



Furste, Zachary, 2025. "Stuff You Can Click: Sensing Infrastructure with Software Emulation." *communication +1*, vol XI, iss I, pp. 1-32.
DOI: <https://doi.org/10.7275/cpo.1939>



Stuff You Can Click: Sensing Infrastructure with Software Emulation

Zachary Aaron Furste, University of Amsterdam, Netherlands, z.a.furst@uva.nl

How can we sense infrastructure? This article begins by considering the role of the body in some recent influential approaches to media infrastructure. The critical work of Lisa Parks, as well as the cartographic project of Kate Crawford and Vladan Joler, prominently feature the eye, whereas the media archaeologies of Wolfgang Ernst and Erkki Huhtamo proceed from the hand's interaction with individual media devices. By contrast, durational "recipes" from Colorado's Media Archaeology Lab and "walking tours" led by Amsterdam's Critical Infrastructure Lab emphasize embodiment in time and (urban) space.

In the context of this taxonomy, the second part of the article describes in-browser software emulation. Detailing the history of emulation and the technical processes that brought it into the browser, I argue that sites like infinitemac.org stage a tactile reckoning with not only software history, but also the underlying techniques of network infrastructure.

Dall'osservatorio: Eyes on infrastructure

In a 2015 essay, Lisa Parks outlined a “stuff you can kick” theory of media infrastructures. The piece convincingly argues that attention is due not only to narratives, symbols, or aesthetics contained in a given piece of media, but also to the “material resources that are arranged and used to distribute audiovisual content.”¹ Along with scholars like Paul Edwards, Geoffrey Bowker, and Nicole Starosielski, Parks has provoked media theory to reconsider an aesthetic bias that privileges the representational in favor of the infrastructural, operational, and microtemporal. As Parks and Starosielski put it, “our current mediascapes would not exist without our current media infrastructures.”²

I am on board for all of this. I would like to note, however, that this “stuff you can kick” formulation invokes, but defers, embodied action and sensation. The essay does not quite, in other words, call on media academics to lace up steel-toed boots. Rather, it elegantly reflects on some filmic and photographic documentation of other laborers—postal workers in Washington, D.C.; electrical linemen in Southern California; police officers and special forces soldiers in Iran—carrying, stretching, and stomping infrastructure.

To be sure, given the constellation of historical and geographical infrastructures in question, scholars of such mediascapes sit at a necessary remove. Kate Crawford’s widely cited *Atlas of AI*, for example, presents a “topographical approach [to offer] different perspectives and scales.”³ Drawing on methods from science and technology studies and art history, Crawford describes the book’s approach as “walking through the many landscapes of computation and seeing how they connect.”⁴ She stresses the counter-hegemonic valence of this mapping and the subjective aspect of visualization: “we gain a better understanding of AI’s role in the world by engaging with its material architectures, contextual environments, and prevailing politics and by tracing how they are connected.”⁵ *Atlas of AI* marks an important intervention into often dematerialized discourses surrounding artificial intelligence.

¹ Lisa Parks, “‘Stuff You Can Kick’: Toward a Theory of Media Infrastructures,” in *Between Humanities and the Digital*, eds. Patrik Svensson and David Theo Golberg (Cambridge, MA: MIT Press, 2015), 356.

² Lisa Parks and Nicole Starosielski, “Introduction” in *Signal Traffic: Critical Studies of Media Infrastructures* (Urbana, IL: University of Illinois Press, 2014), 1.

³ Kate Crawford, *Atlas of AI* (New Haven: Yale University Press, 2021), 11.

⁴ *Atlas of AI*, 11.

⁵ *Atlas of AI*, 12.

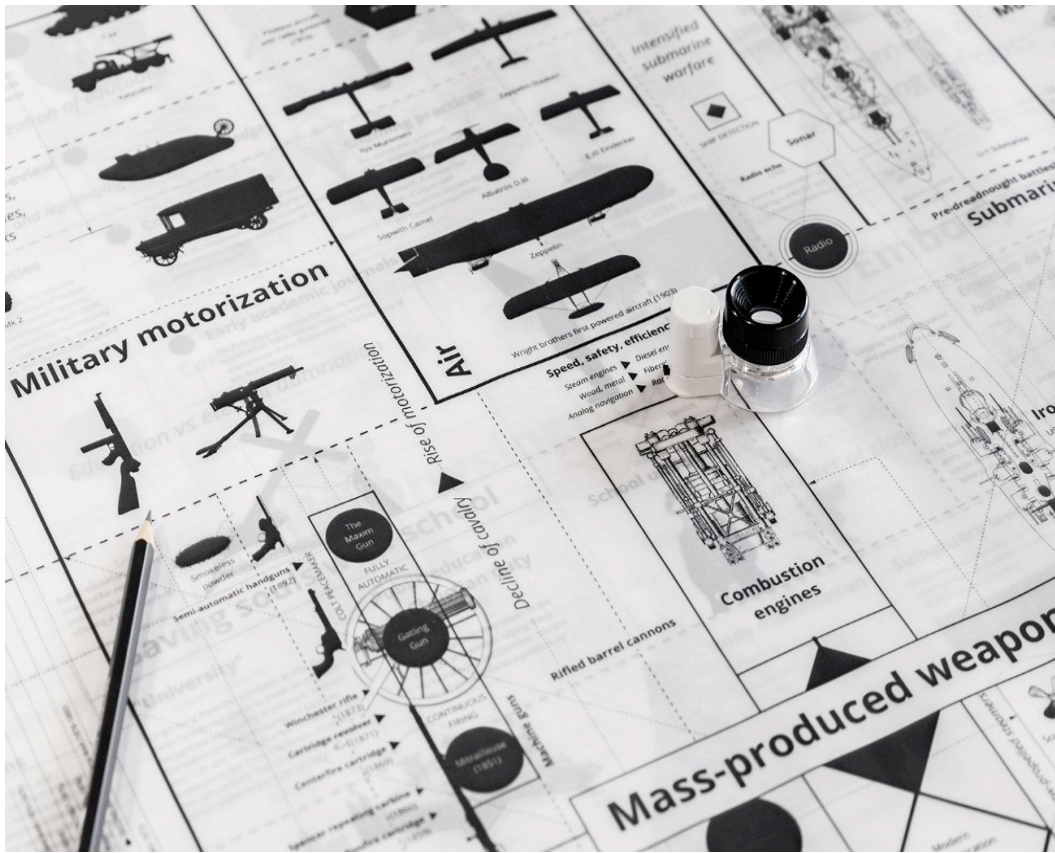


Figure 1 – Documentation of Calculating Empires at the Osservatorio of Fondazione Prada in Milan. Photos by Piercarlo Quecchia. Courtesy Fondazione Prada.

I would again point out the implicit model it employs: documentation, diagramming, presentation to the senses for slow contemplation.

Crawford has recently extended this tracing—the visual apprehension of systems of immense complexity—in *Calculating Empires*, an ongoing installation project in collaboration with Vladan Joler (figure 2). In this work, the “walking through” becomes a bit more literal, even if the landscapes—installed at the Osservatorio of the Fondazione Prada in Milan (November 23, 2023–January 29, 2024) and as part of *Poetics of Encryption* KW Institute of Contemporary Art in Berlin (February 17, 2024–May 26, 2024)—remain fundamentally diagrammatic. Arranged across several centuries, hundreds of topoi, and thousands of custom illustrations, the KW press release describes the project as a “24 meter-long map charting how power



Figure 2 – Documentation of *Calculating Empires* at the Osservatorio of Fondazione Prada in Milan. Photos by Piercarlo Quecchia. Courtesy Fondazione Prada.

and technology have been intertwined since 1500.”⁶ Crawford has stated that the overwhelming aspect of the experience is the point, and that the pair hopes to elicit slow reading and the “radical act of contemplation.”⁷

Hands-on objects

Yet, as the artist (and Crawford collaborator) Trevor Paglen contends, “human visual culture has become a special case of vision, an exception to the rule.”⁸ The “radical act of contemplation” in *Calculating Empires* functions as an important counterpoint to both mainstream modalities of instant algorithmic gratification and the “hyperscale” processing that circulates this culture. It also necessarily elides the processual, freezing

⁶ Press release at <https://www.kw-berlin.de/en/poetics-of-encryption-conversation-kate-crawford>.

⁷ Crawford’s comments in presentation and discussion with Nadim Samman at KW Institute for Contemporary Art, Berlin, February 18, 2024, as part of the *Poetics of Encryption* program.

⁸ “Invisible Images,” *The New Inquiry* (December 8, 2016). <https://thenewinquiry.com/invisible-images-your-pictures-are-looking-at-you/>.

these flows not only into words, illustrations, and diagrams, but also into select material objects.

