Computer Art and the Theory of Computation

Originally published in hz-journal.org

Jim Andrews
vispo.com

Computer Art and the Theory of Computation

Jim Andrews
vispo.com

The Blooming

What I'd like to do is to explore the relevance of the *theory of computation to computer art*. Both of those terms, however, need a little unpacking/explanation before talking about their relations.

Let's start with *computer art*. Dominic Lopes, in *A Philosophy of Computer Art*, makes a useful distinction between *digital art* and *computer art. Digital art,* according to Lopes, can refer to just about any art that is or was digitized. Such as scanned paintings, online fiction, digital art videos, or digital audio recordings. *Digital art* is not a single form of art, just as fiction and painting are different forms of art. To call something digital art is merely to say that the art's representation is or was, at some point, digital. It doesn't imply that computers are necessary or even desirable to view and appreciate the work.

Whereas the term *computer art* is much better to describe art in which the computer is crucial as medium. What does he mean by "medium"? He says "a technology is an artistic medium for a work just in case it's use in the display or making of the work is relevant to its appreciation" (p.15). We don't need to see most paintings, texts, videos or audio recordings on computers to display or appreciate them. The art's being digital is irrelevant to most digital art. Whereas, in computer art, the art's being digital is crucial to its production, display and appreciation.

Lopes also argues that whereas *digital art* is simply not a single form of art, *computer art* should be thought of as a new form of art. He thinks of a form of art as being a kind of art with shared properties such that those properties are important to the art's appreciation. He defines interactivity as being such that the user's actions change the display of the work itself. So far so good. But he identifies the crucial property that works of computer art share as being interactivity.

I think all but one of the above ideas by Lopes are quite useful. The problem is that there are non-interactive works of computer art. For instance, generative computer art is often not interactive. It often is different each time you view it, because it's generated at the time of viewing, but sometimes it requires no interaction at all. Such work should be classified as computer art. The computer is crucial to its production, display, and appreciation.

Lopes's book is quite useful in a number of ways. It's the first book by a professional philosopher toward a philosophy of computer art. It shows us how a philosophy of computer art might look and proceed. But it is a philosophy, in the end, of *interactive* computer art. Which is a more limited thing than a philosophy that can also accommodate *non-interactive* computer art.

Now, why did a professional philosopher writing a philosophy of computer art fail to account for non-interactive computer art in his philosophy? Well, to write a more comprehensive philosophy requires an appreciation of the importance of programmability. For it is programmability, not interactivity, that distinguishes computer art from other arts. And it is at this point that we begin to glimpse that some understanding of the theory of computation might be relevant to an understanding and appreciation of computer art.

I'll return to this point in another section. I've given you some idea of what I mean by computer art. Now let's have a look at the theory of computation.

It was inaugurated in the work of the mathematician and logician Alan Turing in 1936 with his world-shaking paper entitled "On Computable Numbers, with an Application to the Entscheidungsproblem". This is one of the great intellectual documents of the twentieth century. In this paper, Turing invented the modern computer. He introduced us to what we now call the *Turing machine*, which is an abstract idea, a mathematization, an imaginary object that has all the theoretical capabilities of modern computers. As we know, lots of things have changed since then in how we think about computers. However, the Turing machine has not changed significantly and it is still the bedrock of how we think about computers. It is still crucial to our ability to think about the limits of the capabilities of computers.

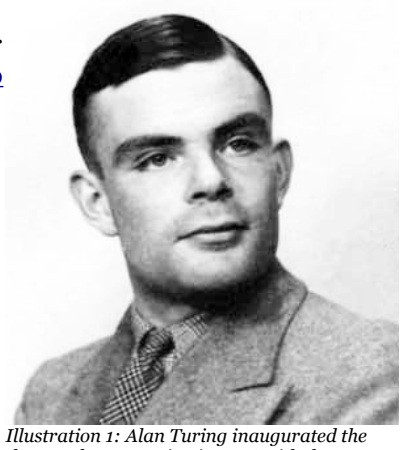

Illustration 1: Alan Turing inaugurated the theory of computation in 1936 with the most humble but powerful manifesto of all time.

