Point of View is Worth 80 IQ Points

A. Kay

TRANSCRIPT OF TALK BY ALAN KAY

Today I wish to talk about some directions I think the field of object-oriented programming is going in. In fact it's very hard for me to think solely in terms of object-oriented programming, because my focus has always been on the idea that there is a person in the loop and that what the person has on their mind and what they want to do is the most important thing. That is not true for all types of computing, but it is for the type of computing I'm interested in.

A characteristic of computing in the 60s and 70s which, I gather from discussions with a number of chief information officers in the largest companies in the US still applies in many large organisations, is that most of the MIPS are still central, and central time-sharing, shared operating systems, etc, are used, with interaction still being mostly by keyboard - you have to remember and type in order to do things. But a lot of the country has shifted over to the PARC/Mac paradigm, which has the MIPS at the users desk. This is a big change. And what I think is even more interesting is that in a few more years we are going to have another, as startling, change I believe, and characterised partly by the MIPS going to be wherever - in California of course we'll weave the MIPS into our tee-shirts. That will be part of the metaphor and instead of "what you see is what you get", it's going to be "what you need is what you get". Instead of "see and point" its going to be "ask and tell".

Larry Tesler and I came up with the view that what users do now is largely editing, which is sort of lay out, but we think it is going to be something more like orchestration in the future. From mechanical printing to laser printing to no printing - that's when you can carry your computing resources around with you. (Incidentallt, the only definition of portability that I accept is that something is portable if you can carry something else too. In other words, O.5 herniations per block is not portability!). A large change I think that we're going to have, is the move from the professional programmers first to having a lot of programming done by experienced users and then to end-user programming. End-user programming is something that those of us at PARC in our group were most interested in. This was because we thought that if there's any analogy at all between learning about computers and print literacy, then the analogy would be that reading (something that I call access literacy), is having a skill to be able to access material made up by other people, and in a sense that's what this

attempt at the mouse and icons and windows interface attempted to do. So when you learn how to use that interface you're learning the equivalent of reading. And the result was, on the PARC systems and MacIntosh, that if you've learnt the interface to one system you've learnt 70 or 80 percent of the interface to any system. This was an attempt to have the user interface universal across applications - but that's only one face of the three faces of literacy.

Another face of literacy is creative literacy - the equivalent of writing. We just didn't believe that computing would be a way of life for people unless they could actually create things for themselves on their computers. That is what Adele and I and the other people in our group at PARC spent most of our effort on - wondering what you had to do to the user interface and the language in order to let people create significant tools, without having to become professional computer scientists. We were not completely successful; this is an on-going question that we've looked at for twenty years now.

However we now have the Hypercard system, which some of you are familiar with, that now has more than 750,000 users - it is given away with every Macintosh that is sold. By the end of next year there'll be more Hypercard programmers than there are programmers for all other systems combined, and that is an interesting change in what it actually means. (The statistics we have now indicate that of the 750,000 Hypercard users 70 per cent of them, about 500,000, write programs in Hypertalk.) So that adds an enormous amount of programmers, just in the one year. So very shortly a lot of the programming that people do is going to be done by the end-users. However, it's going to be done in a framework of design systems that are produced by professional designers.

So we think that the applications programming industry is going to go from making turnkey applications, which they do now, to scripted applications; so that you'll be able to get an application out of the box, use it and after a couple of weeks, when you start getting ideas about how you wish to change it, you will be able to do easily.

A context for this talk concerns two of the ways that human beings have extended themselves over the last several hundred thousand years: one of them is by building tools that amplify. Some of these tools have been physical tools like the lever and the wheel, and other tools have been figurative tools such as language and mathematics. But I think of all these amplifying tools as being

extensions of the gesture. They all help us to manipulate. Even mathematics is a way of taking stuff that is too abstract to even think about, and making it into something more concrete so that we can manipulate the symbols as though they were real things. So the end-word for amplifying tools is "manipulation".

