1

U.S. Operating Systems at Mid-Century

The Intertwining of Race and UNIX

TARA MCPHERSON

University of Southern California

I begin with two fragments cut from history, around about the 1960s. This essay will pursue the lines of connection between these two moments and argue that insisting on their entanglement is integral to any meaningful understanding of either of the terms this volume's title brings together: the internet and race. Additionally, I am interested in what we might learn from these historical examples about the very terrains of knowledge production in the post-World War II United States. The legacies of mid-century shifts in both our cultural understandings of race and in digital computation are still very much with us today, powerful operating systems that deeply influence how we know self, other and society.

Fragment One

In the early 1960s, computer scientists at MIT were working on Project MAC, an early set of experiments in Compatible Timesharing Systems for computing. In the summer of 1963, MIT hosted a group of leading computer scientists at the university to brainstorm about the future of computing. By 1965, MULTICS (Multiplexed Information and Computing Service), a mainframe timesharing operating system, was in use, with joint development by MIT, GE, and Bell Labs, a subsidiary of ATT. The project was funded by ARPA of the Defense Department for two million a year for eight years. MULTICS introduced early ideas about modularity in hardware structure and software architecture.

In 1969, Bell Labs stopped working on MULTICS, and, that summer, one of their engineers, Ken Thompson, developed the beginning of UNIX. While there are clearly influences of MULTICS on UNIX, the later system also moves away from the earlier one, pushing for increased modularity and for a simpler design able to run on cheaper computers.

In simplest terms, UNIX is an early operating system for digital computers, one that has spawned many offshoots and clones. These include MAC OS X as well as LINUX, indicating the reach of UNIX over the past forty years. The

system also influenced non-UNIX operating systems like Windows NT and remains in use by many corporate IT divisions. UNIX was originally written in assembly language, but after Thompson's colleague, Dennis Ritchie, developed the C programming language in 1972, Thompson rewrote UNIX in that language. Basic text-formatting and editing features were added (i.e. early word processors). In 1974, Ritchie and Thompson published their work in the *Journal of the Association for Computing Machinery*, and UNIX began to pick up a good deal of steam.[1]

UNIX can also be thought of as more than an operating system, as it also includes a number of utilities such as command line editors, APIs (which, it is worth noting, existed long before our Google maps made them sexy), code libraries, etc. Furthermore, UNIX is widely understood to embody particular philosophies and cultures of computation, "operating systems" of a larger order that we will return to.

Fragment Two

Of course, for scholars of culture, of gender and of race, dates like 1965 and 1968 have other resonances. For many of us, 1965 might not recall MULTICS but instead the assassination of Malcolm X, the founding of the United Farm Workers, the burning of Watts, or the passage of the Voting Rights Act. The mid-1960s also saw the origins of the American Indian Movement (AIM) and the launch of the National Organization for Women (NOW). The late 1960s mark the 1968 citywide walkouts of Latino youth in Los Angeles, the assassinations of Martin Luther King, Jr. and Robert F. Kennedy, the Chicago Democratic convention with its police brutality, the Stonewall Riots, and the founding of the Black Panthers and the Young Lords. Beyond the geographies of the United States, we might also remember the Prague Spring of 1968, Tommie Smith and John Carlos at the Mexico Summer Olympics, the Tlatelolco Massacre, the execution of Che Guevara, the Chinese Cultural Revolution, the Six-Day War, or May '68 in Paris, itself a kind of origin story for some genealogies of film and media studies. On the African continent, thirty-two countries gained independence from colonial rulers. In the U.S., broad cultural shifts emerged across the decade, as identity politics took root and counter-cultural forces challenged traditional values. Resistance to the Vietnam War mounted as the decade wore on. Abroad, movements against colonialism and oppression were notably strong.

The history just glossed as 'Fragment One' is well known to code junkies and computer geeks. Numerous websites archive oral histories, programming manuals, and technical specifications for MULTICS, UNIX, and various mainframe and other hardware systems. Key players in the history, including Ken Thompson, Donald Ritchie and Doug McIlroy, have a kind of geek-chic celebrity status, and differing versions of the histories of software and hardware development are hotly debated, including nitty-gritty details of what really

counts as "a UNIX." In media studies, emerging work in "code studies" often resurrects and takes up these histories.

Within American, cultural and ethnic studies, the temporal touchstones of struggles over racial justice, anti-war activism, and legal history are also widely recognized and analyzed. Not surprisingly, these two fragments typically stand apart in parallel tracks, attracting the interest and attention of very different audiences located in the deeply siloed departments that categorize our universities.

But Why?

In short, I suggest that these two moments cut from time are deeply interdependent. In fact, they co-constitute one another, comprising not independent slices of history but, instead, related and useful lenses into the shifting epistemological registers driving U.S. and global culture in the 1960s and after. Both exist as operating systems of a sort, and we might understand them to be mutually reinforcing.

This history of intertwining and mutual dependence is hard to tell. As we delve into the intricacies of UNIX and the data structures it embraces, race in America recedes far from our line of vision and inquiry. Likewise, detailed examinations into the shifting registers of race and racial visibility post-1950 do not easily lend themselves to observations about the emergence of object-oriented programming, personal computing, and encapsulation. Very few audiences who care about one lens have much patience or tolerance for the other.