And that is precisely what the theory of computation addresses: the limits of the capabilities of computers. Not so much today's limits, but theoretical limits. The theory of computation shows us what is *theoretically* possible with computers and what is theoretically impossible. "Theoretically impossible" does not mean "probably won't happen". It means "will absolutely never ever (not ever) happen as long as the premises of the theory are true".

Since we are dealing with matters of art, let's first get a sense of the poetry of the theory of computation. It's a little-appreciated fact that Turing devised the Turing machine in order to show that there are some things it will never be capable of doing. That is, he devised the modern computer not to usher in the age of computers and the immense capabilities of computers, but to show that there are some things that no computer will ever do. That's beautiful. The theory of computation arises not so much from attempts to create behemoths of computation as to understand the theoretical limits of the capabilities of computing devices.

If you wish to prove that there are some things that no computer will ever do, or you suspect that such things do exist, as did Turing—and he had good reason for this suspicion because of the earlier work of Kurt Gödel—then how would you go about proving it? One way would be to come up with a computer that can do anything any conceivable computer can do, and then show that there are things it can't possibly do. That's precisely how Turing did it.

Why was he more interested in showing that there are some things no computer will ever

do than in inventing the theoretical modern computer? Well, in his paper, he solves one of the most famous mathematical problems of his day. That was closer to the intellectual focus of his activities as a mathematician and logician, which is what he was. The famous problem he solved was called the Entscheidungsproblem, or the decision problem. Essentially, the problem, posed by David Hilbert in 1928, was to demonstrate the existence or non-existence of an algorithm that would decide the truth or falsity of any mathematical/logical proposition. More specifically, the problem was to demonstrate the existence or non-existence of an algorithm which, given any formal system and any proposition in that formal system, determines if the proposition is true or false. Turing showed that no such algorithm can exist.

At the time, one of the pressing problems of the day was basically whether mathematics was over and done with. If it was theoretically possible to build a computer that could decide the truth or falsity of any mathematical/logical proposition, then mathematics was finished as a serious intellectual enterprise. Just build the machines and let them do the work. The only possibly serious intellectual work left in mathematics would be meta-mathematical.

However, Turing showed that such an algorithm simply cannot exist. This result was congruent with Kurt Godel's earlier 1930 work which demonstrated the existence in any sufficiently powerful formal system of true but unprovable propositions, so-called undecidable propositions. In other words, Godel showed that there are always going to be propositions that are true but unprovable. Consequently, after Godel's work, it seemed likely that no algorithm could possibly exist which could decide whether any/every well-formed proposition was true or false.

We glimpse the poetics of the theory of computation in noting that its historical antecedant was this work by Gödel on unprovable truths and the necessary incompleteness of knowledge. The theory of computation needed the work of Gödel to exist before it could bloom into the world. And let us be clear about the nature of the unprovable truths adduced by Godel. They are not garden-variety axioms. Garden-variety axioms, such as the parallel postulate in geometry, are *independent*. That is, we are free to assume the proposition itself or some form of the negation of the proposition. The so called *undecidable* propositions adduced by Gödel are true propositions. We are not at liberty to assume their negation as we are in the case of independent axioms. They are (necessarily) true but unprovably true. And if we then throw in such a proposition as an axiom, there will necessarily always be more of them. Not only do they preclude the possibility of being able to prove every true proposition, since they are unprovable, but more of them cannot be avoided, regardless of how many of them we throw into the system as axioms.

Sufficiently rich and interesting formal systems are necessarily, then, incomplete, in the sense that they are by their nature incapable of ever being able to determine the truth or falsity of all propositions that they can express.

So the theory of computation begins just after humanity becomes capable of accomodating a deep notion of unprovable truth. Of course, unprovable truth is by no means a new concept! We have sensed for millenia that there are unprovable truths. But we have only recently been able to accommodate them in sensible epistemologies and mathematical analysis. Unprovable truths are fundamental to poetry and art, also. We know all too well that reason has its limits.

