

Tangible Computation Bricks: Building-blocks for Physical Microworlds

Timothy S. McNerney
+1 617 630 9244
mc@media.mit.edu

Draft: Please do not distribute!

Version: 9/11/00
(cleanup 2.2 of CHI 2001 submission)

ABSTRACT

Learning activities that involve computation do not necessarily need to involve a computer. The approach presented in this paper avoids user interfaces which need a screen, keyboard, and pointing device, offering instead physical building-blocks augmented with embedded microprocessors. Using these techniques offers computational learning experiences without the cognitive overhead required by many computer-based learning tools. This paper introduces a general-purpose, constructive tangible user interface device, the Tangible Computation Brick. It describes several tangible programming languages that can be implemented with a handful of Bricks and applied in a variety of domains. The paper focuses on one language that was designed for scientific exploration, and discusses informal observations on test subjects. This system is easy to teach to novices, and it is successful at allowing students to focus on the learning task at-hand instead of being distracted by the user interface.

Keywords

Tangible interface, debugging, programming, functional programming, science learning, children, educational toy

INTRODUCTION

A number of computer programming paradigms and an even greater number of programming languages have been developed. The vast majority are text-based languages, but a notable few are graphical. National Instruments LabVIEW is perhaps the most successful graphical programming environment to date. It presents the user with a familiar signal-flow model of data acquisition and processing. Its strength is that it closely parallels how scientists and engineers work with actual test equipment, thereby providing a natural framework for scientific inquiry, much like electronic spreadsheet programs successfully built on traditional accounting practices using paper spreadsheets.

When people do scientific and engineering exploration in a hands-on way, they are necessarily interacting with the

physical world. With the sensors they have at-hand, the researchers acquire data for analysis and visualization using software running on computers.

When the sensors are in-hand and the software is on-screen, there is not only a physical separation but a cognitive separation between the physical world (where the sensors are), and the virtual world (where the programs are). One solution to this problem is to move programs (and the building-blocks they are made of) into the physical world.

The author has implemented such a system by embedding Logo [23] programmable microprocessors into small, custom-built LEGO[®] bricks. This system is a powerful example of a constructive tangible user interface that promises to make the task of programming for scientific experimentation more accessible to newcomers and more efficient for seasoned veterans.

DIGITAL CONSTRUCTION SETS

Since the early days of Logo, Seymour Papert, his colleagues and students have been building self-contained virtual environments called *microworlds* so children can explore scientific concepts such as gravity and planetary orbits in a simplified setting. Tangible Computation Bricks make it easy to build *physical microworlds* by bringing together physically manipulable computational objects with real-world sensors.

AN EXAMPLE

Figure 3 shows a Digital Construction Kit designed for doing science experiments that explore the concepts of distance, speed, and scaling measurements to make them more meaningful to us.

Let's take a quick tour of the kit, starting from the top of Figure 3 and moving counter clockwise. The unlabeled brick at the top is a battery brick that provides a base to build on and supplies power to the entire stack. *Display* and *Click* let you see and hear what is going on inside the computation. *Measure* and *Trigger* let you bring sensor readings into the stack of computational Bricks. *Multiply* and *Subtract* let you do arithmetic. *Count* lets you count events, and *Counts/t* (pronounced "counts per unit time") counts events for a specific period of time (e.g. 10 seconds). Also visible in this figure is a *knob card* (next to *Measure*), a sensor interface card (already inserted into *Trigger*), two *constant*