Early forays in new media theory in the late 1990s did not much help this problem. Theorists of new media often retreated into forms of analysis that Marsha Kinder has critiqued as "cyberstructuralist," intent on parsing media specificity and on theorizing the forms of new media, while disavowing twenty-plus years of critical race theory, feminism and other modes of overtly politicized inquiry. Many who had worked hard to instill race as a central mode of analysis in film, literary, and media studies throughout the late twentieth century were disheartened and outraged (if not that surprised) to find new media theory so easily retreating into a comfortable formalism familiar from the early days of film theory.

Early analyses of race and the digital often took two forms, a critique of representations *in* new media, i.e. on the surface of our screens, or debates about access to media, i.e., the digital divide. Such work rarely pushed toward the analyses of form, phenomenology or computation that were so compelling and lively in the work of Lev Manovich, Mark Hansen, or Jay Bolter and Richard Grusin. Important works emerged from both "camps," but the camps rarely intersected. A few conferences attempted to force a collision between these areas, but the going was tough. For instance, at the two Race and Digital Space events colleagues and I organized in 2000 and 2002, the vast majority of participants

and speakers were engaged in work in the two modes mentioned above. The cyberstructuralists were not in attendance.

But what if this very incompatibility is itself part and parcel of the organization of knowledge production that operating systems like UNIX helped to disseminate around the world? Might we ask if there is not something *particular to the very forms* of electronic culture that seems to encourage just such a movement, a movement that partitions race off from the specificity of media forms? Put differently, might we argue that the very structures of digital computation develop at least in part to cordon off race and to contain it? Further, might we come to understand that our own critical methodologies are the heirs to this epistemological shift?

From early writings by Sherry Turkle and George Landow to more recent work by Alex Galloway, new media scholars have noted the parallels between the ways of knowing modeled in computer culture and the greatest hits of structuralism and post-structuralism. Critical race theorists and postcolonial scholars like Chela Sandoval and Gayatri Spivak have illustrated the structuring (if unacknowledged) role that race plays in the work of poststructuralists like Roland Barthes and Michel Foucault. We might bring these two arguments together, triangulating race, electronic culture, and post-structuralism, and, further, argue that race, particularly in the United States, is central to this undertaking, fundamentally shaping how we see and know as well as the technologies that underwrite or cement both vision and knowledge. Certain modes of racial visibility and knowing coincide or dovetail with specific ways of organizing data: if digital computing underwrites today's information economy and is the central technology of post-World War II America, these technologized ways of seeing/knowing took shape in a world also struggling with shifting knowledges about and representations of race. If, as Michael Omi and Howard Winant argue, racial formations serve as fundamental organizing principles of social relations in the United States, on both the macro and micro levels (1986/1989: 55), how might we understand the infusion of racial organizing principles into the technological organization of knowledge after World War II?

Omi and Winant and other scholars have tracked the emergence of a "race-blind" rhetoric at mid-century, a discourse that moves from overt to more covert modes of racism and racial representation (for example, from the era of Jim Crow to liberal colorblindness). Drawing from those 3-D postcards that bring two or more images together even while suppressing their connections, I have earlier termed the racial paradigms of the post-war era "lenticular logics." The ridged coating on 3-D postcards is actually a lenticular lens, a structural device that makes simultaneously viewing the various images contained on one card nearly impossible. The viewer can rotate the card to see any single image, but the lens itself makes seeing the images *together* very difficult, even as it conjoins them at a structural level (i.e. within the same card). In the post-Civil Rights

U.S., the lenticular is a way of organizing the world. It structures representations but also epistemologies. It also serves to secure our understandings of race in very narrow registers, fixating on sameness or difference while forestalling connection and interrelation. As I have argued elsewhere, we might think of the lenticular as a covert mode of the pretense of "separate but equal," remixed for mid-century America (McPherson 2003: 250).

A lenticular logic is a covert racial logic, a logic for the post-Civil Rights era. We might contrast the lenticular postcard to that wildly popular artifact of the industrial era, the stereoscope card. The stereoscope melds two different images into an imagined whole, privileging the whole; the lenticular image partitions and divides, privileging fragmentation. A lenticular logic is a logic of the fragment or the chunk, a way of seeing the world as discrete modules or nodes, a mode that suppresses relation and context. As such, the lenticular also manages and controls complexity.

And what in the world does this have to do with those engineers laboring away at Bell Labs, the heroes of the first fragment of history this essay began with? What's race got to do with that? The popularity of lenticular lenses, particularly in the form of postcards, coincides historically not just with the rise of an articulated movement for civil rights but also with the growth of electronic culture and the birth of digital computing (with both—digital computing and the Civil Rights movement—born in quite real ways of World War II). We might understand UNIX as the way in which the emerging logics of the lenticular and of the covert racism of colorblindness get ported into our computational systems, both in terms of the specific functions of UNIX as an operating system and in the broader philosophy it embraces.

Situating UNIX

In moving toward UNIX from MULTICS, programmers conceptualized UNIX as a kind of tool kit of "synergistic parts" that allowed "flexibility in depth" (Raymond 2004: 9). Programmers could "choose among multiple shells. . . . [and] programs normally provide[d] many behavior options" (2004: 6). One of the design philosophies driving UNIX is the notion that a program should do one thing and do it well (not unlike our deep disciplinary drive in many parts of the university), and this privileging of the discrete, the local, and the specific emerges again and again in discussions of UNIX's origins and design philosophies.

Books for programmers that explain the UNIX philosophy turn around a common set of rules. While slight variations on this rule set exist across programming books and online sites, Eric Raymond sets out the first nine rules as follows:

1. Rule of Modularity: Write simple parts connected by clean interfaces.
2. Rule of Clarity: Clarity is better than cleverness.