The more we know about ourselves, the more we come to acknowledge and understand our own limitations. It's really only when we can acknowledge and understand our own limitations that we can begin to do something about them. The first design for a so-called Turing-complete computer—that is, a computer that has all the theoretical capabilities of a Turing machine—pre-dated Turing by a hundred years: Charles Babbage's Analytical

Engine. But Babbage was never able to create his computer. It was not only a matter of not being able to manufacture the parts in the way he wanted, but he lacked the theory of computation that Turing created. A great theory goes a long way. The Turing machine is simplicity itself. Children can understand it. We think of computers as intimidatingly complex machines, but their operation becomes much more understandable as Turing machines.

We could have had computers without the theory of computation, but we wouldn't have understood them as deeply as we do, wouldn't have any sense of their theoretical limitations—concerning both what they can and can't do. And we wouldn't have been able to develop the technology as we have, because we simply wouldn't have understood computers as deeply as we do now, wouldn't have been able to think about them as productively as we can with the aide of a comprehensive theory. Try to put a man on the moon without Newtonian mechanics. It might be doable, but who would want to be on that ship? Try to develop the age of computers without an elegant theory to understand them with? That sounds more like the age of the perpetual blue screen.

Gödel's incompleteness theorems are not logically prior to Turing's work. In other words, Turing's work does not logically depend on Gödel's work—in fact, incompleteness can be deduced from Turing's work, as computer scientists sometimes point out. But it was Gödel's work that inspired Turing's work. Not only as already noted, but even in its use of Georg Cantor's diagonal argument. Gödel's work was the historical antecedent of Turing's work. Gödel's work established a world view that had the requisite epistemological complexity for Turing to launch a theory of computation whose epistemological capabilities may well encompass thought itself.

The theory of computation does not begin as a manifesto declaring the great capabilities of computers, unlike the beginnings of various art movements. Instead, it begins by establishing that computers cannot and simply will never ever solve certain problems. That is the main news of the manifesto; it means that mathematics is not over, which had been a legitimate issue for several years. Were computers going to take over mathematics, basically? Well, no. That was very interesting news. You don't often get such news in the form of mathematical proofs. News that stays news. The other news in the manifesto is almost incidental: oh, by the way, here is a mathematization of all conceivable machines—here is the universal machine, the machine that can compute anything that any conceivable machine can compute.

His famous paper is the foundation for the theory of computation. He put the idea of the computer and the algorithm in profound relation with Gödel's epistemologically significant work. He wrote the philosophical foundation for the theory of computation, establishing that it does indeed have important limitations, and he also provided us with an extraordinarily robust mathematization of the computer in the form of the Turing Machine.

Turing's paper is significant in the history of mathematics. We see now that the development of the computer and the theory of computation occurs after several hundred years of work on the "crisis of foundations" in mathematics and represents a significant harvest or bounty from that research. At least since the seventeenth century, when bishop Berkeley famously likened Newton's treatment of some types of numbers in calculus to "the ghosts of departed quantities", and especially since the birth pains in the eighteenth century of non-Euclidean geometry, mathematicians had understood that the foundations of mathematics were vaguely informal, possibly contradictory, and needed to be formalized in order to provide philosophical and logical justification and logical guidelines in the development of mathematics from first principles.

There's a straight line from that work to the work of Frege, Cantor, and Gödel. And thence to Turing. The theory of computation, it turns out, needed all that work to have been done

before it could bloom. It needed the philosophical perspective and the tools of symbolic logic afforded by that work. Because the theory of computation is not simply a theory of widgets and do-dad machines. At least since the time of Leibniz in the seventeenth century, the quest to develop computing devices has been understood as a quest to develop aides to reason and, more generally, the processes of thought.

The Turing Machine and the theory of computation provide us with machines that operate, very likely, at the atomic level of thought and mind. Their development comes after centurys of work on the philosophical foundations of mathematics and logic. Not to say that it's flawless. After all, it's necessarily incomplete and perhaps only relatively consistent, rather than absolutely consistent. But it's good enough to give us the theory of computation and a new age of computers that begins with a fascinatingly humble but far-reaching paper entitled "On Computable Numbers, with an Application to the Entscheidungsproblem" by Alan Turing.

It changes our ideas about who and what we are. Computer art, without it, would be utterly different. Just as would the world in so many ways.

As a computer artist, I see this history as part of the intellectual heritage of computer art. It's not simply a history of the development of a machine. It's a rich philosophical history that is fundamentally concerned with who and what we are.

