Programming and Programming Languages

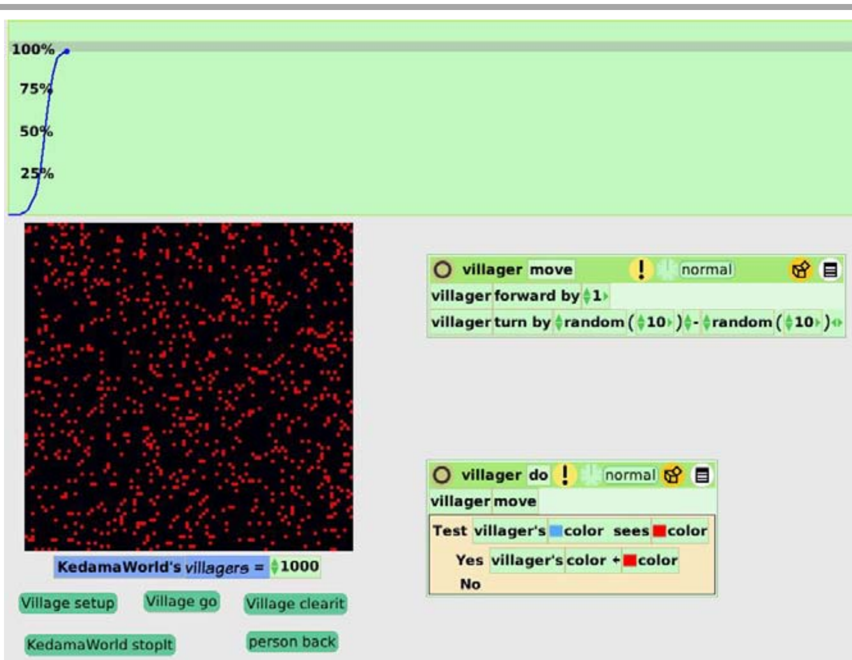
Alan Kay

VPRI Research Note RN-2010-001

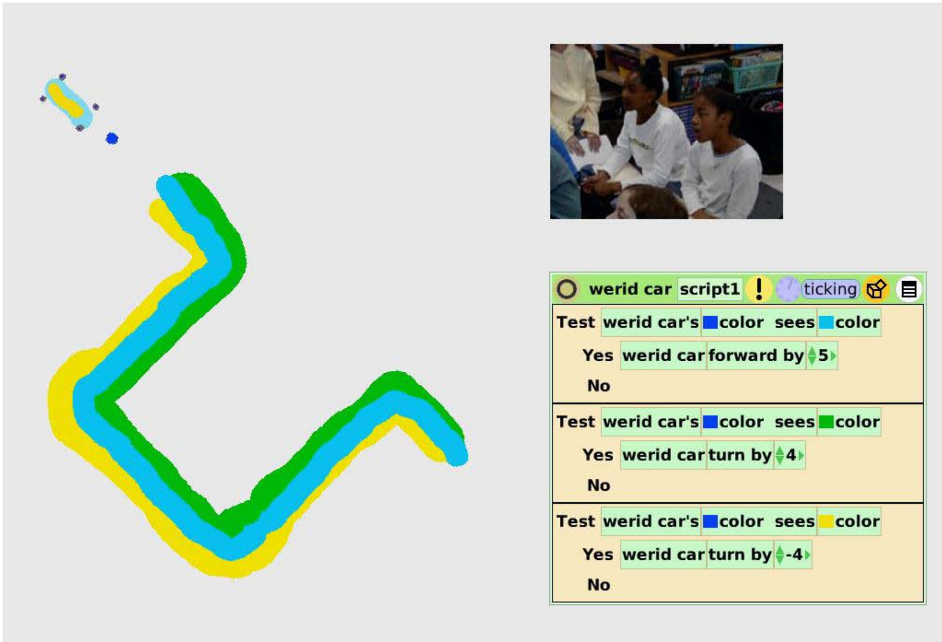
This material is based upon work supported in part by the National Science Foundation under Grant No. 0639876. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Viewpoints Research Institute, 1209 Grand Central Avenue, Glendale, CA 91201 t: (818) 332-3001 f: (818) 244-9761