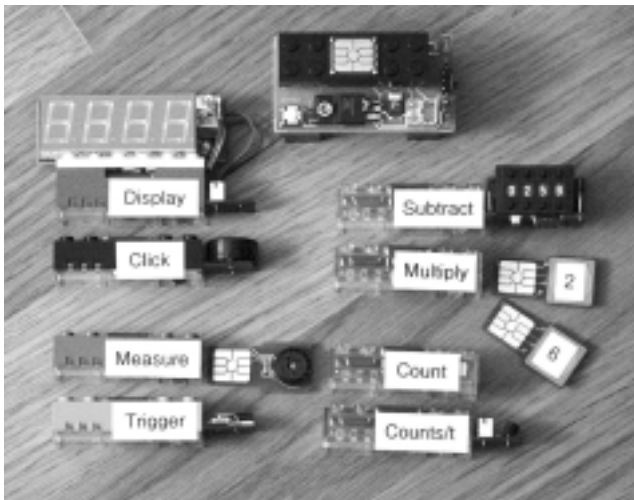


Figure 1: A Digital Construction Kit

cards, 2 and 6, (next to *Multiply*), and a *thumb-wheel card* for “dialing up” arbitrary 1-4 digit numbers.

Figure 2 shows an example of the language in action (a full demonstration can be found in the Video Figure included in the CHI 2001 video program). This experiment was inspired by my colleague's daughter who asked her father to show how a bicycle trip computers work inside. The problem is that if you open one up there is nothing to see, so the best way of explaining how they work is to build one ourselves.

Here data flows from the bottom up. A magnet sensor on the toy bicycle is plugged into the *Trigger* Brick. Every time the bicycle wheel goes around, the magnet moves past the sensor. *Trigger* detects this and sends a message up to *Click*, which beeps once and forwards the message to *Counts/t*, which in turn, tallies messages for 10 seconds and passes on the total to *Display*. The extra display in the lower right is connected to *Counts/t*. It is used to view the internal state of the counter inside *Counts/t*, which would be otherwise invisible between 10 seconds clock “ticks.”

DEBUGGING AS A CENTRAL ACTIVITY

One of the things that is great about this construction set is that it naturally encourages children to start simple and to add complexity on top of already working structures. But getting things to work is more than half the challenge. A necessary part of computational exploration is *debugging*, the process of figuring out why things don't work as expected.

Immediacy

In “Debugging and the Experience of Immediacy” [40], Ungar, Lieberman, and Fry enumerate fundamental properties of good debugging tools and beautifully articulate a central tenet that motivated the development of the Tangible Computation Bricks: “A good user interface brings you face-to-face with whatever is being manipulated or experienced.”

Introspection: Debugging in a Closed System

An important property of the Digital Construction Kit is its ability to debug itself. The *Click* and *Display* bricks can be used to examine intermediate data in a stack, and as previously mentioned, the extra display is used to examine the inner workings of *Count/t*. The bottom line: there is nothing that can be hidden from these debugging tools.

The Kit makes it easy to understand the details of individual building-blocks and small assemblies before moving to bigger projects. For example, *Click* can be used to *listen* to the difference between *Trigger* and *Measure*: Placed above *Measure*, *Click* will beep continuously. Put it above *Trigger*, and it beeps only after high-to-low transitions of the input data stream. This can be demonstrated in isolation by using the easily-manipulated *knob card* in place of a sensor.

Consistency

The *constant*, *knob*, *sensor*, and *thumbwheel cards* can be used interchangeably. If you want to understand how the thumbwheels work, insert it in the measure card, and stack a *Display* on top.

PHYSICAL DESIGN

The physical design of the Brick evolved over a number of months. A first prototype of the stackable, batteryless microprocessor concept was built inside a standard 4x2 LEGO brick. It had only a single LED, and no card slot. Power was supplied by a special brick at the base of the stack, as it is today.