The second way that we have extended ourselves over the years is by what I call "goal-cloning", which is simply being able to convince other people to work on our projects. Lewis Mumford calls this making mega-machines. He wrote a book called called "Techniques and Civilization" which was about how most of the machinery of mankind has involved human beings as its basic parts and how our nervous systems are set up to allow ourselves to be persuaded and to be interested in persuading. And the end-word for that kind of extension for me is "management".

When computer interfaces started being thought about in the late fifties both of these ways of doing things were considered. McCarthy wrote a paper called "The Advice Taker" and the premise behind The Advice Taker was that it wasn't going to be too many years, McCarthy said, before we had information utilities that were the equivalent of power and lighting utilities and the amount of information we could get through these utilities would be so vast that we couldn't do it through a conventional browsing tool. No such tool would enable us to use this amount of information effectively. What we would need is a thing that he called the Advice Taker, which today we would call an "Agent".

The Advice Taker was a semi-intelligent computer process that could learn incrementally; and you could give it more and more advice and part of its duties would be to go off and do things. About 1965 Selfridge called these kinds of semi-intelligent computer processors agents. It has only been in the last decade or so that there has been a big pressure to start thinking about what agents might be, because it is only in the last decade or so that we have had pervasive networking to the extent where the tool approach, for example as provided by the Mac, is starting to become unwieldly. We noticed it at PARC, both when we hooked up to the Ethernet for the first time, and when Smalltalk started getting large. Smalltalk provided a Browser, which is a very nice way of locating relevant code, and was wonderful in its day. But when Smalltalk got larger and larger the Browser got less and less useful.

I realised this firsst when I went away on a trip for a month and came back and they had redone the system and nothing was where it was, and thus for the first time I got a chance to use the Browser as an uninitiated user would use it. I realised that the Browser was mainly useful if you already knew where things were - and then you could find them very quickly. This is true of Engelbart's system as well - if you knew were the stuff was you could get to it very rapidly; if you didn't know where it was it wasn't of much help.

When we start including the humans in the loop we have to start asking ourselves questions about how do human mentalities function, and what are they are made up of. The most important result with regard to user interfaces is a formulation which has been made by several people, but our God in the 70s was an American psychologist by the name of Jerome Brunner, who said that another way of looking at Piaget's theories is that instead of there being stages of development, we actually have multiple mentalities, and we have a change in dominance between one mentality and another. He came up with a three-mentality theory, which he didn't claim was all the mentalities we had, but that there are three major ones. One was basically muscular, one was an iconic one (visual and figurative) and one was a symbolic one.

When we started thinking about the user interface, in the beginning we started thinking a lot about what could we do with images. Gradually we realised that what we wanted to achieve was some kind of synergy between the three Brunner mentalities. The user interface that we're familiar with today has a kinesthetic component that involves you tactilely - that's the "doing" mentality (what Brunner called the "enactive mentality"). It is interesting that pointing is useful - of course there are things on the street to point at, but from a psychological point of view pointing is even more important because it is the kinesthetic mentality that places you in the world. There's a British neurophysiologist by the name of Oliver Sachse that writes books like "The man who mistook his wife for a hat". I don't know if any of you have read it, but basically that book is about what happens to people when they have damage to their brain and parts of the internal world that they live in are removed from them.

One of the most striking things is that it is far preferable, according to the reports, to be blinded than it is to retain your sight and have part of your body image removed from your mentality. For instance, people who have had an

injury to their brain that removes their leg, not physically from them but mentally, report waking up in the middle of the night with someone else's leg in bed with them, because it isn't part of their body. (We can touch our nose because our body knows where our different parts are - it's something that we aren't even aware of.) Although this thing that's in bed with them looks like a leg, they can hardly see that their leg is attached to them. We don't see what is on our retina, we see what we are able to interpret through our knowledge of the world. For them there's a kind of hazy place where this thing joins onto the body they know is theirs. So part of the reason why pointing is so important is not just to indicate things on the screen, but because it is a tactile way of involving a person in the world and, of course, the windows and the icons are there because of the two main features of the iconic system.

