Either/Or → And

07

Terry W. Knight
Massachusetts Institute of Technology, USA

07.1

Keywords:
computational design, rules, algorithms, Kandisky, Klee, non-representational art, creativity

Dr. Terry W. Knight
Associate professor of design and computation,
Department of Architecture MIT,
77 Massachusetts Avenue,
Cambridge MA 02139
USA
tknight@mit.edu
+1 617 253-8044

Introduction

This paper is about computational design, that is, about using rules and algorithms to create designs. I have therefore given a rule for the title of the paper: Either/0r → And. This means: the words "either/or" become, are replaced with, or are transformed into, the word "and". This rule encapsulates a theme that runs through all of the writings of the Bauhaus legend, Wassily Kandinsky. Kandinsky, and his colleague at the Bauhaus, Paul Klee, were pioneers of abstract, nonrepresentational art. In their writings and teachings, Kandinsky and Klee examined in depth the basic elements of composition in art and architecture. They proposed sophisticated and subtle theories of form and form production. It is thus fitting to revisit them in a discussion of morphology and design, the theme of this session.

Kandinsky and Klee covered a lot of ground in their writings. Designers, theorists, writers, and teachers have turned to them again and again over the years for their insights in many areas. When I recommended to one of my graduate students that he read Klee's Pedagogical Sketchbook, I was drawn into rereading this book myself, and then further into rereading more of Klee's and Kandinsky's writings. Not surprisingly, I found that, in many ways, they anticipated some fundamental concerns of computation and design today. So I will frame my discussion of computation and design in terms of some important and timely ideas from Kandinsky and Klee.

Foremost among Kandinsky's concerns was the reconciliation of seemingly opposing or contradictory issues and agendas. He writes of the many misguided dualisms in the art theory and pedagogy of his time, and the pressing need to bring opposing trends into one "Great Synthesis", to make connections between apparently widely separated realms. Kandinsky maintained that wherever there is talk of "*either* this *or* that", work must strive for "this *and* that". "Dispersion", he writes, must be replaced by "integration". "Either-or" must give way to "and". He believed that a new integrative trend in art was just beginning, and that "it is within art that this new trend--"*and*" first becomes apparent, but it also dawns here and there in other areas, and slowly backs will be turned on "*either-or*".

What were the dualisms that preoccupied Kandinsky? Have they been resolved? Are they relevant to design today, in particular to computational design? Here I will look at three dualisms that Kandinsky and Klee addressed, but from the perspective of computation and design. I will also look at two new dualisms that Klee and Kandinsky did not consider.

These are ones that have arisen in recent years specifically with respect to computation and design. The first three dualisms are posed as rules in which *either/or* is replaced with *and*. These are rules which can be applied in computational design. They reconcile and integrate issues often seen as incompatible in computational design. The last two dualisms are posed as possible rules. Whether and how the components of these rules can be reconciled, or whether they are fundamentally incompatible, are open questions.

The five rules are:

1 *either* analysis *or* synthesis → analysis *and* synthesis
2 *either* form *or* content → form *and* content
3 either calculation or intuition → calculation *and* intuition
4 *either* emergence *or* predictability → emergence *and* predictability ?
5 *either* intelligibility *or* productivity → intelligibility *and* productivity ?

I will consider each of these rules in turn. But first, some background.

07.2

The most well known and influential writings of Kandinsky and Klee come from the 1920s. Work on the first formal theories of computation was going on at the same time. The first publications on computation appeared in the 1930s. The names of the pioneers of computation will be familiar to some: Turing, Church, Godel, and others. These mathematicians, logicians, and philosophers proposed models of what it means formally, theoretically, and philosophically to compute something, or to solve something algorithmically. They laid the foundations for all subsequent work on computation, and for all the various paths that computation has taken over the past century.