With the task of programming clearly in mind, it seemed important to include an interchangeable parameter mechanism, and thus the card slot was born. The Brick became a device with two distinct *affordances* [21]:

- Stacking bricks (see Figure 3)
- Inserting cards (see Figure 4)

The Pros and Cons of Assembly Constraints

The Bricks only stack in one direction: up. Many people have made the very same comment/suggestion, “wouldn't it be great if you could build 2-dimensional or branching structures...” While it is beyond the scope of this paper to speculate where this might lead, the author feels obliged to

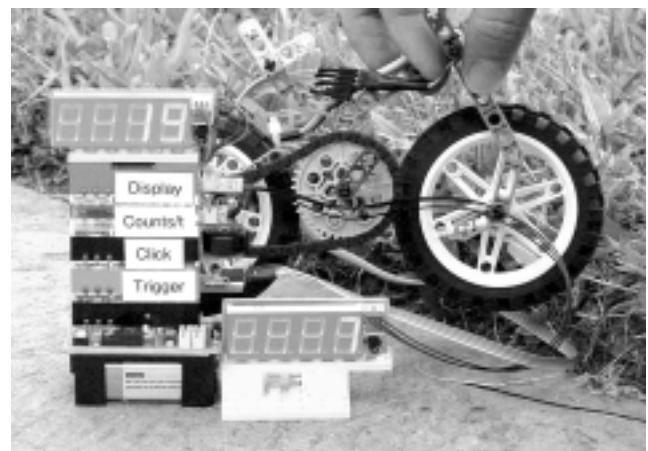


Figure 2: Bicycle trip computer microworld in action

caution that introducing this sort of complexity is not without cost. Two of the main goals of this research is ease of explanation and ease of learning. Although the linear stacking constrain does limit expression, its simplicity is a clear benefit for the novice.

LANGUAGE DESIGN

Although the Tangible Computation Bricks might be intriguing by the nature of their simplicity and flexibility, they become worthy of note only when they can be used to build a compelling tangible language and application. Experiments with a number of different languages were needed to find a language that met these goals:

- A language that is useful even with a small vocabulary.
- A system whose behavior can be explored, both for learning and for debugging, using resources built into the language itself (introspection). And finally
- An application that truly justifies implementing a computational language in a physical, tangible medium.

Language Types

Simple Sequential

For the “Dance Craze Buggies” demonstration [6] visitors used the Bricks to teach new dances to a pair of toy cars. This dance language was strictly sequential, containing no control structure like *repeat*. The parameter cards were used to specify repetition and direction. For example, a “Small step” Brick with a -3 card meant “take three small steps back” (cha-cha-cha!).

Sequential with control

The microwave oven language introduced simple control structure, *Repeat* and *End Repeat*, that could be used to repeat a (sub)sequence of commands some number of times. An example of *Repeat* can be seen in Figure 3 (The *End Repeat* is implied).

Prioritized Behavior Mixing

For a demonstration to LEGO executives, my colleague Bakhtiar Mikhak built an very different kind of language for controlling a toy robot. Instead of specifying an action



Figure 3: Affordance: Stacking bricks together



Figure 4: Affordance: Inserting a card

in a sequence, each Brick represented a simple behavior such as *seek light* or *avoid obstacles* [7]. Which Bricks were stacked on top of the robot *and* the order in which they were stacked determined the overall behavior of the robot. (No parameter cards were used.)

Functional Programming

The final and most successful tangible programming language was inspired specifically by Abelson and Sussman’s *streams* model of computation [1], and more broadly by the field of functional programming [14]. This was also the first completely distributed language implementation, where the Bricks not only *specify* the computation, but also *perform* the computation.

Early applications used the Tangible Computation Bricks as mere symbols in a grammar, sentences to be parsed and interpreted by a Cricket connected to the base of the stack. Indeed the early language implementation worked in just this way. Figures 3 and 4 illustrate the microwave oven programming language. The Cricket in the 1999 oven-of-the-future demonstration waited for a stack of Bricks to be placed on its receptor base, scanned the bricks to determine their order and parameters, and then directed the oven to execute the instructions specified by the Bricks. One slightly disconcerting “bug” was that the oven continued to cook even when the “program” was removed, clearly indicating to the user that the Bricks weren’t directly controlling the oven.