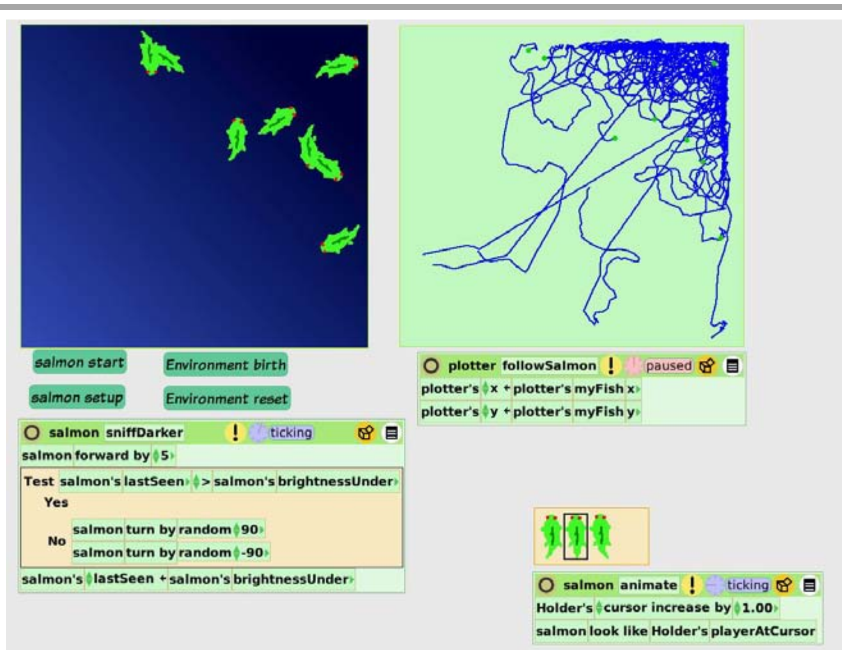
Many thousands of programming languages have been designed and implemented in many styles. Too many for 45 minutes! So let's look at a few styles starting with the Etoys language and environment made for 11 year olds. See Etoys, Children and Learning and Etoys Authoring and Media for some of the things I showed in the live demo.



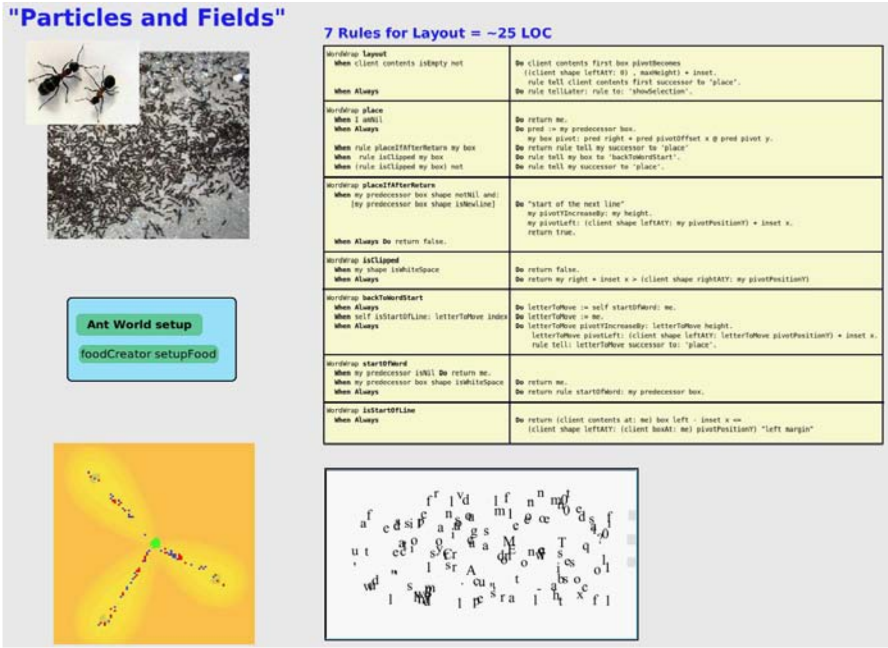
We can easily make thousands (millions) of objects and program them all at once. This shows that contagion can be delayed by changing the time constant (by having more or less villagers) but not eliminated. All the villagers eventually get sick and die. Being able to program "massively parallel" and easily is important. Be critical of languages that can't do this easily.



