Sub setPosition(newx, newy, newz)
    setX newx
    setY newy
    setZ newz
End Sub

Sub rMove(deltax, deltay, deltaz)
    setPosition (x + deltax), (y + deltay), (z + deltaz)
End Sub

Sub walkVisible(step)
    render
    RandomizeDirection(randomizeRotation)
    NormalizeDirection(direction)
    setPosition (x + (step*direction(0))), (y + (step*direction(1))), (z + (step*direction(2)))
End Sub

Sub walk(step)
    RandomizeDirection(randomizeRotation)
    NormalizeDirection(direction)
    setPosition (x + (step*direction(0))), (y + (step*direction(1))), (z + (step*direction(2)))
End Sub

Sub walkLine(step)
    dim oldPosition
    oldPosition = array(x, y, z)
    RandomizeDirection(randomizeRotation)
    NormalizeDirection(direction)
    setPosition (x + (step*direction(0))), (y + (step*direction(1))), (z + (step*direction(2)))
End Sub

Function SeekConstrainedBuildings(arrBBox, arrObjects, arrSeeds)
    SeekConstrainedBuildings = VbFalse
    dim arrSeed, i, currentDistance, intCountConnections,
    if Not blnMove Then Exit Function
    For i=0 To NumberOfSteps
        If blnMove Then Exit For
        ConstrainToBBox(arrBBox)
        If blnVisible Then
            walkVisible(step)
        Else
            walk(step)
        End If
    Next
    intCountConnections = 0
    For Each arrSeed In arrSeeds
        currentDistance = Rhino.Distance (array(x, y, z))
        if currentDistance<radius Then
            blnAllow = VbTrue
        For Each strObject In arrObjects
            If Rhino.IsPointInSurface (strObject, blnAllow) Then
                exit for
            Exit If
        Next
        If blnAllow Then
            If intCountConnections<intConections The
                strLine = rhino.addLine (array(x, y, z), arrDivPts = Rhino.DivideCurveLength (s)
                If IsArray(arrDivPts) Then
                    For Each arrDivPt In arrDivPts
                        For Each strObject In arrObjects
                            If Rhino.IsPointInSurface (s) blnAllow = VbFalse
                        End If
                    Next
                End If
            End If
        End If
    Next
End Function

Acconci Studio, Script for Landmark Wales ’07.
Into a Forest Of Script

Today's form of the obsessional question "Am I alive or dead?" is "Am I a machine (does my brain really function as a computer) or a living human being (with a spark of spirit or something else that is not reducible to the computer circuit)"

— Slavoj Žižek

What could be more rational than a computer program? Software seems, at first glance, to be a methodical means of accomplishing complex but still clearly definable tasks. Being digital, and therefore ultimately translatable into unambiguous electronic signals, computer programs appear to be instruments of control, themselves controllable and their effects fully describable. But there is another, more shadowy side to computer code, an aspect that it sometimes shows when it appears in contemporary culture. The showering green glyphs in The Matrix movies, for example, are the medium of a vast communal hallucination force-fed to humans in lieu of conscious thought. In William Gibson's novel Pattern Recognition, characters who have lost loved ones use the mediation of software to work out the pain and repressions of their grief. And in the real-life hysteria of Y2K, the specter of code run amok, possessed by unseen bugs, was expected to induce a digital apocalypse. In these instances, computer code is seen to frustrate its creators' intentions and interfere, for good or ill, with human thought and feeling.

This aspect of software was suggested as early as 1964 in Dennis Crompton's visionary "Computor City," published in Archigram 5. Wires crisscross in space and a few enigmatically labeled conduits weave through the network, but the axonomic image gives little evidence of human habitation. Unrolling along the top of the drawing, a string of commands referencing infrastructures of transportation and construction, birth and death rates, commerce, and leisure presages the "thought process" of a powerful digital "mind" that is silently monitoring and controlling the city. Its ultimate purposes are unknown, however, and the cryptic text extends off the page as if to show that the system exceeds rational human understanding and control.