The kind of computation I do is with shape grammars. Shape grammars were one of the earliest computational systems developed specifically for design. They were also one of the earliest *visual* computational systems for design. With a shape grammar, designs are constructed or computed using rules made up of 2 or 3 dimensional shapes. A computation with a shape grammar is a sequence of shapes where each shape is generated from the previous shape by applying a rule. By contrast, the computational models proposed by Turing and others were entirely textual or verbal. They computed with rules made up of symbols and numbers. The differences between computing with shapes as opposed to symbols and numbers cannot be overemphasized. These differences have been considered in depth by one of the inventors of shape grammars, George Stiny.

Many other computational systems for design are being used and developed today. My discussion of the rules or dualisms above will be in relation not just to shape grammars but to computation in general.

1 *either* analysis *or* synthesis → analysis *and* synthesis

Kandinsky equates analysis with "dissection" and synthesis with "connection". Analysis has to do with taking things apart and viewing the resulting parts in isolation. Synthesis has to do with putting disparate things together. Kandinsky's pictorial theory requires both analysis and synthesis. He sees "analysis as a means to synthesis."[1] Theory, he says, needs:

> 1. To specify the primary elements and designate the more diverse and complicated that issue from them--the analytical part,
> 2. The possible laws for the grouping of those elements within a work--the synthetic part.[2]

Synthesis has the narrow meaning above for Kandinsky. More broadly and importantly, though, synthesis is the putting together and connecting of disparate ideas and thoughts. For Kandinsky, the ultimate goal of art pedagogy is to make one final connection or synthesis--the "Great Synthesis", as he called it. In the Great Synthesis, "The irreconcilable is reconciled. Two opposing paths lead to one goal--analysis, synthesis. Analysis + synthesis = the *Great Synthesis*".[3]

Klee puts his own twist on analysis and synthesis. This is an emphasis on process--on understanding both analysis and synthesis in terms of growth or development. He first introduces the notion of analysis through simple examples from chemistry. He then writes, "In our business [art] the motives for analysis are naturally different [from those in chemistry]. We do not undertake analyses of works because we want to copy them or because we suspect them [as in chemistry]. We investigate the methods by which another has created his work, in order to set ourselves in motion."[4] In other words, we use analysis to set ourselves in motion to do something of our own, to do synthesis. Klee continues, "One particular kind of analysis is the examination of a work with a view to the stages of its coming-into-being. This kind I call the analysis of 'genesis'".[5] For Klee, the core of both analysis and synthesis is the dynamic processes that take us from the conception of a work of art to its completion. Klee describes these processes as akin to evolutionary or biological processes. A work of art evolves step-by-step like a living organism. For Klee, composition is, in essence, a kind of organic computation.

How are analysis and synthesis reconciled in computational design? At a fundamental level, computational systems simultaneously embody both analysis and synthesis. The relationship between computation and synthesis is straightforward. Computational systems are, by definition, creative or synthetic. They compute, generate, create, or synthesize things. A shape grammar, for example, creates designs. Shape grammars and other computational systems also capture nicely the process view of synthesis so integral to Klee's theory of art. The relationship between computation and analysis is perhaps not so obvious. However, analysis is as intrinsic to computation as synthesis. Computational systems are, themselves, analyses or descriptions of the things they generate. For example, the applications of rules in a shape grammar provide analytic decompositions--what George Stiny calls topologies[6]--of the designs they generate. Shape grammar computations are dynamic descriptions in the sense that Klee had in mind for analysis.

Computational systems inherently embrace and unify the analytic and the synthetic. Applications of computational systems also embrace the analytic and the synthetic, but at a different level. In analysis applications, computation is used to analyze or model known or existing things. In synthesis applications, computation is used to create or synthesize novel, unknown things. Often, these two kinds of applications overlap.