3. Rule of Composition: Design programs to be connected to other programs.
4. Rule of Separation: Separate policy from mechanism; separate interfaces from engines.
5. Rule of Simplicity: Design for simplicity; add complexity only where you must.
6. Rule of Parsimony: Write a big program only when it is clear by demonstration that nothing else will do.
7. Rule of Transparency: Design for visibility to make inspection and debugging easier.
8. Rule of Robustness: Robustness is the child of transparency and simplicity.
9. Rule of Representation: Fold knowledge into data so program logic can be stupid and robust. (2004: 13)

Other rules include the Rules of Least Surprise, Silence, Repair, Economy, Generation, Optimization, Diversity, and Extensibility.

These rules implicitly translate into computational terms the chunked logics of the lenticular. For instance, Brian Kernighan wrote in a 1976 handbook on software programming that "controlling complexity is the essence of computer programming" (quoted in Raymond 2004: 14). Complexity in UNIX is controlled in part by the "rule of modularity," which insists that code be constructed of discrete and interchangeable parts that can be plugged together via clean interfaces. In *Design Rules, Vol. 1: The Power of Modularity*, Carliss Baldwin and Kim Clark argue that computers from 1940 to 1960 had "complex, interdependent designs," and they label this era the "premodular" phase of computing (2000: 149). While individuals within the industry, including John von Neumann, were beginning to imagine benefits to modularity in computing, Baldwin and Clark note that von Neumann's ground-breaking designs for computers in that period "fell short of true modularity" because "in no sense was the detailed design of one component going to be hidden from the others: all pieces of the system would be produced 'in full view' of the others" (2000: 157). Thus, one might say that these early visions of digital computers were neither modular nor lenticular. Baldwin and Clark track the increasing modularity of hardware design from the early 1950s forward and also observe that UNIX was the first operating system to embrace modularity and adhere "to the principles of information hiding" in its design (2000: 324).

There are clearly practical advantages of such structures for coding, but they also underscore a world view in which a troublesome part might be discarded without disrupting the whole. Tools are meant to be "encapsulated" to avoid "a tendency to involve programs with each other's internals" (Raymond 2004: 15). Modules "don't promiscuously share global data," and problems can stay "local" (2004: 84–85). In writing about the Rule of Composition, Eric Raymond advises programmers to "make [programs] independent." He writes, "It should

be easy to replace one end with a completely different implementation without disturbing the other" (2004: 15). Detachment is valued because it allows a cleaving from "the particular . . . conditions under which a design problem was posed. Abstract. Simplify. Generalize" (2004: 95). While "generalization" in UNIX has specific meanings, we might also see at work here the basic contours of a lenticular approach to the world, an approach which separates object from context, cause from effect.

In a 1976 article, "Software Tools," Bell Lab programmers Kernighan and Plauger urged programmers "to view specific jobs as special cases of general, frequently performed operations, so they can make and use general-purpose tools to solve them. We also hope to show how to design programs to look like tools and to interconnect conveniently" (1976b: 1). While the language here is one of generality (as in "general purpose" tools), in fact, the tool library that is being envisioned is a series of very discrete and specific tools or programs that can operate independently of one another. They continue, "Ideally, a program should not know where its input comes from nor where its output goes. The UNIX time-sharing system provides a particularly elegant way to handle input and output redirection" (1976b: 2). Programs

> can profitably be described as filters, even though they do quite complicated transformations on their input. One should be able to say
>
> program-1 ... | sort | program-2 ...
>
> and have the output of program-1 sorted before being passed to program-2. This has the major advantage that neither program-1 nor program-2 need know how to sort, but can concentrate on its main task. (1976b: 4)

In effect, the tools chunk computational programs into isolated bits, where the programs' operations are meant to be "invisible to the user" and to the other programs in a sequence (1976b: 5): "the point is that this operation is invisible to the user (or should be). . . . Instead he sees simply a program with one input and one output. Unsorted data go in one end; somewhat later, sorted data come out the other. It must be *convenient* to use a tool, not just possible" (1976b: 5). Kernighan and Plauger saw the "filter concept" as a useful way to get programmers to think in discrete bits and to simplify, reducing the potential complexity of programs. They note that "when a job is viewed as a series of filters, the implementation simplifies, for it is broken down into a sequence of relatively independent pieces, each small and easily tested. This is a form of high-level modularization" (1976b: 5). In their own way, these filters function as a kind of lenticular frame or lens, allowing only certain portions of complex datasets to be visible at a particular time (to both the user and the machine).

The technical feature which allowed UNIX to achieve much of its modularity was the development by Ken Thompson (based on a suggestion by Doug McIlroy) of the pipe, i.e., a vertical bar that replaced the "greater than" sign in

the operating system's code. As described by Doug Ritchie and Ken Thompson in a paper for the Association of Computing Machinery in 1974 (reprinted by Bell Labs in 1978),

> A *read* using a pipe file descriptor waits until another process writes using the file descriptor for the same pipe. At this point, data are passed between the images of the two processes. Neither process need know that a pipe, rather than an ordinary file, is involved.

