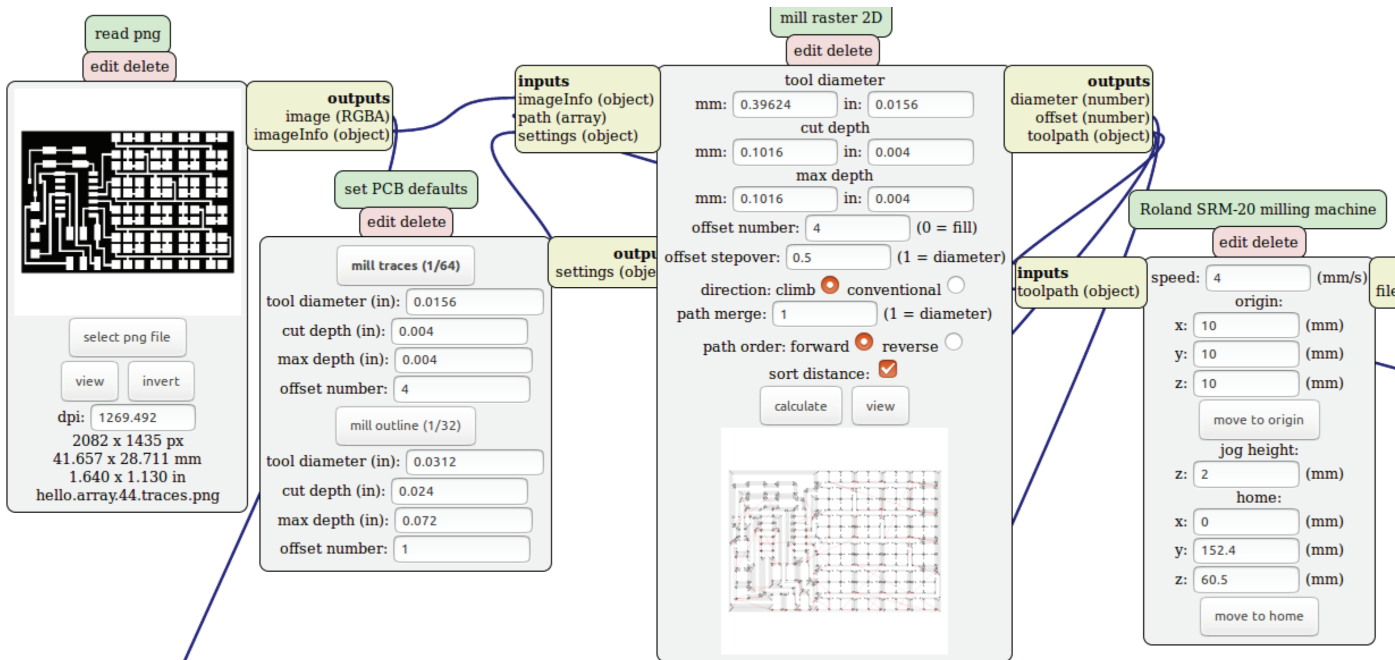
Mods: Browser-Based Rapid Prototyping Workflow Composition

Nadya Peek
University of Washington
Machine Agency

Neil Gershenfeld
MIT Center for Bits and Atoms



1 Modules from left to right:
1. Input design file (here in the form of a .PNG image file representing traces and pads).
2. Select preset parameters for the milling job. This can be done with a presets module such as the one here called set PCB defaults that specifies a 1/64″ end mill and 4 milling contours at 0.004″ depth.
3. Generate the toolpath using the parameters specified.
4. Set up the material and set the origin using the machine.
5. (cropped out) Run the toolpath on the machine by spooling through a WebSocket.

ABSTRACT

Software is shared through files and libraries, but workflows are not. To be able to share workflows for rapid automation, we developed an extensible environment for running CAD, CAM, and machine control. We present Mods, a browser-based environment for data handling, toolpath planning, and machine execution. Users compose modules (either existing modules or new modules they contribute) into workflows for machine automation sequences in a dataflow environment. The modules themselves run client side, implementing the functions used by the modules (such as toolpath planning algorithms or image analysis) in JavaScript, which runs in the browser. The physical machines are connected to a JavaScript server, which listens to commands from the client over a WebSocket connection. Together, these software modules make up an extensible and simple-to-use alternative to traditional CAD/CAM machine control environments.