In the installation in Milan, the black box diagrams of *Calculating Empires* are accompanied by display cases filled with artifacts. One case contains silicon wafers, punch cards, an NVIDIA GPU, a spy camera, and a polygraph machine (figure 3). Another lays out 28 samples of the minerals used in the production of computational technology (figure 4). In this way, the exhibition activates the viewer's perception beyond the two-dimension vector-based illustrations and charts, connecting to theories of the materiality of "things."⁹ Inhering in the lithium and graphite samples, of course, are not only the labor processes of excavation, circulation, and transformation into computational machinery, but also geological histories.¹⁰ A viewer thus walks through the "black box" to arrive at familiar museological displays which present the "thingness" behind the nodes in Crawford and Joler's diagrams.



Figure 3 – Display of physical objects in *Calculating Empires*, Osservatorio of Fondazione Prada in Milan. Photos by Piercarlo Quecchia. Courtesy Fondazione Prada.

⁹ See, e.g., Bill Brown, "Thing Theory," *Critical Inquiry* 28, no. 1 (2001): 1-22.

¹⁰ See Jakko Kemper, "Deep Time and Microtime," *Theory, Culture, & Society* (2024); Jussi Parikka, *A Geology of Media* (Minneapolis: University of Minnesota Press, 2015).

While this approach is diagrammatic in its placement of objects into the visual rhetoric of a museum display, it is careful to avoid uncritical pretense to “objective” documentation; Crawford has cited the speculative practice of Aby Warburg’s *Bilderatlas Mnemosyne* as an inspiration.¹¹ Still, the only physically manipulable component of the installation, tellingly, are the blueprints of the diagrams awaiting the visitor an architect table (figure 1). *Calculating Empires* asks a viewer to trace, chart, view, apprehend, and understand.

I do not wish to discount the contribution of these interventions by Crawford and Joler (nor of Parks and other advocates of an infrastructural approach to media studies). Instead, I note a productive tension between the visual/representational and the material/infrastructural. Clearly, artists and writers should engage infrastructure with the tools they have. But, if as Wolfgang Ernst has argued, “the essence of technical media is revealed only in their temporal operations,” is something left on the table, so to speak, by contemplative, diagrammatic, visually oriented approaches?¹²



Figure 4 – Display case with elements and minerals in *Calculating Empires*, Osservatorio of Fondazione Prada in Milan. Photos by Piercarlo Quecchia. Courtesy Fondazione Prada.

¹¹ Kate Crawford in conversation with Nadim Samman at KW, Berlin, Germany, February 24, 2024, part of *Poetics of Encryption*.

¹² *Chronopoetics* (New York: Rowman & Littlefield, 2016), vii.

The inclusion of material objects recalls a strain of media archaeology that extends a kind of *Wunderkammer* approach to collecting curious objects. These collections, contra conventional museum exhibitions, invite hands-on engagement with their wonders. In addition to serving as a staple of museum education programming, this tactile disposition has spurred scholars in the field of media archaeology. Ernst argues that chronopoetics not only should shape how we interpret technical media, but time itself. His paradigmatic case is a radio receiver which, when it operates, does so across the very same circuits and so in a manner processually identical to when it was first used.

Ernst's chronopoetics illuminates the microtemporal and operative nature of contemporary media. Still, it shares with other media archaeological approaches something of a bias toward individual media objects. So too with the wonderful *Media Archaeological Fundus* curated by Ernst at Humboldt Universität in Berlin, which inventories discrete objects.¹³ YouTube videos documenting the use of each object, Ernst's hands emerging from offscreen to operate a machine in their way isolate each apparatus as an individual unit of study, even if broader systems (such as terrestrial AM transmission or PAL video, as in figure 5) are implied.

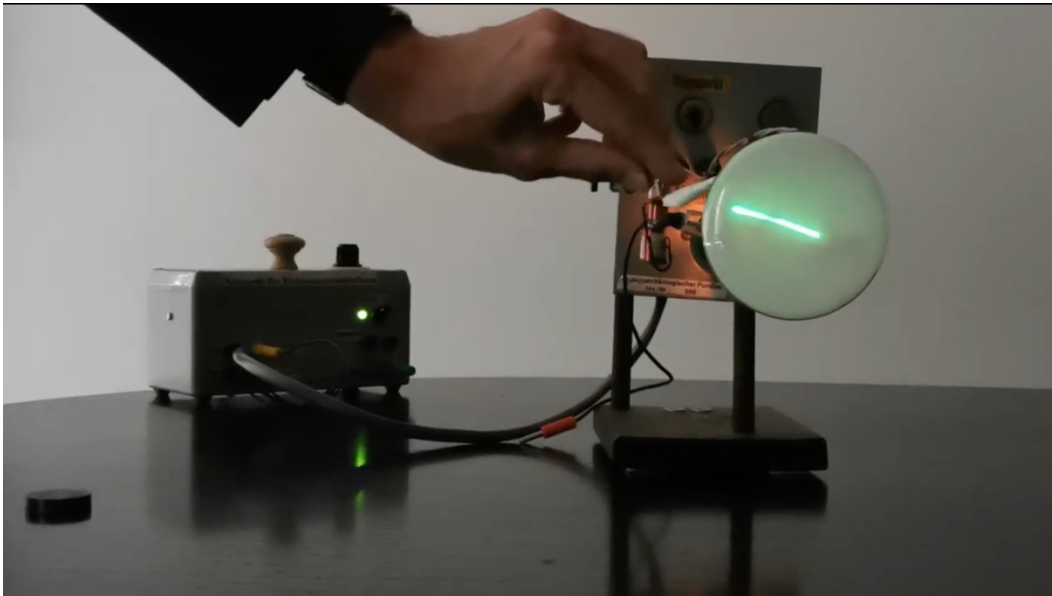


Figure 5 – Screenshot from YouTube video in which a hand demonstrates the magnetic deflection of a cathode ray tube from the Media Archaeological Fundus at Humboldt University in Berlin.

¹³ <https://www.musikundmedien.hu-berlin.de/de/medienwissenschaft/medientheorien/fundus/media-archaeological-fundus>. See also Darren Wershler, Lori Emerson, and Jussi Parikka, *The Lab Book* (Minneapolis: University of Minnesota Press, 2022): 73-78 & 94-92.

Erkki Huhtamo's research, to name another example, is intimately connected to his own collection of historical media devices; his essays and books are replete with images captioned with "from the author's collection."¹⁴ Huhtamo, whose emphasis on the discursive "topoi" surrounding media practice is often framed in distinction to the strict technological focus of Kittler and Ernst, shares with his German counterparts an investment in the value of particular objects as evidence for a new media history. In Huhtamo's hands, a particular media device acts as a "rare survivor opens a peephole into a lost media cultural moment which it helps bring back to life."¹⁵



Figure 6 – Screenshot from a YouTube video by The Daily Bruin showing Erkki Huhtamo operating a Mutoscope, one of the early moving image devices in his collection.

To recap the taxonomy that I've developed of aesthesis with media objects: the critical essay by Parks uses the tool of visual analysis and historical argumentation to interpret recordings related to media infrastructures. Crawford's *Atlas* picks up on this, emphasizing a speculative and embodied cartographic tradition. As an exhibition, this approach is architecturally enframed and intentionally overwhelming with diagrammatics to provoke contemplation and understanding in a viewer. Ernst stresses the manipulability and operability of media devices. Huhtamo collects the oddities to augment media history and unsettle assumptions about contemporary

¹⁴ See, e.g. the preface to *Illusions in Motion* (Cambridge, MA: MIT Press, 2013). See also "Artifacts of Media Archaeology: Inside Professor Erkki Huhtamo's Office" video by *Daily Bruin*, <https://www.youtube.com/watch?v=Ks9tyaft7Gs/>.

¹⁵ Erkki Huhtamo and Doron Galili, "The Pasts and Prospects of Media Archaeology." *Early Popular Visual Culture* 18, no.4 (2020): 337.

media cultures by restoring lost *topoi* from the past, though in both cases these object-inventories tend to lose focus on infrastructural flows.

Into the infrastructural flow

The rich recent discourse of hacking, “media labs” and other kinds of hands-on media archaeological practice suggests other possibilities for engaging technical systems.¹⁶ Lori Emerson at the Media Archaeology Lab at the University of Colorado Boulder has emphasized the “recipes” and “experiments” possible with legacy devices. The MAL’s “Other Networks” projects, for example, take up communication networks that existed before (or alongside of) TCP/IP, the backbone of the Internet.



