Computers in Art an d Design Education — Past, Present and Future.

Tony Longson
California State University
Los Angeles, California.
tony@design.calstatela.edu

Abstract

A backward look at the use of computers in Art and Design Education is used to speculate about the future in an attempt to anticipate and determine events. The author sites practical examples, and raises more questions than he knows how to answer.

Keywords: *Art and Design education, technology, software-packages, Object-Oriented, creativity, problem-solving, browser, collaboration, tailored environment.*

1. Introduction

Design consists largely of problem solving. In a 1972 interview [1] Charles Eames answers the question “What are the boundaries of design?” with another question - “What are the boundaries of problems?”

Art too involves problem solving. What distinguishes it from design and promotes it to one of Mankind’s highest endeavors is that an Artist also has to invent the problem to be solved.

Computers are problem solving tools and, if any justification were needed, this alone is sufficient for using them in the visual arts.

In order to solve a problem the computer needs to be programmed. This provides a creative potential for the artist or designer - visual problems need to be expressed in a structured way through a language with its control structures and variables, algorithms need to be invented. The process requires one to think of the creative problem in new ways and in doing so innovative creative solutions will inevitably show up.

2. PAST (1960’s - 1986)

I started to use computers in my art in 1973 and was annoyed to find that they could make better drawings in seconds than I could make in weeks. There was a precious quality to my drawings that the computer had no regard for. Taking the skill out of my work made me look more closely at the content. My second thought was a feeling of excitement that I could define rules that would condition what the image would be. My arts heritage was the English Constructivist movement, the Systems Group and the Constructionists. And my third thought (all this within an hour or so) was that I had to teach my students how to use computers in their work.

The tools at our disposal were limited: BASIC or FORTRAN with simple device control such as moveto and lineto, output was a small green CRT or, when the ink didn’t run dry, a pen plotter.

Despite, or perhaps because of, the limitations, the work that people produced was extraordinary both in its originality and its integrity. Examples of this work were shown in the Siggraph ‘86 Art Show curated by Patric Prince[2].

I was visiting lecturer to London’s Slade School Postgraduate Experimental Area lead by Malcolm Hughes. A handful of students working with an eviscerated Data General mini-computer included Chris Briscoe, who pioneered commercial computer animation in England, and Paul Brown who has been a major influence in education. Brown’s early work can be seen at his web site.

The tools settled out into three categories: 2D graphics packages, 3D graphics packages and ‘Paint Programs’. The first two were really the domain of scientists and engineers as they worked in conjunction with a programming language. The Paint Programs were something of an anomaly. Developed by scientists who presumably had some idea that they would be useful it was up to artists to demonstrate the fact.

While people were asking whether computers could be used creatively (Yes), artists like David Em (Jet Propulsion Laboratory) and Darcy Gerbarg (New York Institute of Technology) were quietly getting on with it. The Paint Program at JPL was written by Jim Blinn who used it to make texture maps for the moons and planets in his simulation of the solar system. As I recall, he usually got his brother (an artist) to do the work.

The next question up for discussion was “should artists and designers know how to program?” I said yes, and was derided for it! In fact I had already established two successful educational initiatives in Computer Graphics for Artists and Designers based on that premise. One was at West Coast University in Los Angeles (now defunct), and the other - where I still teach - is in the Art Department at California State University, Los Angeles. I think they were the first of their kind. People were lining up to get in [3].

Frankly it wasn’t too hard to make that decision. I had taught myself to program and enjoyed the unique feeling of control over creative questions that it offered; I could see excellent work from other artists; and... there wasn’t anything else to teach! Software packages didn’t exist and the use of computers in Design was just beginning. In 1984 the Macintosh suggested a potential for design but didn’t deliver until 1986.

3. PRESENT (1986-now)

It was the invention of the Laser Printer and its page description language, Postscript, that delivered the promise of computers in Design -_Output.

Desktop Publishing became a practicality. Commercial success lead to cheaper computers. Cheaper memory lead to more colors (one of the rare examples in life where you get more out than you put in,) more colors created a need for paint programs, 2D and then 3D modeling and animation packages became common, and HyperCard established a practical basis for interactive design. Specialized software packages proliferated - and that is what people began to teach.