Feedback and correction is a powerful idea. These two 11 year old girls came up with this nice program to always keep their programmable car on the road. Most systems use this idea whether biological or human made.

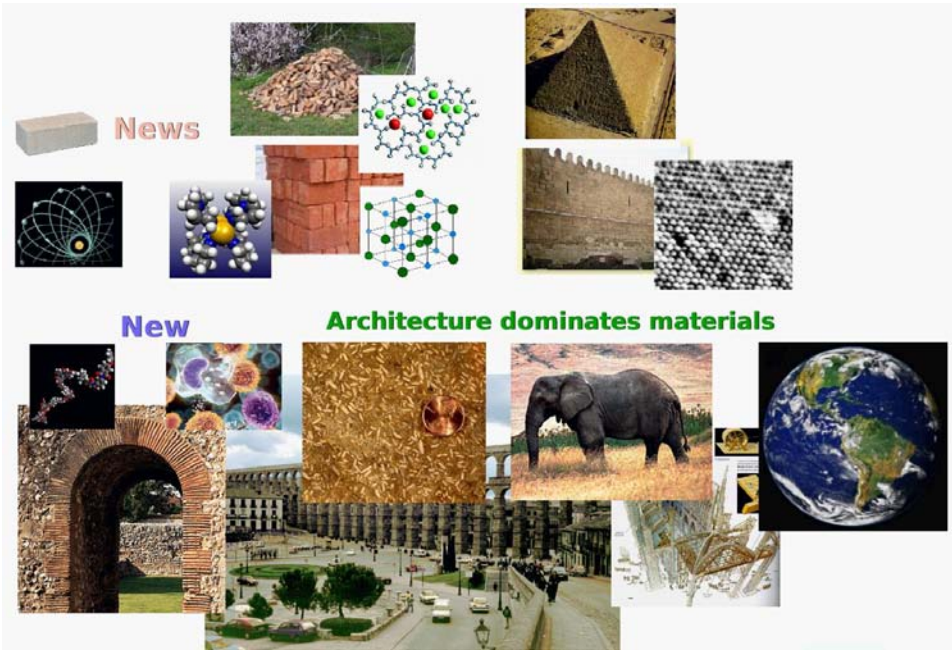


This idea can be used to follow gradients. The salmon are "looking for dark". The basic scheme is "if things are OK, keep going, otherwise do something random and keep going". All life uses this, and so does evolution. And so does the Ethernet, and with a few more elements, so does the Internet.

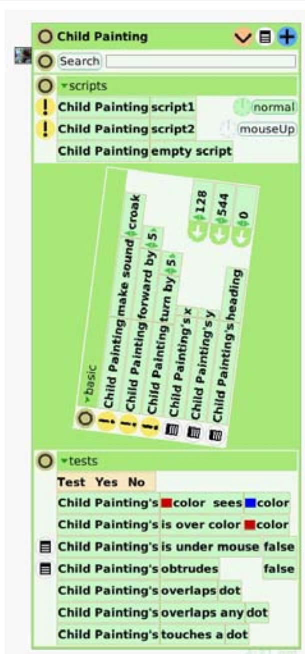


Ants are made efficient by food-carrying ants laying down a scent trail that other ants can follow upstream to quickly find the food and then go downstream to quickly find the nest. This is “loose coupling” between objects. The same idea can be used to program a text editor in just 35 lines of code. Point of view is worth 80 IQ points!