INTRODUCTION

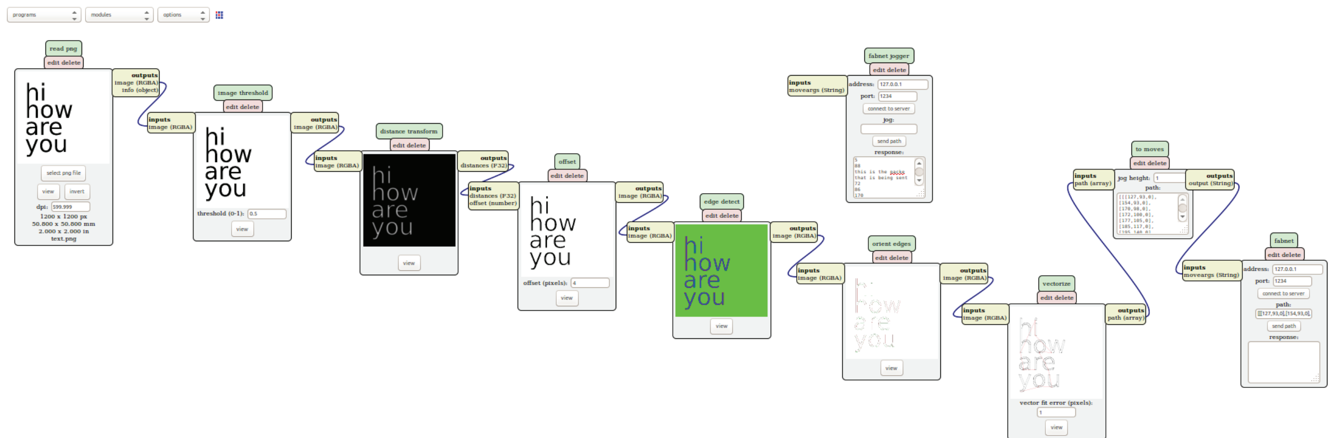
The workflows for running digital fabrication machines can be tedious; the user needs to interface among CAD softwares and their representations, the machine's CAM software and its possible file formats, and load these onto the machine that is going to run the toolpath, all while making sure the material stock accommodates the digital design. Yet digital fabrication is a highly active current field of research (Malone and Lipson 2007; Coleman and Cole 2017; Lewis 2006; Zoran and Paradiso 2013; Rivers, Moyer, and Durand 2012; Gramazio and Kohler 2014; Mellis et al. 2013). Machine users constantly develop new workflows depending on what machine they are using, what file format their design is in, what file formats the machines expect, and how material-specific operations such as zeroing and part placement take place.
Software is one realm where rapid prototyping is already widespread in practice. However, usability lags when it comes to the software driving automation equipment (including digital fabrication machines). We are saddled with buggy print drivers, file name length limitations, or confusing limitations (Soler et al. 2017; Louis-Rosenberg 2016; Coleman et al. 2016). Adding functionality such as sensor feedback complicates matters yet further. Once workflows are finally perfected it is hard to share them with other machine users, except through out-of-band documentation: you can share libraries for designs, or settings for slicers, or machine instructions for a zeroing process, but you cannot share something that includes all those parts in one place.

To be able to share and reuse the workflows that users of CAD/CAM develop, we developed an open-source, extensible, event-driven environment for creating machine actions called Mods. Mods is a framework in which users can author workflows using a dataflow programming language.