Greenberg, Modernism, Computation and Computer Art

In a short but influential piece of writing by Clement Greenberg called *Modernist Painting* written in 1960—and revised periodically until 1982—the art critic remarked that

> "The essence of Modernism lies, as I see it, in the use of characteristic methods of a discipline to criticize the discipline itself, not in order to subvert it but in order to entrench it more firmly in its area of competence."

Such sweeping generalizations are always problematical, of course. But I want to use the Greenberg quote to tell you an equally problematical story about the birth of the theory of computation and, thereby, computer art. Humor me. It's Clement Greenberg. Come on.

The work I've mentioned by Gödel and Turing happened in the thirties, toward the end of modernism, which was roughly from 1900 till 1945, the end of World War II. So it's work of late modernism. Though, of course, Greenberg's version of modernism has a different timeline.

Let's grant Greenberg clemency concerning his conceit, for the moment, that the "essence"—itself a word left over from previous eras—of modernism, of the art and culture of that era, at least in the west, involved a drive to a kind of productive self-referentiality or consciousness of the art itself within the art itself. What work could possibly be more exemplary of that inclination than the work by Gödel and Turing that I've mentioned?

Turing's paper in which he invents the modern computer and solves the Entscheidungsproblem is profoundly meta-mathematical; the Entscheidungsproblem, as already noted, is a problem of meta-mathematics. And its argument also involves

interesting self-referenciality, as is pointed out in plato.stanford.edu/entries/turing:

> Turing's proof can be recast in many ways, but the core idea depends on the self-reference involved in a machine operating on symbols, which is itself described by symbols and so can operate on its own description.

Self-reference is crucial also to Godel's proof in which, among other things, the proposition "This proposition is not provable" is shown to be necessarily true but, yes, unprovable, and exemplary of a kind of proposition which Godel calls "undecidable". Self-reference is an implicit mode of meta-mathematics because mathematics/logic is used to inquire into the nature or properties of mathematics/logic, though that doesn't need to be any more deeply self-referential than using language to inquire into the properties of language; what else are you gonna use? Your big toe? I don't think so.

So we can see the work of Godel and Turing as a kind of profound culmination of modernism's "use of characteristic methods of a discipline to criticize the discipline itself". Greenberg sees modernism as involving a meta mode of art and thought, this growing self-critical, self-aware tendency in art. Modernism culminates in the work of Godel and Turing and the consequent development of the computer, a machine that operates, as it were, at the atomic level of thought. The culmination of that self-critical, self-aware meta mode of modernism in which the work of art is aware of itself, in a sense, is the creation of a type of machine that may indeed quite literally possess the capability of becoming self-aware.

The computer age and, of course, computer art commences at the end of the modern era, signaled by the end of a world war, the invention of the theory of computation, and the atomic bomb, an understanding of the fierce chemistry of the atom, the utterly micro, at the level of the chemistry of the sun, of Apollo, of Ra, the yay very large. Looking inward and looking outward.

That knowledge of the sun's chemistry and its harnessing in the creation of the atomic bomb does indeed reveal important things about our own nature, but the whole subject is not centrally about humans and human capacities. Whereas the theory of computation is a theory inaugurated by Turing as very explicitly being about human capacities. Recall that the Entscheidungsproblem, or the decision problem, was to demonstrate the existence or non-existence of an algorithm that would decide the truth or falsity of any mathematical/logical proposition. A large part of the difficulty of this problem was in coming to the best possible formulation of what we mean by "algorithm". As we read at plato.stanford.edu/entries/turing:

> Turing's purpose was to embody the most general mechanical process as carried out by a *human* being. His analysis began not with any existing computing machines, but with the picture of a child's exercise book marked off in squares. From the beginning, the Turing machine concept aimed to capture what the human mind can do when carrying out a procedure.

From the start, the whole project of computing has been about *us*. And involves abstracting the process of thought into its constituent atoms, as it were. Computer art, a new form of art, goes beyond Greenberg's meta imperative into an art that could possibly create works in which the objects *do actually think*. And *do* actually create art. The human as the meta artist; the program as the artist. A situation where, no, art is not over and the only serious artistic work left is meta art, but the liveliness and self-consciousness of the object envisioned in Greenberg's vision of modernism is taken to the next level in the age of computers.