In this way, the ability to construct a pipeline from a series of small programs evolved, while the "hiding of internals" was also supported. The contents of a module were not central to the functioning of the pipeline; rather, the input or output (a text stream) was key. Brian Kernighan noted "that while input/output direction predates pipes, the development of pipes led to the concept of tools—software programs that would be in a 'tool box,' available when you need them" and interchangeable.[2] Pipes reduced complexity and were also linear. In *Software Tools*, Kernighan and Plauger extend their discussion of pipes, noting that "a pipe provides a hidden buffering between the output of one program and the input of another program so information may pass between them without ever entering the file system" (1976a: 2). They also signal the importance of pipes for issues of data security:

> And consider the sequence
>
> decrypt key <file | prog | encrypt key > newfile
>
> Here a decryption program decodes an encrypted file, passing the decoded characters to a program having no special security features. The output of the program is re-encrypted at the other end. If a true pipe mechanism is used, no clear-text version of the data will ever appear in a file. To simulate this sequence with temporary files risks breaching security. (1976a: 3)

While the affordances of filters, pipes, and hidden data are often talked about as a matter of simple standardization and efficiency (as when Kernighan and Plauger argue that "Our emphasis here has been on getting jobs done with an efficient use of people" (1976a: 6)), they also clearly work in the service of new regimes of security, not an insignificant detail in the context of the Cold War era. Programming manuals and UNIX guides again and again stress clarity and simplicity ("don't write fancy code"; "say what you mean as clearly and directly as you can"), but the structures of operating systems like UNIX function by hiding internal operations, skewing "clarity" in very particular directions. These manuals privilege a programmer's equivalent of "common sense" in the Gramscian sense. For Antonio Gramsci, common sense is a historically situated process, the way in which a particular group responds to "certain problems posed by reality which are quite specific" at a particular time (1971: 324). I am here arguing that, as programmers constitute themselves as a particular class of

workers in the 1970s, they are necessarily lodged in their moment, deploying common sense and notions about simplicity to justify their innovations in code. Importantly, their moment is over-determined by the ways in which the U.S. is widely coming to process race and other forms of difference in more covert registers, as noted above.[3]

Another rule of UNIX is the "Rule of Diversity," which insists on a mistrust of the "one true way." Thus UNIX, in the word of one account, "embraces multiple languages, open extensible systems and customization hooks everywhere," reading much like a description of the tenets of neoliberal multiculturalism if not poststructuralist thought itself (Raymond 2004: 24). As you read the ample literature on UNIX, certain words emerge again and again: modularity, compactness, simplicity, orthogonality. UNIX is meant to allow multitasking, portability, time-sharing, and compartmentalizing. It is not much of a stretch to layer these traits over the core tenets of post-Fordism, a process which begins to remake industrial-era notions of standardization in the 1960s: time–space compression, transformability, customization, a public/private blur, etc. UNIX's intense modularity and information-hiding capacity were reinforced by its design: that is, in the ways in which it segregated the kernel from the shell. The kernel loads into the computer's memory at startup and is "the heart" of UNIX (managing "hardware memory, job execution and time sharing"), although it remains hidden from the user (Baldwin and Clark 2000: 332). The shells (or programs that interpret commands) are intermediaries between the user and the computer's inner workings. They hide the details of the operating system from the user behind "the shell," extending modularity from a rule for programming in UNIX to the very design of UNIX itself.[4]

Modularity in the Social Field

This push toward modularity and the covert in digital computation also reflects other changes in the organization of social life in the United States by the 1960s. For instance, if the first half of the twentieth century laid bare its racial logics, from "Whites Only" signage to the brutalities of lynching, the second half increasingly hides its racial "kernel," burying it below a shell of neoliberal pluralism. These covert or lenticular racial logics take hold at the tail end of the Civil Rights movement at least partially to cut off and contain the more radical logics implicit in the urban uprisings that shook Detroit, Watts, Chicago, and Newark. In fact, the urban center of Detroit was more segregated by the 1980s than in previous decades, reflecting a different inflection of the programmer's vision of the "easy removal" or containment of a troubling part. Whole areas of the city might be rendered orthogonal and disposable (also think post-Katrina New Orleans), and the urban Black poor were increasingly isolated in "deteriorating city centers" (Sugrue 1998: 198). Historian Thomas Sugrue traces the increasing unemployment rates for Black men in Detroit, rates which rose dramatically from the 1950s to the 1980s, and maps a

"deproletarianization" that "shaped a pattern of poverty in the postwar city that was surprisingly new" (1998: 262). Across several registers, the emerging neoliberal state begins to adopt "the rule of modularity." For instance, we might draw an example from across the Atlantic. In her careful analysis of the effects of May 1968 and its afterlives, Kristin Ross argues that the French government contained the radical force of the uprisings by quickly moving to separate the students' rebellion from the concerns of labor, deploying a strategy of separation and containment in which both sides (students and labor) would ultimately lose (2004: 69).

Modularity in software design was meant to decrease "global complexity" and cleanly separate one "neighbor" from another (Raymond 2004: 85). These strategies also played out in ongoing reorganizations of the political field throughout the 1960s and 1970s in both the Right and the Left. The widespread divestiture in the infrastructure of inner cities might be seen as one more insidious effect of the logic of modularity in the post-war era. But we might also understand the emergence of identity politics in the 1960s as a kind of social and political embrace of modularity and encapsulation, a mode of partitioning that turned away from the broader forms of alliance-based and globally-inflected political practice that characterized both labor politics and anti-racist organizing in the 1930s and 1940s.[5] While identity politics produced concrete gains in the world, particularly in terms of civil rights, we are also now coming to understand the degree to which these movements curtailed and short-circuited more radical forms of political praxis, reducing struggle to fairly discrete parameters.