Today, as the direct manipulation of computer programming languages becomes a reality for experimental architec-

tural practices, we might expect to find the “rational” behavior of software haunted by a similar capacity to overflow its context and give rise to the unanticipated. The recent “Scriptedbypurpose” exhibition in Philadelphia2 showcased a number of visionary projects generated with the aid of computer algorithms. Sure enough, many of these script-generated projects could plausibly be read as expressing anxieties and fantasies associated with the computerization of the contemporary environment: dizzying cybernetic landscapes may reflect unease about the proliferation of code-based machines; or perhaps they indicate a god fantasy, evoking a digital garden generated by the fiat of an architectural demiurge that initiates self-perpetuating processes of growth and evolution.

But such iconographic readings of code-generated architecture do little to illuminate how the software functions in these projects. More interesting for those who would analyze the operation of algorithms is the stipulation by the curators of “Scriptedbypurpose” that all scripts be displayed along with the image boards and fabricated objects, effectively asserting that, as a component of the architectural text (which might also include drawings, models, diagrams, and buildings), this new form of notation is not simply a transparent means of access to architectural form, but also imposes its own conceptual framework and methodological constraints like any other representational medium. Scripting has emerged as the newest stage in a progression of attempts to free architecture from the hegemony of drawing by devising ever-more-abstract machinic processes for generating form. This effort took off with the diagrammatic houses of Peter

Eisenman in the 1970s, which subordinated drawings to orthographic diagrams, and continued in the '90s as Greg Lynn used digitally modeled force fields to replace static topological configurations with complex, distorted topological entities. As these efforts now progress to the manipulation of code itself, one effect has been a shift from an architecture conceived as externally imposed form to one in which the work seems to acquire a vitality of its own, the turgid contours of the design (if we can still call it that) emerging from a process in which the architect's role as author appears more laissez-faire.

The usefulness of a computer algorithm might seem to lie in its being universally comprehensible and executable. In practice, however, these characteristics are at best local and limited. Even experienced programmers find it difficult to work with someone else's poorly documented code. Moreover, scripts are only high-level instructions that must be interpreted by another software program, itself comprising millions of lines of code, and translated into a lower-level machine code to be executed by the operating system. These layers of software are each in flux: for example, the specifications of scripting languages change as new functions are added and obsolete ones deprecated. Inevitably, sections of this enormous, multitiered code base are recycled from previous versions, forcing programmers to struggle against software rot, the gradual deterioration of functionality as sporadic updating causes components to interact in unintended ways. Considering the maddening problems that can arise from trying to open design files on a computer with even a slightly different configuration of software, it is no exaggeration to say that in just a few years' time, it will be impossible to find the requisite equipment to execute many of today's scripts. Thus, despite the functional transparency that their open-source presentation might suggest, architectural scripts seem to lack technology's basic property of reliability and allow countless opportunities for the unexpected to surface, arising in hidden subroutines to interfere with the design process in the form of emergent behaviors and organizations. And yet, if the notion of machine code as fully rational and predictable seems to have gone the way of modernism's promised mechanized utopia, this is by no means necessarily a bad thing: the unpredictability of digital modeling software, with its often quirky surprises, is arguably a necessary condition of its usefulness to designers.

Emergent spatial organization is a theme of the Landmark Wales '07 design of Acconci Studio, a concept for
a city suspended over a river on “stilts.” The rendered images portray a pseudobiological milieu of organically curved surfaces rippling through space, covered with tiny cilia that would generate energy from the wind. According to the project text, the city “will be like a flock of flying birds, where there’s no leader, where each bird knows what to do without being told,” these rule-based interactions resulting in the emergence of unpredictable patterns. “Never mind the metaphors,” the text continues. “The method of the future city will come from physics, mathematics, computer software.”¹ How, the viewer might protest, could something as chaotic as a city emerge from the rational operation of a computer program?