Why not teach software packages? They offer a well-defined body of information, usually come with documentation and tutorials, and are easy to justify to administration. Furthermore there are hundreds of job adverts which say “must know PageMaker”, and hundreds of students who want to get that job. As a consequence you’ll see academic courses in Universities across the world with titles like “Advanced Photoshop”. It sounds reasonable but it is a mistake.

Academic courses in Art and Design are about creativity and problem solving, the tools are secondary. Also the tools are out of date in a few months, so that a student who know menus and commands and keyboard shortcuts for one program will be at a loss when the new version comes out.
An alternative is to teach the principles behind the software and to set this in a creative context where practical projects encourage an exploration and understanding of the concepts. The principles behind the software will survive. Teaching a student the principles of computer graphics means that she will be able to transfer her knowledge to whatever environment of software and hardware she finds herself in.

Computer Graphics software seems to separate out into categories: drawing programs such as Illustrator or Freehand characterized by a 2D coordinate system with transformations operating on object-oriented lines and shapes; raster based graphics such as Photoshop characterized by pixel-level manipulation, layers, and image-processing techniques; 3D modeling and animation programs such as CINEMA4D or Maya characterized by 3D coordinate systems, 3D objects with transformations applied to them, rendering, and the notion of change over time; and Interactive Design which may include any of the other categories but is characterized by non-linear presentation of ideas and information in a context which is easy to negotiate.

It is not surprising that these software packages had their origins in the 2D, 3D and paint programs of the early days. Even Interactive Design, proposed by Ted Nelson in the 1960’s, had most of its design principles raised and codified before the Internet made it a reality.

I try to tailor projects to encourage an exploration and understanding of the principles behind the software. I make the projects “self-starting”. Creativity is a cycle of activity and review. Often students find it difficult to start the process so I invent projects which will have them producing objects and images without much effort. Usually the projects are rule-based. When they have an image in front of them that has not required too much effort on their part students are usually more open to critically evaluating it and making changes, and so they quickly become involved in the creative process. Once the image or object has a life of its own the students can respond to it creatively.

I also try to invent projects which will allow the students to juxtapose their work with someone (or everyone) else in the class.

You might agree that the idea of teaching principles (not packages) is good but argue that at some point the student needs to be shown details of the software package such as which menu item to select or window to open. I would counter that if the student can’t figure it out for herself then there is something wrong with the software package. As consumers we wouldn’t dream of buying a domestic appliance such as a washing machine or a car whose operation was not self-evident. Software should be the same. I usually take a naive approach to teaching software explaining what they might expect the package to do and putting responsibility on to the student to find out how to do it. This encourages self-reliance, and a problem-solving approach.

As artists and designers we tend to push software and hardware to its limits which means that things can go wrong, so problem-solving is an important talent. I encourage it.

3.1. Trends

Paul Brown spoke of the future as the invisible places beyond an "event horizon" [4]. Extending his analogy I suggest that we build a platform of current knowledge and use that vantage point to get a glimpse of what might lay ahead.

Programs have become larger as a result of cheaper memory. Developers wish to provide more functionality and so there's an overlap between what used to be distinct software categories. As a trend this is good but the consequence is an unweildy and expensive software package. The concept of 'plugins' has gained popularity as a way of extending the functionality of a software package without making it monolithic. Many software developers allow for and encourage them. Browsers thrive on third-party plugins to extend their scope.

Object Oriented Programming is becoming the standard, and implies a flexibility that will allow for problems and solutions of which we can't yet conceive of. CINEMA 4D is an example of object-oriented animation software. The manual apologizes at one point:

> "This is not meant to challenge your imagination but is simply the result of CINEMA 4D being a totally open system. Nobody knows the type of animation effects the future might bring and (what) you may wish to assign them to."

Web-based Objects are taking off.

4. FUTURE (now to the next Millennium)

4.1. A Disclaimer.

Predicting what Art and Design will look like in the future is oxymoronic. As soon as you describe it - it exists (in the present.) Witness the failure of Disney's Tomorrowland, and Mr. Spock's three-dee chess board. It's safer to invent a retro-future such as Kubrick's Louis Quinze interior in 2001 A.D.

