30
REGULARITY

REPETITION IN SPACE
THE GRID IN MODERN ART
THE COMPUTER SCREEN
REPETITION IN TIME
REPETITION IN PROCESS
PERFORMING THE LOOP

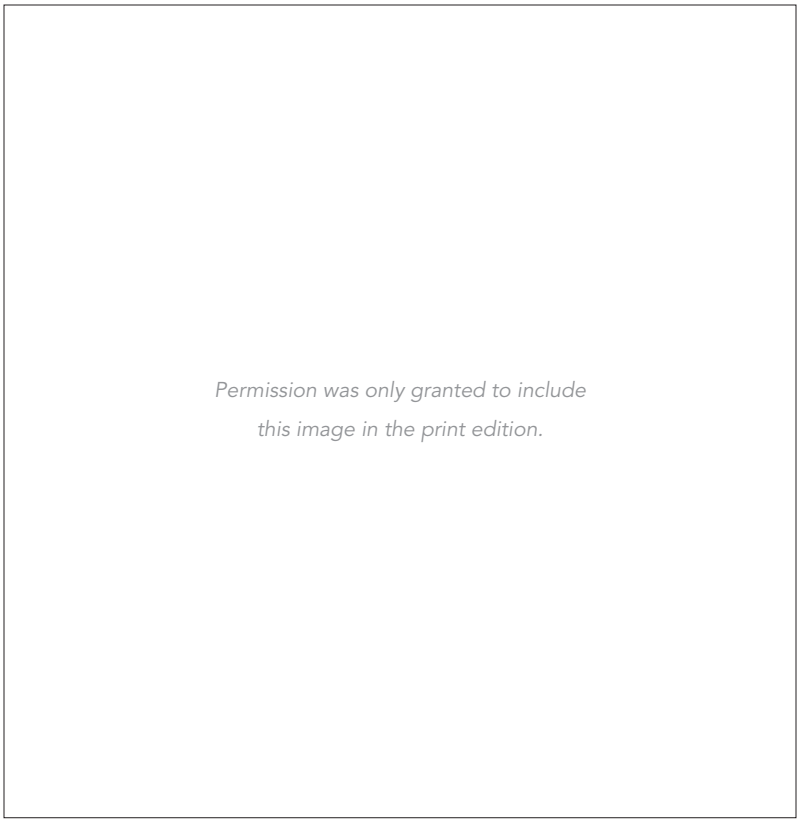

Figure 30.1
Vera Molnar, *Untitled (Quatre éléments distribués au hasard)*. Collage on cardboard, 1959, 75 × 75 cm. Paris, Centre Pompidou-CNAC-MNAM. © bpk | CNAC-MNAM | Georges Meguerditchian.

In 1959 artist Vera Molnar created *Untitled (Quatre éléments distribués au hasard),* a collage similar to `10 PRINT` (figure 30.1). A variant of the `10 PRINT` program shipped with the first Commodore 64s in 1982 (figure 30.2). And in 1987, Cyril Stanley Smith more or less recreated `10 PRINT`'s output from a reduced, random arrangement of Truchet tiles (figure 30.3). How did the same essential mazelike pattern come to appear in all of these different contexts in the twentieth century?

```
{64}   10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

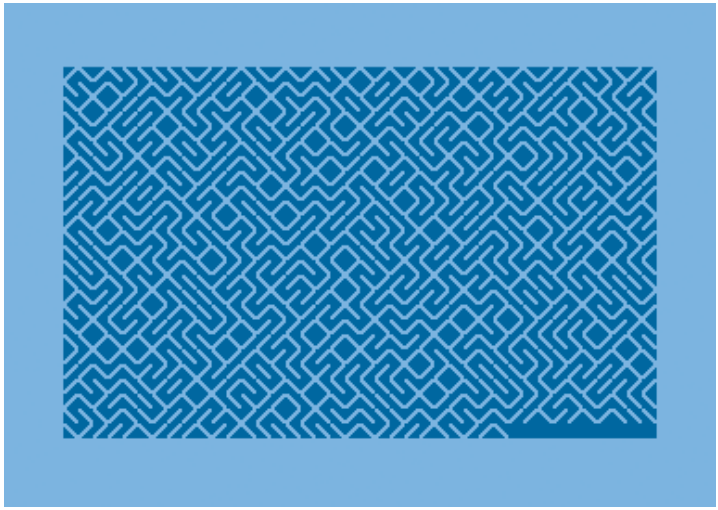

Figure 30.2
Random maze program from the *Commodore 64 User's Guide*, 1982.

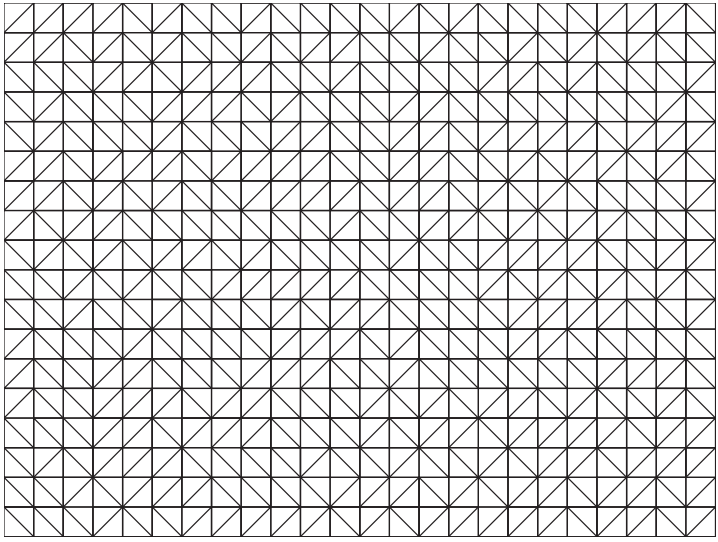

Figure 30.3
Truchet's four tiles placed in random orientations by Cyril Stanley Smith in 1987. The solid coloring was removed to show the formal connection to the `10 PRINT` pattern.

The repetitions of the 10 PRINT process are connected to two categories of artistic tradition and to the flow of control in computer programs. The first tradition within the arts is in the domain of craft, particularly pattern-based crafts such as needlework and ornamental design. The second is the creation of complex patterns using repeated procedures and a small number of elements. In this way, the aesthetic of 10 PRINT parallels experiments in painting, sculpture, sound composition, video art, performance, experimental animation, and dance. In both cases, these artistic practices owe their success to factors that also make 10 PRINT compelling: the continual repetition of a simple rhythmic procedure or rule across a regular space or time signature creating a complex and stimulating gestalt. In its minimalist and constructivist strains the world of art confronts the constraints and regularity of the *technē* of programming, which makes room for a formal definition of a repeating process that a computer can carry out. In all of its newfangled (for the 1980s) sophistication, 10 PRINT ties the computer to the homespun tradition of handicraft: stitching, sewing, and weaving.

This intersection of design craft, art, and computation is not accidental, for 10 PRINT is a demonstration of the generative qualities of repeated procedure. 10 PRINT was written and published at a time when the art world was turning to explore the constraints and possibilities of the systematization of creativity in an age of Taylorism and Fordism, of which the computational machine is itself an expression. Situating 10 PRINT not only within twentieth-century art, but also in the larger traditions of formal experimentation and craft culture can help to explain how the personal computer is a site of procedural craft.

This chapter explores the first of two formal aspects of the 10 PRINT program that give it its compelling visual power. This chapter focuses on regularity, while the next one deals with randomness. Although the pattern of 10 PRINT cannot be established at a glance, the program is nothing if not regular. It works regularly in space, time, and process—and each of these aspects of regularity is examined in the discussion that follows. Spatial regularity is considered, beginning with tilings, continuing through the history of the grid, and ending with a discussion of the computer screen. Artistic repetition in time, particularly in music and performance, is considered next. Then, repeating processes and the programming constructs that support them are discussed.

REPETITION IN SPACE

In a classic, provocative text, *The Sense of Order*, E. H. Gombrich (1994) wrestles with the tensions between pleasing repetition and uninteresting redundancy. As he reflects on pavement designs he notes the pleasure in encountering one whose pattern cannot be fully grasped. Gombrich explains this desire for variation or complexity in terms of the information theory emerging at the time, which posits that information increases in step with unpredictability (9). He goes on to speculate that the viewer examines patterns by trying to anticipate what comes next. "Delight," he writes, "lies somewhere between boredom and confusion" (9). Consider, again, the Labyrinth at Chartres as one such balance of the two.

`10 PRINT` no doubt offers similar delights, thanks to its creation of a complex pattern from a simple random alternation. As Gombrich later argues, the greatest novelties computers bring to visual design and variation are not only their ability "to follow any complex rule of organization but also to introduce an exactly calculated dose of randomness" (1994, 94). In this view, computers prove to be entrancing weavers, and the design of `10 PRINT`, as a work of pattern rather than paths, may be less like the work of Daedalus than that of Arachne.

Patterns are inextricably tied to a process of repetition. This notion is clearly demonstrated in Gombrich's commentary on "the hierarchical principle" by which units are "grouped to form larger units, which in turn can easily fit together into larger wholes" (1994, 8), or a *gestalt*. The sum of the pattern then is the result of a process. This interrelationship of pattern, perceived whole, and process becomes clear in his discussion of paving and of various methods for selecting stones. By extension, visual design relies on the process of repeating patterns across space, even if these patterns are not drawn as individual units. The regulated backdrop or foundation of these orderly patterns in Euclidean space is the grid.

The grid provides a framework within which human intuition and invention can operate and that it can subvert. Within the chaos of nature, regular patterns provide a contrast and promise of order. From early patterns on pottery to geometric mosaics in Roman baths, people have long used grids to enhance their lives with decoration. In Islamic culture, the focus on mathematics and prohibition on representational images led to the most advanced grid systems of the time, used to decorate buildings and

religious texts. Grids have also long been used as the basis for architecture and urban planning. For example, it is impossible to imagine New York, the one-time city of the future, without the regular grid of upper Manhattan. (Broadway breaks this grid in ways that form many of the city's most notable public spaces.) The grid is also the basis for our most intellectual play, from chess to go, whether the design submits to or reacts against it.

The grid has proved essential to the design of computers from the grid of vacuum tubes on the ENIAC (1946) to the latest server farms that feed data to the Internet. A new era of more reliable computing was spawned in the 1950s by a grid of ferrite rings called core memory (figure 30.4). This technology works by addressing each ring on the grid to set its charge to clockwise or counterclockwise to store one bit of information. Because the information is stored as a magnetic force, it maintains its state with or without power. The grid is an essential geometry of computation.

The two-dimensional regularity of the grid is essential to the impact of `10 PRINT`, as removing a single character from the program reveals. Taking out the semicolon that indicates that each character should be drawn immediately to the right of the previous one, the symbol that wraps the program's output continually rightward across the screen, makes the importance of the grid clear (see figure 30.5):