Figure 7 – Documentation of a Media Archaeology Lab's experiment showing Lori Emerson's Twitter feed transmitted over VHF radio to 8 analog televisions.

¹⁶ See Wershler, Emerson, and Parikka, *The Lab Book*.

Workshops have dealt with short wave radio, smoke signals, semaphore, and analog TV broadcasts (figure 7).¹⁷

Expanding avenues of engagement with media technology has clear pedagogical advantages, allowing students with different learning styles to encounter historical material across different technological paradigms. It also centers focus on the processual and interlocking aspects of mediascapes.

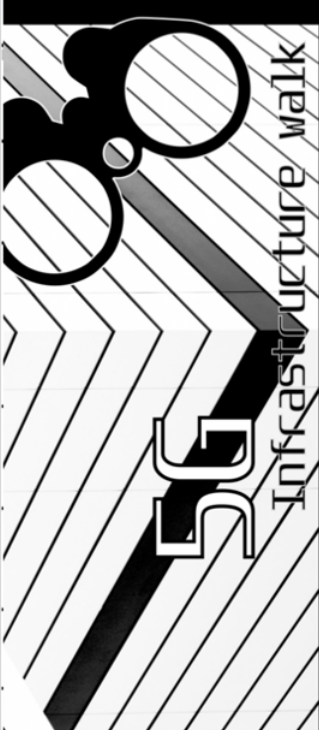
Likewise, the Critical Infrastructure Lab based at the University of Amsterdam hosts “infrastructure walks” to observe wireless equipment in urban settings. Taking groups through Berlin and Amsterdam, scholars Niels ten Oever and Maxigas also occasion embodied experiences in which familiar cityscapes are refigured in terms of the web of technologies hiding in plain sight (figure 8). These walks put into literal motion the urban media archaeological approach outlined by Shannon Mattern.¹⁸

In the rest of this article, I consider another onramp to embodied, processual engagement with material infrastructure: in-browser software emulation on sites like infinitemac.org where one can boot an emulated version of Mac 7.6 in a web browser tab alongside the one containing, for example, an email inbox. Initially, this process seems directly opposed to media archaeology’s interest in the materiality of obsolete media forms. Doesn’t cleaving the GUI from its material basis reinscribe the very fantasy of “user experience” that this work is at pains to problematize?

To be sure, vital aspects of the material culture of computing drop away in emulation. Yet, as I will argue, in-browser emulations in fact stage a tactile reckoning with not only software history, but also the underlying techniques of network infrastructure. This reckoning recasts media theoretical assumptions about the “stack” of hardware and software that produce sensory experience. To arrive at this encounter, however, I will first explain software emulation and trace the technical and historical developments that took it into the web browser.

¹⁷ Emerson, “Table of contents for Other Networks: A Radical Technology Sourcebook.” <https://loriemerson.net/2024/03/14/table-of-contents-for-other-networks-a-radical-technology-sourcebook/>. See also <https://othernetworks.net/>.

¹⁸ See *Code and Clay, Data and Dirt* (Minneapolis: University of Minnesota Press, 2017).



5G Infrastructure Walk

The **5G Infrastructure Walk** is brought to you by the People's 5G Lab, an independent sub-project of IN-SIGHT (University of Amsterdam) in which Niels ten Oever, Maxigas, and Jeroen de Vos seek to construct critical interpretative frames by analyzing governance processes and implementations that make sense of the 5G phenomena and its infrastructural ecology.




IN-SIGHT investigates standard making in relation to democratic values and practices. It asks how the public sphere is governed today through the standardization of the digital and how to support societal values in the creation of standards.

Frequencies
 4G → 800 + 2600 MHz
 5G → 700 MHz, 3500 MHz
 C2000 → 380 MHz
 DECT → 1880-1900 MHz
 GSM → 880-960 MHz
 1710-1780 MHz
 1805-1875 MHz
 WIFI → 2412-2472 MHz
 5180-5320 / 5500-5700 MHz

More information about the project:
<https://in-sight.it/wavesofinterference/>

Resources

Smartphone apps:

- * **WiFi Analyzer** 
- * **Architecture of Radio** 
- * **Public Eye** 

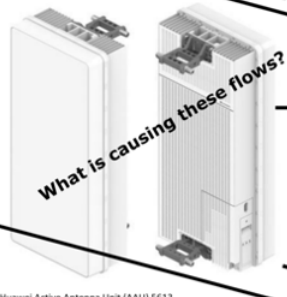
Websites:

- * **Public Eye**
<https://druktebeeld.amsterdam.nl/>
- * **Sensoren register**
<https://sensorenregister.amsterdam.nl/>
- * **Antennekaart**
<https://antennekaart.nl/>
- * **Antenneregister**
<https://antenneregister.nl/>

Site 1: FLOWS
 In front of Johan Cruyff Arena - Entrance H

What kind of flows do you see?

2.1.1. 5G base station



What is causing these flows?


Technical standard	3GPP Release 15
Frequency Band	3400Mhz - 3800Mhz
TX RX channel	64T64R
Polarization	+45°, -45°
Gain (dBi)	24

What are their functions?

Who uses them?

Site 2: LAYERS
 Inside Entrance P1 - Arena

5Ghz wireless access point



What flow devices can you observe?

How are data flows being communicated?

Can you find out more?

How would you show them on a map?

Site 3: VISIBILITY
 At the map, across Perry Sport




Figure 8 – “A handout was created” by Maxigas for the Critical Infrastructure Lab's walks, CCo.

Running the past

The archive of executable history – historical software – shares some similarities with audiovisual archives. Unlike paper documents, photographs, or material objects, but rather like “time-based” audiovisual archives, software is durational: it unfolds in sequence. Software is “executable” like a sound recording or video is “playable.” Whereas a video or audio clip, as essentially a fixed function of frequencies on a time-axis, can be transcoded and repackaged to play in different formats – on a mobile, a gallery’s media player, a streaming website – effectively repackaging software often requires consideration of the “user interaction” constituent of software experience. A software archive must not only map the times, pixels, frequency spectrum across formats for a given re-presentation, in other words, but also input (like keyboard strokes or mouse clicks) and the processual transformations occasioned thereby. Refactoring code into similar packages in new languages, interpreters, and runtime environments is one approach to achieving the same algorithmic ends.

But it’s not only these ends that concern historians and archivists of software; it’s also the texture and structure of user experience. Hence, while video or photo documentation of a legacy system is useful for providing details about interface design choices or program features, these formats necessarily transform manipulable environments into grids of pixels (image) or video timelines. Indeed, many of these systems predate reliable screenshotting/capture and/or ubiquitous high-resolution photography, the archive is paltry to begin with.

Moreover, given the scarcity and unreliability of legacy hardware – think of demagnetized floppy disks, corrupted hard drives, broken peripherals – there are substantial pragmatic obstacles to simply pointing, for example, an iPhone at the screen of a Mac from the 1980s to inscribe its software into an archival record.

This last deliberately naïve hypothetical raises the question of just what is particular to software: it is in principle a set of instructions independent from the actual machines that run it. Software emulation is a field of techniques that puts this principle into practice. The basic configuration for software emulation has a “host” operating system—typically something contemporary—running a “guest” operating system. It thus trades on Turing equivalence, that is the capacity for a computer that can be implemented by a Turing Machine, to implement any arbitrarily complex set of computations, given unlimited time.

Turing Completeness and the related Church Turing thesis are invested in the logical force of computation. These perspectives are often framed in terms of numbers and calculation: two machines are Turing equivalent if they can produce the same

(numerical) output given the same (numerical) input.¹⁹ An interest in the “texture” of user experience described above, however, draws emulation from the dispositive domain of symbolic logic to the sensorium of phenomenology. Hence, while it is afforded by the most fundamental advances in computer science, software emulation has been extensively developed, like much of digital culture, in that quintessential experiential domain of computation: gaming. Richard Rinehart and Jon Ippolito have described the collaborative enthusiast culture that gave rise to emulators developed in the 1990s by retro gamers.²⁰

This development, moreover, converged with a BBS and Usenet culture in which emulators and ROMs were freely distributed, leading to lawsuits like Sega vs MAPHIA in 1993.²¹ This scene marked a shift from an earlier for-profit model in which companies sold floppy disks containing emulators through ads in computer

```
i:Exit  -:PrevPg <Space>:NextPg v:View Attachm. d:Del r:Reply j:Next ?:Help
Date: 26 May 1994 20:22:51 GMT
From: Ron Asbestos Dippold <rdippold@qualcomm.com>
Subject: ADMIN: Welcome to comp.emulators.{announce,apple2,c64,misc}

Welcome to the new comp.emulators groups.
Note the followup to comp.emulators.announce.

FAQ keepers or Emulator Authors please see the Periodic Info Postings
section near the bottom of this post.

These are the four groups:

comp.emulators.announce  Emulator news, FAQs, announcements (moderated).
comp.emulators.apple2    Emulators of Apple // systems.
comp.emulators.cbm       Emulators of C-64 and other Commodore systems.
comp.emulators.misc      Emulators of miscellaneous computer systems.

The hoped for purpose of these groups is to take much of the
discussion of emulators that is spread all over Usenet and concentrate
it in one place, or at least to create groups where those looking for
information will be able to find it. Note that it is the system that
is being _emulated_ that determines the posting location. All
discussion of Apple // emulators, for Unix, Mac, IBM, Amiga,
etc. systems would be appropriate in comp.emulators.apple2.

- - 1/1: Ron "Asbestos" Dippo  ADMIN: Welcome to comp.emulators.{an -- (10%)
```

Figure 9 – The announcement for the comp.emulators Usenet group with the goal of drawing together “discussion spread all over Usenet.”