Within the history of computation, different strands of computation can be identified that focus more or less on analysis or synthesis. For example, early production systems from the 40s and 50s, such as those of Post in logic and those of Chomsky in linguistics, focused on analysis. Chomsky developed his generative grammars to model existing natural languages, not to generate new languages. A different strand of computation, one with a biological bent, has a decidedly different focus. This strand begins with Von Neuman's cellular automata of the 50s, continues to the evolutionary computational methods begun in the 70s, and then up to artificial life systems of today. Computational systems along this line

have been used to model existing phenomena. But they are geared to do much more. They are geared toward the creation of the new and the novel. And in recent years, they have been increasingly tapped for their creative potential. Evolutionary and artificial life systems are used today to synthesize new and compelling life-like forms, new forms of art, architecture, and engineering. These biologically-inspired computational design efforts would likely have caught the attention of Paul Klee.

Where do shape grammars sit in the assortment of computational methods and applications? Shape grammars are production systems like those of Post and Chomsky. But like other "biological" kinds of computation, shape grammar applications are equally at home in analysis and synthesis, and have had much success in both areas. Shape grammars have been developed to analyze historic and contemporary design languages in virtually all areas of visual and spatial design. They have also been used in studio or classroom settings to create new and original designs. Recent applications of shape grammars merge analysis and synthesis by beginning with the grammatical analysis of known designs as a way toward the development of grammars for new designs. In these efforts, analysis is used "to set ourselves in motion".

The space syntax techniques developed by Bill Hillier and others use computation of a nongenerative, nonrule-based kind, and thus fall outside the definition of computation I am using here. However, like early production systems, space syntax is computation that is analytically oriented. But space syntax techniques are often embedded in creative design processes to evaluate proposed designs and to point to new design possibilities.

2 *either* form *or* content → form *and* content

Here is a more familiar dualism, both for the nonrepresentational art that Klee and Kandinsky pioneered in the 20s, and for computational design today.

Klee and Kandinsky were unequivocal about form *and* content, not form *or* content. Klee's understanding of form is process-oriented. For Klee, the study of form is the study of "form-making". Klee considers form as "genesis, growth, essence".[7] "Form", he says, "is set by the process of giving form, which is more important than form itself."[8] He uses the term *Gestaltung* for the study of form because it "emphasizes the paths to form rather than the form itself."[9] These views are not unlike the design computation point of view! And like many a design computationalist, Klee wishes to "avoid the misconception that a work consists only of form."[10] Content or expression is essential. Content is the impetus for form. However, content is impossible without the appropriate forms to hold or represent it. Klee maintains that "what must be stressed even more at this point [than the misconception that work consists only of form] is that . . . the profoundest mind, the most beautiful soul, are of no use to us unless we have the corresponding forms to hand."[11]

Kandinksky frames content and form in a similar way. Kandinsky describes content and form as the inner and outer--or internal and external--elements of a work of art. He writes, "The work of art is an inevitable, inseparable joining together of the internal and external elements, of the content and the form."[12] Content, the inner element, is "the emotion in the soul of the artist."[13] Like Klee, Kandinsksy realizes that content (soul) is not enough. He continues, "[In order] for the content, which exists first of all 'in abstracto,' to become a work

of art, the second element--the external [form]--must serve as its embodiment."[14] However, it is always content that leads form: "Complete harmony between 'form' and 'content', where form = content and content = form, can only exist if it is the content that creates the form."[15]

Computational design comprises both form and content, but is indifferent to which takes the lead. Because computation is often used to generate numerous, diverse, and unpredictable forms, in seemingly mysterious and spontaneous ways, computation sometimes gives the impression that form reigns over all. But as Kandinsky observed "every form has inner content."[16] This is true for forms that are computed as well as for those that are not. The rules of a shape grammar, for instance, embody content. Expression, meaning, purpose, aesthetics, and so on all go into the creation of shape rules. For example, the Palladian shape grammar generates villa plans with a compositional element called an enfilade--this is the continuous line of windows and doors from one facade of a villa to the opposite one. Enfilade rules are not only about form--the impetus for these rules are climate and site conditions. Here, it is clearly "content that creates form".