```
10 PRINT CHR$(205.5+RND(1)) : GOTO 10
```

As a column of diagonal lines, the output does not form a maze and the vibrant pattern that encourages our eyes to dance across the screen is not established (figure 30.5). The essential process of `10 PRINT` in time is a single, zero-dimensional coin flip to pick one of two characters; when this recurs in time, it becomes a one-dimensional stream of diagonal lines that either flows quickly down the left side (if the semicolon is omitted) or moves right to wrap around to the next position below the current line and to the left. The visual interest of this program results from wrapping this one-dimensional stream of tiles into the two-dimensional grid.

Truchet Tiles

Imagine the diagonal character graphics in `10 PRINT` are painted on a set of square ceramic tiles, of the sort used for flooring. Each tile is painted

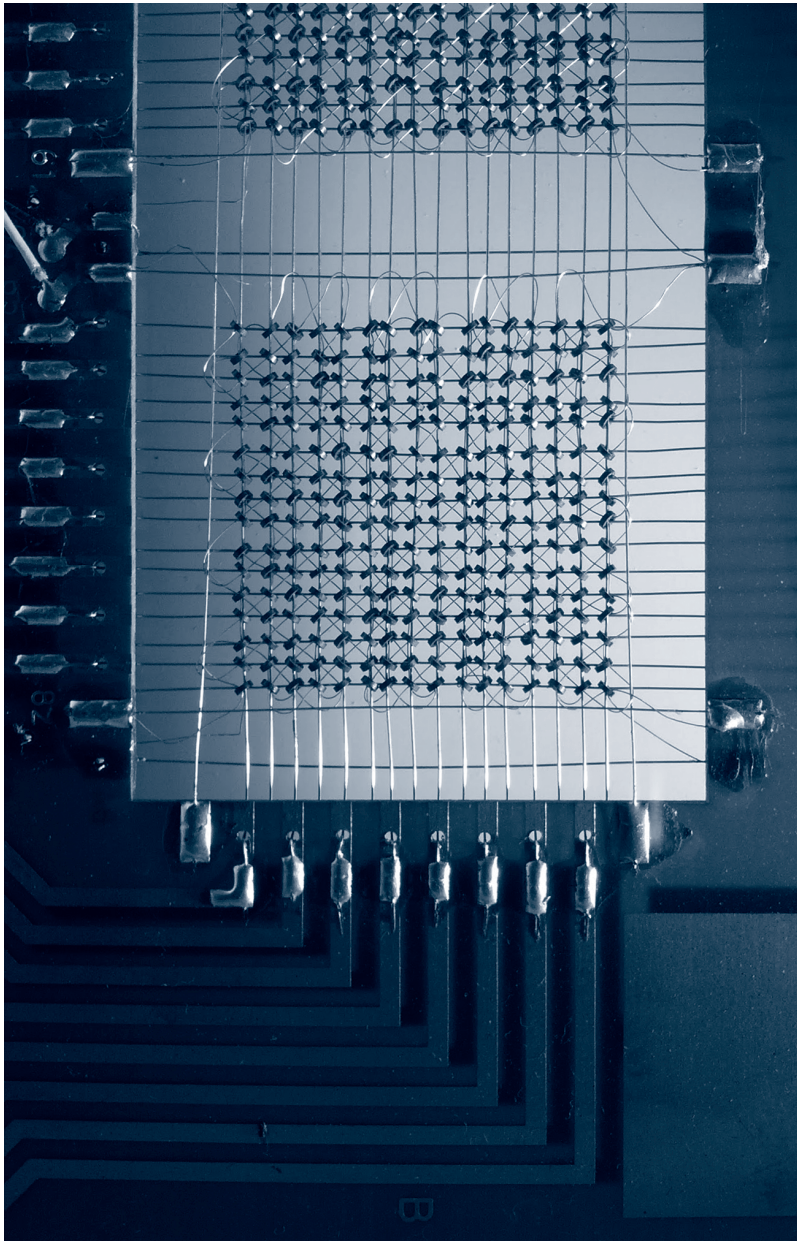

Figure 30.4
Magnetic core memory.

Figure 30.5
This screen capture from the `10 PRINT` variation without the semicolon shows the importance of the two-dimensional grid as a defining characteristic of the program.

with a black diagonal line dividing two white triangles. A tile can be rotated in two orientations, so that the diagonal line appears to be a backslash or a forward slash. Now imagine painting one of the two triangles black. Each tile can now be rotated in four different orientations, like a black arrow pointing at each of four corners. Repeatedly placing tiles down in the same orientation will create a pattern (figure 30.6). Two tiles can be placed next to each other to create one of sixteen unique formations, and laying down any such pair repeatedly will again produce patterns. Indeed, any unique grouping of tiles (whether 2 × 1, 4 × 4, etc.) can serve as a building block for larger regularity.

Now, imagine a whole floor or tapestry covered with a regular pattern of these repeating tiles. This thought exercise suggests the power of the Truchet tile, so named because the Dominican priest Sebastien Truchet first described what he called the "fecundity of these combinations" in 1704, after experimenting with some ceramic tiles he came across at a building site for a château near Orléans (Smith and Boucher 1987, 374).

Matching a single Truchet tile with another, and another, and another,

and so on, a designer is able to create an incredible array of patterns. The interplay between the direction of each tile and the varying repetition of black and white—of positive and negative—produces symmetrical designs that can range from grid-like patterns to mesmerizing, almost three-dimensional illusions. Unlike earlier, Islamic patterns or Celtic designs, which both relied on multiple-sized shapes, the Truchet tile uses only a single size and a single shape (Smith and Boucher 1987, 378). In his original 1704 essay, Truchet provides examples of thirty different patterns, barely evoking the aesthetic possibilities of his tiles, though he notes that he "found too great a number to report them all" (374). Truchet's work would be the inspiration for a later book, Doüat's modestly named *Methode pour faire une infinite de desseins differents . . .* [*Method for Making an Infinity of Different Designs . . .*] which in turn had a considerable impact on eighteenth-century European art (373).

Yet all of Truchet's and Doüat's examples are regular patterns, symmetrical and repetitive. The historian of science Cyril Stanley Smith observed in 1987 that even more compelling designs can be generated from Truchet tiles if dissymmetries are introduced. What happens when the regularity of a Truchet pattern is interrupted by randomness? Smith provides one example, a block of Truchet tiles arranged at random (figure 30.3). The lattice of the basic grid is still visible, but randomness has made its mark, leaving imperfections that disrupt any nascent pattern. Unlike the symmetrical examples Truchet and Doüat give, there is no resolution to the structure. The center cannot hold, and neither can the margins. Smith next pushes the limits of the Truchet tiles' regularity by omitting solid coloring from the tiles, leaving only the black diagonal line. The four possible orientations of any given tile are then reduced to two.

These modified Truchet tiles generate a design that looks unmistakably like the output of `10 PRINT`, a program published a half decade before Smith and Boucher's article. The grid still remains—indicating the edges of each tile—but the diagonals no longer seem to bound positive or negative space. Instead, they appear to be the walls of a maze, twisty little passages, all different. In this Truchet tile-produced artifact the dynamic between regularity and its opposite come into play, suggesting that regularity is not an aspect of design that exists in isolation, but rather can only be defined by exceptions to it, by those moments when the regular becomes irregular. Rather than celebrating that `10 PRINT` "scooped" Smith,

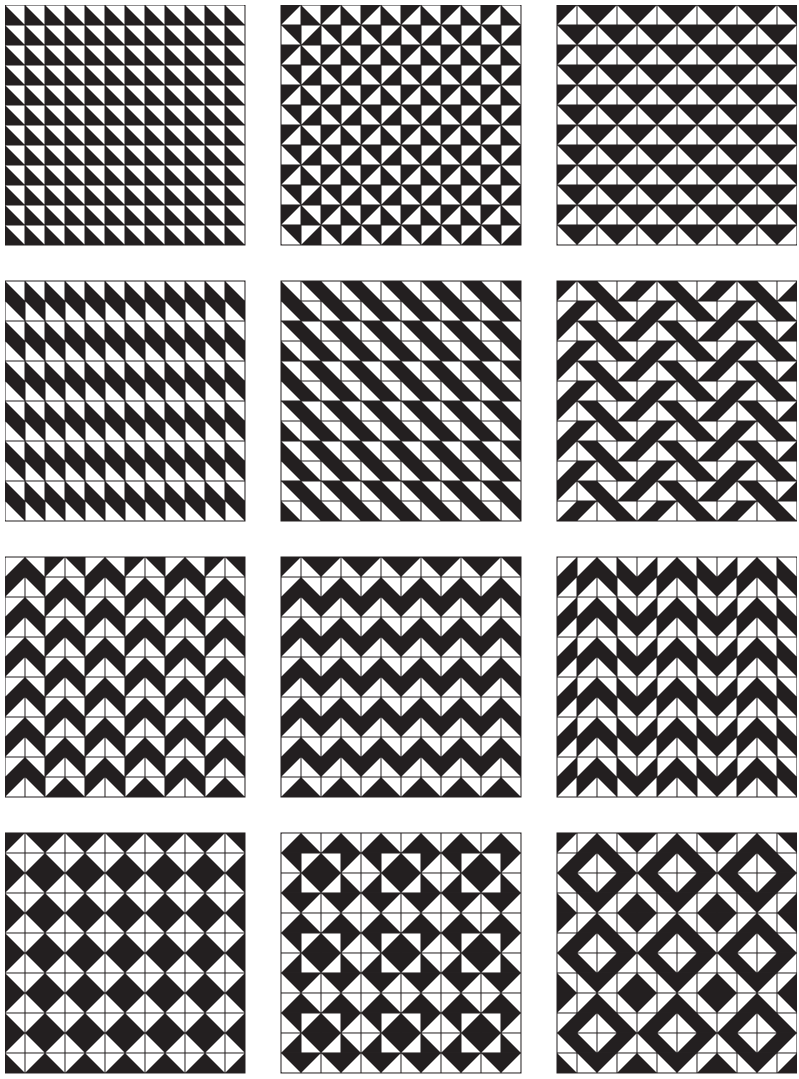

Figure 30.6

Patterns from Sébastien Truchet's "Mémoire sur les combinaisons," 1704. Each 12 × 12 pattern redrawn above is constructed from smaller patterns using one tile design, half black and half white cut across the diagonal.