Let me be clear. By drawing analogies between shifting racial and political formations and the emerging structures of digital computing in the late 1960s, I am not arguing that the programmers creating UNIX at Bell Labs and in Berkeley were *consciously* encoding new modes of racism and racial understanding into digital systems. (Indeed, many of these programmers were themselves left-leaning hippies, and the overlaps between the counterculture and early computing culture run deep, as Fred Turner has illustrated.) I also recognize that their innovations made possible the word processor I am using to write this article, a powerful tool that shapes cognition and scholarship in precise ways. Nor am I arguing for some exact correspondence between the ways in which encapsulation or modularity work in computation and how they function in the emerging regimes of neoliberalism, governmentality and post-Fordism. Rather, I am highlighting the ways in which the organization of information and capital in the 1960s powerfully responds—across many registers—to the struggles for racial justice and democracy that so categorized the U.S. at the time. Many of these shifts were enacted in the name of liberalism, aimed at distancing the overt racism of the past even as they contained and cordoned off progressive radicalism. The emergence of covert racism and its rhetoric of colorblindness are not so much intentional as systemic. Computation

is a primary delivery method of these new systems, and it seems at best naïve to imagine that cultural and computational operating systems don't mutually infect one another.

Thus we see modularity take hold not only in computation but also in the increasingly niched and regimented production of knowledge in the university after the Second World War. For instance, Christopher Newfield comments on the rise of New Criticism in literature departments in the Cold War era, noting its relentless formalism, a "logical corollary" to "depoliticization" (2004: 145) that "replaced agency with technique" (2004: 155). He attributes this particular tendency in literary criticism at least in part to the triumph of a managerial impulse, a turn that we might also align (even if Newfield doesn't) with the workings of modular code (itself studied as an exemplary approach to "dynamic modeling systems" for business management in the work of Baldwin and Clark cited above).[6] He observes as well that this managerial obsession within literary criticism exhibits a surprising continuity across the 1960s and beyond. Gerald Graff has also examined the "patterned isolation" that emerges in the university after World War II, at the moment when New Criticism's methods take hold in a manner that de-privileges context and focuses on "explication for explication's sake." Graff then analyzes the routinization of literary criticism in the period, a mechanistic exercise with input and output streams of its own (1989: 227). He recognizes that university departments (his example is English) begin to operate by a field-based and modular strategy of "coverage," in which subfields proliferate and exist in their own separate chunks of knowledge, rarely contaminated by one another's "internals" (1989: 250). (He also comments that this modular strategy includes the token hiring of scholars of color who are then cordoned off within the department.) Graff locates the beginning of this patterned isolation in the run-up to the period that also brought us digital computing; he writes that it continues to play out today in disciplinary structures that have become increasingly narrow and specialized. Patterned isolation begins with the bureaucratic standardization of the university from 1890 to 1930 (1989: 61–62), but this "cut out and separate" mentality reaches a new crescendo after World War II as the organizational structure of the university pushes from simply bureaucratic and Taylorist to managerial, a shift noted as well by Christopher Newfield. Many now lament the over-specialization of the university; in effect, this tendency is a result of the additive logic of the lenticular or of the pipeline, where "content areas" or "fields" are tacked together without any sense of intersection, context, or relation.

It is interesting to note that much of the early work performed in UNIX environments was focused on document processing and communication tools and that UNIX is a computational system that very much privileges text (it centers on the text-based command line instead of on the Graphical User Interface, and its inputs and outputs are simple text lines). Many of the

methodologies of the humanities from the Cold War through the 1980s also privilege text while devaluing context and operate in their own chunked systems, suggesting telling parallels between the operating systems and privileged objects of the humanities and of the computers being developed on several university campuses in the same period.

Another example of the increasing modularity of the American university might be drawn from the terrain of area studies. Scholars including Martin W. Lewis and Kären Wigen have recently mapped the proliferation of area studies from the onset of the Cold War era to the present. They show how a coupling of government, foundations and scholars began to parse the world in finer and finer detail, producing geographical areas of study that could work in the service of the "modernization and development" agenda that was coming to replace the older models of colonial domination, substituting the covert stylings of the post-industrial for the overtly oppressive methods of earlier regimes. By 1958, government funding established university-based "area-studies centers" that grew to "some 124 National Resource Centers [by the 1990s] . . . each devoted to the interdisciplinary study of a particular world region" (Lewis and Wigen 1999: 164). John Rowe has convincingly argued that area studies thrived by operating through a kind of isolationism or modularity, with each area intently focused within its own borders.

Lev Manovich has, of course, noted the modularity of the digital era and also backtracked to early twentieth-century examples of modularity from the factory line to the creative productions of avant garde artists. In a posting to the Nettime list-serve in 2005, he frames modularity as a uniquely twentieth-century phenomenon, from Henry Ford's assembly lines to the 1932 furniture designs of Belgian designer Louis Herman De Kornick. In his account, the twentieth century is characterized by an accelerating process of industrial modularization, but I think it is useful to examine the digital computer's privileged role in the process, particularly given that competing modes of computation were still quite viable until the 1960s, modes that might have pushed more toward the continuous flows of analog computing rather than the discrete tics of the digital computer. Is the modularity of the 1920s really the same as the modularity modeled in UNIX? Do these differences matter, and what might we miss if we assume a smooth and teleological triumph of modularity? How has computation pushed modularity in new directions, directions in dialogue with other cultural shifts and ruptures? Why does modularity emerge in our systems with such a vengeance across the 1960s?