MICROPROCESSOR ARCHITECTURE

Bricks

At the core building-block of the system is a programmable microprocessor embedded in a plastic 6x2 LEGO System® brick. The Brick’s microprocessor architecture is based on that of the Cricket. [20]. The Cricket is one in a series of small, battery-powered microcontrollers developed at MIT that inspired the commercially successful LEGO Mindstorms RCX “programmable brick.” Although the Cricket was initially designed with children in mind, just like its

commercial cousin, it has become popular among older students and adults for prototyping and experimenting.

A Cricket has two analog sensor ports, two motor control ports, a serial peripheral interface bus, a piezoelectric beeper, a wireless infrared transceiver, but perhaps more importantly, it all fits neatly on top of a 9v battery holder. Not only can programs written in Logo on a PC be downloaded into a Cricket via the infrared link, but the user can observe and debug programs inside a Cricket by typing Logo statements at an interactive command-line interface.

The design of the Brick begins with a stripped down, battery-less Cricket and adds three LEDs (in place of motors), and a second serial peripheral bus. This additional bus port allows the Brick to respond to communications from the Brick below. The special card slot in the side of the Brick serves a number of (electronic) interface purposes:

- Analog sensor input
- Non-volatile memory interface (e.g. constants)
- Beeper output
- Serial peripheral bus interface (e.g. display)
- Capacitive touch sensor
- Infrared wireless communication (e.g. downloading)

Each Brick has an ISO “smart card” connector on the top and bottom that allows *communication* with other Bricks.

Cards

A variety of tiny cards were designed to easily, but firmly, slide into the Brick’s card slot. Some cards were built specifically with language-based applications in mind:

- EEPROM card, with capacitive touch sensor [25]
- Knob card
- Thumb-wheel numeric input card
- Real-time clock card

while other cards were designed specifically to restore certain essential Cricket functionality that didn’t fit inside the plastic shell:

- Infrared transceiver and peripheral bus card
- Analog sensor port card
- Beeper card

RELATED WORK

Programming Languages for Children

The work describe here sits squarely between visual programming languages [24, 32], tangible interfaces [12, 15, 34, 35, 37, 39], and “end-user” programming [33]. It focuses specifically on children as programmers. A number of programming systems have been designed for children: Logo [2, 23], ObjectLogo [9], the TORTIS Button Box and Slot Machine [24], ToonTalk [17], Cocoa (a.k.a. KidSim) [32], Agentsheets [11, 26], and AlgoBlock [34, 35], just to name a few. This work follows previous research done at the Epistemology and Learning group at the MIT Media Lab, where researchers have developed a number of computational toys [18, 29, 30] designed with education in mind. The Tangible Computation Bricks project was heavily influenced by the methodology and research agenda of the “Beyond Black Boxes” (BBB) initiative, designed to encourage kids to explore science by building their own scientific apparatus.

The author has been particularly inspired by fundamental principles of human-centered design espoused by Don Norman [21], who hopes that, one day, computers will be so embedded in everyday objects as to be rendered invisible [22], by graphic designer and visual thinker, Edward Tufte [36], and lastly, but not least, by the work of Guy Steele, Hal Abelson, and Gerry Sussman, who developed not only a powerful, expressive programming language, Scheme, but more importantly, a way of thinking about computation which allows programmers to capture how they *think* about a problem, not merely how to solve it [1].

Tangible User Interfaces

Ullmer and Ishii define tangible user interfaces as “user interfaces employing physical objects, surfaces, and spaces as representations and controls for computationally mediated associations.” [38] They refer to these physical objects as “tangibles.” The system described here, like many tangible user interfaces, is composed of a collection of *tangibles*. In the earlier terminology of Fitzmaurice, Buxton, and Ishii, the Tangible Computation Bricks system is a “graspable user interface.” [10]

Unlike systems where absolute position and orientation of objects in space control the interaction (e.g. Illuminating Light [39]) the system presented here uses sequence and juxtaposition (e.g. mediaBlocks [37]), and constructive assembly to convey meaning, configuration, and topology (e.g. Geometry-defining processors [3], Triangles [12], and MERL Blocks [4]). The Triangles and MERL Blocks are always used as input devices for specifying topology to a traditional computer (e.g. SGI workstation), whereas the Tangible Computation Bricks are designed specifically to be stand-alone and portable.