```
10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

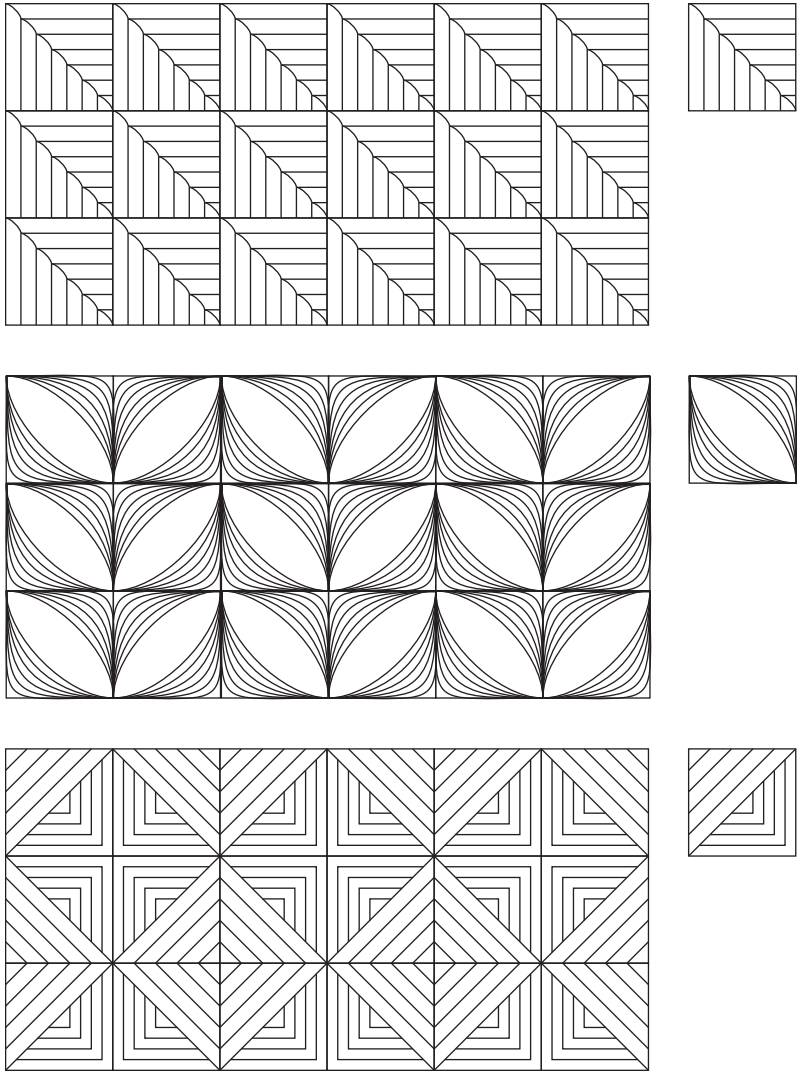

Figure 30.7

Examples of litema patterns from South Africa. These patterns are typically etched into the plastered mud walls on the exterior of homes. The patterns are constructed by repeating and rotating a single square unit.

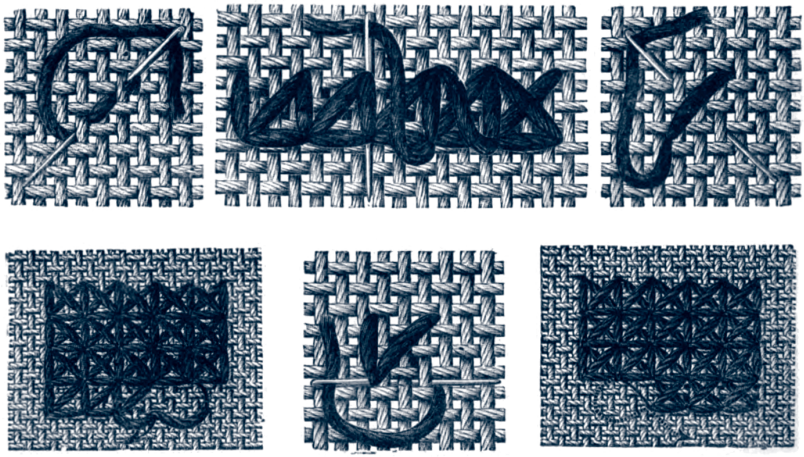

Figure 30.8
Examples of stitchwork from The Square Pattern technique from *The Young Ladies' journal Complete guide to the work-table*.

it seems appropriate to note that there are several ways up the mountain—or into the maze—of this particular random and regular pattern; one was discovered at Commodore, another by taking a mathematical perspective on tiling patterns and their aesthetics.

Textiles and Craft

The experiments of Truchet and Doüat did not introduce the idea of creating patterns out of simple variations on shapes. Such practice is commonplace across many forms of design, particularly in the realm of ornament, where both regular and irregular patterns have long been created. Franz Boas documented compelling examples of theme and variation of Peruvian weavers, for example (cited in Gombrich 1994, 72). The Kuba of Zaire create patterns of a complexity that has puzzled electrical engineers, patterns with the mazelike passageways of `10 PRINT` and yet of a far greater intricacy (Huang et al. 2005). Or consider the murals of the Sotho women of South Africa, decorative geometric murals known as *litema* (figure 30.7). This technique, documented as early as 1861, involves assembling networks of squares made of painted mud and etched with fingers and sticks

(Gerdes 1998, 87–90). In fact, the decorative arts have long held this secret to `10 PRINT`. Such techniques are detailed in the examples of fancy work in the 1885 *The Young Ladies' journal Complete guide to the work-table* (figure 30.8). The examples therein demonstrate the orthogonal basis for stitchwork that is evocative of the grid of the computer screen.

The hundreds of techniques define patterns ranging from simple grids to complex emergent patterns. As Mark Marino argues elsewhere (2010), these pattern books and instructive texts, primarily aimed at young women, provided models of fundamental processes similar to the role of the computer manuals and magazines such as *RUN*. Many of the techniques result from a repeated process with instructions, similar to that indicated by a computer program. For example, the Square Pattern technique (figure 30.8) in the Fancy Netting chapter is defined as a pair of operations that are repeated:

> No. 6.— SQUARE PATTERN
>
> For this pattern:—
> 1st Row: Work one plain row.
> 2nd Row: One ordinary stitch, and twist the thread twice round for the large square. Repeat to the end of the row.
>
> The first and second rows are repeated alternately. Arrange the stitches so that a long stitch always comes under a short stitch.

Such examples demonstrate that while the systematic theorization of patterns such as the one produced by `10 PRINT` may emerge periodically, the production of those patterns is deeply woven into the traditions of decorative craft. The fundamental role of shared techniques for process and pattern place computer programming squarely in the realm of *technē,* artistic craft. As in the *Commodore 64 User's Manual,* this text promotes the execution of a set of instructions collected as a technique. On the surface, the parallels between teaching needlecraft and programming are striking. The programmers, however, are not taught to repeat the procedure but instead, initially, to repeat a formal description of the procedure by typing it into the machine—which then does the repeating for them. It is the very automation of the process that makes `10 PRINT` possible; the program

operates less like handy stitching and more like the machinery of the Jacquard Loom.

Prior to that loom, during or near the second century BCE, China gave birth to a loom "that made it possible to create a pattern in fabric . . . called a drawloom because [it] allowed the warp threads to be drawn up individually to create the design to be woven" (10). That loom, however, was irregular: "the arrangement of the individual warp threads was different for every single row of weaving" (10). By contrast, the loom designed by Joseph-Marie Jacquard was regular and programmable (12). Such a machine relied on an exacting degree of regularity. Of course, much has been made of the Jacquard Loom as the prototypical computer, for example James Essinger's book *Jacquard's Web: How a Hand-Loom Led to the Birth of the Information Age* (2004). The core similarity in these early accounts were the punch cards, which were automatically applied to the control system and which served as patterns for the loom to follow. Earlier punch card looms have been discovered and attributed to J. B. Falcon, B. Bouchon and Vaucanson, whose invention of a mechanical duck is a bit more widely known (Zemanek 1976, 16). According to Essinger, Falcon's punch cards were "clumsily made and unreliable" (36).

Commercial-grade textiles require up to four thousand cards strung together—a far cry from the two statements on the one line of `10 PRINT` (figure 30.9). The cards are applied to a bar, an "elongated cube," full of "hundreds of identical holes . . . to accommodate the tips of needles," which are raised according to the selections on the punch card. As the bar turns with each pick of the shuttle, it moves down the material as if moving down a computer screen. Regularity made it possible for the Jacquard Loom to draw its intricate patterns. But the use of the cards as a pure pattern and the inability to regulate the flow of control meant that patterns have to be defined exhaustively rather than through concise programs. In other words, the number of cards is proportional to the size of the pattern being woven. While needlework instructions demonstrate the role of repeated process and pattern over somewhat regulated space, the loom regulates time and space without, in effect, repeating the process.