One is that the iconic system is incredibly good at remembering images. The primary experiment on this was done by Haber in the late sixties; Haber showed 2500 images to people spaced about ten seconds apart. It took four or five hours to show these images to people and in fact for some of the people he spaced it out over several days because he thought it would be boring. It turned out that it wasn't boring at all. The people who sat for four or five hours just seeing different slides were quite happy. (This is an explanation as to why American television is so popular!) And he discovered that, even after a month, all of his subjects had better than 90 percent recall of these random images, even to the extent of being able to tell what was in an occluded part of an image.

This is an experiment you can do yourself if you have cable television. Get a friend to dial in a movie at random and see within how many frames of the picture you recognise whether you have seen that movie before. Most people can say within a few seconds "Yes, I've seen that movie before - I know what is going to happen next." It can be a movie you haven't seen for 18 years and you've seen just sixty or seventy frames out of the middle of it, and yet your visual memory is so incredibly good at retrieving that it will be able to let you know what will happen next. The other thing is that it is about four times as efficient to find one of a hundred images randomly scattered on a bulletin board than it is to find a word in a list of a hundred items - it is done by a different part of the brain.

So all these things argued for having a panorama. The other thing is that the iconic system is modeless in the sense that it is always flitting about from one

thing to another. That mode of thinking tends to be fairly creative, but somewhat unconstructive, in that because there is lots of visual material for you to look at and you're not fixating on any of it for a given time, you don't get blocked on a particular area. In fact there were studies by Covington and Crutchfield at Berkeley in the sixties which showed that if people were doing problem solving and you seeded the room by putting visual metaphors and hints to the solution on the wall they would do much better, just because they would be glancing around and taking in more from the environment.

The other thing a visual system likes to do is to compare things and of course that's why we have multiple windows. Not just to have separate applications in different windows, but so that you can have several views of the same application.

And then, finally, you'd like all this to have a symbolic thing to be translated into - so you go from the concrete to the abstract. It is this jump from the concrete to the abstract that we've been the least successful at, though Smalltalk was one of the first attempts to find a way of symbolically representing the kind of things you can indicate concretely on the screen.

So all these things led most successfully to the first two issues of dealing with images and we wound up with an interface that was rather universal in its appeal. Here is an example of how universal it actually has proven to be. This film is of a 22-month old young girl who has never lived in a world not densely populated by MacIntoshs. Her mother is my accountant and both her mother and father work at home and each has their own MacIntosh. So she grew up in the middle of MacIntoshs. (When I found out she was interested in computers I gave her an Apple II - which she rejected!) This is a good example of this idea that technology is all that stuff that wasn't around when you were born. We didn't grow up to think of the pencil and crayon and our language and clothes and stuff as being technology because it was just part of the environment, so she doesn't think of the MacIntosh as technology.

As with Adele Goldberg's children, she literally sat on her mother's lap while her mother was working and she started learning to manipulate the MacIntosh and here she is, two months before her second birthday, using this interface. In fact it is not too unusual, even though she is better at the mouse than most people think, for her to use visible menu commands, for example in MacPaint here.

When I saw this I was intrigued, but I wasn't amazed until I saw what happened next. As she needs a fresh sheet of paper she goes up to the "close" box on the window, saves her old drawing with the pop-up menu, and goes to the pull-down menu to get a "new" page, and there she is, off and rolling again. To our amazement when we tested her out (this is an old movie of her father's; we then took about nine hours of video tape of her using things on the Macintosh) we discovered that she was about 70 percent literate in the Mac user interface - which meant that she could start up any of the applications, including a complicated one like PageMaker. She could make marks of some kind, she could print those marks out, she could save those marks and get them back, and so she could manoeuvre around pretty well. Of course she knew what she was going after "new". She knew that those blotches in the top of that pull-down menu were called "new", but she read it the way she would read Chinese. She doesn't yet read English, but she knew that that particular combination of things meant "new" and that was something she could get. So this is an interesting example of something that goes beyond simply being a tool - it actually starts looking like a "medium".

That was one of the things we were interested in - what was the computer if it were media? Part of it is that media has to have some access forms and ways to manipulate things - and we think that in order for it to be completely literate it has to have a way of symbolically translating.