However we can anticipate a working (and teaching) environment, and possibly even help to make it happen.

Software packages will cease to exist as we now know them. Applications will consist of objects and messaging. You won't necessarily know what computer they are on. For any particular task you'll be assembling (without much effort) the appropriate modules. You won't own software any more, you'll rent it, paying a fraction of a penny for each object each time you use it. Equally your own objects may be sought after items and provide some revenue. Forget about software upgrades, or new versions of the operating system. Any one of the millions of the components of which they consist could be updated transparently at any moment in time.

Browsers will probably be the only software on your computer. They will be the portal to your web-based software modules, and they won't look like they do now (no clunky windows and that anachronism known as scrolling).

The result of this will be a trend to tailored environments. As a designer you'll have your own set of tools (like craftsmen of old) which you will hone and refine. These tools will result from the kind of work that you do and will reflect your creative sensibilities. They will be an analog of your distinct style.

Think of the potential for teaching. Principles will still abide, but now you can design a tailored environment appropriate to who and what you are teaching.

As students mature creatively you'll teach them how to build their own tools (if they haven't already found out!) Is this back to programming? Yes, in the sense that you'll have control over your own solutions to problems. But not in the sense of sequential programming with a few simple control structures and variables. Object-Oriented Programming demands a different mental agility and the need to know details of huge libraries of existing objects and their syntax. Good user interfaces will facilitate this.

4.2. Output - put out.

More of the things designed on the computer will use the computer as a means of delivery. Physical output, will diminish for several years until the Rapid Prototyping Systems become consumer appliances. The Prestigious Prix Ars Electronica Competition had a category called "sculpture" for several years eventually to replace it with “Web-based media”. Conincidentally this was the year I planned to submit!

4.3. Collaboration - the Good, Bad, and the Ugly

Early Siggraph conferences often showcased collaborative paint programs. You could add to an image on a monitor at the same time as someone in an adjacent booth or across the world. Somehow the results were always disappointing (the Bad). Perhaps a lack of clear goals lead to what was a competitive rather than collaborative exercise.

Algorithmically generated images have been posted on the Net which one can rank (the Ugly.) Subsequent generations of the art work spawn from images that were most popular - Aesthetic Darwinism, perhaps, or more likely the cyber equivalent of Velvet Elvis.

As part of the Electronic Theater at Siggraph some years ago, Rachel and Loren Carpenter set up a collaborative visual venture (the Good) which continues to astonish me. Each member of the audience had a wand with reflective stickers on each side, green and red, which they could hold up to signal a binary decision. The collective audience response was grabbed by a camera at the front of the auditorium processed in some way and revealed on the cinema screen.

In the first exercise a large circle was drawn on the screen. With the whole screen representing a map of our seats in the auditorium we were asked to signal one color if we thought we were inside the circle, and another if we were outside. Within a second or so the circle filled in. In a couple of seconds more the edges of the circle were well-defined and stable. How did this happen with such authority? Each person made a single binary decision. The decision was based on some knowledge of their location in space. They could perhaps see what decisions their immediate neighbors had made. The image on the screen provided some general feedback but not specific to an individual’s decision.

Though in its consequence this example may seem trivial I think the implications are profound. The notion could be extended to creative tasks using the Internet both as the equivalent of the auditorium and of the screen. Collaborators would provide far more sophisticated decisions than in our example, and their role in the task would be correspondingly significant.

The nature of creative problems that could be addressed and the design answers we may see are boundless.

References

[1] Paul Goldberger. The Eames Team. In *The New Yorker,* pages 90-99, New York, May 1999.

[2] *Art Show Catalog*, Siggraph ACM, Dallas, 1986.

[3] Judy Brown. *Teaching Computer Graphics: An Interdisciplinary Approach.*, pages 189-190 Siggraph ACM, Anaheim, 1987.

[4] *Computer Animation Workshop.*,California State Summer School for the Arts, Humboldt, California, 1990.

Acknowledgements

To Bob Holzman for enabling the Present and Future in my career and to my Stepfather, Ken Dixon.