`10 PRINT` can be imagined as the complete method of craft programmed into the computer—as it was not fully programmed into the loom. The loop offers a way for the weavers of the computer screen to shift their emphasis from a fixed template, traversed once, to a more intricate

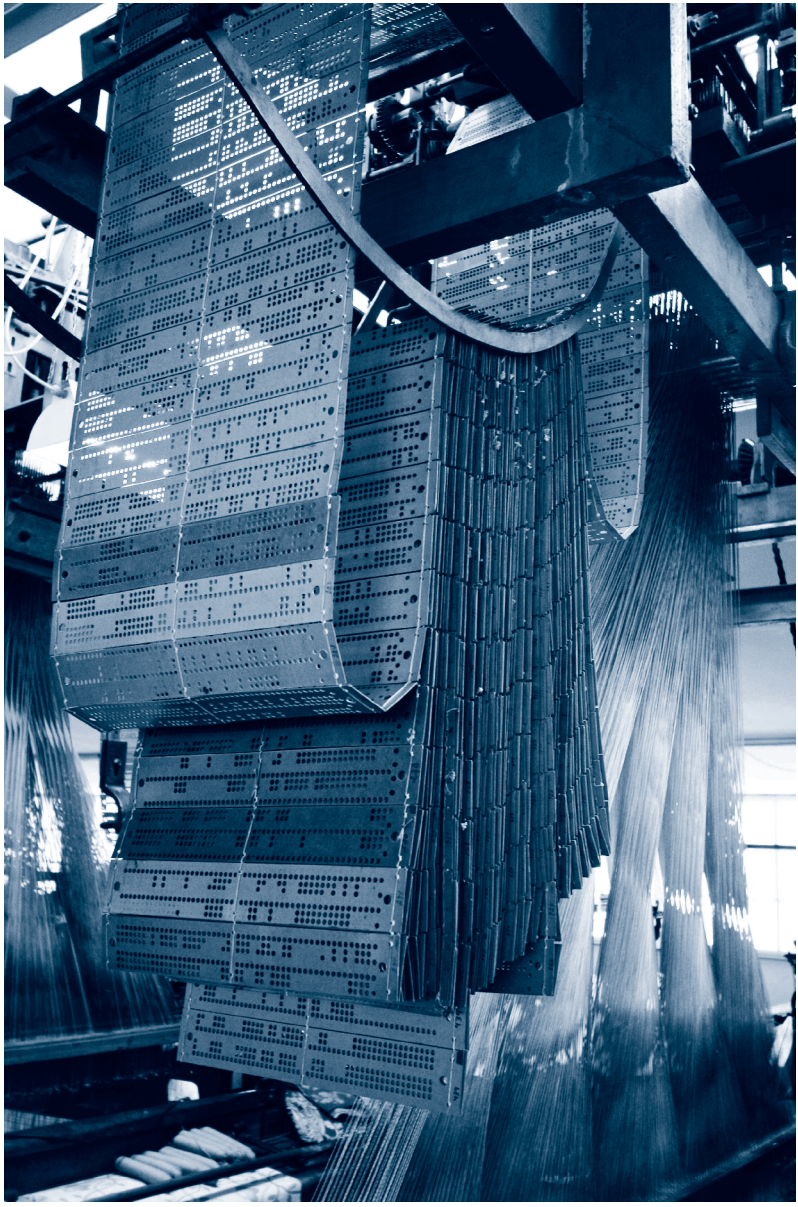

Figure 30.9

Punch card-operated loom at the Sjølingstad Uldvarefabrik in Sjølingstad, Norway. Courtesy of Lars Olaussen, Creative Commons Attribution-NonCommercial-ShareAlike 2.0 Generic.

model of process. `10 PRINT` demonstrates the power of the computational machine to rapidly prototype a repeated pattern, and since it executes the pattern itself, the incipient programmer is freed to experiment with variations and extensions of that process.

THE GRID IN MODERN ART

In the 1960s and 1970s, artists moved away from abstract expressionism, the dominant current of the 1950s, and its preference for raw emotion. Newer movements such as op-art and minimalism along with the continued line of constructivism in Europe engendered a body of rational, calculated visual art that utilized grids and even spacing to define order. A tour through any major American modern art museum will reveal Frank Stella's canvases of regular lines, Ad Reinhardt's hard-edge grids of barely distinguishable tones, Carl Andre's grids of metal arranged on the floor, Donald Judd's regularly spaced steel fabrications, Dan Flavin's florescent matrices, and Agnes Martin's exquisite, subtle grids on canvas. This list could continue for pages as it moves forward in history; the point has no doubt been made. This American tendency to move toward minimal forms was expressed well by Ad Reinhardt in "Art-as-Art" in 1962: "The one object of fifty years of abstract art is to present art-as-art and as nothing else, to make it into the one thing it is only, separating and defining it more and more, making it purer" (Rose 1991, 53).

In Europe at the same time, a massive number of artists were working with grid systems, and they were often doing so with more explicit focus and rigor. This energy was frequently channeled into groups that formed in different cities. For example, there was GRAV (François Morellet, Julio Le Parc, et al.) in Paris, ZERO (Heinz Mack, Otto Piene, et al.) based in Düsseldorf and extending a wide net across Europe, The Systems Group (Jeffrey Steele, Peter Lowe, et al.) in London, and the Allianz in Zurich (Max Bill, Richard Paul Lohse, et al.). The most iconic artist to work with grids might be the optical artist Victor Vasarely, whose grids were mesmerizingly distorted. His work was so systemized that he invented a notation system to enable a team of assistants to assemble his works using instructions and modular, prefabricated colored pieces. Although it is difficult to discern by just looking at the work, there was tension between the artists who worked

toward the primacy of mathematical form and those who maintained a desire to imbue subjectivity and emotion in their geometric compositions.

In critiquing of the former category of art, Ferreira Gullar, a Brazilian poet and essayist, wrote the 1959 "Neo-Concrete Manifesto" declaring that it was dangerous for art to be concerned only with "objective problems of composition, of chromatic reactions, of the development of serial rhythms, of lines or surfaces" (Zelevanksy 2004, 57). Gullar's manifesto harkens back and reimagines works of the early twentieth century by artists such as Wassily Kandinsky, Kasimir Malevich, and Alexander Rodchenko. Within the specific context of the grid, pioneers Piet Mondrian, Theo Van Doesburg, and other artists affiliated with De Stijl abandoned representation entirely. Van Doesburg et al. coined the term "concrete art" to categorize works that are conceived without reference to nature and symbolism. The manifesto "The Basis of Concrete Painting" published in April 1930 stated, "The work of art should be fully conceived and spiritually formed before it is produced. It should not contain any natural form, sensuality, or sentimentality. We wish to exclude lyricism, dramaticism, symbolism and so forth" (Fabre and Wintgens Hotte 2009, 187). Van Doesburg continued in "Elementarism (The Elements of the New Painting)" from 1932: "One must not hesitate to surrender our personality. The universal transcends it. . . . The approach to universal form is based on calculation of measure and number" (187). Representative works such as Mondrian's *Composition with Red, Blue, Black, Yellow, and Gray* (1921) and Van Doesburg's *Counter-Composition VI* (1925) were composed exclusively with orthogonal lines to form a grid. Works from this time also experiment with rotating the grid 45 degrees to create a more dynamic composition. This formal technique manifests itself, of course, in `10 PRINT`.

While the `10 PRINT` program came out of the computer culture and not the art world, it has an uncanny visual resemblance to prior works of twentieth-century art. Paul Klee, a Bauhaus professor and highly influential artist (1879–1940), produced works in the 1920s that seemed to resume Truchet's and Doüat's experiments. In his concise *Pedagogical Sketchbook,* published in 1925, Klee presents his thoughts on quantitative structure, rhythm, repetition, and variation. His *Variations (Progressive Motif),* painted in 1927, demonstrated his theories as a visual composition. He divided the 40cm-square canvas into a grid of nine units, where each unit contains a pattern of parallel lines, with some exceptions, which run vertical, horizon-

tal, or diagonal. More insight into this painting is found in his notebooks, as published in *The Thinking Eye* in 1964. Klee discusses the difference between natural and artificial measurement as the difference between idiosyncratic and rational order. More important, he discusses tension and dynamic density through the linear and progressive spacing of parallel lines. Through these visual contrasts in *Variations,* Klee explores the same aesthetics questions that can arise from `10 PRINT`. First he created an artificial grid to work within; then he populated each square with ordered but variable patterns. Klee didn't have the advantage of motion that is afforded to `10 PRINT`, but he simulated it through the expansion and contraction of parallel lines within his grid.

In France, a group of like-minded artists within and around GRAV (Groupe de Recherche d'Art Visuel) were exploring variations within grids. François Molnar and Vera Molnar worked on a series of images in 1959 that presented a visual system strikingly similar to `10 PRINT.` In the essay "Towards Science in Art," published in the anthology *DATA: Directions in Art, Theory and Aesthetics* in 1968, François Molnar published the images *Simulation d'une série de divisions de Mondrian à partir de trois au hasard* and *Quatre éléments au hasard*. Both are 24 × 24 unit grids with one of a few possible forms painted into each grid unit with black gouache. As the titles suggest, a random process defines the elements in each square. Their *Composition Stochastique* of the same year systematizes the random component by producing a modular set of two elements—left and right diagonals that are placed within a 10 × 10 unit grid. In the illustrations for the essay, they feature a 1 percent, 5 percent, 30 percent, and 50 percent ratio of left to right diagonal lines to show the result of chaos intruding upon order. Given that this is a 100 unit grid, these percentages correspond to precisely 1, 5, 30, and 50 units in each figure. In the 50 percent figure, the only substantive difference with our `10 PRINT` program is the variation on the core elements. So, just as a mathematician independently described the output of `10 PRINT` in 1987, a team of artists working in Paris produced the fundamental algorithm for `10 PRINT` in 1959—twenty-three years prior to the printing of the *Commodore 64 User's Guide*.

In her 1990 essay entitled "Inconceivable Images," Vera Molnar wrote that she was thinking about *Composition Stochastique* as a computer program, because she had access to a machine:

> To genuinely systematize my research series I initially used a technique which I called machine imaginaire. I imagined I had a computer. I designed a programme and then, step by step, I realized simple, limited series which were completed within, meaning they did not exclude a single possible combination of form. As soon as possible I replaced the imaginary computer, the make-believe machine by a real one.

Across the Atlantic in the 1960s the American artist Sol LeWitt embarked on decades of work exploring grids and regular structures. In 1968, LeWitt started making drawings directly on walls, rather than on paper or canvas that would be placed on the wall. In this return to the scale of frescos, his drawings within grids integrated into architecture to transform the space (Singer 1984). His *Wall Drawing 291* from 1976 is a striking work, with a strong similarity to `10 PRINT`. Instead of the binary decision within `10 PRINT`, LeWitt's drawing allows for horizontal and vertical lines, to create four choices for each grid element. LeWitt's work is encoded as an algorithm—another similarity with `10 PRINT`. A difference is that the instructions are in English, rather than BASIC:

> 291. A 12" (30cm) grid covering a black wall. Within each 12" (30cm) square, a vertical, horizontal, diagonal right or diagonal left line bisecting the square. All squares are filled. (The direction of the line in each square is determined by the draftsman.)

This grid-based wall drawing wasn't an isolated work within LeWitt's output. He created dozens of similar drawings, each with slightly different rules and allowing for varied lines including arcs and dotted lines.