Now what I would like to do is to talk to you a little bit about some of the projects that we are doing. I'm going to say a few words about a new language that we have been involved in called Playground, which is an object-oriented language, but has a different object orientation from the memory-based orientation of earlier languages This was the sort of way we thought when we did Smalltalk - it had cells, or conceptual computers, on a network, sending messages to each other, and objects were known by their behaviour. What I would like do is change that, and make it even more modular, because there is a small problem with this way of thinking about things and that is that the abstraction that we find in object-oriented languages like Simula and Smalltalk is basically a data abstraction. The state becomes modular and hidden, so that you cannot tell when you are looking at an object whether there is actual state there, or whether it is a fake (or virtual) state. But the thing that is not hidden very well is control. (This is true in Prolog as well, since Prolog has this fake modularity which was like magic until you actually have to understand it. Then it just throws

you off a cliff, where all of a sudden you have to understand much, much more, and there is no gradual transition of having to have some notion in your mind as to how control is going to be passed around.)

So I started wondering just how much code we were actually writing in Smalltalk to deal with the control not being modular. The Hypercard system is an example of making control a little bit more modular but it doesn't quite answer it enough. So a few years ago I started thinking about what if we had a language where the objects do not send messages; what they are able to do is indicate that they are interested in noticing things, i.e. the various types of event going on around them.

Part of the effort in designing a language like this is to avoid getting into some of the traps that event-driven systems get you into. One of the most annoying traps is that event-driven systems tend to require you to use modes. Very often a particular event you are interested in will trigger more than one action. For instance, a simple example would be if you have an object whose job it is to move a corner or something. You can imagine two kinds of actions which you may want to do with it. One action is where it is part of the corner of a window and when the mouse is in it a movement of the mouse causes the corner of the window to move. So the basic implicit loop that's going on there is the one that we call the "apple-mouse-still-down" loop. You can imagine the other possibility of where the corner mover object is like a button that you are trying to move around. Hypercard has a mode for that. There is a mode for moving buttons as buttons, and there's a mode for using buttons as something that gives you control. The reason Hypercard has a mode for that is because if you tried to write this as a script, you would find you needed two different scripts, both of which are driven by the mouse-still-down event. The only difference between those two scripts is what started you off.

The problem is that event-driven systems don't handle contexts very well. Event sensors, things like "on conditions", etc., tend to be on or off, and tend not to be conditioned by things that have happened before. So one of the difficulties that we have to solve is finding a way of dealing with that problem - and that is one of the things we try to do in the Playground design. (The other thing about Playground is that, like the original Smalltalk system, it is mainly aimed at very young children - in fact in the context of a project called the Vivarium, that I'm going to talk about in a minute.)

The second issue I want to deal with is the question of what it means to use this iconic mentality of trying to explain things. This was something which was a kind of siren song for many years in the seventies and it is still not clear exactly what the place of iconic description actually is beyond the recognition of images. I want to explore a couple of those issues before I talk further about Playground. So here are some symbolic representations and some iconic representations - the idea is that when iconic representations are really working well, they not only indicate a relationship, they often show why the relationship holds. So for instance when we say 14 + 18 = 32, we are indicating a relationship, but when we take two rulers of different lengths and show they can be concatenated together to make one of a third length, we are showing why that is true. The same thing with 11 × 5. We do multiplication as areas and the distributed law as the concatenation of areas. We can even do the things like proportion, and more complicated things.

So here is an example of multiplying X + Y which is equivalent to taking a square that has X + Y on the side and the result of it is $2XY + Y^2$. This result becomes particularly apparent, even if you grab the square with the mouse and drag it out, because no matter how you drag this thing, no matter what size it is, the inherent relationship always stays the same.