I have here suggested that our technological formations are deeply bound up with our racial formations, and that each undergo profound changes at mid-century. I am not so much arguing that one mode is causally related to the other, but, rather, that they both represent a move toward modular knowledges, knowledges increasingly prevalent in the second half of the twentieth century. These knowledges support and enable the shift from the overt standardized bureaucracies of the 1920s and 1930s to the more dynamically modular and

covert managerial systems that are increasingly prevalent as the century wears on. These latter modes of knowledge production and organization are powerful racial and technological operating systems that coincide with (and reinforce) (post-)structuralist approaches to the world within the academy. Both the computer and the lenticular lens mediate images and objects, changing their relationship but frequently suppressing that process of relation, much like the divided departments of the contemporary university. The fragmentary knowledges encouraged by many forms and experiences of the digital neatly parallel the lenticular logics which underwrite the covert racism endemic to our times, operating in potential feedback loops, supporting each other. If scholars of race have highlighted how certain tendencies within poststructuralist theory simultaneously respond to and marginalize race, this maneuver is at least partially possible because of a parallel and increasing dispersion of electronic forms across culture, forms which simultaneously enact and shape these new modes of thinking.

While the examples here have focused on UNIX, it is important to recognize that the core principles of modularity that it helped bring into practice continue to impact a wide range of digital computation, especially the C programming language, itself developed for UNIX by Ritchie, based on Thompson's earlier B language. While UNIX and C devotees will bemoan the non-orthogonality and leakiness of Windows or rant about the complexity of C++, the basic argument offered above—that UNIX helped inaugurate modular and lenticular systems broadly across computation and culture—holds true for the black boxes of contemporary coding and numerous other instances of our digital praxis.

Today, we might see contemporary turns in computing—neural nets, clouds, semantics, etc.—as parallel to recent turns in humanities scholarship to privilege networks over nodes (particularly in new media studies and in digital culture theory) and to focus on globalization and its flows (in American studies and other disciplines). While this may simply mean we have learned our mid-century lessons and are smarter now, we might also continue to examine with rigor and detail the degree to which dominant forms of computation—what David Golumbia has aptly called "the cultural logic of computation" in his recent update of Frankfurt School pessimism for the twenty-first century—continue to respond to shifting racial and cultural formations. Might these emerging modes of computation be read as symptoms and drivers of our "post-racial" moment, refracting in some way national anxieties (or hopes?) about a decreasingly "white" America? We should also remain alert to how contemporary techno-racial formations infect privileged ways of knowing in the academy. While both the tales of C.P. Snow circa 1959 and the Sokal science wars of the 1990s sustain the myth that science and the humanities operate in distinct realms of knowing, powerful operating systems have surged beneath the surface of what and how we know in the academy for well over half a decade. It would be foolish of us to believe that these operating systems—in this paper best categorized by UNIX and its many close siblings—do not at least partially over-determine the very critiques we imagine that we are performing today.

Moving Beyond Our Boxes

So, if we are always already complicit with the machine, what are we to do?

First, we must better understand the machines and networks that continue to powerfully shape our lives in ways that we are ill-equipped to deal with as media and humanities scholars. This necessarily involves more than simply studying our screens and the images that dance across them, moving beyond studies of screen representations and the rhetorics of visuality. We might read representations seeking symptoms of information capital's fault lines and successes, but we cannot read the logics of these systems and networks solely at the level of our screens. Capital is now fully organized under the sign of modularity. It operates via the algorithm and the database, via simulation and processing. Our screens are cover stories, disguising deeply divided forms of both machine and human labor. We focus exclusively on them increasingly to our peril.

Scholars in the emerging field of "code studies" are taking up the challenge of understanding how computational systems (especially but not only software) developed and operate. However, we must demand that this nascent field not replay the formalist and structuralist tendencies of new media theory circa 1998. Code studies must also take up the questions of culture and meaning that animate so many scholars of race in fields like the "new" American studies. Likewise, scholars of race must analyze, use and produce digital forms and not smugly assume that to engage the digital directly is in some way to be complicit with the forces of capitalism. The lack of intellectual generosity across our fields and departments only reinforces the "divide and conquer" mentality that the most dangerous aspects of modularity underwrite. We must develop common languages that link the study of code and culture. We must historicize and politicize code studies. And, because digital media were born as much of the Civil Rights era as of the Cold War era (and of course these eras are one and the same), our investigations must incorporate race from the outset, understanding and theorizing its function as a "ghost in the digital machine." This does not mean that we should "add" race to our analysis in a modular way, neatly tacking it on, but that we must understand and theorize the deep imbrications of race and digital technology even when our objects of analysis (say, UNIX or search engines) seem not to "be about" race at all. This will not be easy. In the writing of this essay, the logic of modularity continually threatened to take hold, leading me into detailed explorations of pipe structures in UNIX or departmental structures in the university, taking me far from the contours of race at mid-century. It is hard work to hold race and computation together *in a systemic manner*, but it is work that we must continue to undertake.

We also need to take seriously the possibility that questions of representation and of narrative and textual analysis may, in effect, be a distraction from the powers that be—the triumph of the very particular patterns of informationalization evident in code. If the study of representation may in fact be part and parcel of the very logic of modularity that such code inaugurates, a kind of

distraction, it is equally plausible to argue that our very intense focus on visuality in the past twenty years of scholarship is just a different manifestation of the same distraction. There is tendency in film and media studies to treat the computer and its screens as (in Jonathan Beller's terms) a "legacy" technology to cinema. In its drive to stage continuities, such an argument tends to minimize or completely miss the fundamental material differences between cinematic visuality and the production of the visual by digital technologies.