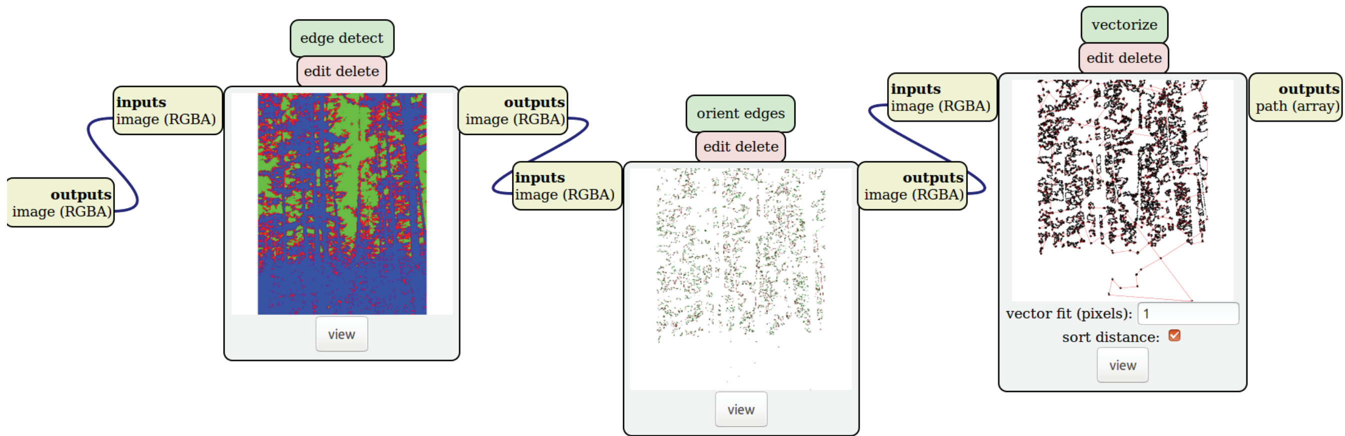
BACKGROUND

Dataflow programming research was originally motivated in the 1970s by the need for massive parallelism, which was considered at odds with von Neumann Architecture's global program counter and global updatable memory (Johnston, Hanna, and Millar 2004). Researchers proposed dataflow programming as executing commands once all operands were available and using only local memory (Davis and Keller 1982; Johnston et al. 2004). These methods were exceptionally suited for visual programming, as programs could be represented by nodes and graphs. Visual programming dataflow languages, such as National Instruments Labview, were subsequently popularized in the 1990s and continue to be used to this day (Bitter, Mohiuddin, and Nawrocki 2006). In computer-aided design, dataflow programming mimics a designer's modelling workflow. The ability to generalize these workflows into parametric designs in CAD makes visual dataflow programming languages such as Rhino3D's Grasshopper very popular (Bachman 2017). We draw inspiration from these dataflow languages and design practices in the development of Mods.

Event-driven software architectures with asynchronous I/O are now commonly considered best practice for (online) applications with both high throughput and scaling requirements (Tilkov and Vinoski 2010). Node.js is a JavaScript runtime environment using the Chrome V8 engine, which can be used for both server-side and client-side scripting and facilitates real-time communication (Dahl 2017). The



2 A Mods program for toolpath generation. This program uses the modules "read png," "threshold image," "distance transform," "offset," "edge detect," "orient edges," "vectorize," "to plotter moves," and finally connects to the physical machine through the "fabnet" module. More generally, this program reads in an image, thresholds it, produces a machine toolpath of that threshold, and sends it to the machine.



3 A closer look at the "edge detect," "orient edges," "vectorize" modules. Each image can also be opened in a separate window to inspect the toolpath more closely.

V8 engine does just-in-time (JIT) compilation of JavaScript to native machine code, offering massive speedup in comparison to interpreted JavaScript (Google 2017). Mods code would be impossibly slow without JIT compilation, and interfacing between the client and server sides would be very difficult without Node.js. Furthermore, by using JavaScript's typed arrays we avoid converting at every access and can take advantage of highly optimized web file readers available in libraries such as OpenGL (Group 2017).

These recent (since 2010) browser and JavaScript technologies make the development of Mods as a browser-based system possible. We also drew inspiration from other platforms moving into the browser, such as the creative programming language Processing, which was adapted into P5.js (McCarthy, Reas, and Fry 2015), or the Scratch programming language for children (Resnick et al. 2009). The remixing that takes place in these online communities can lead to improved computational thinking and learning (Dasgupta et al. 2016), and we modelled our remixing possibilities using insights from these studies.

We are interested in the full workflow from idea through manufacturing. We are specifically interested in low-volume production and real-time feedback in the fabrication process, similar to Willis et al. describe in "Interactive Fabrication" (2011). While dataflow programming exists for CAD, the hooks to CAM are limited. G-Code generators such as 3D-printer slicers or machine-specific postprocessors are well developed for existing machines, but don't easily generalize to nonstandard machines. For executing machine commands, platforms like GRBL and TinyG do a stellar job for G-Code interpretation, but G-Code has severe limitations (for example, G-Code does not have conditional statements). Finally, there are middleware suites such as Willow Garage's ROS or IBM's Node Red, which create a structure for passing messages between services, but do not implement the functionality we want in digital fabrication workflows. These limitations are why we decided to explore what an accessible system with CAD/CAM machine control in one place could look like.

USING MODS

Before going into the implementation details of Mods, we describe how a user might perform an example fabrication task using Mods. For example, if a user would like to subtractively machine a copper-clad board using a milling machine such as a Roland SRM-20 to produce a circuit, this would be done in the following steps and as illustrated in Figure 1.