Tangible Programming

In [34] Suzuki and Kato coin the term “tangible programming language.” In their AlgoBlock system, built to study how children learn programming through collaboration, each hand-sized block is roughly equivalent to one Logo statement, for example “go forward,” and “turn right 90.” The tangible programs assembled by the children direct an on-screen submarine around an obstacle course. It is ironic that in AlgoBlock, the programming language is brought into the physical arena, while the problem domain stays on the screen.

USER TESTING

Although *no formal* user testing of the Tangible Computation Bricks has been conducted at this writing, the author has had the opportunity to conduct informal one-on-one study sessions of 20 minutes each with five children, ages 4, 6, 9, 11, and 13 who had not used the Bricks before. Additionally, valuable insights and feedback have been gained from the approximately one hundred adults who participated in or simply witnessed demonstrations of the technology at several stages of its development.

Only cursory testing was performed on the 4 and 6-year olds once it became clear that were not ready for the challenge. After a brief demonstration, they were offered the Bricks and cards to play with and assemble, and then offered a working system with the toy bicycle to play with.

The 4-year-old girl discovered every possible way of assembling non-functional creations, and found it engaging to turn the bicycle crank and watch the display count revolutions. The 6-year-old boy demonstrated an understanding of proper assembly techniques.

For the 9, 11, and 13-year-old, all of whom had some computer experience, a typical session involved an introduction to the collection of Bricks and cards in the Digital Construction Kit by the experimenter, and an incremental demonstration of how some of the bricks worked. Then the subject was asked to construct a device that counted revolutions. After a second demonstration resulting in the configuration shown in Figure 2, the subject was asked to modify the stack to display RPM instead of “revolutions per 10 seconds.” The testing session followed a progression similar to the Video Figure, but with the subject filling in some of the steps on his/her own.

The session with the 9-year-old had the subjective feeling of a tutorial because a significant amount of prompting was required to guide the subject through the requested tasks.

Both the 11-year-old girl and the 13-year-old boy were successful at solving the two problems with little prompting, and the girl launched into an impromptu activity using the Bricks and the bicycle to measure distance along a yardstick.

With such limited testing and no formal comparisons with computer-based science and programming learning tools it is difficult to draw definitive conclusions. It is possible to say however, that children as young as 6 can learn the physical construction aspects of manipulating the Bricks and cards. An 11-year-old from a highly educated family can quickly understand the Digital Construction Kit and figure out how to apply it to new problems, and a 13-year old from the same family might ask questions like, “do you have an *If* brick?” and responds in a post-interview that he likes the way “you can see everything that’s going on.”

From the author’s observations of BBB workshops using computer-based programming tools (e.g. Crickets with Logo or LogoBlocks), after 20 minutes students are just beginning to get familiar with the software tools. So that students have time to complete a project, these workshops generally run for several days.

FUTURE WORK

Getting Under the Hood

A powerful feature of the Tangible Computation Bricks system is its ability to allow researchers and even young programmers to “get under the hood,” and re-program the behavior of any Brick. If you don’t have the right Brick, you can design your own.

Exploring Parallel Computation

Not only can students invent their own tangible programming languages by writing simple programs, they can also explore principles of parallel computation in a simplified programming environment. This is because the Logo program inside each Brick necessarily runs at the same time as the programs in other bricks. In other words each Brick is its own *process*. Issues of synchronization still exist, but

because a Brick can only interact with its neighbors above and below, the conceptual burden is reduced for the student, compared with unbridled parallelism.