It is instructive to compare this work to the performance and video artworks made by Vito Acconci in the 1970s, based on a different kind of “scripting,” in which actions and desires were iterated obsessively and transposed onto a video screen, where they became irrational, taking the form of unfulfillable fantasies, self-mutilation, and other situations that probed the limits of personhood. The psychological power of a work like Theme Song (1971), for example, comes from the instant feedback of video: staring at the camera, Acconci repeats his invitation to “come in here with me,” but it is unclear whether he is addressing the viewer or his own image reflected in a monitor. Subject and object are effectively collapsed into one another, setting up a condition, externalized in the paraphernalia of the work, of narcissism. The technology of camera and monitor is so essential to these artworks as to become part of their psychological structure.⁴

The script-generated architecture of Acconci’s latter-day practice differs substantially from his video art, but here too technology could be viewed as externalizing a psychological condition. The human geographer Nigel Thrift has studied how, through the mediation of software, new kinds of social relationships are guaranteed while other situations now leave room for unexpected emergent happenings. By anticipating and second-guessing our decisions about how we occupy our world, Thrift argues, networked computer technology conditions the phenomenal space of our everyday life to such a degree that it constitutes a “technological unconscious.” In Thrift’s account, the boundary between conscious human organism and inorganic environment is today blurred by “a prepersonal substrate” consisting of real-time digital flows.⁵ A script, then, might be understood as the infrastructure of a designer’s thinking, and thus, in some sense, part of the apparatus of the unconscious.

1. Acconci Studio, Landmark Wales ’07 project text from “Scripted by purpose” exhibition.
To understand how this theory helps to analyze script-generated architecture, consider the Grotto project of Aranda/Lasch, a proposal for a temporary summer pavilion consisting of digitally formed boulders arranged by a tiling program. The architects devised a script based on Voronoi tiling to create an organizational system that, in their words, "defies a conventional sense of order." In their book Tooling, Benjamin Aranda and Chris Lasch propose a catalogue of virtual rule systems that precede form, codified thought processes that can be modified, repurposed, and concatenated into larger programs. In the case of open-source scripts, a code object becomes a collective palimpsest of many different programmers' mental states over time. To execute these cognitive "recipes" is not possible for the human mind alone, but requires the agency of computers to iterate their formal operations to a degree at which (conscious) formalism is no longer apparent. The manner in which their Grotto rejects apparently rational organization while being, in fact, highly ordered is linked with the designers' unspoken biases and desires: according to Aranda and Lasch, "Shapes are never unwilled figures. Deep within them is a struggle between the predilections of the architect and the inherent properties of the geometries encountered." The intervening layer that mediates between the architect's instincts and the project's geometric and material constraints turns out to consist of code.

Anthony Vidler has linked Greg Lynn's attempts to overcome conceptual oppositions in architectural discourse through the fluid effects of digital modeling to the surrealist
informe, a term Georges Bataille deployed not as a concept but as an “operation” that aimed in various ways to dismantle the structural logic of modernist form. Yet Vidler acknowledges that biomorphic architecture “is in fact highly formalized... What is being sought is not so much a non-formal outlet... but more a rejection of the formal-dialectical method... in favor of an all-subsuming ‘solution’ in formal continuity.”7 The manipulation of code today, then, might best be characterized as advancing the effort begun by Lynn, not in the direction of Bataille’s informe—whose proneness toward base materiality was never part of Lynn’s project—but rather toward a kind of hyperformalism, in which digital technology allows formal games to be played out beyond the capacity of the human consciousness to discern or read formal order. We can regard this architecture as a species of automatic writing, in which the designer’s partial “letting go” of conscious aesthetic control allows complex forms to emerge from the hidden operation of the algorithm, like the oneiric figures produced by the unconscious through the métonymic short-circuits of free-association.