Programmability

I said earlier that it's programmability, not interactivity (or anything else) that is the crucial matter to consider in computer art. I want to explain and explore that claim in this section.

What makes computer art computer art? We've seen that there is a great deal of art that appears on computers that could as well appear on a page or on a TV, in a canvas or on an album. I'm calling that art digital art and computers are not crucial to the display or appreciation of it.

The idea I want to capture in the notion of 'computer art' is art in which computers are crucial for the production, display and appreciation of the art, art which takes advantage of the special properties of computers, art which cannot be translated into other media without fundamentally altering the work into something quite different than what it was on the computer, art in which the computer is crucial as medium.

And we've noted that interactivity is not sufficient to characterize the special properties of computer art, where interactivity means that the display itself is actually changed/affected by the actions of the user/viewer. We can imagine theatre, for instance, that is responsive to input from the audience. Interactivity is not unique to computers. But there's no question it's a very important part of many works of computer art. Also, although interactivity is not unique to computers, interactivity is the basis of a whole type of computer art from games to instrument-like devices to communication devices. Whereas interactivity does not have such a crucial and definitive role in other forms of (non-computer) art. The computer's ability to respond almost instantaneously to increasingly complex situations with sometimes deeply considered, highly conditional responses is, well, unique. Interactivity itself is not, per se, unique to computing, but some of the ways computers can react interactively are unique. Computers can guide us instantaneously through a whole interactive virtual world of illusions at every point and maintain the illusion of that world's existence by constructing it, moment to moment, based on our navigational decisions. So there is no question that interactivity is often the crucial property of computer art. In that it's the interactivity we often think about most deeply in our appreciation of the work. Interactivity can be experiential, immediate, engrossing and, yes, immersive in ways that art rarely is.

However, as I pointed out, there are types of computer art that art not interactive at all. Generative computer art is often not interactive. It's often different each time you see it. The art is generated by a computer program. This sort of work needs to be considered as computer art because the art requires a computer to be generated at all.

So if interactivity is not sufficient to characterize the special properties of computer art, what is?

Well, in both interactive computer art and generative art, the computer's programmability is crucial. Interactivity requires programmability. Interactivity requires conditional responses. That's fundamental to programming. And, in generative art, the art is different each time because of conditional programming.

Programmability is what separates computers from other types of machines. It's important to understand this to have any idea about what a computer really is. Non-programmable machines may have some very slight ability to react conditionally. The light in a refrigerator comes on only when the door is opened and goes off only when the door is closed. That's conditional behavior. When you put a penny in a pop machine you

get no pop but you do if you put in enough money. That's also conditional behavior.

But computers can be as labyrinthine as we ourselves are, sometimes, in our conditional responses to life. Computers can make decisions. What sort of decisions? Think of the sort of decision computers can make as being like atoms. They're small and you don't notice them. But put lots of them together and you get the sort of decisions humans make, just like when you put lots of atoms together you get stuff we can see. Basically, computers can only decide if two numbers are equal or one number is bigger than another number. And then do different things depending on the outcome. But just like everything is made out of atoms, all decisions can be made of lots of smaller decisions.

It's programmability that allows computers to exhibit conditional behavior that ranges from the obviously just plain-old mechanical, like a fridge or a toaster's behavior is mechanical, to decisions and behavior predicated on millions of lines of programming that not even the programmers can anticipate. Deep Blue, the famous chess program, plays a better game of chess than any of its programmers.

Historically, the drive to create programmable machines arose from the need to make machines flexible. When you go to the time, trouble, and expense of building a machine to perform a task, you would like it to be capable of related tasks. If the machine is a loom, for instance, to weave fabric, you would like the loom to be capable of weaving many different types of cloth that display many different types of patterns. Rather than having to have a different machine for each of them. How do you achieve the utmost in such flexibility? Well, you make the action of the machine, at any stage in its operation, dependent on decisions carried out by a program that guides the machine's actions toward the completion of the particular desired task. And this program is what you change when you want the machine to do something different. You don't change the machine; you change the program.