Computational design can handle content, and relationships between form and content, in other ways. Content can be expressed through form; it can be also be expressed and described through words, numbers, or symbols-- through poetry, for example. Textual or symbolic representations of content can be manipulated and generated computationally. (Recall that the earliest computational systems were textual or symbolic.) Computations on representations of content can be connected or networked to other computations--for example, computations on form. These networked computations are often called "parallel computations." Parallel computations are computations that take place simultaneously. With parallel computation, shape rules that encode form can be linked to text rules that encode content. Links between rules can be defined freely: content can drive form, form can drive content, or the two can operate independently. For example, a parallel grammar has been developed recently for the Malagueria housing designs of the architect Alvaro Siza. The parallel grammar generates house designs and simultaneously generates descriptions of the costs, layouts, and other programmatic features of houses. In this case, content (specified by a user) drives the form of a house.

Evolutionary or genetic computation can be considered a kind of parallel computation where form is linked to, and driven by, content. Here, content is specified by objectives called fitness functions.

3 *either* calculation *or* intuition → calculation *and* intuition

These two concepts are always difficult to define. Kandinsky and Klee equate calculation with conscious processes, thinking, reasoning, logic, and mathematics. They equate intuition with subconscious processes, inspiration, and the irrational. Are calculation and intuition at odds then with one another? Not for Klee:

> We construct and keep on constructing, yet intuition is a good thing. You can do a good deal without it, but not everything. Where intuition is combined with exact research it speeds up the progress of research. Exactitude winged by intuition is at times best.[17]

Kandinsky speaks similarly, but in a biological vein:

> . . . the actual making of art has a perpetual goal, the creation of artistic entities, which resemble organisms. Would it be permissible to create organisms that lack head or heart? Head or heart, the conscious or the subconscious, calculation and intuition--what might be thrown overboard?[18]

Neither, of course. Kandinsky calls instead for "for a balance of creative forces, which may be divided up under two schematic headings--intuition and calculation"[19]. He insists on "the necessary, simultaneous application of intuition and calculation."[20], and warns "Woe betide him who relies solely on mathematics--on reason."[21]

Where are intuition and calculation in computational design? Are they balanced? The calculation component is obvious. With a shape grammar, rules apply step by step to add and subtract shapes in much the same way that numbers are added and subtracted in an arithmetical calculation. Other kinds of calculation--other kinds of conscious, logical, or mathematical processes--underlie other kinds of computational design systems.

The role of intuition in computational design is perhaps not as obvious. But computational design, especially good computational design, is impossible without it. Intuition, inspiration, and guesswork--all of these play a part in every phase of computational design. Intuition is critical in designing a computational system. Whatever subconscious processes go into the making of a great building or work of art, also go into the making of a great shape grammar. Intuition is critical in using a computational system. Whenever user input or choice is called for, so is intuition. The user of a shape grammar, for instance, may have to choose rules to apply and how to apply them. The user of an evolutionary design system may have to choose or define fitness functions that drive the generation of designs. Intuition is as crucial to these decisions as calculation. Intuition is also critical in selecting from the output of a computational design system. Whenever multiple design solutions are presented, the evaluation and choice of solutions require intuition.

Kandinsky tells an instructive story about the role of calculation in art, where calculation is equated with thinking. The story carries a lesson for computational design as well. Kandinsky recalls a drawing instructor he had when he was a grammar school boy. The instructor would often tell his pupils: "Boys, drawing is a difficult thing. It's not like Latin or Greek--here, you have to think!"[22] Much later, as a young man, Kandinsky studied drawing under another, then famous, art instructor. This teacher required students to know anatomy thoroughly in order to draw. But the teacher insisted: "woe betide you if you think about anatomy in front of the easel! When he's working, the artist shouldn't think!"[23] Kandinsky concludes: "I have followed these two suggestions to this day and have remained true to them to the end."[24]

4 *either* emergence *or* predictability → emergence *and* predictability ?