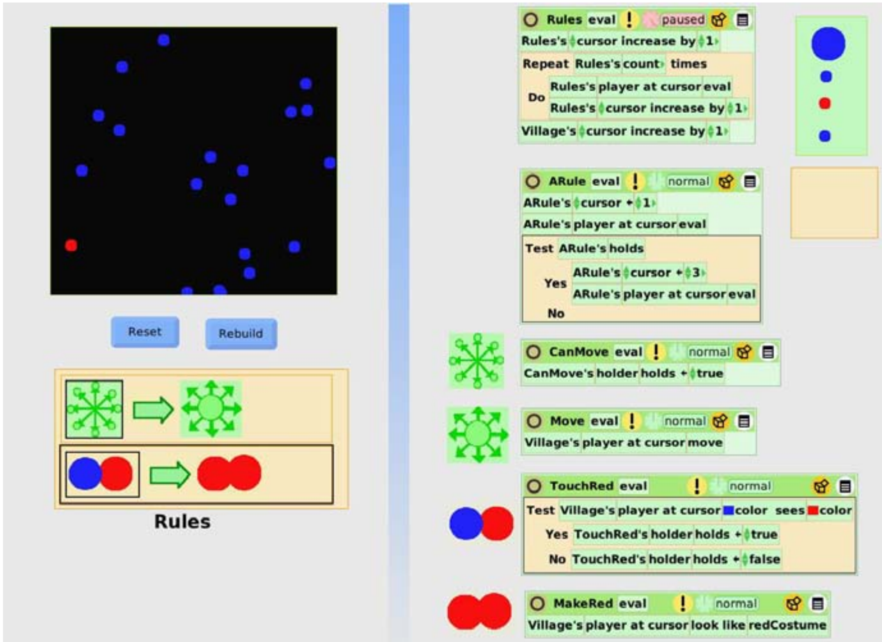
The future of programming is not like most programming today. But will be more like making biological systems. The Internet is one of the few systems organized this way. It is the most scalable and robust human artifact on the Earth.



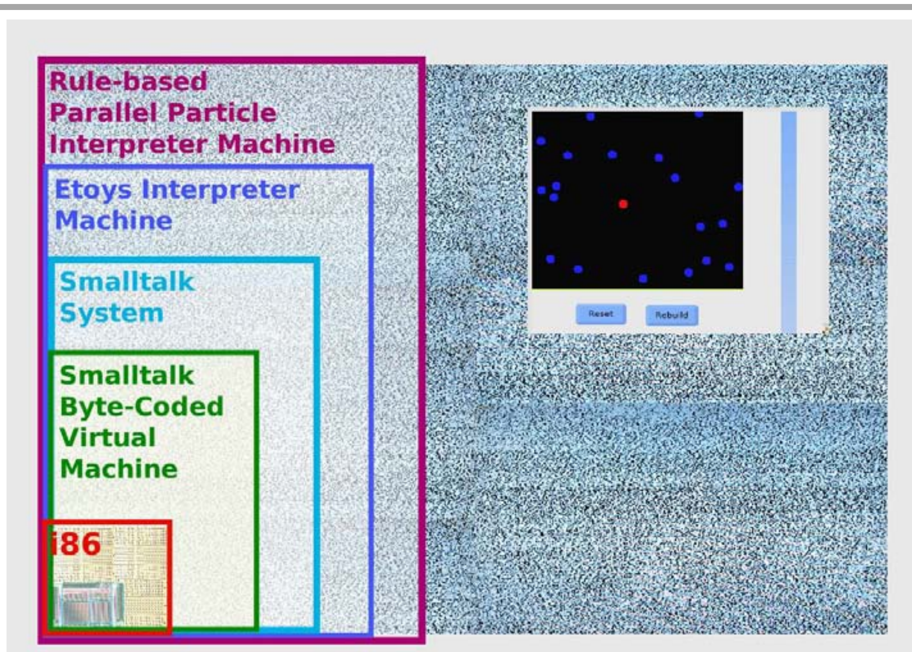
Building blocks can be organized simply – with results that are like the building block – walls and pyramids are like piles and stacks. Many molecules are like atoms. Or, completely new much more powerful structures can be built that are not like the parts – arches and life – this is what we need to try to do when we design systems and programming languages!



The children's programming environment is made from the very same simple materials as the toy cars they draw. And all is dynamic and "alive". All interactive development environments for programming could be like this. If you are working with one that isn't then you are working with a language and environment that have serious flaws!



It is easy to make a simple interpreter, even in the children's system. Here is an example of a rule-based language for controlling the particles doing the epidemic example. All the scripts needed are on the right.