So iconic representations are very powerful; there is even a branch of geometry done by the Indians that only admitted proofs that were obvious by construction. You weren't allowed to argue in the proofs; you had to demonstrate and the only word you were allowed to use was "Behold". Many of the original Greek proofs before Euclid, the Pythagorian proofs, were basically iconic proofs. Here's this iconic bubblesort that I talked about yesterday. When you start to try indicating a program this way, what happened to all that clarity. By combining icons, icons start acting a lot more like symbols than like pictures. All of a sudden this program starts to become something that you have to start figuring out like anything else. The reason for this is a peculiar property of icons, and that is that when you start trying to indicate meaning by combining icons the icons start acting a lot like more like symbols than like pictures. Whatever meaning they might have individually, as soon as you start wanting a person to deduce a meaning from a combination of them much of the clarity goes away, because all of a sudden the part of the brain that understands images is not able to come out with very strong inferences as to what the combinations of images

actually mean. The iconic mentality seems to be good at dealing with configurations as things, but not with configurations that have much analysis to them. So a program like this can be understood by looking at it for a bit but it is nonetheless a little bit more obscure than you would really like it to be.

Here is an iconic programming system done by David Smith, one of our graduate students at PARC, who later went on to become one of the designers of the STAR. The idea behind it was that instead of trying to represent meaning iconically what he tried to do is to allow it to be much easier to understand whether your program was giving you the right result. So the idea of this system was that you constructed a result that you wanted - so that this programming was sometimes called concrete programming. You constructed for yourself - just as you can construct a document with a word-processor. And if its a WYSIWIG system then it is rare to make a mistake. People don't usually make a lot of typos on a MacIntosh because you can see exactly on the screen what it is going to be, what the result of each edit is. This system was called Pygmalion and had many interesting features.

Another graduate student of ours, Alan Borning, did a system called Thinglab, which again allowed you to program things by direct connstruction; what it assembled as the knowledge base was a concatenation of constraints, so if you had separate constraints on things, and put them together, the system was supposed to give you the entailment of all the different constraints.

This is the famous Sketchpad bridge problem which was done simply by construction. The act of constructing the bridge and having a simple rule on one of the I beams is enough for it to calculate the stresses and strains on all the other members of the bridge. The same thing with this electric circuit. The components have completely linear descriptions, so on a resistor the constraint is simply Ohm's law. The system doesn't know about electricity. Similarly, a battery has a simple constraint and so forth. The entailment of connecting the things together is enough to force the system into satisfying the constraints, to essentially invent an electro-motive force. So it actually invents something like voltage. That sounds fancier than it actually is, but that's the effect of it, so that in order to satisfy everything the meters will sense messages that are equivalent to volts and amps, and will show what is wrong with the system. This is another way of doing iconic programming.

A simple way of doing iconic programming is in this music system we did, in which time flows across from left to right and you simply draw in what you want to have happen at each instance of time and the system goes along with it. As Adele Goldberg pointed out yesterday, this is right on the borderline of programming, because essentially you are just parameterising something that is already there, you are not doing much of a construction. Another system that people have tried involves using something like neural nets. This is basically a data flow diagram, where you are basically hooking up components and showing in which direction is the flow. Again, these have the problem that they are fairly easy to think about when you are making them but are rather complex when you are actually going back and look at something you did a couple of weeks ago or somebody else's thing - they are almost incomprehensible.

This is another interesting notation. How many people have seen this notation before? Let's see exactly how obvious this notations is. (Response from floor.) No, but you are close. It is a dance notation called Benesh notation. It is actually more direct and analytic than Laban notation is. When you see the little horizontal strokes, those are where the hands and the feet are, and so each beat actually indicates a body position and a head position and so forth. So I would call this not an iconic notation but a symbolic notation, even though it uses icons. So again it is in that class where even though it uses images to convey you have to take a symbolic view on it.

This is one of my favourite iconic programs. This is Konrad Lorenz's model for emotion in animals. The idea is that as the animal gets more and more stressed the gate opens a little bit more and there is a point where the water starts pouring through into the reservoir down below into the pan that has a weight on it that really opens the gate, and discharges all the water. When the water is all gone you have this long refractory period because the water pouring in from the top and comes in very slowly. This indicates a very complex system very nicely, although it is not very generalisable.