¹⁹ See Kyle Steiglitz, *The Discrete Charm of the Machine* (Princeton: Princeton University Press, 2019), 151-153.

²⁰ *Re-collection* (Cambridge, MA: MIT Press, 2014).

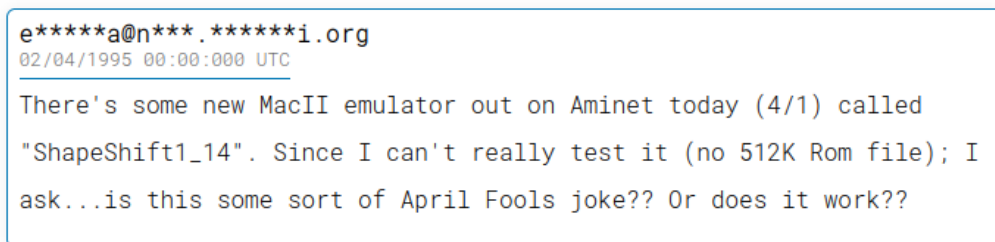
²¹ See *Sega Enterprises LTD. Vs MAPHIA* ruling, <http://www.internetlibrary.com/pdf/Sega-Enterprises-Maphia.pdf>

magazines.²² And while emulation may be afforded by one of the foundational theories of computer science, its practice was often devilishly complicated. Usenet posts from comp.amiga.sys.emulations and comp.emulators.misc contain hundreds of requests for help troubleshooting emulators (figure 9).

Emulating Apple

Users of Commodore Amiga systems represented one of the largest early emulation communities. Along with a robust Usenet discussion, the community benefited from a culture of file sharing. The site Aminet began in 1991 as an FTP server started by Swiss computer scientists. Over the next few years, it would host thousands of freely distributed software files. In 1996, two years after launching its World Wide Web site, its 30,000 files had it claiming to be the world's largest collection of freely distributed software.²³

In addition to games and other software released for the Amiga platform and its AmigaOS operating system, there was significant interest in emulating other systems, chief among them the Apple II and IIe which, as Laine Nooney has detailed, had the “largest library of programs of any microcomputer” available in the early 1980s.²⁴ This was sometimes, as with products like A-Max and Emplant, achieved through additional hardware that would slot into one of the Amiga's ports to translate software into language that the Amiga could run. In 1995, however, a German engineering student and hobbyist developer named Christian Bauer posted ShapeShifter, a software-only emulation of the Apple II, to Aminet.²⁵ There was some

A screenshot of a Usenet post. The header shows the sender as 'e*****a@n***.*****i.org' and the date as '02/04/1995 00:00:000 UTC'. The body of the post reads: 'There's some new MacII emulator out on Aminet today (4/1) called "ShapeShift1_14". Since I can't really test it (no 512K Rom file); I ask...is this some sort of April Fools joke?? Or does it work??'.

```
e*****a@n***.*****i.org
02/04/1995 00:00:000 UTC

There's some new MacII emulator out on Aminet today (4/1) called
"ShapeShift1_14". Since I can't really test it (no 512K Rom file); I
ask...is this some sort of April Fools joke?? Or does it work??
```

Figure 10 – A 1995 Usenet post on comp.sys.amiga.emulations incredulous about the new software-only Apple II emulator.

²² For a firsthand account of these changes to distribution, see “A Concise History of Emulation – Part 1 The Early Years 90’s”, YouTube video by RetroRewind. <https://youtu.be/nCvmlvS5bno/>.

²³ “The history of Aminet” archived version accessed from http://web.archive.org/web/20150220020812/http://wiki.aminet.net/The_history_of_Aminet.

²⁴ *The Apple II Age* (Chicago: University of Chicago Press, 2023), 14.

²⁵ An archival version of the Usenet post is available via Google Groups: <https://groups.google.com/g/comp.sys.amiga.emulations/c/uCuI3rQmaq8/m/e4-xBbp1oD8I>.

incredulity, with one Usenet poster noting the upload date of April 1st: “is this some sort of April Fools joke?? Or does it work??” (figure 10). (A response attested to ShapeShifter’s credibility, citing Bauer’s work on Frodo, a popular Commodore 64 emulator for Amiga.)

In 1999 Bauer would continue his prolific output with Basilisk, a program emulating the Macs of the 1980s and early 1990s on the host platform of either Linux or BeOS. A successor, Basilisk II, was highly popular and thereafter adapted by Lauri Personen for Windows and, since 2008, maintained by a community of volunteers.²⁶ Basilisk’s key contribution is in systematically modeling the 68K Motorola processors found inside Mac machines from the 1980s and early 1990s (figure 11). These processors’ instruction set architecture—the specification of how machine code tells hardware how to perform arithmetic operations, store data in registers and memory, etc.—differs both from the PowerPC that replaced them in the “PowerMacs” from 1994 on, and from those in Apple computers after the 2005 announcement of the switch the Intel-developed x86 platform, still used in most personal computers and servers (though newer Macs tout an “Apple Silicon”-specific custom instruction set, similar to the ARM varieties in most phones and tablets.)

Basilisk II and other software emulators recreate the abstractions taken as foundational by machines like the Apple Macintosh introduced in the famous 1984 Super Bowl ad. Basilisk was not the first program to do so, as emulation was a built-in feature of Mac System 7.1.2 and onwards, allowing newer PowerPC computers to run titles from the company’s back catalog. However, Basilisk’s emulation allowed the software to escape the “walled garden.” Unlike company employees with access to the famously closed-source code of operating systems like Mac OS or Windows, emulator projects tend to be driven by reverse engineering.

With a systematic translation for instructions written for one (guest) instruction set into the language expected by another (the host), the original binary files—say those held on a floppy disk or pre-loaded onto an OEM computer, become once again executable on a newer machine at paltry cost of computation, thanks to the exponential scaling of computer power. If one managed to extract a ROM—Read Only Memory image—from the computer and/or floppy disk as a file, one could execute this file in a properly configured emulator. As mentioned above, though, accessing and/or extracting ROMs and properly configuring an emulator is not a simple affair.

²⁶ “Basilisk II” *emaculation* wiki entry, https://www.emaculation.com/doku.php/basilisk_ii.

```

105 #include "debug.h"
106
107
108 // Constants
109 const char ROM_FILE_NAME[] = "ROM";
110 #if !EMULATED_68K
111 const int SIG_STACK_SIZE = SIGSTKSZ; // Size of signal stack
112 #endif
113 const int SCRATCH_MEM_SIZE = 0x10000; // Size of scratch memory area
114
115
116 #if !EMULATED_68K
117 // RAM and ROM pointers
118 uint32 RAMBaseMac; // RAM base (Mac address space)
119 uint8 *RAMBaseHost; // RAM base (host address space)
120 uint32 RAMSize; // Size of RAM
121 uint32 ROMBaseMac; // ROM base (Mac address space)
122 uint8 *ROMBaseHost; // ROM base (host address space)
123 uint32 ROMSize; // Size of ROM
124 #endif
125
126
127 // CPU and FPU type, addressing mode
128 int CPUType;
129 bool CPUis68060;
130 int FPUType;
131 bool TwentyFourBitAddressing;
132
133
134 // Global variables
135 #ifndef USE_SDL_VIDEO
136 extern char *x_display_name; // X11 display name
137 extern Display *x_display; // X11 display handle
138 #ifdef X11_LOCK_TYPE
139 X11_LOCK_TYPE x_display_lock = X11_LOCK_INIT; // X11 display lock
140 #endif
141 #endif
142
143 static uint8 last_xpram[XPRAM_SIZE]; // Buffer for monitoring XPRAM changes
144
145 #ifdef HAVE_PTHREADS
146 #if !EMULATED_68K
147 static pthread_t emul_thread; // Handle of MacOS emulation thread (main thread)
148 #endif
149
150 static bool xpram_thread_active = false; // Flag: XPRAM watchdog installed
151 static volatile bool xpram_thread_cancel = false; // Flag: Cancel XPRAM thread
152 static pthread_t xpram_thread; // XPRAM watchdog
153
154 static bool tick_thread_active = false; // Flag: 60Hz thread installed
155 static volatile bool tick_thread_cancel = false; // Flag: Cancel 60Hz thread
156 static pthread_t tick_thread; // 60Hz thread
157 static pthread_attr_t tick_thread_attr; // 60Hz thread attributes
158
159 static pthread_mutex_t intflag_lock = PTHREAD_MUTEX_INITIALIZER; // Mutex to protect InterruptFlags
160 #define LOCK_INTFLAGS pthread_mutex_lock(&intflag_lock)
161 #define UNLOCK_INTFLAGS pthread_mutex_unlock(&intflag_lock)