If scripting is akin to automatic writing, the “automaton” appears explicitly in the Olzweg project of R&Sie(n), a competition entry in which the courtyard of a French contemporary art museum would become the site of a labyrinth of recycled glass bars assembled by a construction robot. Never settling into a fixed configuration, the bars would be reshuffled perpetually by the robot in response to an ongoing script. Visitors, their conventional spatial awareness com-

pletely frustrated, would be tagged and tracked via RFID (radio frequency identification) technology to keep them from getting lost in the landscape’s fluctuation. The title of the design, from the German term *holzweg*, refers to rough paths leading through a forest to a clearing. Heidegger uses the term to describe the provisionality of our ways of thinking about the world. Here, the biotic metaphor reflects the flux of code-based cognition in which access is continuously rerouted, refusing to congeal into clear patterns of organization. (The condition of perpetual flux is taken even farther in R&Sie(n)+D’s visionary project *I’ve Heard About*, published in *Log* 10, a megadwelling under continuous modification by a robot that converts the “pre-psychic” chemical stimuli of residents into constructive impulses, literally replacing the conscious will of an individual form-giver with a collective unconscious, politically mediated by code.)

The manner in which scripts interact with the human psyche in these architectural examples perfectly illustrates the enigmatic place of computer code in the contemporary mythos. Despite the waning of the late 20th-century cyberculture fetish, software is more culturally pervasive than ever. The literary theorist N. Katherine Hayles has explored the peculiar status of code as a form of asemiotic writing that circulates beneath the surface of human language, appearing at moments when software acts outside our conscious intentions, just as the unconscious reveals itself through telling slips of the tongue. Hayles posits that with human behavior increasingly caught up in tangles of software, the emergence of computer code as a literary and artistic theme signals an ambivalence about the widespread computerization of the environment and of cognition.8 The cultural objects that explore these anxieties related to code are themselves highly dependent on software for their creation and dissemination. To understand its role in contemporary architecture, then, we must accept that code is far from a rational and transparent instrument subject to complete formalization, but can take on a life of its own as the infrastructure of collective thinking, remembering, and imagining.

Of course, the sense in which software functions as a cybernetic extension of human consciousness is a far cry from the Freudian unconscious. Still, it is remarkable how the efforts of biomorphic architecture to elide the negations of critical discourse with an organismic of virtual forces and flows, morphing and crossbreeding the most disparate typologies into one another, align with Freud’s account of how dreams create “new surfaces” out of logically contra-

dictory ideas by juxtaposing their fragments in time and space, distorting them through psychical processes such as condensation and displacement, and smoothly “combining the whole material into a single situation.”9 Along these lines, writers such as Caroline O’Donnell have linked the diagramming strategies of contemporary architects with Freud’s diagrams of the virtual energies of the unconscious, suggesting a mode of architectural analysis based not so much on discovering represented meanings as on putting “the building on the couch.”10 It is no coincidence that Freud describes the medium of such unconscious processes as a shadowy kind of writing. Derrida finds particular significance in Freud’s analogy of a “Mystic Writing Pad,” which secretly preserves the inscriptions erased from its visible surface, to figure the gap of representation that frustrates attempts at dream interpretation based on a one-to-one correspondence of signifier and signified. In the unending circulation of signs that constitutes the unconscious, “dreams [are] displaced into a forest of script.”11

Just as dreams underscore for consciousness that it is not the whole of mind, the hyperformalist emergences of algorithmically generated architecture suggest that human systems of representation have an inscrutable counterpart in computer code. In many contemporary design practices, software functions as a supplement to architects’ conscious intentions; the resulting environments accord with Vidler’s observation that posthumanist architectural space is “profoundly indifferent to our presence.”12 The apparent indifference of script-generated forms (with their frequently “viral” aesthetics) points to a contamination of human semiotics by a kind of writing that escapes it — and by whose agency the human subject, looking for its reflection in architecture, discovers that it is not fully natural.