To push my polemic to its furthest dimensions, I would argue that to study image, narrative and visuality will never be enough if we do not engage as well the non-visual dimensions of code and their organization of the world. And yet, to trouble my own polemic, we might also understand the workings of code to have already internalized the visual to the extent that, in the heart of the labs from which UNIX emerged, the cultural processing of the visual via the register of race was already at work in the machine.

In extending our critical methodologies, we must have at least a passing familiarity with code languages, operating systems, algorithmic thinking, and systems design. We need database literacies, algorithmic literacies, computational literacies, interface literacies. We need new hybrid practices: artist-theorists; programming humanists; activist scholars; theoretical archivists; critical race coders. We have to shake ourselves out of our small field-based boxes, taking seriously the possibility that our own knowledge practices are "normalized," "modular," and "black boxed" in much the same way as the code we might study in our work. That is, our very scholarly practices tend to undervalue broad contexts, meaningful relation and promiscuous border crossing. While many of us "identify" as interdisciplinary, very few of us extend that border crossing very far (theorists tune out the technical, the technologists are impatient of the abstract, scholars of race mock the computational, seeing it as corrupt). I'm suggesting that the intense narrowing of our academic specialties over the past fifty years can actually be seen as an effect of or as complicit with the logics of modularity and the relational database. Just as the relational database works by normalizing data—that is by stripping it of meaningful context and the idiosyncratic, creating a system of interchangeable equivalencies—our own scholarly practices tend to exist in relatively hermetically sealed boxes or nodes. Critical theory and post-structuralism have been powerful operating systems that have served us well; they were as hard to learn as the complex structures of C++, and we have dutifully learned them. They are also software systems in desperate need of updating and patching. They are lovely, and they are not enough. They cannot be all we do.

In universities that simply shut down "old school" departments—at my university, German and Geography; in the UK, Middlesex's philosophy program; in Arizona, perhaps all of ethnic studies—scholars must engage the vernacular digital forms that make us nervous, *authoring* in them in order to better understand them and to recreate in technological spaces the possibility of doing the work that moves us. We need new practices and new modes of

collaboration; we need to be literate in emerging scientific and technological methodologies, and we'll gain that literacy at least partially through an intellectual generosity or curiosity toward those whose practices are not our own.

We must remember that computers are themselves encoders of culture. If, in the 1960s and 1970s, UNIX hardwired an emerging system of covert racism into our mainframes and our minds, then computation responds to culture as much as it controls it. Code and race are deeply intertwined, even as the structures of code work to disavow these very connections. Politically committed academics with humanities skill sets must engage technology and its production, not simply as an object of our scorn, critique, or fascination, but as a productive and generative space that is always emergent and never fully determined.

Notes

1 UNIX developed with some rapidity, at least in part because the parent company of Bell Labs, AT&T, was unable to enter the computer business due to a 1958 consent decree. Eric Raymond notes that "Bell Labs was required to license its nontelephone technology to anyone who asked" (2004: 33). Thus a kind of "counterculture" chic developed around UNIX. Eric Raymond provides a narrative version of this history, including the eventual "UNIX wars" in his *The Art of UNIX Programming* (2004). His account, while thorough, tends to romanticize the collaborative culture around UNIX. For a more objective analysis of the imbrications of the counterculture and early computing cultures, see Fred Turner's *From Counterculture to Cyberculture* (2006). See also Tom Streeter (2003) for a consideration of liberal individualism and computing cultures.

2 This quote from Kernighan is from "The Creation of the UNIX Operating System" on the Bell Labs website. See www.bell-labs.com/history/unix/philosophy.html

3 For Gramsci, "common sense" is a multi-layered phenomenon that can serve both dominant groups and oppressed ones. For oppressed groups, "common sense" may allow a method of speaking back to power and of re-jiggering what counts as sensible. Kara Keeling profitably explores this possibility in her work on the Black femme. Computer programmers in the 1970s are interestingly situated. They are on the one hand a subculture (often overlapping with the counterculture), but they are also part of an increasingly managerial class that will help society transition to regimes of neoliberalism and governmentality. Their dreams of "libraries" of code may be democratic in impulse, but they also increasingly support post-industrial forms of labor.

4 Other aspects of UNIX also encode "chunking," including the concept of the file. For a discussion of files in UNIX, see *You Are Not a Gadget* by Jaron Lanier (2010). This account of UNIX, among other things, also argues that code and culture exist in complex feedback loops.

5 See, for instance, Patricia Sullivan's *Days of Hope* (1996) for an account of the coalition politics of the South in the 1930s and 1940s that briefly brought together anti-racist activists, labor organizers, and members of the Communist Party. Such a broad alliance became increasingly difficult to sustain after the Red Scare. I would argue that a broad cultural turn to modularity and encapsulation was both a response to these earlier political alliances and a way to short-circuit their viability in the 1960s. My *Reconstructing Dixie* (2003) examines the ways in which a lenticular logic infects both identity politics and the politics of difference, making productive alliance and relationality hard to achieve in either paradigm.

6 To be fair, Newfield also explores a more radical impulse in literary study in the period, evident in the likes of (surprisingly) both Harold Bloom and Raymond Williams. This impulse valued literature precisely in its ability to offer an "unmanaged exploration of experience" (2004: 152).