The relationship between emergence and predictability in art was not considered explicitly by either Kandinsky or Klee. However, notions of emergence and predictability underlie much of their writings, and are intrinsic to any creative design process. Computation has amplified these two aspects of design, and brought to the fore questions about their reconcilability.

Emergence is a concept widely associated today with computation. However, the origins of the concept date back to the 19th century. In 1843, the British philosopher John Stuart Mill noted a unique property of chemical bonding: he observed that the sum of the properties of individual chemical components does not produce or predict the effects of actually combining the components. Mills's work led to the development of a new school of philosophy called British Emergentism. George Henry Lewes coined the term "emergence"

in 1875 to describe unpredictable effects or characteristics such as those observed by Mill.[25] British Emergentism was most active around the time that Klee and Kandinsky were working out their new theories of abstract art. Perhaps it is not coincidental then that Kandinsky chose an analogy from chemistry to explain the emergent properties that arise from the combination of the basic components of art:

> . . . the sounds and characteristics of the component elements [of a composition] produce in individual instances a sum total of qualities not covered by the former. Comparble facts are not unknown in other sciences, e.g., chemistry: the sum of the component elements when separated is not the same as the total produced by their combination. In such cases, we are perhaps confronted with an unknown law, whose indistinct features strike us deceptively.[26]

The concept of emergence in vogue in recent years retains the hallmarks of emergence laid out more than a century ago. In a recent book devoted entirely to the subject, John Holland writes that emergence "occurs only when the activities of the parts do not simply sum to give activity of the whole. For emergence, the whole is indeed more than the sum of its parts."[27] Frequently given examples, by Holland and many others, of systems that embody emergence are complex, nonlinear, self-organizing systems ranging from economic systems to ant colonies to board games. Holland's observation that the emergent behavior of such systems gives the "sense of much coming from little"[28] and that "This feature also makes emergence a mysterious, almost paradoxical, phenomenon"[29] recalls Kandinsky's sense of a vague, unknown law behind emergence.

In keeping with his empiricist views, the 19th century philosopher Lewes hoped that some day the mysterious, "unseen process" of emergence could be expressed in a mathematical formula.[30] Lewes's belief might be satisfied by today's computational models of emergence. In fact, the new wave of interest in emergence is attributable in part to the work of mathematicians and computer scientists on computation, and probably more so to the engineering of fast computers to support computation. Rule-based systems from cellular automata to shape grammars to genetic algorithms and artificial life can be used to generate and understand a wide range of emergent behaviors and objects. With some systems, emergence is hierarchical: simple rules direct local interactions among low-level elements to generate complex higher-level, global objects or behaviors. With other systems--for example, shape grammars--emergence is nonhierarchical: emergent objects are those that are not explicitly represented or identified in the rules used to generate them. Emergent objects may be simple, complex, or anything in between.

The seemingly spontaneous emergence of phenomena not explicitly built into rules makes computational emergence engaging to the beholder, and also very attractive for creative design. Indeed, computational systems have great potential in design exploration to generate new, "emergent" designs. These designs range from new but conventional designs that can be generated more readily with computation than without, to novel and unconventional designs that are near impossible to create without computation.

Emergence is a natural consequence of computation. Unpredictability, which often goes hand in hand with emergence, is also a natural consequence of computation. The Turing machine--one of the earliest and most famous embodiments of computation--is inherently unpredictable. The Turing machine is a theoretical description of a computing device. It was defined by the mathematician Alan Turing in the 1930s. A wide range of questions about the

behaviors and outcomes of Turing machines have been shown to be unanswerable or undecidable. The same is true of any computational system equivalent to a Turing machine, from cellular automata to shape grammars. Given any shape grammar, for example, there is no general method for predicting how the rules of the grammar will behave and what they will generate. Thus, predictability, which is not a consequence of computation, may be at odds with emergence, which is.