While many artists and critics in the twentieth century were clearly obsessed with the grid, not all have celebrated it. The critic Rosalind Krauss put the grid into a different context in her 1979 essay "Grids" (Krauss 1979). She acknowledges the proliferation of the grid but criticizes it as a dead end: "It is not just the sheer number of careers that have been devoted to the exploration of the grid that is impressive, but the fact that never could exploration have chosen less fertile ground." She continues, "The grid declares the space of art to be at once autonomous and autotelic." Through pursuing pure visual exploration like variations on grids, Krauss argued that

SCREENSAVERS AND COMPUTER DREAMS

While no one has ever claimed that `10 PRINT` might be used as a screensaver to prevent phosphor burn-in on CRT monitors and televisions, its noninteractive, endlessly looping nature coupled with its pleasing and changing image make it a close cousin to this type of program. If one had to place `10 PRINT` into a familiar software category, "screensaver" would not be a bad one to choose.

The earliest screensaver dates to 1968, when researchers at Stanford University programmed the text "Take Me, I'm Yours" to appear at random locations on an open terminal screen of Stanford's Artificial Intelligence Lab's time-sharing system, signaling the terminal was free to use. In 1973, engineers at the famed Xerox PARC lab created screensavers with bouncing and zooming graphics, moving screensavers beyond mere text (Davenport 2002, 65). In the early 1980s, commercial screensavers followed, such as Berkeley Systems' bestselling *After Dark* collection. By the height of the screensaver craze in 1996, the design of computer monitors made phosphor burn-in nearly impossible. Yet screensavers lived on in roles that went beyond their original utilitarian or aesthetic function. The SETI (Search for Extraterrestrial Intelligence) project released software for PCs that harnessed unused computer cycles to process radio waves from outer space—all the while displaying a screensaver on the computer display. Along different lines, Nancy Davenport created the "May Day" screensaver that captures the repetitive and often image-based nature of contemporary political protests.

There are provocative parallels of the screensaver in the history of art. David Reinfurt (2009) considers a wide variety of moving visual and mechanical pieces, including Marcel Duchamp's *Precision Optics* projects, Alexander Calder's mobiles, phased oscilloscope displays, Brion Gysin and Ian Sommerville's *Dreammachine*, and generated visual and sound art. Reinfurt sought to find interesting connections and not to trace a genealogy or demonstrate influence, but there appears to be a compelling pre-war antecedent of these programs.

Perhaps the most intriguing protoscreensavers are found even earlier, in the form of Duchamp's 1933 *Rotoreliefs*. This set of six discs, each printed on both sides with offset lithography, allowed the purchaser to do something with a 33 RPM turntable when it was not being used for its primary purpose: listening to music. Of course, the disc caused additional wear on one's turntable and did not "save" it, but screensavers have seldom been truly valued as savers. Just as a screensaver pack of-

fers several options to suit the mood of the non-computer-using viewer, *Rotoreliefs* offered twelve options for the viewer, the nonlistener. One side of the third disc features "Poisson Japonais" and—just as the screensaver would later make the monitor into a simulation of a fish tank—makes the turntable into a fish bowl.

This twist on the electronic device (initially the turntable, later the television or computer monitor) calls attention to its being a piece of furniture; it also simulates the activity of a living creature using electricity and technology. It shows us our powerful media technologies made mute, circling, and amusingly off-kilter. The relationship to technology that is suggested by the aquarium screensaver (or fish Rotorelief) is one of odd juxtaposition and low-key looping motion, probably not far off from the effect of spinning a bicycle wheel that has been fixed in a stool.

One very emblematic screensaver combines life and technology in an even more curious way. In Berkeley Systems' *After Dark 2.0*, the "Flying Toasters" screensaver features toasters flitting across the screen using their small (birdlike) wings. The toaster, that single-purpose device used only to cause bread to undergo the Maillard reaction, drifts lazily through space alongside . . . pieces of toast. This screensaver suggests that androids *do* dream of electric sheep and that computers, when they snooze, have visions of lower forms of technology in flight. The *After Dark 2.0* toasters, remarkably, are not pure technological artifacts—they are cyborg toasters with organic wings. While the scene they take part in is amusing, it also at least risks calling attention to the limitations of the computer, which, despite its general-purpose capabilities, does not help to prepare food, is entirely inorganic, and, of course, does not fly. The computer may be capable of symbolic manipulations and machine dreams, but there are realms into which it cannot go, realms towards which it is left to aspire. Perhaps all of this is not imagined by the average computer nonuser observing toasters in flight, but any of it which is will contribute to the absurdity of the image and the pleasure of the onlooker.

Some screensavers create very abstract patterns that have no simple interpretation, even as something like an abstract maze. Others, like the early Windows "Starfield" and the Windows 95 "Maze" (which shows movement through a 3D, RPG-like maze) suggest that the computer is a vehicle for exploration. These screensavers live alongside those that play with our perceptions of life, inviting us to think about how technologies relate to creatures like fish and birds. But in addition to all

of these, there are screensavers that accumulate structures in a way that suggest industrious, technical production. The Windows 95 "Pipes" screensaver is a fairly famous example. It assembles 3D tangles of multicolored pipes which become impossibly dense and intricate. This screensaver does more plumbing work in a few minutes than Nintendo's Mario has done in his lifetime. The busy visuals show that the computer is hard at work, even though its user is not interacting with it. While the image is pleasing to look at, it also projects a more serious image than the frivolous flying toaster or simulated fish tank.

While `10 PRINT` is extremely abstract, its generation process seems to be one of furious and constant construction. `10 PRINT` suggests that the computer is a maker of structures, is tireless at producing these at a regular rate, and can create patterns that are both pleasing to view and perplexing to walk through. While the Windows 95 "Maze" screensaver provides the viewpoint of Theseus (or perhaps the Minotaur), `10 PRINT` shows us the maze as seen by Daedalus: from the mind's eye of the architect, the viewer shares the imagination of a structure that is continually in the process of being built.

the visual arts abandoned narrative and discourse and moved into cultural isolation.

During the era of our `10 PRINT` program, in wake of the Vietnam war and social movements of the late 1960s and early 1970s, the larger emphases within visual arts communities had moved away from minimalism and constructivism (and their variants) to focus back on expressive and realistic painting and the emerging acceptance of photography. The visual work created for early home computers and games systems like the Atari VCS and Commodore 64, however, were highly constrained by the technical limitations of the hardware and therefore had more in common with the visual art of prior decades.

A chief explanation for these uncanny similarities is the grid itself. In "Designing Programmes," the Swiss designer Karl Gerstner (1964) asks, "Is the grid a programme? Let me put it more specifically: if the grid is considered as a proportional regulator, a system, it is a programme par excellence." Gerstner's encounters with the computer led him to theorize the

regulated space as a program itself. The grid systematizes artistic creation even as it presents a challenging and yet generative platform for experimentation, whether on canvas, the dance floor, or a computer screen.

THE COMPUTER SCREEN

While the traditions of twentieth-century art and earlier craft traditions are significant for `10 PRINT`, the program functions the way it does because of the circumstances of technology, the history of the Commodore 64's display, and the types of regularity it supports. Again, the grid acts as a program to determine the final output.

The Commodore 64's video image is a grid 320 pixels wide and 200 pixels tall. This accommodates an array of characters or, in the terminology of Commodore 64 hi-res graphics, attribute cells. Specifically, the grid is 40 attribute cells wide and 25 high. (There are other graphics modes that offer advantages, but for understanding 10 PRINT, this array of characters or attribute cells is most important.) The 40 × 25 grid contains exactly 1,000 characters, each represented by a byte. This fits nicely into one kilobyte (which equals 1024 bytes)—in fact, it is the largest grid that is forty characters wide and occupies 1024 bytes or less.

Economically and conveniently, the Commodore 64 could be taken out of its box and hooked to an ordinary television. It was an idea that could be seen in Steve Wozniak's Apple I, introduced in April 1976. Later, the more widespread Apple II could also be connected to a television if one used an inexpensive RF modulator, purchased separately. (This component was left off the Apple II as a workaround; the FCC would not otherwise approve the computer, which would have produced too much interference.) The Apple II was the main precedent in home computing—other early home computers such as the TRS-80 Model II and Commodore PET had built-in monitors—but the idea was not original to Apple. At Atari, television engineer and employee #1 Al Alcorn had designed a *Pong* cabinet that, rather than using an expensive commercial CRT (cathode ray tube), incorporated an ordinary black-and-white television that was initially bought from a retail store. Wozniak, who did the original design for the Atari arcade game *Breakout*, knew about this trick of using a TV as a monitor. Videogame consoles (including Atari's 1977 VCS, later called the

Atari 2600, and the Magnavox Odyssey by Ralph Baer, introduced in 1972) would typically hook to televisions, too.

The rectangular form of the television image had its origins in the movie screen, which was rectangular due to the material nature of film and apparently obtained its 4:3 aspect ratio thanks to a gesture by Thomas Edison, one which resulted in a frame that was four perforations high. While the aspect ratio of film changed and diversified over the years, television in the United States (standardized in the NTSC format in 1953) and many computer monitors, through the 1980s and 1990s, used the 4:3 ratio.

Although composite monitors were available for the Commodore 64, the relationship between that system and the television was clear and was the default for those setting up their computer and hooking it up to a display. For a Commodore 64 purchased in the United States, the system's video output usually terminated in a NTSC television. But the computer display did not begin there: it has a heritage that included at least two output methods, ones that seem unusual today.

As one of this book's ten coauthors has noted, "Early interaction with computers happened largely on paper: on paper tape, on punch cards, and on print terminals and teletypewriters, with their scroll-like supplies of continuous paper for printing output and input both" (Montfort 2004). The standard output devices for computers through much of the 1970s were print terminals and teletypes. Output was not typically produced on pages of the sort that come from today's laser printers, but on scrolls of standard or thermal paper. The form factor for such output was not a standard 8½ × 11-inch page, but an essentially endless scroll that was typically 80 columns wide.

Teletypes were used to present the results of the first BASIC programs written at Dartmouth in the 1960s, and they were the typical means of interacting with important early programs such as *Eliza* and *Adventure*. With such a system for output, there was no need for an automated means of saving and viewing the "scrollback"—a user could actually pick up the scroll of output and look at it. Of course, this sort of output device meant that animation and other effects specific to video were impossible.

