CHI EA '10* (New York, New York, USA, Apr. 2010), 4099.

[7] Kay, A. 2005. *Squeak Etoys, Children & Learning*.

[8] Klemmer, S.R., Hartmann, B. and Takayama, L. 2006. How bodies matter. *Proceedings of the 6th ACM conference on Designing Interactive systems - DIS '06* (New York, New York, USA, Jun. 2006), 140.

[9] Millner, A. and Baafi, E. 2011. Modkit. *Proceedings of the 10th International Conference on Interaction Design and Children - IDC '11* (New York, New York, USA, Jun. 2011), 250–253.

[10] O'Sullivan, D. and Igoe, T. 2004. *Physical Computing: Sensing and Controlling the Physical World with Computers*. Thomson Publishing Group.

[11] Papert, S. 1980. Teaching Children Thinking. 5, (1980), 353–365.

[12] Resnick, M., Silverman, B., Kafai, Y., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E. and Silver, J. 2009. Scratch. *Communications of the ACM*. 52, 11 (Nov. 2009), 60.

[13] Rosenbaum, E., Eastmond, E. and Mellis, D. 2010. Empowering programmability for tangibles. *Proceedings of the fourth international conference on Tangible, embedded, and embodied interaction - TEI '10* (New York, New York, USA, Jan. 2010), 357.

[14] Solos, H., Parkes, A.J. and Ishii, H. 2004. Topobo : A Constructive Assembly System with Kinetic Memory. (2004).

[15] Solos, H., Parkes, A.J. and Ishii, H. 2004. Topobo : A Constructive Assembly System with Kinetic Memory. (2004).

[16] Spark Core: *https://www.spark.io/*.

[17] Stefik, A. and Siebert, S. 2013. An Empirical Investigation into Programming Language Syntax. *ACM Transactions on Computing Education*. 13, 4 (Nov. 2013), 1–40.

[18] Suzuki, H. and Kato, H. 1995. Interaction-level support for collaborative learning. *The first international conference on Computer support for collaborative learning - CSCL '95* (Morristown, NJ, USA, Oct. 1995), 349–355.

[19] Zuckerman, O. and Resnick, M. 2005. Extending Tangible Interfaces for Education : Digital Montessori-inspired Manipulatives. (2005).