- Open the program "SRM-20 PCB" at http://mods.cba.mit.edu. Part of this program is shown in Figure 1.
- Input design file (image of the board)
- Specify the machining parameters (depth of cut, number of offsets, percentage of offset, climb or conventional, etc.)
- Calculate toolpath (using other modules we will later describe in Figure 3)
- Zero machine (by moving the toolhead close using computer commands and then manually zeroing the bit)
- Start the milling job.

For this workflow we assume that the SRM-20 milling machine is connected to the user's computer and that the Mods server is already installed and running there. The Mods server can be stopped and restarted at any time by the user. This is only one example of a Mods workflow—many others are possible.

```
var server_port = '1234'
var client_address = '127.0.0.1'
var server = {}
var WebSocketServer = require('ws').Server
wss = new WebSocketServer({port:server_port})
wss.on('connection',function(ws) {
	if (ws._socket.remoteAddress != client_address) {
		console.log("connection	rejected	from	"+ws._socket.
	remoteAddress)
		wss.close()
		return
		}
	else {
		console.log("connection	accepted	from	"+ws._socket.
	remoteAddress)
		}
	ws.on('message',function(msg) {
		eval(msg) }
		}
	}
```

4 We have developed machine-specific Node.js servers controlling different machines. A generic Node.js server might evaluate any messages passed through a WebSocket from a Mods module. Mods modules for different machines may be more specific with what messages need to be evaluated (e.g., lpr -P milling-machine [data]), and handle details like hardware flow control or ports permissions when connecting a machine. For example, to interface with the Roland MDX-20 milling machine, we need to set up a serial port with a baud rate of 9600, RTSCTS flow control, and the correct port name. These attributes are specified in a "serial" module available in the repository.

```
function worker() {
	self.addEventListener('message',function(evt) {
		var h = evt.data.height
		var w = evt.data.width
		var t = evt.data.threshold
		var buf = new Uint8ClampedArray(evt.data.buffer)
		var r,g,b,a,i
		for (var row = 0; row < h; ++row) {
			for (var col = 0; col < w; ++col) {
				r = buf[(h-1-row)*w*4+col*4+0]
				g = buf[(h-1-row)*w*4+col*4+1]
				b = buf[(h-1-row)*w*4+col*4+2]
				a = buf[(h-1-row)*w*4+col*4+3]
				i = (r+g+b)/(3*255)
				if (a == 0)
					val = 255
				else if (i > t)
					var val = 255
				else
					var val = 0
				buf[(h-1-row)*w*4+col*4+0] = val
				buf[(h-1-row)*w*4+col*4+1] = val
				buf[(h-1-row)*w*4+col*4+2] = val
				buf[(h-1-row)*w*4+col*4+3] = 255
				}
			}
			self.postMessage({buffer:buf.buffer},[buf.buffer])
			})
		}
```

5 The image threshold module will spawn this web worker to execute thresholding (making a color or greyscale image black and white).

MODS SOFTWARE ARCHITECTURE

Mods consists of modules that can be connected together in a dataflow graph. Users can use existing modules (provided in a repository), write their own modules (using a module template), or modify existing modules (directly in the browser). The modules can be connected together into programs. To run, the main source mods.js sets up a container within the browser where programs of modules can run. Each module includes initialization, event handling, presentation, and application. This means that the functionality of the module and how it will be rendered are both included in each single module. A module can be edited, reloaded, and saved from within Mods. Programs (collections of modules and their connections) can be saved as well. This all happens client side—the module is all its parts and does not use any online resources. Mods.js provides the container, while modules each have a closure. When the modules are loaded by mods.js, they spawn HTML5 web workers to complete their tasks. This means that although Mods runs in a browser, it does not need to be connected to a server for running the computational modules—they are standalone.

We have developed machine-specific Node.js servers controlling different machines, as shown in Figure 4. The example program in Figure 3 follows a typical design file (here a .png) to toolpath workflow (here in coordinates). This is the same workflow used in Figure 1. When a module is added or updated, it triggers events on the modules that are connected downstream. If we look at the code that makes up the modules, we find the module's inputs, outputs, interface code, and functions. For example, the threshold module is shown in Figure 5. Each module will start working upon receiving the operands, or more specifically once an event has been triggered because the inputs have been modified. If the inputs are modified again, previous workers are terminated.