An argument has been advanced that the modern computer screen also has an important heritage in the round CRT display of the radar screen (Gere 2006). The SAGE early warning system, the PDP-1, and the system on which Douglas Engelbart did the "mother of all demos" all sported

round CRTs, as did early televisions. It is notable that the first two systems that may have been videogames in the modern sense, *Tennis for Two* by William Higginbotham and *Spacewar* by Steve "Slug" Russell, Martin "Shag" Graetz, Alan Kotok, and others were both created for circular CRT displays. While radar and some other images were actually round, as the early cathode ray tube was, the television signal and the page were not. What was, for radar, a radial display eventually gave way in computing to the rectangular, grid format that was adhered to by both page and television image.

REPETITION IN TIME

While `10 PRINT` would be impossible without the regularity of space, it would also be wholly other without regularity of time and process. The program is as much the product of ordered isometric shapes across a grid as it is the repeated placement of those shapes. Gombrich notes that "Everything . . . points to the fact that temporal and spatial orders converge in our experience. No wonder language speaks of patterns in time and of rhythms in space" (1994, 10). He continues to examine simple mechanical temporal rhythms from the pendulum's swing to the turn of the cog. As a bridging example between spatiality and temporality, he notes the way a regular configuration of stairs' height and depth in a staircase lead to a regular climb up the steps.

As Gombrich develops the notion of temporal repetition and regularity, he quickly transitions into a discussion of process. Whether a clock ticking or a person climbing the stairs, the temporal regularity is the result of a repeated process. Gombrich then moves to a discussion of work, by referencing K. Bucher's *Work and Rhythm*, which insists "on the need for timed movement in the execution of joint tasks," for example workers loading bricks onto wheelbarrows (Gombrich 1994, 10). The ticking clock does more than set the hours of labor on the factory floor: it epitomizes the regular movement of the workers. Gombrich continues, "And here again it is not only the simultaneous movement that is ensured by rigid timing. Even more important is the possibility inherent in any order of constructing a hierarchy of movements or routines to ensure the performance of more complex tasks" (10). This formulation suggests the relationship between

regulated time and instruction, hierarchies of movements and routines, which recalls Taylorist models of production as well as programming. While a full investigation of those connections lies outside the aim of this book, it is important to note the fundamental role of processed instructions in producing rhythms in time and space.

Process and the appearance of motion are essential to `10 PRINT`. The still images that show a moment in the program's run, the sort that are reproduced in this book, document the program to some extent but are an incomplete representation. A full understanding of the program comes only through experiencing the pattern building one unit at a time and the surprise of the unexpected sequences and connections that form into a maze as the left or right line is randomly drawn.

The visual arts at their most traditional, represented by frescoes, stone and bronze sculptures, and canvases, are static. A viewer creates motion by moving around a work, eyes exploring the surface, but the object is still. The thrust of machines into life at the beginning of the twentieth century was an inspiration to painters (the Futurists), photographers (Étienne-Jules Marey, Eadweard Muybridge), and sculptors who used motors to create motion. The origin of integrating physical movement into artworks in the twentieth century is often credited to Naum Gabo for his *Kinetic Construction (Standing Wave)* from 1919–1920. This sculpture created a virtual volume in space by mechanically hitting a metal rod near the base to send a wave through the object. Gabo was thrilled to bring motion into his art and his enthusiasm led to "The Realistic Manifesto," cowritten with his brother, Antoine Pevsner, in 1920. After rejecting the traditional foundations of art they declared:

> We renounce the thousand-year-old delusion in art that held the static rhythms as the only elements of the plastic and pictorial arts. We affirm in these arts a new element, the kinetic rhythms as the basic forms of our perceptions of real time. (Quoted in Brett and Nash 2000, 228)

With kinetic rhythm as the base of all new art, Gabo's *Kinetic Construction* is an ideal demonstration. It is a machine without visual interest or relation to the sculpture of the time. It performs the same motion precisely over and over. The work of art is reduced to a rhythmic repetition. While other pioneers of motion in art such as Marcel Duchamp and Alexander Calder

worked with motors to create regular machines, by the middle of the century the dominant form of motion had shifted to the type of chance motion experience through the wind moving a Calder or Rickey mobile or the anarchic mechanical chaos of Jean Tingely. The essence of `10 PRINT` lies in the relationship between both forms.

The next phase of the pairing motion and repetition in visual art brings us closer to `10 PRINT`. Artists began to create works for screens, first with film and later for CRT screen with video. Akin to the minimalist sculptures referenced above, there was a proliferation of minimal gestures within experimental film and animation. Starting in the 1960s, artists including Lillian Schwartz, John Whitney, Norman McLaren, Bruce Nauman, Richard Serra, and Paul Sharits explored repetitive physical movements and abstract motion with rigor. *The Flicker* (1965), a film by Tony Conrad, stands out for its clarity. As shown earlier, without the semicolon in `10 PRINT`, each diagonal line could be seen as a panel of a film strip. It's only a small leap to imagine the left line of the program as an unexposed film frame (clear) and the right line as the maximum exposure (black) to bring the fundamental mechanism of `10 PRINT` close to *The Flicker*. The fundamental difference is the larger arc within Conrad's work. The pace at which the projection flips from pure light to black is slower at the beginning and end of the film to give it a beginning and end, while `10 PRINT` maintains the same pace, does not vary in any way as it begins, and continues running until interrupted.

Simultaneously with the exploration of repetition in film, a host of composers based musical works on repetition. Like film and video, musical performance is temporal, but unlike these linear media, performances and `10 PRINT` unfold in real time, each step happening in the moment and potentially informed by the present state. *Piano Phase* (1967) by Steve Reich is an iconic sound work built on repetition. In this approximately twenty-minute-long composition, two pianists start by playing the same twelve-note melody continuously. One pianist plays the sequence faster so it moves out of phase until they are back in phase, but the faster pianist is playing the second note, while the slower is on the first note. This continues until the faster pianist has complete a full loop and both are again playing the same sequence at the same time. The piece iterates further from that point, but the same phasing technique is used until the end. The concept is the same as in Reich's later and simpler piece *Clapping Music* (1972), which is clapped by two performers and varies only in rhythm. In `10 PRINT`,

new forms emerge from the program's decision to display the left or right line, but in *Piano Phase* and *Clapping Music,* new sonic forms emerge by playing the same sequence over and over, with performers playing at a different speeds.

REPETITION IN PROCESS

The artworks in this chapter engage with regularity as a style and technique; computers employ regularity as a necessary paradigm of their existence. The execution of a computer program, even one that is riddled with bugs or does numerous complex and interesting things, is nothing if not regular. In fact, it is the regularity of computer processes that many of the artworks discussed in the chapter are reacting to and against. Even more than in the Ford factory, regularity becomes a paradigm of the computational age, to be explored and resisted because it is the central logic for even the most basic computational literacy. While the assembly line might put many goods in the hands of twentieth-century consumers, families did not need to contemplate assembly lines to consume these goods. Even for workers actually in a factory, the flow of the factory would be defined elsewhere. However, to write even the most rudimentary program, a person must understand and engage the regularity of the machine. Consequently, it is worthwhile to articulate the process of flow and control that allows this regularity to become such a generative space.

Part of what gives programs their power is that they can be made even more regular by repeating a sequence of instructions. This repetition can be accomplished in two main ways: in a loop that continues for a certain number of iterations or in an unbounded loop. These two loops correspond to two types of branching, the conditional and unconditional branch. To understand the unbounded loop of `10 PRINT` and the specific legacy of `GOTO` is to understand the essentials of the flow of control in computer programs.

To explain the loop, it is necessary to first juxtapose it with the alternative: not having a loop and letting a program progress in the usual sequence. In any imperative programming language, commands are processed in a particular order that relates to the left-to-right then top-to-bottom path that Western readers' eyes take along a page. If one types two

PRINT commands directly into the Commodore 64's BASIC interpreter, with a colon between them, like so:

```
PRINT "FIRST": PRINT "SECOND"
```

the result is

```
FIRST
SECOND
```

The command on the left is executed first, then, the command on the right. In executing the following program, the top-most, left-most command is run first, then the one to the right, and then the one on the next line, so that

```
10 PRINT "FIRST": PRINT "SECOND"
20 PRINT "THIRD"
```

prints FIRST, SECOND, and THIRD, in that order. Since BASIC uses line numbers to determine the sequence, the order in which these two lines are typed is completely irrelevant to how the program runs.

There are some important ways to complicate this straightforward program flow. All of these ways involve branching, which causes program flow to shift to some other command rather than continuing along to the subsequent one. The same could be said of the low level of machine code and its execution by the processor. A machine language program is a sequence of numbers typically processed in order of appearance. An executing machine language program, however, like a high-level program in BASIC or another imperative language, can either continue to process the next instruction or can move out of sequence to process a different instruction. The standard case is when a processor continues to the next machine language instruction, just as the reader of a text moves to the next word or line. This involves incrementing the program counter so that it points to the place where the next instruction is located in memory. If the current instruction is one that can change the flow of control, however, the program may jump to a new memory location and a new piece of code. The branch or jump is the key operation that is used to build a loop.

Conditional and Unconditional Branching

There are two essential ways that the flow of control can change and a program can branch, continuing from a new point. An unconditional branch always directs the program to jump to a new location, like the instruction "Go directly to Jail. Do not pass Go" in Monopoly. The assembly mnemonic for an unconditional branch is `jmp`; the corresponding BASIC keyword is `GOTO`; packaging together a branch away from a line of code and then a subsequent branch that returns to the original line constitutes a subroutine, implemented in BASIC using `GOSUB` and `RETURN`. When an unconditional branch is used to point back to an earlier instruction, it can cause repetition in process as in the case of `10 PRINT`.

The other type of branch is a conditional branch, a type of instruction that is critical to general-purpose computing. There are many different types of conditional branches in assembly, since there are many different types of conditions. `beq` is "branch if equal," for instance: when used after a comparison (`cmp`), the branch will be taken only if the compared values are equal. `bmi`, "branch if minus," checks to see if the last computation resulted in a negative value. In BASIC, using the `IF . . . THEN` statement is the most straightforward way to accomplish a conditional branch, as this program demonstrates:

```
10 INPUT A$
20 IF A$ = "1" THEN PRINT "YOU TYPED ONE!" : END
30 PRINT "SOMETHING ELSE..."
```

If, after running this program, the user types just the digit "1" and then presses RETURN, all of the statements on line 20 will be executed. "YOU TYPED ONE!" will be printed and then the program will terminate as `END` instructs. This is another way to change the flow of the program, of course: use `END` or `STOP` to terminate the program. If `STOP` is used, the `CONTINUE` command can be issued to pick up where the program left off.