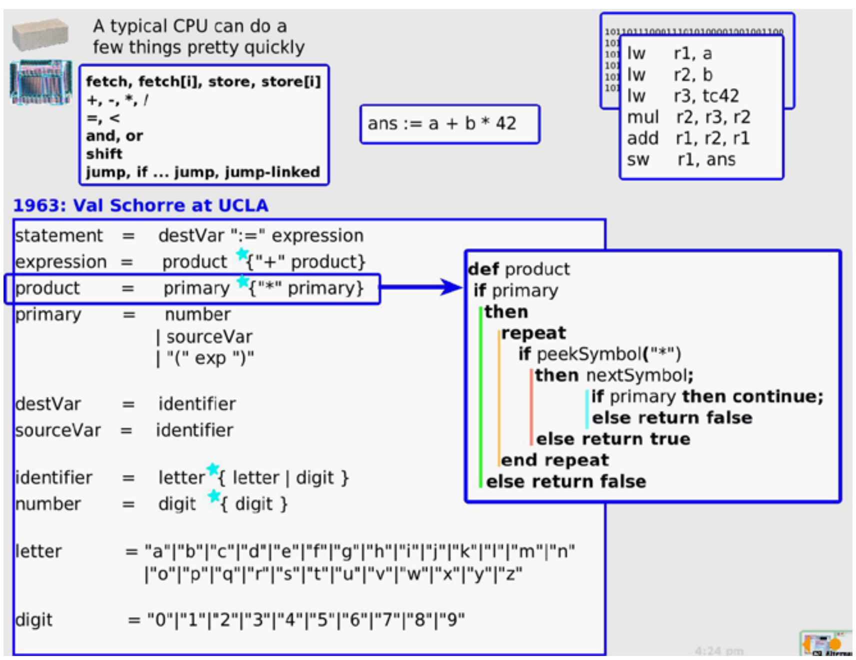


Here is the situation of the previous "slide". The blue "noise" are the bits in the main memory of the computer. This presentation is actually given in the children's system, where the new interpreter we just made is built on the Etoys system which is built from Smalltalk, which runs on a virtual machine made for Smalltalk that ultimately runs on i86 machine code which is interpreted by microcode running in the CPU with simple logic.

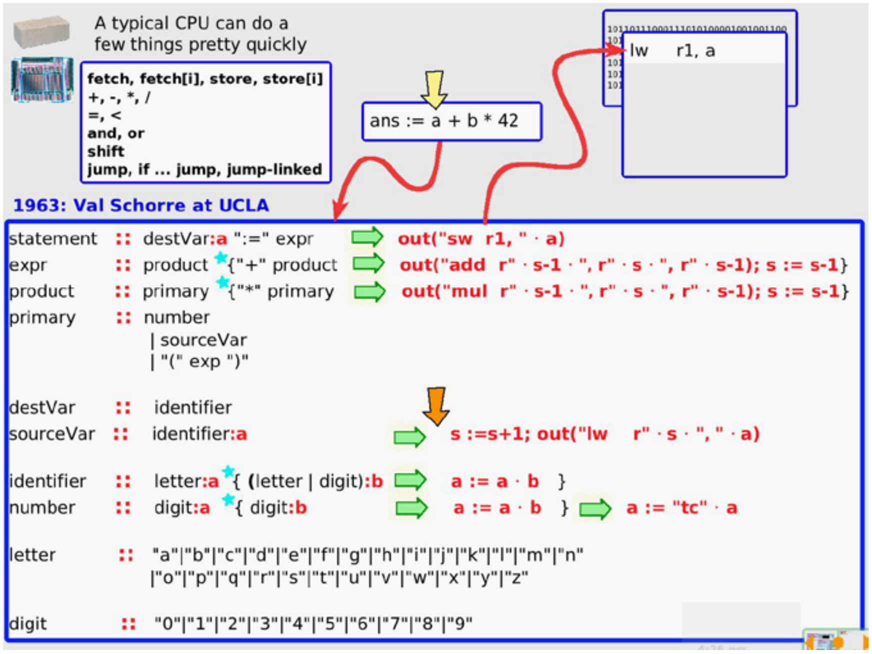
Virginia Pham CS1:

… how can something you type be decoded into machine language, and how does the computer know what to do?