CONCLUSION

This paper discusses the physical, electronic, and linguistic evolution of the Tangible Computation Bricks that led up to the Digital Construction Kit built using this flexible technology. It also discusses successful design properties of the Kit. Limited user testing suggests that the tangible user interface system described here is accessible and appropriate for learning about computation and for scientific exploration by 11-year-old children and up.

ACKNOWLEDGMENTS

Many thanks to the LEGO Group for their generous in-kind support for this project. *Full acknowledgements will be included in the final version of the paper.*

REFERENCES

1. Abelson, H., Sussman, G. J., with Sussman, J., *Structure and Interpretation of Computer Programs*, MIT Press and McGraw-Hill, 1985. Second Edition, 1996.
2. Abelson H., diSessa, A., *Turtle Geometry: The Computer as a Medium for Exploring Mathematics*, MIT Press, 1981.
3. Anagnostou, G., Dewey, D., and Patera, A., Geometry-defining processors for engineering design and analysis, in *The Visual Computer*, 5, pp. 304-315, 1989.
4. Anderson, D., Frankel, J., Marks, J., et al., Building Virtual Structures with Physical Blocks (demo description), in *Proceedings of UIST'99*, CHI Letters, vol. 1, no. 1, 1999.
5. Begel, A., *LogoBlocks: A Graphical Programming Language for Interacting with the World*, S.B. Thesis, MIT Department of Electrical Engineering and Computer Science,
<http://abegel.www.media.mit.edu/people/abegel/begelaup.pdf>
1996.
6. Borovoy, R., *Dance Craze Buggies*,
<http://el.www.media.mit.edu/people/borovoy/cars/>
1998.
7. Braitenberg, V., *Vehicles, experiments in synthetic psychology*, MIT Press, Cambridge, Massachusetts, 1984.
8. Brooks, R. A., *A Robust Layered Control System For a Mobile Robot*, AIM-864, MIT AI Lab, 1985.
9. Drescher, G. L., Object-Oriented Logo, *Artificial Intelligence and Education*, Ablex Publishing, Norwood, NJ, pp. 153-165, 1987.
10. Fitzmaurice, G., Ishii, H., and Buxton, W., Bricks: Laying the Foundation for Graspable User Interfaces, in *Proceedings of CHI'95*, pp. 442-449, ACM Press, 1998.
11. Gindling, J., Ioannidou, A., Loh, J., Lokkebo, O., and Repenning, A., LEGOsheets: A Rule-Based Programming, Simulation and Manipulation Environment for the LEGO Programmable Brick, *Proceedings of IEEE*