USER-MADE MODS

Mods has been tested by novices using small-format digital fabrication machines such as laser cutters, milling machines, and 3D printers (Peek et al. 2017). Users can access Mods by going to http://mods.cba.mit.edu or running Mods locally. We host a repository of open-source modules including video, image, .stl, and sensor input modules, modules that directly import from CAD tools like Solidworks, machine code output modules, and server code for connecting to digital fabrication machines. However, this of course does not cover all the applications users might be interested in. Therefore, users can augment Mods with custom modules. Users have, for example, made their own modules for liquid handling machines that are used in biological experiments. Those modules expose functionality such as pipetting, mixing, or creating buffers.

DISCUSSION

The main reason we developed Mods was to avoid debugging drivers to run digital fabrication machines. We fear that trying lots of different ancient versions of the .dxf file format to be able to get a toolpath right for a waterjet or other digital fabrication machine is very familiar to thousands of architects. While we acknowledge that our system is hardly ready to replace all CAM and machine control software, we hope to demonstrate that it is possible to capture the workflows that are developed each time a user figures out how to run a particular machine for a particular geometry, material, or specification. Especially within architecture, where low-volume production is the norm and highly precise fabrication of thousands of unique parts is required, we hope to increase workflow efficiency.

We chose to implement Mods to run in the browser to be able to keep connections to machines working with an increasingly post–operating system world. We specifically did not implement Mods as a Grasshopper plugin so that we could more easily interface with the machines through serial/USB/ethercat/etc. connections maintained from the Mods server. While Grasshopper is a powerful tool, it is still difficult to connect it directly to machine tools.

Performance is of concern when moving to interpreted languages. Toolpath planning has historically been a computationally intensive task. To measure the performance of Mods, we created several benchmarking modules. To benchmark processing power we calculate π to a specified decimal point, and to benchmark connectivity we time a roundtrip to the server. On a typical-performance laptop on which we also wrote this paper, we measured 1033 Mflops and 9.8 ms round trip with the server. This is on par with performance of compiled C on the same machine, so we conclude that our implementation does not suffer from interpreted language slowdown.

CONCLUSION

In summary, Mods is a browser-based event-driven environment for data handling, toolpath planning, and machine execution. It makes creating workflows for generating automation sequences for machines easier by allowing users to compose modules into programs in a dataflow environment. This allows users take advantage of the precision of digital fabrication machines without their historical constraints. The modules themselves run client side, implementing the functions used by the modules (such as toolpath-planning algorithms or image analysis) in JavaScript, which runs in the browser. The physical machines are connected to a JavaScript server, which listens to commands from the client over a WebSocket connection. Together, these software modules make up a simple-to-use and simple-to-extend alternative to traditional CAD/CAM machine control environments. We believe this is a step towards harnessing the precision of machines for the creativity of individuals.

ACKNOWLEDGEMENTS

We'd like to thank Naveed Ihsanullah, Jason Weathersby, Eric Rescorla, Martin Best, and Andreas Gal (for Firefox) and Kenneth Russell and Jochen Eisinger (for Chrome/Chromium) for finding and fixing issues with their engines that emerged during the development of Mods.

REFERENCES

Bachman, David. 2017. *Grasshopper: Visual Scripting for Rhinoceros 3D*. New York: Industrial Press.

Bitter, Rick, Taqi Mohiuddin, and Matt Nawrocki. 2006. *LabView: Advanced Programming Techniques*, 2nd ed. Hoboken: CRC Press.

Coleman, James, and Shannon Cole. 2017. "By Any Means

Necessary: Digitally Fabricating Architecture at Scale." In *Disciplines & Disruption: Proceedings of the 37th Annual Conference of the Association for Computer Aided Design in Architecture*, edited by T. Nagakura, S. Tibbits, M. Ibanez, and C. Mueller, 190–201. Cambridge, MA: ACADIA.

Coleman, James, Craig Long, Andrew Manto, and Trygve Wastvedt. 2016. "Lots of parts, lots of formats, lots of headache." *XRDS* 22 (3) 54–57.

Dahl, R. 2017. "Node.js." https://nodejs.org/en/.

Dasgupta, Sayamindu, William Hale, Andrés Monroy-Hernandez, and Benjamin Mako Hill. 2016. "Remixing as a Pathway to Computational Thinking." In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*, 1438–49. San Francisco, CA: CSCW.