Now in order to study this stuff at Apple we set up a project called Vivarium, which is modelled in many ways on the project that Adele and I ran at PARC. Here instead of asking the question, as we did at PARC; "What would it be like if everybody had their own Dynabook, their own personal computer; would they be able to use it, what would they build on it?", this project assumes that that

has already happened, that everybody is going to have their own Dynabook, and that they will be able to do a substantial amount with it. What we are interested in is another question, which is what would it be like for people to make their own artificial intelligences. As before, we find that question impossible to answer in the adult world. We didn't know what it would be like. We had no sense from the state of AI that there is anything extrapolatable from the current state of AI to be able to do even semi-intelligent things. AI has drifted seriously and badly since 1970 when the government quit being interested in basic research and got more interested in results. The results of course gave us expert systems, which can do things, but which have a very uninteresting model, not a model from which you can get much. So as we did at Xerox, we said well what would it be like if we try to translate this notion of making an artificial intelligence into the world of a child.

We asked ourselves, what is a thing that is semi-intelligent, not as intelligent as a human, resourceful, able to survive in a hostile environment, that can learn and all those things - and we came up with animals. Children like animals so the project got turned into saying what would it be like to make a user interface and a programming language in which children could create animals - so that they could study animals and then sit down, as people use a desk top publishing system today to make drawings or documents, but instead what they would do is desk-top animal construction - desk-top Frankensteins, if you will. In particular we were interested in what it would be like for children to be able to program animal mentalities.

We have 50 adults working on it, and an entire school with three hundred children in classrooms ages about five to eleven, and have this project that stretches out over about six years, starting its third year now. I thought I'd like to show you some things that we've done, and maybe the best way is to first give you a glimpse of what the Playground system looks like.

This is an early version that shows one of the features, which is that everything you do in Playground is shareable by everybody else that is on the network. So here are some people who are supposed to be in different geographic locations working on a collaborative project. In this system what we decided to do is that, regardless of what else it could do or we wanted it to have, you could chose to share anything. Anything you do on any system can be shared in real-time with anybody else, regardless of what is done. The other thing that the system

contracts to do is to animate any object regardless of whether the object is even something as small as a letter 'a'. So, in this system, as in Smalltalk. everything really is an object. Unlike Smalltalk, one of the things we contract to do with an object is to deal with it in real-time and to share it over the network. The other thing we want to be able to do is to build very complicated things with it.

I am going to give you two examples of the use of it. First is a system done by one of our graduate students at MIT which was a precursor to Playground but which is very similar, so we've used a lot of ideas from it. It is called Agar, and the graduate students at MIT had it making animals that had been well described, such as Tinberger's sticklebacks and herring gulls, and Ed Wilson's ants. There are approximately 25 kinds of animals whose behaviours have been worked out in great detail and the apparent modularity of whose behaviour has been separated out. But nobody has ever built any computer models of this behaviour. So one of the things we were interested in is could we build models of such animals.

Here is an example of that with some ants. The ant has about twenty behavioural modules, according to the ethologists. When you deal with control, as I mentioned previously, the amount of actual information you have to write down to do a decent ant was about two pages of code, because each of the modules is quite independent. So what we see here is an attempt to go from making smart modules to having modules which are not so smart and try to get an architecture that combines the modules to produce the "smarts". It's an architectural attempt at doing mentalities. I'll just show you one other example of one of the ways in which we program in this system. Note that with the system that he had, although the code looks different from Simula and Smalltalk, it is attached to the separate little agent objects in a way that it is similar to Smalltalk and Simula.

One of the things we are experimenting with is a way of being able to specify a larger view of what we are interested in having happen. This is an example from doing a MacWrite-style paragraph editor. We are actually specifying situations that we want the individual objects to take notice of. So for instance in this situation what we are saying is here are two objects which are next to each other and what we want them to notice is that, if the mouse has this particular cursor and the button goes down in this area, then we want them to put an insertion

bar between the two of them. Here we are saying "If you happen to be the last character (that is what this bracket means) and you see the I beam next to you and a character is typed on the keyboard, then whatever that character was, it should go after you and it should become the last character."

But these are not actions imposed from the outside, this is the way the programmer is thinking - the programmer is saying: "This is the panorama that I am dealing with and what I am doing is showing these objects what situations I want them to be interested in" and they internalise that information and then do it. The program for doing this text editor is about ten lines of code, for doing both the editing of the characters themselves and the justification of the text.