- Symposium on Visual Languages*, Darmstadt, Germany, pp. 172-179, IEEE Computer Society Press, September 1995.
12. Gorbet, M., Orth, M. Ishii, H., Triangles: Tangible Interface for Manipulation and Exploration of Digital Information Topography, *Proceedings of CHI '98*, ACM Press, 1998.
 13. Hansen, W. J., "The 1994 Visual Languages Comparison," in *1994 IEEE Symposium on Visual Languages*, pp. 90-97, IEEE Computer Society Press, Los Alamitos, CA, 1994.
 14. Henderson, P., *Functional Programming: Application and Implementation*, Prentice-Hall, Englewood Cliffs, NJ, 1980.
 15. Ishii, H., and Ullmer, B., Tangible Bits: Toward Seamless Interfaces between People, Bits and Atoms, *Proceedings of Conference on Human Factors in Computing Systems (CHI '97)*, pp. 234-241, ACM, Atlanta, March 1997.
 16. Jagadeesh, J., and Wang, Y., LabVIEW (product review), in *Computer*, February 1993.
 17. Kahn, K., Drawings on Napkins, Video-game Animation, and Other Ways to Program Computers, *Communications of the ACM*, vol. 39, no. 8, pp. 49-59, August 1996.
 18. Martin, F., Resnick, M., LEGO/Logo and Electronic Bricks: Creating a Scienceland for Children, *Advanced Educational Technologies for Mathematics and Science*, (David Ferguson, ed.), Springer-Verlag, Berlin, Heidelberg, 1993.
 19. Martin, F. Colobong, G. L., Resnick, M., *Tangible Programming with Trains*,
<http://el.www.media.mit.edu/projects/trains/> 1998.
 20. Martin, F., Mikhak, B., Silverman, B., MetaCricket: A Designers' Kit for Making Computational Devices, forthcoming, *IBM Systems Journal*.
 21. Norman, D. A., *The Psychology of Everyday Things*. Basic Books, New York, 1988.
 22. Norman, D. A., *The Invisible Computer*, MIT Press, Cambridge, Massachusetts, 1998.
 23. Papert, S., *Mindstorms: children, computers, and powerful ideas*, Basic Books, 1980.
 24. Perlman, R., Using computer technology to provide a creative learning environment for preschool children, Logo Memo no. 24, AI Memo 260, MIT AI Lab, May 1976.
 25. Post, R., *E-Broidery: An Infrastructure for Washable Computing*, S.M. Thesis, MIT Media Lab, 1999.
 26. Repenning, A., Sumner, T., Agentsheets: A Medium for Creating Domain-Oriented Visual Languages, *IEEE Computer*, v. 28 no. 3, pp. 17-25, 1995.
 27. Repenning, A., and Ambach, J., Tactile Programming: A Unified Manipulation Paradigm Supporting Program Comprehension, Composition, and Sharing, in *Proceedings of Visual Languages 1996*, IEEE Computer Society, 1996.
 28. Resnick, M., *Turtles, Termites and Traffic Jams: Explorations in Massively Parallel Microworlds*, MIT Press, 1994.
 29. Resnick, M., Behavior Construction Kits, *Communications of the ACM*, v. 36, No. 7, pp. 64-71, July 1993.
 30. Resnick, M., Martin, F., Berg, R., Borovoy, R., Colella, V., Kramer, K., Silverman, B., Digital Manipulatives: New Toys to Think With, *Proceedings of CHI'98*, ACM Press, 1998.
 31. Resnick, M., Silverman, B. S., Begel, A., Martin, F., Welch, K., Logo Blocks,
<http://fredm.www.media.mit.edu/people/fredm/projects/cricket/logo-blocks/> 1998.
 32. Smith, D. C., KidSim: Programming Agents Without a Programming Language, *Communications of the ACM*, v. 37, No. 7, pp. 55-67, July 1994.
 33. Soloway, E., and Spohrer, J., *Studying the Novice Programmer*, Lawrence Erlbaum, Hillsdale, NJ, 1989.
 34. Suzuki, H., Kato, H., AlgoBlock: a Tangible Programming Language, a Tool for Collaborative Learning, *Proceedings of 4th European Logo Conference*, pp. 297-303, Athens, 1993.
 35. Suzuki, H., Kato, H., Interaction-Level Support for Collaborative Learning: AlgoBlock—An Open Programming Language, *Proceedings of CSCL'95*, 1995.
 36. Tufte, E. R., *Envisioning Information*, Graphic Press, Cheshire, Connecticut, 1990.
 37. Ullmer, B., Ishii, H., and Glas, D., mediaBlocks: Physical Containers, Transports, and Controls for Online Media, *Proceedings of SIGGRAPH'98*, ACM Press, 1998.
 38. Ullmer, B. and Ishii, H., Emerging Frameworks for Tangible User Interfaces, forthcoming, *IBM Systems Journal*.
 39. Underkoffler, J., Ishii, H., Illuminating Light: An Optical Design Tool with a Luminous-Tangible Interface, *Proceedings of CHI'98*, pp. 452-549, ACM Press, 1998.
 40. Ungar, D., Lieberman, H., Fry, C., Debugging and the Experience of Immediacy, *CACM*, vol. 20, no. 4, pp. 38-43, 1997.
 41. *The author's master's thesis*