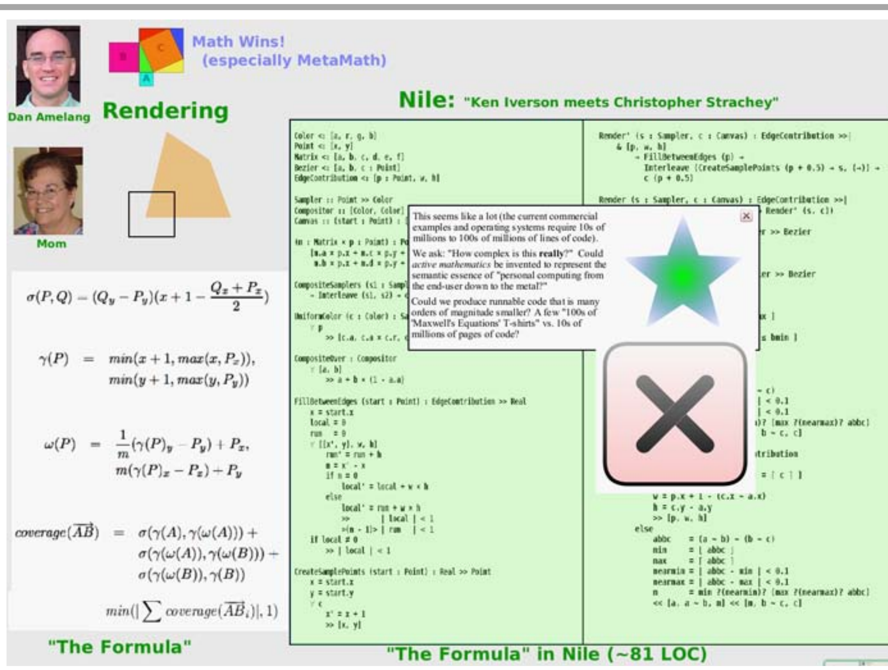
A good question from a CS1 student!



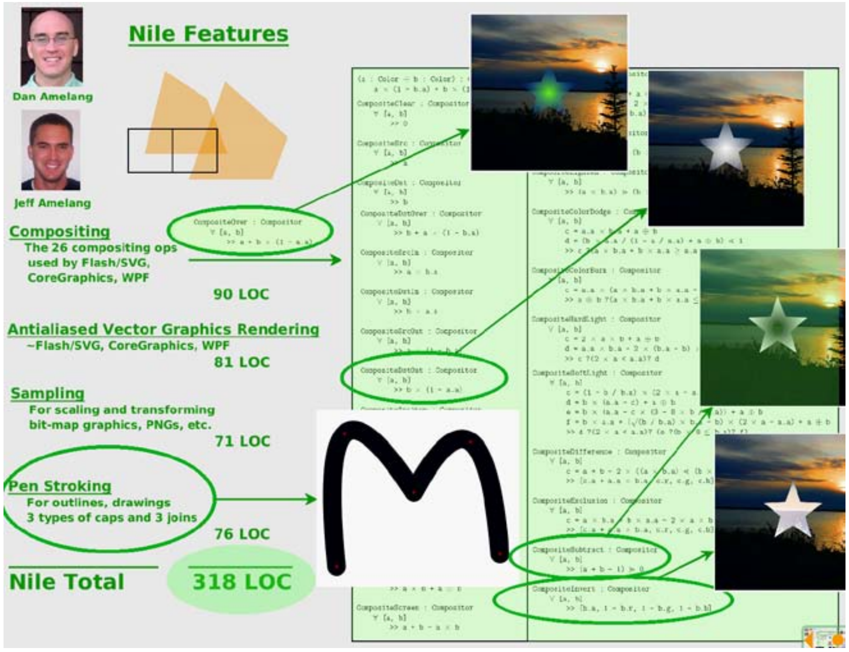
Val Shorre at UCLA in 1963 noticed that the mathematical expression of a higher level programming language could itself be turned into a programming language. The circled line of metalanguage is equivalent to the procedure on the right.