Davis, A. L., and R. M. Keller. 1982. "Data Flow Program Graphs." *Computer* 15 (2): 26–41. DOI: 10.1109/MC.1982.1653939.

Google. 2017. "Chrome V8." https://developers.google.com/v8/.

Gramazio, Fabio, and Matthias Kohler, eds. 2014. "Made by Robots: Challenging Architecture at a Larger Scale". Special issue, *Architectural Design* 84 (3).

Group, K. 2017. "OpenGL." https://www.opengl.org/.

Johnston, Wesley M., J. R. Paul Hanna, and Richard J. Millar. 2004. "Advances in Dataflow Programming Languages." *ACM Computing Surveys* 36 (1): 1–34.

Lewis, J. A. 2006. "Direct Ink Writing of 3D Functional Materials." Advanced Functional Materials 16 (17): 2193–204.

Louis-Rosenberg, Jesse. 2016. "Drowning in Triangle Soup: The Quest for a Better 3-D Printing File Format." *XRDS* 22 (3): 58–62.

Malone, Evan, and Hod Lipson. 2007. "Fab@home: The Personal Desktop Fabricator Kit." *Rapid Prototyping Journal* 13 (4): 245–55

McCarthy, Lauren, Casey Reas, and Ben Fry. 2015. *Getting Started with p5.js: Making Interactive Graphics in JavaScript and Processing*. San Francisco, CA: Maker Media.

Mellis, David, Sean Follmer, Björn Hartmann, Leah Buechley, and Mark D. Gross. 2013. "FAB at CHI: Digital Fabrication Tools, Design, and Community." In *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, 3307–10. Paris: CHI EA.

Page, Mitchell. 2017. "A Robotic Fabrication Methodology for Dovetail and Finger Jointing: An Accessible & Bespoke Digital Fabrication Process for Robotically-Milled Dovetail & Finger Joints." In *Disciplines & Disruption: Proceedings of the 37th Annual Conference of the Association for Computer Aided Design in Architecture*, edited by T. Nagakura, S. Tibbits, M. Ibanez, and C. Mueller, 456–63. Cambridge, MA: ACADIA.

Peek, Nadya, James Coleman, Ilan Moyer, and Neil Gershenfeld. 2017. "Cardboard Machine Kit: Modules for the Rapid Prototyping of Rapid Prototyping Machines." In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 3657–68. Denver, CO: CHI.

Resnick, Mitchel, John Maloney, Andrés Monroy-Hernandez, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, and Brian Silverman. 2009. "Scratch: Programming for All." *Communications of the ACM* 52 (11): 60–67.

Rivers, Alec, Ilan E. Moyer, and Frédo Durand. 2012. "Position-correcting Tools for 2D Digital Fabrication." *ACM Transactions on Graphics* 31 (4): 88:1–88:7.

Soler, Vicente, Gilles Retsin, and Manuel Jimenez Garcia. 2017. "A Generalized Approach to Non-Layered Fused Filament Fabrication." In *Disciplines & Disruption: Proceedings of the 37th Annual Conference of the Association for Computer Aided Design in Architecture*, edited by T. Nagakura, S. Tibbits, M. Ibanez, and C. Mueller, 562–71. Cambridge, MA: ACADIA.

Tilkov, Stefan, and Steve Vinoski. 2010. "Node. js: Using JavaScript to Build High Performance Network Programs." *IEEE Internet Computing* 14 (6): 80–83

11

Willis, Karl D. D., Cheng Xu, Kuan-Ju Wu, Golan Levin, and Mark D. Gross. 2010. "Interactive Fabrication: New Interfaces for Digital Fabrication." In *Proceedings of the Fifth International Conference on Tangible, Embedded, and Embodied Interaction*, 69–72. Funchal, Portugal: TEI.

Zoran, Amit, and Joseph A. Paradiso. 2013. "FreeD: A Freehand Digital Sculpting Tool." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2613–16. SIGCHI.

Nadya Peek develops unconventional digital fabrication tools, small scale automation, networked controls, and advanced manufacturing systems. Spanning electronics, firmware, software, and mechanics, her research focuses on harnessing the precision of machines for the creativity of individuals. Nadya is an assistant professor at the University of Washington in the Human-Centered Design and Engineering department where she directs the Machine Agency.

Prof. Neil Gershenfeld is the Director of MIT's Center for Bits and Atoms. His unique laboratory is breaking down boundaries between the digital and physical worlds, from creating molecular quantum computers to virtuosic musical instruments.