Bibliography

Baldwin, Carliss and Kim Clark. 2000. *Design Rules, Vol. 1: The Power of Modularity*, Cambridge, MA: MIT Press.

Beller, Jonathan. 2009. "Re: Periodizing Cinematic Production." Post to IDC Listserve. September 2. Archived at https://lists.thing.net/pipermail/idc/2009-September/003851.html.

Bolter, Jay and Richard Grusin. 2000. *Remediations: Understanding New Media*. Cambridge, MA: MIT Press.

"The Creation of the UNIX Operating System" on the Bell Labs website. Available at: www.bell-labs.com/history/unix/philosophy.html.

Galloway, Alex. 2006. *Protocol: How Control Exists after Decentralization*. Cambridge, MA: MIT Press.

Golumbia, David. 2009. *The Cultural Logic of Computation*. Cambridge, MA: Harvard University Press.

Graff, Gerald. 1989. *Professing Literature: An Institutional History*. Chicago, IL: University of Chicago Press.

Gramsci, Antonio. 1971. *Selections from the Prison Notebooks*. Translated and edited by Q. Hoare and G. Nowell Smith. London: Lawrence and Wishart.

Hansen, Mark B.N. 2000. *Embodying Technesis: Technology Beyond Writing*. Ann Arbor: University of Michigan Press.

Keeling, Kara. 2007. *The Witch's Flight: The Cinematic, the Black Femme, and the Image of Common Sense*. Durham, NC: Duke University Press.

Kernighan, Brian and Rob Pike. 1984. *The Unix Programming Environment*. Englewood Cliffs, NJ: Prentice Hall.

Kernighan, Brian and P.J. Plauger. 1976a. *Software Tools*. Reading, MA: Addison-Wesley.

——. 1976b. "Software Tools." *ACM SIGSOFT Software Engineering Notes* 1.1 (May): 15–20.

Kernighan, Brian and D.M. Ritchie. 1978. *The C Programming Language*. Englewood Cliffs, NJ: Prentice Hall. Second edition 1988.

Kinder, Marsha. 2002. "Narrative Equivocations Between Movies and Games," in Dan Harries, ed., *The New Media Book*, London: BFI.

Landow, George. 1991. *Hypertext: The Convergence of Contemporary Critical Theory and Technology*. Baltimore, MD: Johns Hopkins University Press.

Lanier, Jaron. 2010. *You Are Not A Gadget: A Manifesto*. New York: Knopf.

Lewis, Martin W. and Kären Wigen. 1999. "A Maritime Response to the Crisis in Area Studies." *The Geographical Review* 89.2 (April): 162.

McPherson, Tara. 2003. *Reconstructing Dixie: Race, Place and Nostalgia in the Imagined South*. Durham, NC: Duke University Press.

Manovich, Lev. 2002. *The Language of New Media*. Cambridge, MA: MIT Press.

——. 2005. "We Have Never Been Modular." Post to Nettime Listserve, November 28. Archived at www.nettime.org/Lists-Archives/nettime-l-0511/msg00106.html.

Newfield, Christopher. 2004. *Ivy and Industry: Business and the Making of the American University, 1880–1980*. Durham, NC: Duke University Press.

Omi, Michael and Howard Winant. 1986/1989. *Racial Formation in the United States: From the 1960s to the 1980s*. New York: Routledge.

Raymond, Eric. 2004. *The Art of UNIX Programming*. Reading, MA: Addison-Wesley. 2004.

Ritchie, Dennis. 1984. "The Evolution of the Unix Time-sharing System," *AT&T Bell Laboratories Technical Journal* 63.6 (2): 1577–1593. Available at: http://cm.bell-labs.com/cm/cs/who/dmr/hist.html

Ritchie, D.M. and K. Thompson. 1978. "The UNIX Time-Sharing System." *The Bell System Technical Journal* 57.6 (2, July–August).

Ross, Kristin. 2004. *May '68 and Its Afterlives*. Chicago: University of Chicago Press.

Rowe, John Carlos. Forthcoming. "Areas of Concern: Area Studies and the New American Studies," in Winfried Fluck, Donald Pease, and John Carlos Rowe, eds, *Transatlantic American Studies*. Boston: University Presses of New England.

Salus, Peter H. 1994. *A Quarter-Century of Unix*. Reading, MA: Addison-Wesley.

Sandoval, Chela. 2000. *Methodology of the Oppressed*. Minneapolis: University of Minnesota Press.

Spivak, Gayatri. 1987. *In Other Worlds: Essays in Cultural Politics*. New York: Routledge.

Streeter, Thomas. 2003. "The Romantic Self and the Politics of Internet Commercialization." *Cultural Studies* 17.5: 648–668.

Sugrue, Thomas J. 1998. *The Origins of the Urban Crisis: Race and Inequality in Post-War Detroit*. Princeton: Princeton University Press.

Sullivan, Patricia. 1996. *Days of Hope: Race and Democracy in the New Deal Era*. Chapel Hill, NC: UNC Press.

Turkle, Sherry. 1997. *Life on the Screen: Identity in the Age of the Internet*. New York: Simon and Schuster.

Turner, Fred. 2006. *From Counterculture to Cyberculture: Stewart Brand, the Whole Earth Network, and the Rise of Digital Utopianism*. Chicago: University of Chicago Press.

Race After the Internet

Edited by
Lisa Nakamura
and
Peter A. Chow-White

Routledge
Taylor & Francis Group
NEW YORK AND LONDON