```

Figure 11 – Screenshot from Basilisk II’s GitHub repository showing a Motorola 68K processor (like those used in the Apple II) emulated for Unix in C++.

JavaScript eats the world

The enthusiasts in the 1990s writing emulation software for old gaming consoles and PC operating systems were doing so at the dawn of what Michael Lewis would term “the new new thing”: multimedia transmitted over the world wide web.²⁷ During the mid- to late-90's boom of emulation that coincided with stronger processors, files began to be distributed not over floppy disks sent through the post or BBS servers, but personal websites (figure 12).²⁸ And while the developers involved tended to prefer a minimalist aesthetic for distributing their files and listing their projects, there was an appetite for a scripting language to add some dynamism to the static experience of HTML.

JavaScript was famously authored in 10 days in May 1995 by Brendan Eich, who'd been recently hired by Netscape to pursue Marc Andreessen's “rallying cry” that “Netscape plus Java kills Windows.”²⁹ While the scripting language would end up with only loose semantic connections to its namesake, Java, it would eventually overtake Microsoft's rival Visual Basic Script. In the second half of the 1990s, most major web browsers included an engine for executing these scripting languages. For a complex mix of reasons beyond the scope of this article, JavaScript's adoption was fragmented by the browser wars of the 1990s and the prevalence of Microsoft's ActiveX plugin, Sun's Java Runtime Environment, and Macromedia's Flash, extensions which ran much of the dynamic web content during this period.³⁰

This changed with the spread of AJAX techniques (asynchronous JavaScript and XML) in popular products from the mid-2000s, like Gmail and Google Maps. Processing instructions (written in JavaScript) from a website asynchronously allowed for components to load after other aspects of the page had been rendered by the browser. So, Google Maps would snappily update based on user interaction (e.g., zooming in on a map) without the need for any external plug-ins.³¹ Google's release of

²⁷ *The New New Thing* (New York: Norton, 1999).

²⁸ For an firsthand account of these changes to distribution, see “A Concise History of Emulation – Part 1 The Early Years 90's”, a YouTube video by RetroRewind. <https://youtu.be/nCvmlvS5bno>.

²⁹ Wirfs-Brock and Eich, “JavaScript: The First 20 Years,” *Proceedings of the ACM on Programming Languages* 4 (June 2020), 7.

³⁰ See Wirfs-Brock and Eich, “JavaScript: The First 20 Years,” 52-79. On the prevalence of Flash, see Megan Ankersen, *Dot-Com Design* (New York: New York University Press, 2018): 141-158. On Java in the late-90s, see Mary Brandel, “Java and Windows 95,” *Computerworld* November 22, 1999.

³¹ Jesse James Garrett, “Ajax: a New Approach to Web Applications” (February 18, 2005), archived version accessed via the Internet Archive at <http://web.archive.org/web/20050222032831/http://www.adaptivepath.com/publications/essays/archives/000385.php>.

the V8 JavaScript engine built into the new Chrome browser marked a considerable leap in efficiency further enabling dynamic interaction within the open standards of the web.³²

Chrome’s dramatic increase in rendering speed was achieved with its V8 engine, which, using a technique called Just-in-Time compilation, would turn a developer’s JavaScript into “native machine code”—the kind of commands included in the instruction set architecture detailed above. V8 also used innovations in this compilation process like hidden classes, inline caching, and generational garbage collection to address the challenges posed by the openness and flexibility (dynamic typing) that made JavaScript easier to learn than its stricter counterparts.³³

Similar innovations would be taken up by the WebKit and Gecko JavaScript engines in Safari and Firefox, respectively. By the end of 2008, Chrome, Firefox, and Safari had all introduced Just-In-Time compilers, dramatically increasing the speed of JavaScript.³⁴

The tandem developments of emulation software by an online community and the increased capacity of browser engines to natively run complex web applications would lead, in a few years, to experiments in running legacy systems in the browser. To join these strands, let’s revisit the example of Basilisk II, the emulator behind infinitemac.org.

Previously, running Bauer’s C++ source code (a small part of which is shown in Figure 11.) meant first sending that code through a compiler, an algorithm which would put together a binary readable by a specific “instruction set architecture,” like the x86 of PC desktops since the 1990s. A user looking to download a pre-compiled binary file—one to simply open and execute—would thus select a file corresponding to their machine (figure 12). These files are the result of C++ using an “ahead-of-time” compilation process.

³² See Krik L. Kroeker, “Toward Native Web Execution,” *Communications of the ACM* 52, no. 7 (July 2009): 16-17. <https://doi.org/10.1145/1538788.1538795>.

³³ Mads Ager, “V8: High Performance JavaScript Engine,” talk at *Google I/O 2009*, <https://youtu.be/FrufjFBSOQY>. See also Richard Artoul, “JavaScript Hidden Classes and Inline Caching in V8” *Under the Hood* (blog), April 26, 2015, <https://richardartoul.github.io/jekyll/update/2015/04/26/hidden-classes.html>

³⁴ Alon Zakai, “The History of WebAssembly,” YouTube video, December 3, 2020, <https://youtu.be/XuZtiOCCQTg>.



Figure 12 – Screenshot of download options on a Bauer's personal site for Basilisk II, archived on October 13, 2005, accessed via <https://web.archive.org/web/20051013064429/http://basilisk.cebix.net/>. Note the option do download source code or binaries for different architectures (i386, ppc, x86, and Amiga.)

As we have seen, JavaScript engines such as Chrome's V8 also compile source code, though theirs is typically "just-in-time" compilation. In a 10-year anniversary post about the Internet Archive's emulation service, archivist Jason Scott explains that the site's offerings of historic arcade games and software are enabled by an innovation in this process: a "cross-compiler" named Emscripten made it possible to compile code originally written in C, C++ or Objective-C to JavaScript.³⁵

According to then-Mozilla employee Alon Zakai explained in a paper introducing Emscripten, previous attempts to run other programming languages on the web foundered because they could not "run on some platforms, for example, Java and Flash cannot run on iOS devices such as the iPhone and iPad."³⁶ Zakai presented the paper in 2011, four years after the announcement of the iPhone and a year after release of the first iPad. Projects like Zakai's apprehended a web culture that

³⁵ "A Quarter In, A Quarter-Million Out: 10 Years of Emulation at Internet Archive," September 2023, <https://blog.archive.org/2023/09/20/a-quarter-in-a-quarter-million-out-10-years-of-emulation-at-internet-archive>. For a remarkable set of emulators originally written in JavaScript, see <https://copy.sh/v86>. Emscripten now compiles not to JavaScript, but to WebAssembly, a binary format adopted by major browsers from 2017 to 2019. See Haas, et al., "Bringing the Web up to Speed with WebAssembly," *PLDI 2017*, <https://doi.org/10.1145/3062341.3062363>.