But good design requires both emergence and predictability. A real world design problem requires design solutions that are new, and that predictably satisfy the constraints and criteria of the problem (an architectural program, for example). Computation readily generates new, emergent designs. But because computation is generally unpredictable, designs that satisfy constraints may be difficult to find or generate.

Unpredictability is made tractable in different ways in different computational systems. Evolutionary computation and other heuristic search and optimization techniques are typically implemented on a computer. They use the power and speed of the computer to repetitively generate and test massive numbers of possibilities until satisfactory solutions are found. The problem of predictability is sidestepped with the brute force of the computer, and with the more subtle and careful formulation of criteria to be tested against. Shape grammars, on the other hand, are often implemented by hand. By-hand computation requires the subtle and careful formulation of rules that will predictably meet the constraints of a design problem. While shape grammars in general are unpredictable, useful predictions are often possible for specific or restricted kinds of shape grammars. However, more restricted grammars are less powerful and less likely to exhibit novel, emergent behaviors.

Finding a balance between emergence and predictability in computational design is a delicate task.

5 *either* intelligibility *or* productivity → intelligibility *and* productivity ?

Here is the last dualism I will consider. It is a dualism not considered by Klee and Kandinsky. It is unique, perhaps, to computational design. However, it is deeply intertwined with issues of emergence and predictability. And, it returns, full circle, to issues of analysis and synthesis.

Computation is used both to analyze and interpret existing phenomena, and to invent and synthesize new phenomena. Intelligibility is a requirement of analysis. In analysis, the rules of a computational system must be understandable. They must describe in an intelligible way the things they generate. For instance, the logician Emile Post developed production systems in the 1940s to understand problems of logic and mathematics. To this end, the rules of a production system are required to be intelligible--they must describe, intelligibly, solutions to questions. In the 1950s, Chomsky developed generative grammars to explain the structure of natural languages. The rules of a generative grammar must be intelligible--they must describe intelligibly how sentences are put together. Stiny and Gips's shape grammars were also developed with intelligibility in mind. The rules of a shape grammar are meant to explain the structure of the designs they generate. A long standing criterion of a well-crafted shape grammar (and of old-fashioned computer programs) has been the intelligibility of the rules. If the rules are not understandable, then neither are the designs generated by them.

Computational systems that are geared only toward invention or synthesis have no obligation to be intelligible. Productivity is the requirement of these systems. They are obliged to produce correct, appropriate, or novel results for the problem at hand, and to

produce these results reliably. Evolutionary computation, for example, is oriented toward productivity. And productivity is made possible only through the speed and power of current computers.

Daniel Hillis, a pioneer of massively parallel computing, the inventor of the Connection Machine, and a champion of evolutionary computation, understands well the conflicts between productivity and intelligibility. In a recent book, he talks about his experiments using simulated evolution to develop computer programs for sorting numbers. He doesn't know how his evolved programs work, but he knows that they will reliably produce the kinds of results he wants. In fact, his evolved programs will produce more trustworthy results than traditional programs because they have been tested over many thousands of evolutionary iterations. Hillis writes:

> One of the interesting things about the sorting programs that evolved in my experiment is that I do not understand how they work. I have carefully examined their instruction sequences, but I do notunderstand them: I have no simpler explanation of how the programs work than the instruction sequences themselves. It may be that the programs are not understandable--that there is no way to break the operation of the program into a hierarchy of understandable parts.[31]

Hillis further connects intelligibility to predictability. Evolutionary computation effectively dodges the problem of predictability, but with negative consequences for intelligibility, for the "why" of a design:

> Simulated evolution is a good way to create novel structures, but it is an inefficient way to tune an existing design. Its weaknesses as well as its strengths stem from evolution's inherent blindness to the "Why" of a design . . . evolution chooses variations blindly, without taking into account how the changes will affect the outcome.[32]