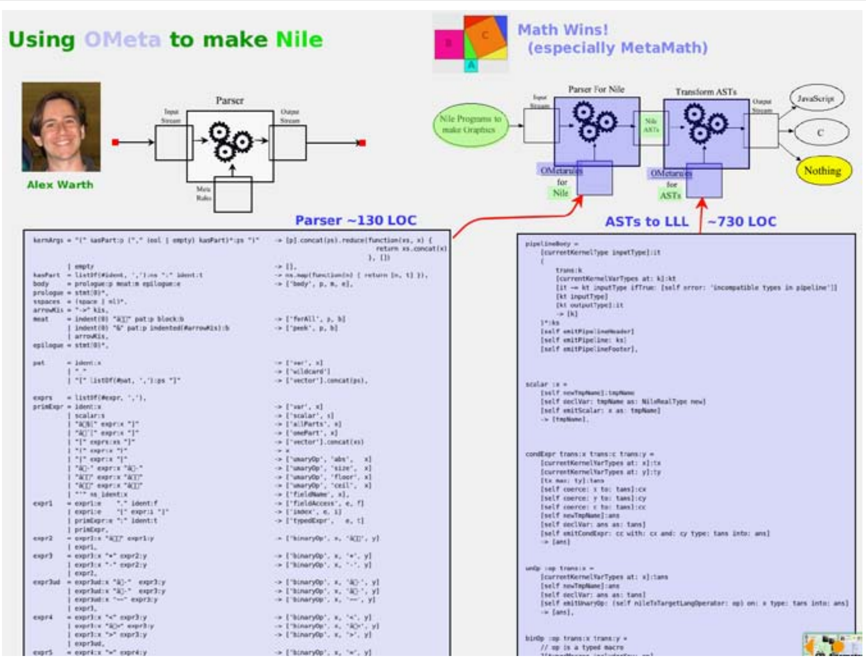
This allows "grammar programming" by adding the stuff in red and writing an interpreter for it. This can be run as a program (starting in the upper left corner of the code) via mostly doing function calls until some concrete symbol in the grammar can be compared with the input. Here "a" has been matched, and the output statement has just put out the first line of the machine code



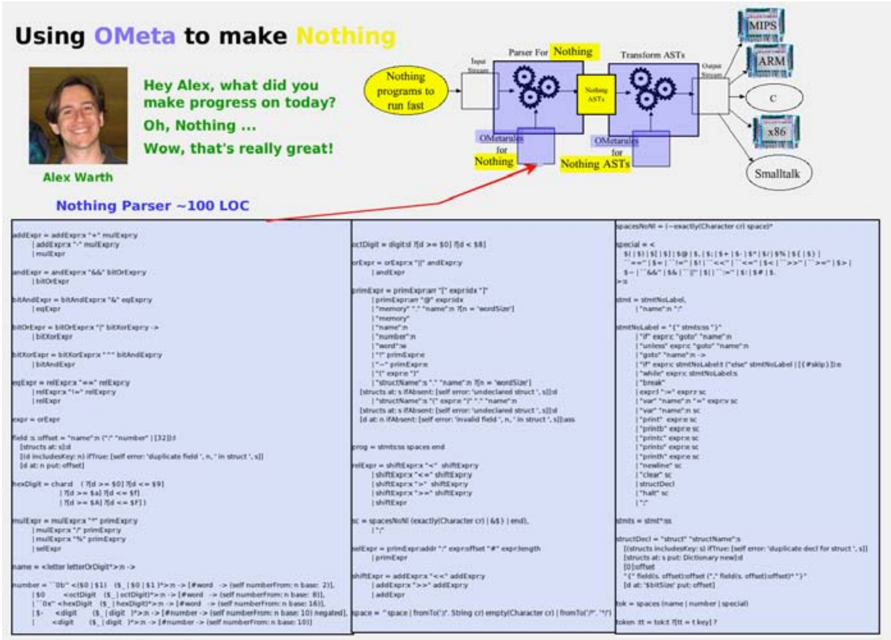
A real example is drawn from a project to go from millions of lines of code to hundreds by inventing new languages. Here's a new math for rendering in 40 lines, and the math in a programmable language called Nile in 80 lines. The language automatically deals with streams of points and pixels with mathematical transformations of coordinates and signal processing.