³⁶ Alon Zakai, "Emscripten: An LLVM-to-JavaScript Compiler," *Proceedings of the ACM International Conference on Object Oriented Programming Systems, Languages and Applications Companion* (October 2011), 301. <https://doi.org/10.1145/2048147.2048224>.

increasingly ran on the html, CSS, and JavaScript universal to web browsers across devices. (The first presentation in this 2011 session was by Brendan Eich, the original author of the JavaScript language: “The JavaScript World Domination Plan at 16 Years.”³⁷) The brief “example uses” section at the end of Zakai’s Emscripten paper include implementations of Python, PDF-rendering, and the Bullet Physics library.³⁸ In just a few days after Zakai’s presentation, Jason Scott posted on his blog a call to bring emulators into the browser using JavaScript.³⁹

Part of Scott’s appeal in his plea for help bringing emulation in the browser was to augment a history comprised of “artifacts” (the kinds of files he archived with archive.org) with “experiences.”⁴⁰ It took a couple years for these “experiences” to load but in January 2013, Scott announced a beta version of JSMESS, which ported the cross-platform emulator MESS to JavaScript (figure 13). The first version offered support for the game consoles Atari 2600, ColecoVision, Fairchild Channel F, Odyssey2, and the Sega Genesis, as well as the Texas Instruments 99 4/a PC.⁴¹ That month also saw the initial commit of an emulator of the OpenRISC1000 processor written in JavaScript.⁴²

More platforms would follow that year, both on archive.org’s emulation project, powered by JSMESS, and elsewhere. Ansgar Grunseid announced Arc, a project putting Linux machines in websites in August 2013.⁴³ That October, the developer and UX designer James Friend used Emscripten to port PCE, an emulator of Classic Mac OS written in C, to the browser.⁴⁴ The pseudonymous copy.sh shared a hand-coded emulator of the x86 instruction set, which initially ran several Linux distributions and later offered a range of operating systems including various releases of Windows and Unix-based systems.⁴⁵

³⁷ <https://doi.org/10.1145/2048147.2048218>

³⁸ Zakai, “Emscripten: An LLVM-to-JavaScript Compiler,” 310-311.

³⁹ “JavaScript Hero: Change Computer History Forever” on *ASCII* by Jason Scott (blog) <http://ascii.textfiles.com/archives/3375>.

⁴⁰ Scott, “JavaScript Hero”

⁴¹ “JavaScript MESS” about page, archived version on January 26, 2013 accessed via <http://web.archive.org/web/20130126213541/http://jsmess.textfiles.com:80/>

⁴² <https://github.com/s-macke/jorik/commit/96a52715447df3ccbef9d3dfcc1bb19903dd4bef>

⁴³ Ansgar Grunseid, “Virtual Machines in the Browser” (blog post), <http://blog.grunseid.com/2013/arc.html>.

⁴⁴ Announced at “PCE.js – Classic Mac OS in the Browser” <https://jamesfriend.com.au/pcejs-classic-mac-os-browser> and still available at <https://jamesfriend.com.au/pce-js/>.

⁴⁵ <https://copy.sh/v86>.

Try the Beta Version!

We're now experimenting with a beta version of the JSMESS program. (A lot needs to be done. Click on the following systems to try them out:

[Atari 2600](#)·[ColecoVision](#)·[Fairchild Channel F](#)
[Odyssey2](#)·[Sega Genesis](#)·[Texas Instruments 99 4/a](#)



Figure 13 – Screenshot of announcement for first Beta of JSMESS on Jason Scott's textfiles.com.

Some of the discourse around these projects carries the thrill of pulling off the impossible, or at least the seemingly ill-advised, demonstrating that well-trod hacker mastery over the latest technologies to absurd ends. Well ahead of the curve, the legendary open-source programmer Fabrice Bellard wrote of his own JSLinux emulator, published to his personal website May 23, 2011: “I did it for fun, just because newer JavaScript Engines are fast enough to do complicated things.”⁴⁶ Comments on the popular discussion site *Hacker News* alternate between glee and awe (figure 14). Glee at the audacious incongruity between an operating system, the software ‘closest to the metal’ of a given machine, and the (increasingly less) humble HTML website,

⁴⁶ “JavaScript PC Emulator – Technical Notes,” blogpost, May 23, 2011. Archived version accessed through <http://web.archive.org/web/20110524162113/http://bellard.org/jslinux/tech.html>

▲ hardwaresofton on Oct 17, 2013 | prev | next [-]

Pretty awesome, I for one don't think the constant stream of "x, rewritten completely in javascript" is tiring (this is not meant to be sarcastic).

[EDIT] - This post doesn't seem to say enough when I look back at it. Wanted to add this:

Seeing posts like this really excites me (and inspires me) about the future of web programming, and programming in general. Anyone that's excited about programming has to get excited about abstraction, and it doesn't get much more abstract than cross-coded/implemented virtualized systems like this. Even if you hate javascript.

Every step people take in blurring lines between systems like this should be exciting, given the large amount of abstraction that had to go into creating something like this.

Figure 14 – A post on Hacker News discussing copy.sh's v86 JavaScript emulator, <https://news.ycombinator.com/item?id=6567967>.

several TCP/IP layers abstracted from the devices that abstract it, all JavaScript, the often-derogated “kiddie” language. Awe at the mastery of not only the nominally “frontend” language and the emulated operating systems, but all the way down the stack to the granular functions of CPUs.

More recent iterations of these projects have drawn on the newly adopted standard of WebAssembly, jointly designed by engineers from Google, Microsoft, Mozilla and Apple.⁴⁷ WebAssembly is a bytecode or portable binary-code format—the “assembly” in its name refers to the low-level language that corresponds to a given instruction set architecture—Power PC and x86 each have their own assembly language. WebAssembly was designed to be platform independent, however, and is instead targeted towards a web browser. Because it's a low-level binary file, it's not a language that developers write in, but instead a “compilation target”—human written source code is fed to a compiler that outputs the webassembly .wasm file.

Before turning to the last example, infinitemac.org and engaging phenomenologically with the experience of computing infrastructure that it occasions, it's worth establishing the stakes of this encounter through a review of the overlapping practices that made it possible.

The principle behind emulation—Turing equivalence—was demonstrated in 1937 but emerged in broader practice in the 1980s and especially the 1990s, when

⁴⁷ See Haas, et al., “Bringing the Web up to Speed with WebAssembly” and Alon Zakai, “The History of WebAssembly.” Zakai stresses the project's resulting from collaboration between major browser vendors Mozilla, Google, and Apple.

personal computer users with a nostalgic connection to an earlier cycle of PCs, started tinkering with running the old software on their ever-more-powerful processors. A culture of sharing, iterating, and hobbyists developed an 1990s run of these emulators, distributing them first over BBS's, FTP and collectively troubleshooting them in Usenet groups.

Along a parallel track, as this scene shifted to HTTP websites, JavaScript was developed and introduced to add dynamic execution capabilities to what had been static webpages. After a decade of false starts and fragmentation despite integration in every major browser, interactive sites on the AJAX model, namely Google Maps, demonstrated the promise of developing applications that 'just ran' in the browser, without separate plugins. With the release and eventual dominance of the iPhone and other touchscreen internet devices that could run standard HTML/JavaScript but not Flash or other plugins, this position solidified. The development of more sophisticated engines for running the JavaScript code, like Chrome's V8 expanded developers' sense of what was possible to run in the browser. Tools for cross-compiling established codebases, like Emscripten, allowed emulators designed to be run on a local machine to be systematically ported into webpages, as with the archivally-minded JSMESS. The adoption of WebAssembly as a binary compilation target that could run on every browser further enhanced the performance of non-JavaScript programs running in a web browser.

"Software is eating the world," declared Marc Andreessen in a much-hyped 2011 *Wall Street Journal* essay published simultaneously on the website of his influential venture capital firm.⁴⁸ At least among the architects of this cosmophagy, software's own recent ancestors were on the menu.

Point and click: digital aesthesis

Infinitemac.org is a hobbyist project run by Mihai Parparita, a San Francisco-based developer who has worked for Slack and Google. The site draws together existing web-adaptations of emulators written to be run locally—like James Friend's port of Christian Bauer's Basilisk II for running Classic Mac in browser that we have seen above—in addition to new compilations of other emulators to WebAssembly. Parparita has documented the project on blog.persistent.info.

⁴⁸ Marc Andreessen, "Why Software Is Eating The World," blogpost, August 20, 2011 <https://a16z.com/why-software-is-eating-the-world/> and *The Wall Street Journal* <https://www.wsj.com/articles/SB10001424053111903480904576512250915629460>.

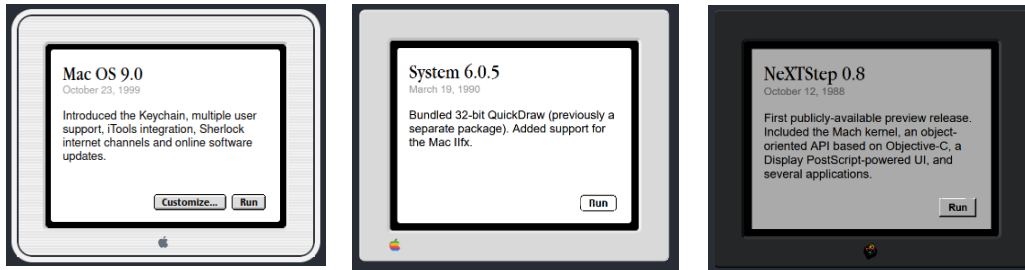


Figure 15 – Screenshots showing stylized icons corresponding to design of the emulated systems.