Margaret Boden, a commentator on artificial intelligence and artificial life, has made similar observations about evolutionary computation. She ties intelligibility to predictability, and also uses it to characterize emergence. She distinguishes between "intelligible emergence" and "unintelligible emergence". Emergence is intelligible if it can be understood by examining the rules that produce it. It is unintelligible if it cannot be understood or predicted, even with access to the rules and to the computation. Boden talks about the work of the digital artist, Karl Sims, who generates images of artificial creatures using genetic algorithms:

> Sims's computer-generated images give us an example of the latter [unintelligible emergence]. . . Sims himself cannot always explain the changes he sees appearing on the screen before him, even though he can access the mini-program responsible for any image he cares to investigate . . . Often he cannot even 'genetically engineer' the underlying LISP expression so as to get a particular visual effect. To be sure, this is partly because his system makes several changes simultaneously, withevery new generation. If he were to restrict it to making only one change, and studied the results systematically, he could work out just what was happening. But when several changes are made in parallel, it is often impossible to understand the generation of the image . . . [33]

As Boden suggests, intelligibility is partly a function of the complexity of a program and the speed of its implementation. Computation done by hand--slow, old-fashioned computation--requires intelligibility. Appropriate and trustworthy results can only be generated by

understanding the rules. Not so for high-speed computation on a computer. But if we are concerned only with synthesis, invention, or results, is intelligibility all that important? And is intelligibility fundamentally incompatible with productivity in much of current computational design anyway?

Answers to these questions may come with more practical experience using computation for designing, and with the ongoing development of computational systems and theories to support them. The complexity of conflicting issues and agendas, either apparent or real, in computational design go well beyond the five dualisms I have considered here. But perhaps in considering these issues we can try, in Kandinsky's words, to "desert the petrified atmosphere of 'either-or' for the flexible, living atmosphere of 'and'"[34].

Acknowledgment

I would like to thank Sotirios Kotsopoulos, whose inquiries into shape grammars prompted me to revisit the writings of Klee and Kandinsky.

Notes

1 Lindsay K C and P Vergo (Eds), 1994 *Kandinsky: Complete Writings on Art* (Da Capo Press) 724
2 *Ibid.* 852
3 *Ibid.* 479
4 Spiller J (Ed), 1961 *Paul Klee: The Thinking Eye* (New York: George Wittenborn) 99
5 *Ibid.* 99
6 Stiny G, 1994, "Shape rules: closure, continuity, and emergence" *Environment and Planning B: Planning and Design* **21** s49-s78
7 Spiller J (Ed), 1973 *Paul Klee Notebooks:Volume 2, The Nature of Nature* (New York: George Wittenborn) 269
8 *Ibid.* 269
9 Spiller, *Paul Klee: The Thinking Eye,* 17
10 *Ibid.* 100
11 *Ibid.* 100
12 Lindsay and Vergo, *Kandinsky*, 350
13 *Ibid.* 349
14 *Ibid.* 349-350
15 *Ibid.* 482
16 *Ibid.* 165
17 Spiller, *Paul Klee: The Thinking Eye*, 69
18 Lindsay and Vergo, *Kandinsky*, 485
19 *Ibid.* 535
20 *Ibid.* 601n.
21 *Ibid.* 758
22 *Ibid.* 735
23 *Ibid.* 736
24 *Ibid.* 736
25 Lewes G H, 1875 *Problems of Life and Mind* (Boston: Osgood and Company), 368
26 Lindsay and Vergo, *Kandinsky*, 592
27 Holland J H, 1998 *Emergence: From Chaos to Order* (Cambridge, MA: Perseus Books) 14
28 *Ibid.* 2
29 *Ibid.* 2
30 Lewes, *Problems*, 370
31 Hillis D W, 1998 *The Pattern on the Stone* (New York NY: Basic Books) 146-147
32 *Ibid.* 148
33 Boden M A (Ed), 1996 *The Philosophy of Artificial Life* (Oxford UK: Oxford University Press) 103
34 Lindsay and Vergo, *Kandinsky*, 724