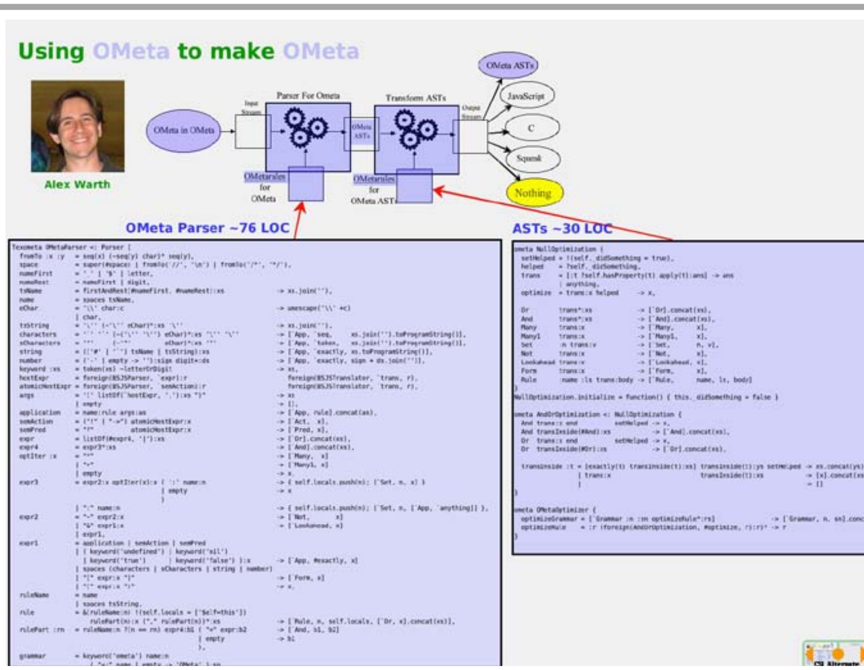
We need a few more things: to compose separately rendered things – there are about 26 of these standardly used – sampling of bit maps for scaling – and pen stroking. All these plus rendering can be expressed in just 318 lines of code in Nile.



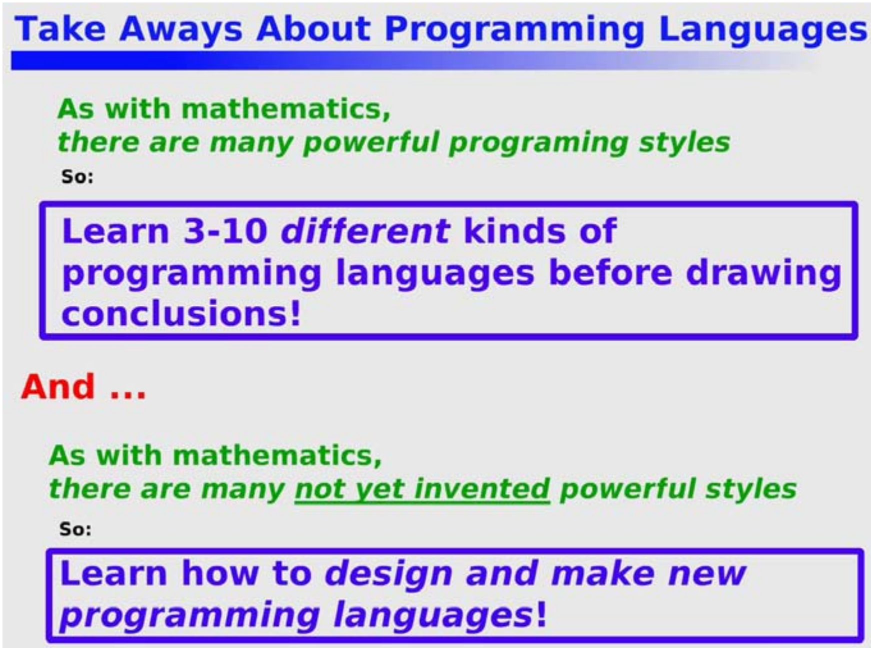
But we have to make Nile, and this can be done in the OMeta language (that is a big brother of the Val Shorre example previously). Dealing with the surface structure of Nile takes about 130 lines of code, and creating optimized output (we like graphics to run quickly) takes about 700. The "target" is a "symbolic computer" - "Nothing" - that only has operations like a simple CPU.



And we have to make our target machine language “Nothing”. This takes about 100 lines of the OMeta language.



And we have to make OMeta itself. This takes about 100 lines of code. There are a few more lines of code required (not shown), but this shows how a completely “from the metal” system can be made in a few thousands of lines of code rather than millions as with the commercial versions without losing any of the actual functionality. In other words: “Math Wins!”



You must ask questions and "be critical via knowledge" -- that is, learn a lot and try to gain perspective on what people are trying to get you to learn. A lot of it might not be a good idea any more (and some of it might never have been a good idea!)