On loading the site, a user is presented with a succession of bootable virtual machines running Mac and NeXT operating systems from 1984 to 2001 (figure 16). In addition to a one- or two-sentence description of the developments present in each emulated release, each is styled with reference to the original PCs that ran them—a “beige box” framing the Chicago typeface and rounded buttons for the mid-80s Macs and the dark frame and Helvetica text on a shadowed button for the NeXT machines (figure 15). Parparita describes this display as a kind of curated gallery, responding to the overwhelming choice of several hundred bootable images at the Internet Archive.

When a user mouses over one of these, perhaps to click on the “run” button, they are presented with an additional button: “customize” (figure 16). This option represents one of *infinitemac*’s unique contributions: the capacity to load particular software and also to save data from an emulated environment. In a blogpost describing the motivations for the project, Parparita mentions the previous web discussed above, but notes that “none of these setups replicated the true feel of using a computer in the 90s. They’re great for quickly launching a single program and playing around with it, but they don’t have any persistence, way of getting data in or out of it, or running multiple programs at once.”⁴⁹ The effort even extends to networking between instances, so that the site supports using AppleTalk or multi-player games across emulators running separately.

So, what of the embodied experience that this produces? One immediately present UX frustration arises when navigating the menus of Mac OS 7.1. To continue seeing a dropdown menu, a user must keep holding down the mouse button after clicking on “File” or “Edit”. The process of selecting thus becomes integrated into the

⁴⁹ Mihai Parparita, “Infinite Mac: An Instant-Booting Quadra in Your Browser,” *persistent.info* blog, March 31, 2022, <https://blog.persistent.info/2022/03/blog-post.html>.



Figure 16 – Screenshot of the first four systems available for emulation on infinitemac.org’s “gallery”. original click, and force is applied in step with the scanning of the eye(s) and the reasoning of the mind.

We can observe, with the help of infinitemac, the switch to “sticky menus” in OS 8 when clicking a menu becomes a one-shot affair. The tactile logic shifts from “let me execute the Edit/Copy command” to “let me browse the available Edit commands to see if there’s one I’d like.” In some senses, this might be a holdover from terminal autocompletion which had existed since the 1960s, for example in the Berkeley Timesharing System and later with the press of a button like with “escape

completion” in TOPS-20, an operating system for the PDP-10.⁵⁰ It may resonate, therefore, with the “tab completion” still vital to the everyday use of Unix shells like bash, fish, and zsh today.

Beyond demonstrating this development in the history of user experience design, this interruption to the browsing process represents a unique sort of glitch: what is perceived as a malfunction in the website’s interactivity is, in fact, the faithful rendering of an old paradigm. In this sense, while it shares some aesthetic sensibilities with Shane Denson’s *Discorrelated Images* or Legacy Russell’s glitch manifesto, this effect arises from a kind of seamlessness.⁵¹ Here, a disorientation stems from the seamless integration of multiple vernaculars, paradigms, systems into the surface of the self-same browser window.

On one hand, this seamlessness can be read as symptomatic of the Silicon Valley fantasy of “frictionless” design, a fantasy that subordinates material reality to ephemeral user experience.⁵² While it pays homage to the physicality of these machines with frames, beeps, and blinking LEDs, no emulation can capture the proper sounds, smells, and tactile sensations of hardware. What’s more, as Parparita has explained, the project involves a constant tradeoff between realistic simulation of these systems (which often operated several times more slowly than their infinitemac counterparts) and accessibility.⁵³ The emulated systems, for example, expect user input from a mouse and keyboard, demanding that Parparita cobble together systems for translating touch input from smartphones and tablets. So, while the look and feel of the UX are present, many of the characteristic aspects of the material technology fade into yet another user interface (figure 18).

On the other hand, the “experience” seamlessly achieved by this scrollable gallery of bootable machines resists assimilation into a daily practice. A user on a touch device may be able to glide down the gallery to select a machine to boot, for example, but once it does boot, they will quickly be forced into the logic of the scrollbar with clickable arrows. The “tap” of both handheld devices and laptop trackpads is in tension, in a related juxtaposition, with the double-clicking expected by something like Mac OS 8. In these situations, like with glitches, routinized tactile engagements with technology are disrupted.

⁵⁰ Dan Murphy, “Origins and Development of TOPS-20.”

<https://web.archive.org/web/20200801165237/http://www.opost.com/dlm/tenex/hbook.html>

⁵¹ Shane Denson, *Discorrelated Images* Durham, NC: Duke University Press, 2020; Legacy Russell, *Glitch Feminism: A Manifesto*, London: Verso, 2020.

⁵² See Jakko Kemper, *Frictionless*, (London: Bloomsbury,) 2024.

⁵³ Comment by Mihai Parparita in Zoom interview with the author, March 8, 2024.

Recent work by Kyle Stine and Jacob Gaboury on the hardware of computer imagery has rightfully grounded the visual experience of the digital in the material processes that produce it.⁵⁴ Likewise, scholars like David Parisi and Rachel Plotnick have persuasively reframed media histories in the tactile and haptic.⁵⁵ In browser emulations transfigure input/output devices like the screen and speakers, but also the mouse/capacitive touch surface (figure 17).



Figure 17 – Photo of four Apple devices running [infinitemac.org](https://blog.persistent.info/2023/03/infinitemac-dot-org.html), posted on the blog of Mihai Parparita, the project's developer. <https://blog.persistent.info/2023/03/infinitemac-dot-org.html>. Photo courtesy of Mihai Parparita.

⁵⁴ Stine, “Critical Hardware: The Circuit of Image and Data,” *Critical Inquiry* 45 (Spring 2019): 762-786; Gaboury, *Image Objects* (Cambridge, MA: MIT Press 2021).

⁵⁵ See, e.g., Parisi, *Archaeologies of Touch* (Minneapolis: University of Minnesota Press, 2018) and Plotnick, “Force, Flatness, and Touch without Feeling,” *New Media & Society* 19, no. 10 (October 2017): 1632-162.

Emulation everywhere

Infinitemac and other emulation projects expose a user to the change over time, and thus contingency of, certain tactile computing formations. More than this, though, the gallery of possible bootable images and the potential to transform a browser window into another machine materializes a fundamental infrastructural technique that undergirds digital culture: virtualization.

It's not just these niche retro-computing projects that seek to turn individual machines into abstractions, repeated across hardware. This is precisely the logic behind every major website, mobile service, and “cloud” phenomenon. What we encounter with in-browser emulation is a middle ground between the specific, “bare metal” materiality of a given machine executing its own machine code, and the nebulous, dematerialized mass of cloud compute. Highly structured, recursively contained, iterated – these adjectives describe the infrastructures of our contemporary apps, too.⁵⁶ Amazon Web Services, social media platforms, food delivery apps, VPNs, dating sites are afforded by “virtual machines.”⁵⁷ Emulation is a subspecies of virtualization, the key technique behind the cloud. Despite seeming like an eccentric combination of technologies, then, it in fact is the rule and not the exception of computational culture.

How, finally, does this all connect with the methodological questions I raised in this article's opening sections on the range of approaches for engaging the media infrastructures that condition our world? Clearly, I have not eschewed either the critical reading of Parks, nor the cartographic diagramming of Crawford. I have, however, tried to present an avenue to these practices sensitive to the embodied and mediated position from which we make either kind of observation. Likewise, I suspect my website-facing approach might be unsatisfying to object-oriented scholars, both in the vein of Huhtamo's technocultural topoi and Ernst's radical operationalism. I certainly do not want to suggest that emulation obviates the need for engagement with historical artefacts. But part of the materiality that we must engage as scholars of media has to do with the arrangement and orchestration of individual machines in broader sequence. This is where going “hands-on” with the internet emulations can be instructive.

Nor can clicking through these hobbyist projects supplant the experimentation with alternative technical imaginaries by Emerson and the Media Archaeology Lab's “Other Networks.” As quirky as these emulations can be, they fall

⁵⁶ See my “On Bare Metal: Recovering Virtualization” (forthcoming.)