What we don't know is whether this is actually a good idea or not, because we have just started testing out the system on real people. We tried it out on a bunch of children in May and all of the teachers this summer. It has worked out on small problems very well, but it will be another nine months or so before we have any idea whether it is a good idea.

The problem with programming languages is they always work extremely well on a few problems. A programming language really is 50,000 solutions to 50,000 problems for 50,000 people you never met. That is one of the reasons why Prolog, for example, turned out for me at least to be quite disappointing. This is because it was an extremely promising approach, but was actually a fairly weak model; so you wind up with code that goes from the Baroque to the Byzantine very rapidly in order to do some of the simplest things.

The second area which we've been interested in is real-time computation, because children, we've noticed, are real-time devices. You can't just generate an answer and take hours to do so because that is not the world they live in. So we also got interested in what it would be like to take some of these mental models that we had been working on and translate them into a simulated physical world. Again we use object-oriented techniques. Here is an example of that using a flight simulator, the most advanced flight simulator in the world, called the Evans and Sutherland CT6. There are only about four of these in the world and fortunately Dave Evans was my thesis advisor, so I was able to call him up on the 'phone and ask him if we could use the machine at night that he had in Salt Lake City. This machine has a computational power of about eight Cray

XMP-48s. (The problem is that God made the Universe with many protons so that if you try to do real-time computer graphics you have to compute them and it takes a long time.)

This is an example of a California kelp bed ecology that we did that has hundreds of animals in it and the animals' mentalities are constructed in the way that I showed you. So here's what this looks like. This is all real-time, it has been computed at the rate of sixty frames per second as you sit in front of it. Well it was extremely painful for us to construct that. It took quite a number of adult experts quite a number of weeks to construct that world, to program it and to get it working. Our aim is in a couple of years to have the children (fifth and sixth graders) be able to produce an environment like that, constructing all the animals in it, not in lots of weeks but in a few days - because when you actually write it down and try to understand what the actual entropy of the code is, if that's a reasonable word, it comes out to be something that is rather simple. The trouble is that it is just extremely difficult to do using today's techniques. So I think this is a good place to quit and to get some questions.

Question: My daughter doesn't like playing chess with computer because it never lets her win. We understand our children well enough to know they need to win sometimes. Do you think it is possible to put that level of understanding of motivation into a program?

Alan Kay: No I don't think so and I don't think that's even my problem at all. Everything I have said and everything we have ever done with a computer is to use it as a medium of construction. I would not even presume in the first place to have a child play chess with a computer, but that is a different problem than being able to use it as construction material for building things. I believe that children learn things by building them .

Question: That's an interesting point, where do you think the difference lies?

Alan Kay: The difference lies in that when children build for themselves a drawing or build for themselves a tool or build for themselves anything else, they are learning the same way as they do when they build something with blocks or they build something from languages, because they are learning by constructing.

There is a whole other way of using computers, which is as a surrogate parent or as a surrogate person, which I don't think is as nearly as fruitful a way of dealing with a child. There is no curriculum that we use at the school which is computer-directed, but every child at the school has a Macintosh which is used as another piece of paper or as another kind of crayon that is used as material for the children. I think having a computer-based curriculum is a bad idea, not a good idea.

But I think that the computer is one of the best construction materials we have ever come up with. So for instance in this project we built this environment to see what it was like to build one, but we are not giving this environment to the children, because that takes all the fun out of it. The children have interactive video disks in the school, but they do the interacting system. The system is Hypercard and they take the video disk and do tracks for themselves, rather than have the teacher do tracks beforehand when they are simply not involved. This is a very different way of looking at things.

We don't think of the computer as very important in the educational process itself except. Musicians know that the music is not in the piano. The musical impulse is inside the person and the piano is there as an amplifier, at best, and in fact most musicians who play keyboard instruments know that keyboard instruments are the least musical of all musical instruments. They have to pretend fiercely that they actually have powers of expression that they don't have. Much of the task of playing the piano in front of an audience is merely a kind of conjuring trick, to convince them that the piano is actually musical.