If the user types nothing or anything other than a single "1" before pressing RETURN, the flow of control moves to line 30; both the first `PRINT` statement and the `END` are skipped. "SOMETHING ELSE . . ." is printed instead. This program, although written differently, does exactly the same thing:


```
10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

```
10 INPUT A$
20 IF A$ = "1" THEN GOTO 40
30 PRINT "SOMETHING ELSE..." : END
40 PRINT "YOU TYPED ONE!"
```

Instead of using the `IF . . . THEN` to directly determine whether two statements (the `PRINT` and `END` statements) should be executed, this one changes the flow of control with `GOTO`. The `GOTO` statement is used to skip ahead past `PRINT "SOMETHING ELSE..."` and `END` to line 40. Although this isn't a very exciting program, it shows that unconditional branching can be used to jump *ahead* in the sequence of lines; there is nothing about `GOTO` that means it *must* be used to repeat or loop.

Although there is no `IF . . . THEN` statement in `10 PRINT`, and the program does not by any interpretation contain a conditional branch, this short program does accomplish a very small-scale sort of variation. By computing `205.5+RND(1)` and passing that value to the function `CHR$`, the program prints either PETSCII character 205 or PETSCII character 206. This variation between two characters is a one-bit variation, a selection from the smallest possible set of options. Yet, in combination with the regularity of a repeating processes and the regularity of the Commodore 64's screen grid, these selections take shape as something evocative and visually interesting.

The Harmfulness of GOTO

Those aware of the discourse in computer science might turn to `10 PRINT` with some trepidation, thanks to a 1968 letter to the editor from famous computer scientist Edsger W. Dijkstra, one that was headlined "Go To Statement Considered Harmful" (EWD 215). Although the title was actually written by the editor—Dijkstra called this article "A Case against the GO TO Statement"—the letter and the sentiment behind it have gained lasting fame. One author called it "probably the most often cited document about any type of programming" (Tribble 2005). As this author explains, Dijkstra's exhortation was written at a time when the accepted way of programming was to code iterative loops, if-thens, and other control structures by hand using goto statements. Most programming languages of the time did not support the basic control flow statements that we take for granted today,

or only provided very limited forms of them. Dijkstra did not mean that *all* uses of goto were bad, but rather that superior control structures should exist and should replace most uses of goto popular at the time.

Indeed, there is an obvious control structure, unfortunately absent from BASIC, which would accomplish the purpose of `10 PRINT`'s `GOTO` without requiring the use of `GOTO`. This is the `while` or `do . . . while` loop, which in this case could simply be used with a condition that is always true so that the loop would always repeat. For instance, if Commodore 64 BASIC had a `DO . . . WHILE` statement and a keyword `TRUE`, one could write:

```
10 DO : PRINT CHR$(205.5+RND(1)); : WHILE TRUE
```

This would certainly be a clearer way to write the program, and it would be widely recognized today as clearer because Dijkstra's view of programming has prevailed. When the creators of BASIC introduced True BASIC in 1983 they included the `DO` loop; its syntax was a bit different from the preceding, but the essential construct was the same.

Although this important construct is missing from early versions of BASIC, it's not obvious that the tiny program `10 PRINT` would have particularly offended Dijkstra. In his letter, he objects to the "unbridled use of the go to statement," but he does not state that *every* use of it is unambiguously bad. He describes the `GOTO`-related problems that arise when programmers are not able to track and understand the values of variables. But `10 PRINT` has no variables, so this particular problem with `GOTO` is not an issue in this particular case.

As mentioned, `GOTO` has a assembly language equivalent, `jmp`. Dijkstra essentially exempted assembly language from his critique of `GOTO`. He recognized that computer programs written in a high-level language are more complicated and nuanced ways of expressing thought, not tied directly to machine function. By creating a new sort of language that does not directly mimic the operation of the processor, it is possible for programmers to think more flexibly and powerfully. Although Dijkstra objected to the way BASIC was designed, he, like the original designers of BASIC, worked strenuously to describe how high-level languages, useful for thinking about computation, could work in machine-independent ways.

Bounded and Unbounded Loops

`10 PRINT` works as it does because of its repetition of the `PRINT` statement, repetition that is unconditional: it will continue until something outside the program interrupts it, such as the user pressing the key labeled RUN STOP or unplugging the computer. This ability to repeat endlessly differentiates the program from its artistic parallels in other media.

At a high level, programs can contain two types of loops: bounded (also called "finite") and unbounded (or "infinite"). `10 PRINT` has the latter kind. If one is writing a program that is intended to produce some result and exit, it would be a mistake to include an unbounded loop, creating a bug that would make the program hang on a particular set of operations. Even among comparisons to repetitions in fine art and craft work, the computer stands alone as a system capable of infinite looping. Nonetheless, the unbounded loop does have legitimate uses. It can be used to keep an application, usually an interactive one, running and possibly accepting input until the system is shut off, rebooted, or interrupted. This can be done, and is done, even on a Commodore 64.

Bounded loops are those that end under certain conditions. The exact conditions vary; some will continue until the program reaches a predefined exit state, while others execute a specific number of times. If there are exit conditions for a loop at all, that suggests the programmer expected some kind of change to be introduced as the program executes.

A common use of finite loops is to create an iterative process. Iteration is a special type of looping where the result of one pass through the loop influences the result of succeeding passes. The simplest example of an iterative process is nothing more than counting: beginning with an initial value of 1, adding 1 to this (that is, incrementing it) to produce 2, performing the same incrementing operation again to yield 3, and so on. If the program goes to some limit, say 10, the loop is bounded:

```
10 A=1
20 PRINT A
30 A=A+1
40 IF A<=10 GOTO 20
```

If line 40 is replaced with `40 GOTO 20`, the loop becomes unbounded. A

BASIC CONSIDERED HARMFUL

Edsger W. Dijkstra, of "Go To Statement Considered Harmful," was not at all a fan of the BASIC programming language in which `10 PRINT` is written. He wrote in a 1975 letter, "How do we tell truths that might hurt?," published in 1982: "It is practically impossible to teach good programming to students that have had a prior exposure to BASIC: as potential programmers they are mentally mutilated beyond hope of regeneration" (EWD 498).

The statement appears amid other one-sentence jabs at programming languages (FORTRAN, PL/I, COBOL, and APL), IBM, and projects to allow "natural language" programming. In a 1984 keynote address, "The Threats to Computing Science," Dijkstra said, similarly, that "the teaching of BASIC should be rated as a criminal offence: it mutilates the mind beyond recovery" (EWD 898). In both cases, his statements are not part of arguments, nor are they elaborated at all. They are simple denunciations of BASIC—no doubt resonant with many computer scientists and no doubt of some tactical value at the time, when the structured programming that predominates today and that Dijkstra was advocating was still being questioned.

Dijkstra's papers contain a single reference to BASIC coinventor John Kemeny. Dijkstra read Kemeny's 1983 article in *Daedalus,* an article that discussed the idea of computer literacy and declared that "the development of 'structured languages' in recent years has been a giant step forward." Dijkstra simply wrote a dismissive

conditional branch is what makes loop bounded—it ends when the condition is met. Therefore, an unconditional branch to earlier in the program corresponds to an unbounded loop.

The bounded loop, and counting up or down to a particular value, is so important in programming that BASIC has its own special syntax for specifying that sort of loop, using the `FOR . . . TO` and `NEXT` statements. The bounded program above could also be written:

```
10 FOR A=1 TO 10
20 PRINT A
30 NEXT
```

sentence about the article, saying it "gave a striking example of superficiality" by comparing computer languages to natural languages (EWD 858).

The context for these statements was a time of formation and fortification of the discipline of computer science. Also important was that at this time, computer scientists were undertaking the development and use of languages that might support first formal verification (the provability of programs) and later the weaker, but still potentially very useful, technique of program derivation (Tribble 2005). BASIC, created for quick interactive use and very amenable to creative production, was not a suitable language for Dijkstra's goals.

Much of Dijkstra's influential thinking (about stepwise programming, for instance) applies most clearly to programs of some complexity. It would be difficult to develop a program as simple as `10 PRINT` using stepwise programming, since it may take about one step to write it. As mentioned, `10 PRINT` is in a sense exempt from Dijkstra's critique of `GOTO` because of its lack of variables. But in another sense, it is one of many extremely simplified programs that could lead programmers to learn programming methods that don't scale. The risk of learning programming via one-liners is that one learns tricks and particular methods instead of gaining an understanding of programming methodologies.

The same program could be written in yet another way:

```
10 A=1
20 PRINT A
30 A=A+1
40 PRINT A
50 A=A+1
...
200 PRINT A
```

Doing so is an inefficient and error-prone way to write a process. The programmer is forced to do a repetitive task that the computer is extremely

well suited to accomplish on its own. Modifying the program to count to 50 takes four times as long as writing the original program, if this way is chosen. In the previous program, one simply changes "10" to "50" in line 10.

Looping and Iterating

`10 PRINT`'s unbounded loop produces iterative effects, even if it is not enacting a purely iterative process. The unconditional branch at the end is what accomplishes this: `10 PRINT` clearly repeats. There is nothing in the code, however, to indicate that `10 PRINT` is an example of iteration in the more mathematical or computational sense. There are no variables that change value from one pass to the next—there are no variables at all, in fact—and the code inside the loop, and the way that code works, never changes. In the most straightforward computing sense, the BASIC program `10 PRINT CHR$(205.5+RND(1)); : GOTO 10` does not iterate.

Nonetheless, watching the program execute on screen shows hints that there is something changing as it runs: the position where a new character is displayed changes every time the loop executes, moving one location to the right until it reaches the fortieth column; then, the display of characters continues on the next line in column 1. The entire contents of the screen also moves up by two lines, with the top two lines disappearing, once the display of characters reaches the bottom left position.

None of this is a direct result of any BASIC statements included in `10 PRINT`. Displaying strings on the screen generally has this effect, since BASIC's PRINT command calls the Commodore 64 KERNAL's CHROUT routine to produce output. So, while `10 PRINT` is not an iterative program, it invokes the iterative behavior of `PRINT`. `10 PRINT` exposes the power of using simple, iterative steps to create a complex construction.

Computing is full of iterations and simple loops. The highest level of an application program is typically the "main loop" that checks for user input and for other events. Turn-based games, such as a chess program that plays against a human player, will typically have a conditional loop for each turn, conditioned upon whether or not the game is over. And frames of a graphics window or the screen overall, whether drawn in OpenGL, Processing, or by some other means, are drawn within a loop that repeats more or less rapidly depending upon the framework.

PERFORMING THE LOOP

Step Piece by Vito Acconci is a performance work based on repetition that is interesting to compare and contrast with `10 PRINT`; it is a defined, repetitive procedure that is carried out by a person rather than a computer. Acconci defined this work in a brief text: "An 18-inch stool is set up in my apartment and used as a step. Each morning, during the designated months, I step up and down the stool at the rate of 30 steps a minute; each morning, the activity lasts as long as I can perform it without stopping."

Since there are months designated for the performance, Acconci defined a bounded loop. *Step Piece* was certainly meant to be the same every morning in particular ways during these months. But it was not the same at every point for a viewer or for Acconci, and will not be the same at every point for someone considering the piece years later. How is repeating the same thing over and over not repetitive? A person's repetitive performance cannot be exactly the same each time. Acconci no doubt stepped more rapidly at some times and then needed to slow down; his foot struck the stool in a slightly different way, producing a different sound. There is no close analogue to this in `10 PRINT`, since a Commodore 64 runs the program the same way each time and the symbols are presented in the same way on the same well-functioning television. The photographic documentation of *Step Piece* shows Acconci in action with one foot on the stool and the other on the way up or down. Just as different screen captures from `10 PRINT` are different, each photo is different—they differ even more in some large-scale ways since they are images of the same person in different postures, not a 40 × 25 grid with different elements in it.

Additionally, as the days pass, Acconci gets better at his repetitions (like a weightlifter doing "reps" to improve strength) while `10 PRINT` writes characters to the screen at the same rhythm and in the same manner as long as the program runs. Some computer programs do actually improve over time—for example, because they cache frequent operations or use machine learning to classify inputs better. A one-line program like `10 PRINT`, however, is an exemplar of the legacy of computers as basic calculating machines that do the same thing with symbols each time they are run. Finally, the repetition does not have the same effect on the viewer because the context of life changes from day to day. Thus, repetitive motion may elicit different thoughts at different times.

FOR . . . TO . . . STEP

The FOR loop in BASIC is a bit more general than is shown in the example on page 96. It can be used not only to increment a variable, but also to change its value each time by any amount. A different increment is set using the STEP keyword. To show only the even numbers between 2 and 10:

```
10 FOR A=2 TO 10 STEP 2
20 PRINT A
30 NEXT
```

If STEP is omitted, it's the same as adding STEP 1. The step value can be set to a negative number or even to zero. By setting the value to 0, an unbounded FOR loop can be created:

```
10 FOR A=1 TO 10 STEP 0
20 PRINT "FOREVER!"
30 NEXT
```

This allows for an alternate version of 10 PRINT that uses a FOR loop instead of GOTO:

```
10 FOR A=1 TO 2 STEP 0 : PRINT CHR$(205.5+RND(1)); : NEXT
```

This one is slightly longer and exceeds the forty characters of a physical line but still a "one-liner" in the sense that it fits into the eighty-character logical line.

In this version, the "10" at the beginning is optional. The three statements can be entered together in immediate mode, just as one can type PRINT "HELLO" and have the PRINT statement executed immediately:

```
FOR A=1 TO 2 STEP 0 : PRINT CHR$(205.5+RND(1)); : NEXT
```

The only reason the line number is needed in the original program is as a point to branch back to. As GOTO goes, so goes the line number.

So, these are at least four ways that *Step Piece* changes through its repetition: (1) human performance changes in subtle ways from repetition to repetition; (2) documentation shows different, nonrepetitive moments; (3) human performance improves over time; and (4) the work is perceived in different contexts, even by the same viewer. *Step Piece* is exemplary here because it seems to be the more or less pure repetition of pure repetition, but actors putting on the same play on different nights encounter a similar type of nonrepeating repetition and all of these four points also apply to the case of plays that run for several performances. Even though `10 PRINT` is a performance by a digital machine, (2) and (4) still apply, so that its repetition also varies in these ways.

The performance piece *Dance*, a collaboration between choreographer Lucinda Childs, composer Phillip Glass, and artist Sol LeWitt, interweaves these performance strands together. Like all of the works already discussed, narrative is absent and the subject is literally movement, hearing, and seeing through variation and repetition (figure 30.10). For three of the five sections of the performance, LeWitt created films of the dancers performing the same sequence of motions that they perform live on stage. He filmed them on top of a grid to reinforce the structure within the choreography. The film is projected onto a scrim at the front of the stage during the performance, and it is synchronized to a recorded version of Glass's score. At times the projected image is enlarged and at times it is slowed down as the film echoes the movements of the live performance. Glass's music for *Dance* features his signature style of repeating and transforming brief passages. The foundation of *Dance* is Lucinda Childs's choreography, which she has referred to as stripped-down ballet. In the first and strongest movement, the dancers move along straight lines from left to right and right to left across the stage. As they quickly follow the line in a mix of running, skipping, and turning, they move through a series of tilts. The perception is they are moving through every permutation within the grammar of movements created for the dance. As with `10 PRINT`, the sound and motion are hypnotic and emphasize the details of the variation rather than larger global structural changes. *Dance,* however, is unlike `10 PRINT` in that it works across multiple media to synthesize a powerful and at times overwhelming aesthetic experience.

Regularity is an important technique of the mid-to-late twentieth century in part because artists explored systematic production and repeated

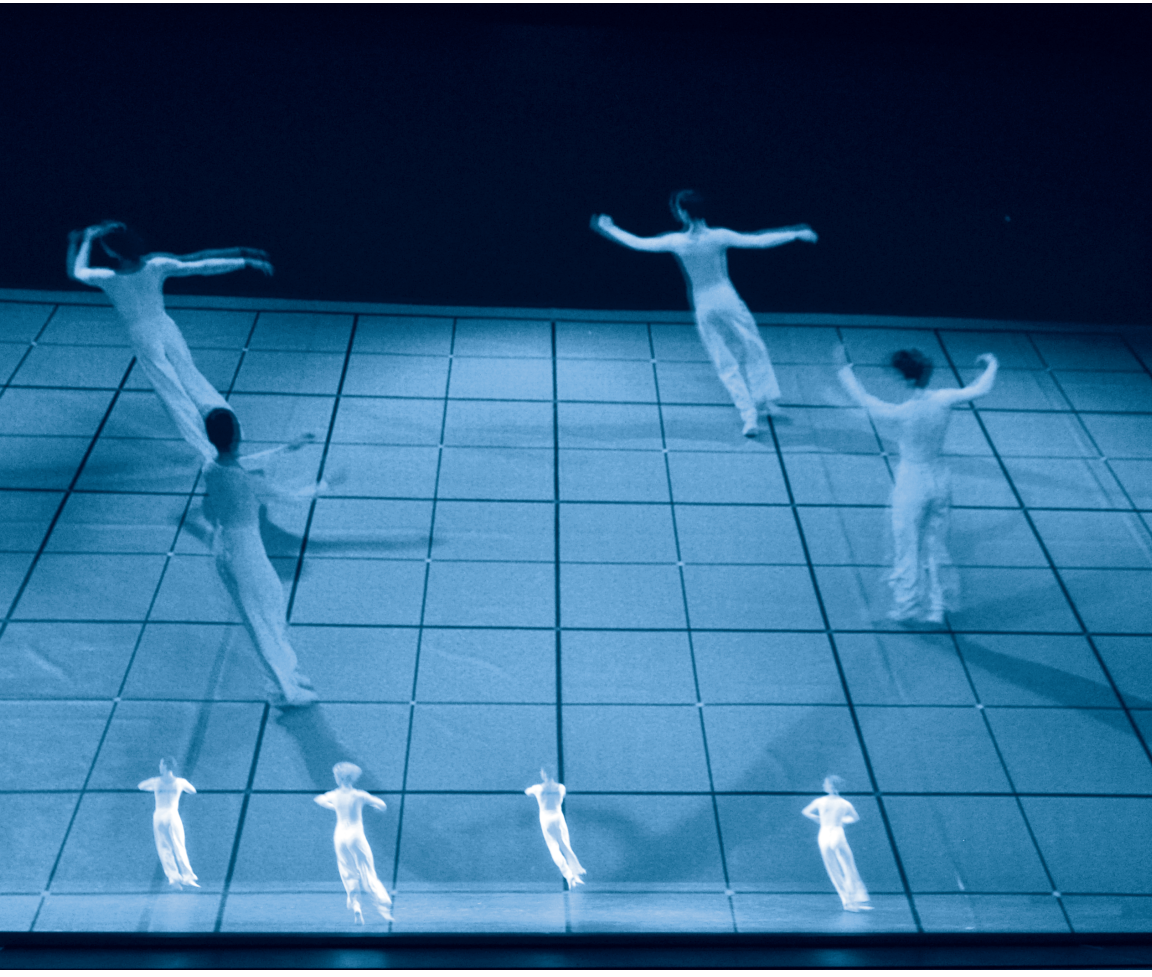

Figure 30.10
Image from the 2009 revival of *Dance*, a 1979 collaboration of Lucinda Childs, Phillip Glass, and Sol Lewitt. Photo by Sally Cohn, ©2009. Courtesy of Sally Cohn.

 10 PRINT CHR$(205.5+RND(1)); : GOTO 10

processes as epitomized by computer programs. The impact of those art forms is in the stripping away of their representational and expressive possibilities, the minimalism in their constructivist techniques. By contrast, `10 PRINT`, as an initial and initiating example for novices at the Commodore 64, does not strip away but offers an introductory glimpse of the effects of the flow and control of computer systems processing an unbounded loop within the constraints of regulated time and space. If the art world was moving away from representation and expression to this minimalism, the novice programmer was encountering `10 PRINT` as the beginning of a movement toward representation and expression, in this case within the world of computer graphics.

In the cases discussed, regularity in time, space, and process becomes programmatic, a proportional regulator that proves to be a generative constraint, producing larger patterns that can be quite unexpected. Through the unrelenting predictability of regularity, the repeated random presentation of two diagonal lines across a grid becomes a quite unanticipated scroll of mazes. As a pedagogical example in the *technē* of programming, `10 PRINT` is both a product of and a demonstration of the force of computational regularity.