Using Art to Create Programs

I have a dream. I am upstairs in the Museum of Modern Art. After passing a preliminary exhibit on the history of programming languages from punch cards to perl, I enter the Room of Sorting Algorithms. The walls are decorated with printouts of some of the most elegant computer program code ever devised, along with the sample outputs. Passing through the Hall of Object-Oriented Programming, I reach the AI Chamber, where I peruse a moving implementation of the A* path-finding method, and an exhilarating flowchart of a design for a neural network…

Computer programming has traditionally been viewed as a scientific discipline, yet few would argue that there is a degree of craftsmanship, creativity and even artistry involved in it. For decades, computers have been used to create works of art, so why have not been attempts to create source code that has artistic value? Are the needs of the computer are inherently non-artistic? Are the creative aspects of programming are mere concessions that we must make because we, as humans, are unable to think in pure ones and zeros. Or could a true work of art be written in a programming language?

Before discussing some of the artistic aspects of computer programming, I should clarify that I am writing specifically about the source code itself, not the compiled product (i.e.: not the software program that is created by the code). It is possible to conceive of a very aesthetically pleasing program that was generated by ugly, sloppy code. There is a change that occurs when code is compiled,

that is akin to translating a book into another language. Code that looks great in C++ may not necessarily perform well on the machine once it has been compiled into binary machine instructions.

1. Language

Though designed to facilitate communication between humans and machines, rather than humans and humans, programming languages are as legitimate as English or Latin. They have rules of grammar, syntax and style. The formatting of paragraphs (functions or classes) and the choices of words (variable names) possess inherent meaning and can evoke certain impressions and feelings when one reads the code. But has anyone written a poem or short story in Java? Has a function ever made someone laugh or cry? The audience might be limited to other programmers, but would that make the endeavor any less artistic? As interaction with machines becomes a larger and larger aspect of our lives, a computer language may be the ideal means to express our thoughts and fears.

In ***Using Computers to Create Art***, Stephen Wilson says one of his book's goals is to make computers more accessible to people with artistic inclinations.[1] He feels that right-brained people need to become more involved in the evolution of technology by contributing creative designs, producing social commentary, and even developing entirely new art forms that put the latest technology to use.[2]Left-brainers, the ones who currently spearhead technological development, tend to obfuscate things with unnecessary complexity, and can become disconnected from the aesthetic and moral issues that impact our society. He compares modern engineers to the priests of Ancient Egypt, who used black boxes adorned with hieroglyphs as props in their mysterious rituals:

> They are so accustomed to ministering to the ordained purposes of the box in the ordained ways that they have lost touch with other possibilities for it. They have also lost the ability to talk to people outside the sacred caste. [3]

This goes to show that the stereotype that computer programmers have poor communication skills is incorrect. Indeed, being a good programmer requires excellent language and communication skills. Programmers simply spend most of their time working in a *different* language from regular people!

2. Aesthetics

In programming, there is a concept of *elegance*, which is a subjective, aesthetic judgment. An algorithm is considered impressive if it accomplishes something powerful in a handful of clearly organized lines of code. The most elegant algorithm may not necessarily be the fastest or the one that requires the fewest lines of code. Rather, it is the one that communicates the idea behind it most effectively to other programmers so they can easily understand and apply it. The individual choices a programmer makes, line by line, ultimately determine whether her program is elegant or not.

Here are two examples of C code that accomplish exactly the same thing. They count down from 100 to 1, and add all the numbers together to compute a final sum (100 + 99 + 98 + ... + 2 + 1).

```
//Program 1
int s = 0;
for (int x = 100; x >= 1; x--)
        s += x;
printf ("The sum is %d", s);
//end Program 1

//Program 2
int sum = 0;
int counter;
while (counter > 0)
        sum += counter;
printf("The sum is %d", sum);  //print the final
result
```

It can be seen that even these small programs require the programmer to make a number of choices. Program 1 uses shorter variable names (s, x), which are concise but more cryptic than those used in Program 2 (counter, sum). Also, Program 1 uses a "for" loop, whereas Program 2 uses a "while" loop. Program 2 has a comment on the last line while program 1 does not. (Comments are some explanatory text preceded by a //. They are ignored by the compiler, and not converted into binary instructions.)

It is easy to imagine how in a larger program thousands of these sorts of decisions need to be made. To make life easier, programmers often create style guides for themselves and their co-workers (called conventions) that help keep their writing consistent. Yet, even with

these aids, there are still individual decisions to be made in nearly every line of code.

3. Emotion

It could be argued that programming is, by its very nature as a dialogue between humans and machines, an exercise in pure intellect. Ideas may be communicated in a program, but not feelings, and it is this emotional element that separates source code from art. Art must have some emotion driving its creation. To this argument, I would respond first that all ideas communicated by humans have some feeling behind them because feelings are a part of our beings. Yet even if we assume that more programming work does not have an emotional component doesn't mean emotion can't or shouldn't play a part in the creation of code. Certainly, the database programmer, who spends most of his day crunching stock market data, does not have much time to express her creativity at work, but not all programming efforts are so tedious. Similarly, the *Joy of Cooking* is not a work of art, but *The Brothers Karamazov* is.

Finally, any programmer that has spent hours fuming over a nasty bug, or who has experienced the elation of finally getting her masterpiece up and running, will vouch that there is plenty of emotion that goes into the programming process. The emotional aspect of programming is merely neglected by many because most programmers consider themselves scientists rather than artists or communicators. This is a notion perpetuated by computer science teachers and by other programmers. Yet, it is quite possible that if programmers began to embrace their artistic side, the world would be gifted with even more innovative, powerful and unique computer applications.

Writing computer code is an act of communication. As such, the thoughts and feelings that drive the act are comparable to the impulses the motivate painters, writers and musicians. Programming is an attempt to take an idea that does not have a concrete existence and express it. Along the way, programmers are required to make thousands of choices, many of which will have an aesthetic impact on their code. The emotions a programmer brings to her work will shape the code. In the end, the source code that gives life to the programmer's abstract ideas can be elegant or inventive or it can be ugly and spaghetti-like.

Is it possible to call a piece of code "beautiful" or "artistic"? While we may never see sheets of source code hanging in the Louvre, I believe the answer is *yes*.

Footnotes

1. Stephen Wilson, *Using Computers to Create Art*. Prentice Hall, 1986. pg. 5
2. Wilson pp. 10-11
3. Wilson pp. 1-2