⁵⁷ Matthew Portnoy, *Virtualization Essentials*, (Hoboken: Wiley, 2023,) xv.

squarely in line with a kind of developer ethos and at times unquestioning acceptance of the teleological view of “software eating the world.” By framing the projects in a spirit of experimentation and alternative possibilities, though, I hope to frame fissures in this teleology. As inventive and efficient as the latest “over the wire” approaches are, of course, they still sit atop the kind of network infrastructure that we would do well to observe on a 5G walk with the Critical Infrastructure Lab.

So, I end with a call for a pluralist, embodied study of media infrastructure: let it be observed by the eye, framed by the brain, manipulated by the hand, and clicked with the finger.



Figure 18 – Wikimedia image "The Apple Mouse," showing designs from 1984 to 2005. CC BY-SA 2.0.

Bibliography

- Acker, Ameila. "Emulation Practices for Software Preservation in Libraries, Archives, and Museums." *Journal of the Association for Information Science and Technology* 72, no. 9 (September 2019): 1148–1160. <https://doi.org/10.1002/pra2.279>.
- Ankerson, Megan. *Dot-com Design: The Rise of a Usable, Social, Commercial Web*. New York: New York University Press, 2018.
- Bellard, Fabrice. "JSLinux – Technical Notes." <https://bellard.org/jslinux/tech.html>.
- Brown, Bill. "Thing Theory." *Critical Inquiry* 28, no. 1 (Autumn 2001): 1–22.
- Crawford, Kate. *The Atlas of AI: Power, Politics, and the Planetary Costs of Artificial Intelligence*. New Haven: Yale University Press, 2021.
- Ernst, Wolfgang. *Digital Memory and the Archive*. Minneapolis: University of Minnesota Press, 2012.
- Ernst, Wolfgang. *Chronopoetics: The Temporal Being and Operativity of Technological Media*. Translated by Anthony Enns. New York: Rowman & Littlefield, 2016.
- Friend, James. "Porting the Basilisk II Classic Macintosh Emulator to the Browser." November 16, 2017. <https://jamesfriend.com.au/basilisk-ii-classic-mac-emulator-in-the-browser>.
- Friend, James. "Porting the PCE Emulator to the Browser." April 11, 2017. <https://jamesfriend.com.au/porting-pce-emulator-browser>
- Friend, James. "Why Port Emulators to the Browser?" October 18, 2013. <https://jamesfriend.com.au/why-port-emulators-browser>.
- Gaboury, Jacob. *Image Objects: An Archaeology of Computer Graphics*. Cambridge, MA: MIT Press, 2021.
- Garrett, Jesse James. "Ajax: a New Approach to Web Applications." *adaptive path* blog. Archived version accessed via <http://web.archive.org/web/20050222032831/http://www.adaptivepath.com/publications/essays/archives/000385.php>
- Haas, Andreas, Andreas Rossberg, Derek L. Schuff, Ben L. Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, Michael Holman, JF Bastien. "Bringing the Web up to Speed with WebAssembly." *PLDI 2017: Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*. <https://doi.org/10.1145/3062341.3062363>.

- Hinkelmann, Franziska. "JavaScript Engines—How Do They Even?" Presentation at *JSConf EU* 2017. <https://youtu.be/p-iiEDtpty6I>.
- Huhtamo, Erkki. *Illusions in Motion: Media Archaeology of the Moving Panorama and Related Spectacles*. Cambridge, MA: MIT Press, 2013.
- Ippolito, Jon. "Emulation." In *Debugging Game History: A Critical Lexicon*. Edited by Henry Lowood and Raiford Guins. Cambridge, MA: MIT Press, 2016.
- Jain, Shashank Mohan. *WebAssembly for Cloud: A Basic Guide for Wasm-Based Cloud Apps*. New York: Apress, 2022.
- Kemper, Jakko. "Deep Time and Microtime: Anthropocene Temporalities and Silicon Valley's Longtermist Scope." *Theory, Culture & Society* (2024). <https://doi.org/10.1177/02632764241240662>.
- Kittler, Friedrich. "Es gibt keine Software." *Draculas Vermächtnis: Technische Schriften*. Leipzig: Reclam Verlag, 1993.
- Kroeker, Kirk L. "Toward Native Web Execution." *Communications of the ACM* 52, no. 7 (July 2009) 16-17. <https://doi.org/10.1145/1538788.1538795>.
- Lantinga, Sam. "SDL: Making Linux Fun." IBM developerWorks. September 1, 1999. Accessed on February 22, 2024 via web.archive.org <http://web.archive.org/web/20030511174315/http://www-106.ibm.com/developerworks/library/l-making-linux-fun>.
- Lewis, Michael. *The New New Thing: A Silicon Valley Story*. New York: Norton, 1999.
- Mackenzie, Adrian. "Java™: The Practical Virtuality of Internet Programming." *New Media & Society* 8, no. 3 (2006): 441-465. <https://doi.org/10.1177/1461444806061954>.
- Mattern, Shannon. *Code and Clay, Data and Dirt: Five Thousand Years of Urban Media*. Minneapolis: University of Minnesota Press, 2017.
- Nooney, Laine. *The Apple II Age: How the Computer Became Personal*. Chicago: University of Chicago Press, 2023.
- Paglen, Trevor. "Invisible Images (Your Pictures Are Looking at You)." *The New Inquiry*, December 8, 2016. <https://thenewinquiry.com/invisible-images-your-pictures-are-looking-at-you>.
- Parikka, Jussi. *A Geology of Media*. Minneapolis: University of Minnesota Press, 2015.
- Parisi, David. *Archaeologies of Touch: Interfacing with Haptics from Electricity to Computing*. Minneapolis: University of Minnesota Press, 2018.

- Parisi, David, Mark Paterson, and Jason Edward Archer. "Haptic Media Studies." *New Media & Society* 19, no. 10 (October 2017): 1513–1522.
- Parks, Lisa. "'Stuff You Can Kick': Toward a Theory of Media Infrastructures." In *Between Humanities and the Digital*, edited by Patrik Svensson and David Theo Golberg, 355–373. Cambridge, MA: MIT Press, 2015.
- Parks, Lisa, and Nicole Starosielski, eds. *Signal Traffic: Critical Studies of Media Infrastructures*. Urbana: University of Illinois Press, 2015.
- Parparita, Mihai. "Infinite Mac: An Instant-Booting Quadra in Your Browser." *persistent.info* (blog), March 31, 2022. <https://blog.persistent.info/2022/03/blog-post.html>.
- Plotnick, Rachel. "Force, Flatness, and Touch without Feeling: Thinking Historically about Haptics and Buttons." *New Media & Society* 19, no. 10 (October 2017): 1632–1652. <https://doi.org/10.1177/1461444817717510>.
- Pogue, David. "OS 8 and Why It's Great." *Macworld* 14, no. 12 (December 1997).
- Rinehart, Richard and Jon Ippolito. *Re-collection: Art, New Media, and Social Memory*. Cambridge, MA: MIT Press, 2014.
- Scott, Jason. "JavaScript Hero: Change Computer History Forever." *ASCII* (blog). October 28, 2011. <http://ascii.textfiles.com/archives/3375>.
- Scott, Jason. "Microcomputer Software Lives Again, This Time in Your Browser." (blog), *Internet Archive*, October 25, 2013. <https://blog.archive.org/2013/10/25/microcomputer-software-lives-again-this-time-in-your-browser>.
- Scott, Jason. "A Quarter In, A Quarter-Million Out: 10 Years of Emulation at Internet Archive." *Internet Archive Blog*, September 20, 2023. <https://blog.archive.org/2023/09/20/a-quarter-in-a-quarter-million-out-10-years-of-emulation-at-internet-archive>.
- Steiglitz, Kyle. *The Discrete Charm of the Machine: Why the World Became Digital*. Princeton: Princeton University Press, 2019.
- Stine, Kyle. "Critical Hardware: The Circuit of Image and Data." *Critical Inquiry* 45 (Spring 2019): 762–786.
- Wershler, Darren, Lori Emerson, and Jussi Parikka. *The Lab Book: Situated Practices in Media Studies*. Minneapolis: University of Minnesota, 2022.

Wirfs-Brock, Allen and Brendan Eich. “JavaScript: The First 20 Years.” *Proceedings of the ACM on Programming Languages* 4, issue HOPL, article 77 (June 2020): 1-189. <https://doi.org/10.1145/3386327>.

Zakai, Alon. “Emscripten: An LLVM-to-JavaScript Compiler.” *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems, Languages and Applications Companion*, 301-312. New York: ACM, 2011. <https://doi.org/10.1145/2048147.2048224>.

Zakai, Alon. “The History of WebAssembly.” YouTube video. Published December 3, 2020. <https://youtu.be/XuZt1OCCQTg>.