In another section, we'll look more closely at just what a program is, and what a computer is, exactly. We'll discuss the idea of the Turing machine, the universal computer, and precisely how it works. It's quite simple, actually. Children can understand it. The Turing machine is an imaginary, theoretical computer. It is our model of every computer that has ever existed and may ever exist. In any case, the Turing machine is the ultimate jackpot of machine flexibility. In the 75 years that have elapsed since it was first invented by Turing, nobody has been able to come up with a similarly imaginary, theoretical computing device that can do more than the Turing machine.

The Turing machine is profoundly flexible, as a machine. It is flexible to the point that there is no proof, and probably never will be, that there exist thought processes of which humans are capable and computers are not. Which is to say that it's very likely that computers are flexible to the point of being capable of thought itself.

And this flexibility is completely predicated on programmability. Programmability, not interactivity or anything else, is what distinguishes computers from other machines. Programmability is the fundamental distinguishing characteristic of computers.

Well so what? Of what importance is this to computer art?

Recall that I'm defining computer art to be art in which the computer is crucial as medium. That is, the computer is crucial to the display and appreciation of the work. The art can't be displayed (even if it is sonic) on a device that isn't hooked to a computer; it can't run properly or be properly appreciated without it.

If you think about what that means, I think you have to come to the conclusion that such art requires a computer for its display and appreciation because the art relies on something about computers that the art can't get from any other devices. What could that be? It could be interactivity. But recall that there is lots of computer art that isn't

interactive in the slightest. In that case, it has to be programmability. And even when it is interactivity, as we've seen, computerized interactivity is completely predicated on programmability.

So the relation of computer art and programmability is very strong. Computer art is simply not computer art without programmability. The programmability of computers is one of the main things–or, in some cases, the main thing–that computer artists use to distinguish their art from every other type of art.

Evolution and the Universal Machine

Having recently been trying to be less of a fossil concerning knowledge of evolution, I've watched all sorts of truly excellent documentaries available online. In several of them, it was said that Darwin's idea of evolution through natural selection is the best idea anyone's ever had. Because it's been so powerfully explanatory and has all the marks of great ideas in its simplicity and audacious, unexpected and absolutely revolutionary character.

It's definitely a good one. That's for sure. But I'll tell you an idea that I think is right up there but is nowhere near as widely understood, perhaps permanently so. It's Turing's idea of the universal machine. Turing invented the modern computer. This was not at all an engineering feat. It was a mathematical and conceptual feat, because Turing's machine is abstract, it's a mathematization of a computer, it's a theoretical construction.

What puts it in the Darwin range of supreme brilliance are several factors. First and foremost, it shows us what is almost certainly a sufficient (though not a necessary) model of mind. There is no proof, and probably never will be, that there exist thought processes of which humans are capable and computers are not. This is a source of consternation for many people–very like Darwin's ideas were and, in some quarters, still are.

The reason why such proof will likely never be forthcoming is because it would involve demonstrating that the brain or the mind is capable of things that a Turing machine is not–and a Turing machine is a universal machine in the sense that a Turing machine can almost certainly execute any *finitistic algorithm*. A finitistic algorithm can be completed in finitely many steps where each step takes a finite time to finish.

Turing has given us a theoretical model not only of all possible computing machines, which launched the age of computing, but a device capable of thought at, as it were, the atomic level of thought. I don't really see that there is any reasonable alternative to the idea that our brains must function as information processing machines. The universality of Turing's machine is what allows it to encompass even our own brains.

Additionally, another reason to rank Turing's idea very high is that, mathematically, it is extrordinarily beautiful, drawing, as it does, on Godel's marvelous ideas and also those of Georg Cantor. Turing's ideas are apparently the culmination of some of the most beautiful mathematics ever devised.

Darwin's ideas place us in the context of "deep history", that is, within the long history of the planet. And they put us in familial relation with every living thing on the planet in a shared tree of life. And they show how the diversity of life on our planet can theoretically emerge via evolution and natural selection.

Darwin's ideas outline a process that operates in history to generate the tree of life. Turing's ideas outline a process that can generate all the levels of cognition in all the critters thought of and unthought. Darwin gives us the contemporary tree of life; Turing gives us the contemporary tree of knowledge.