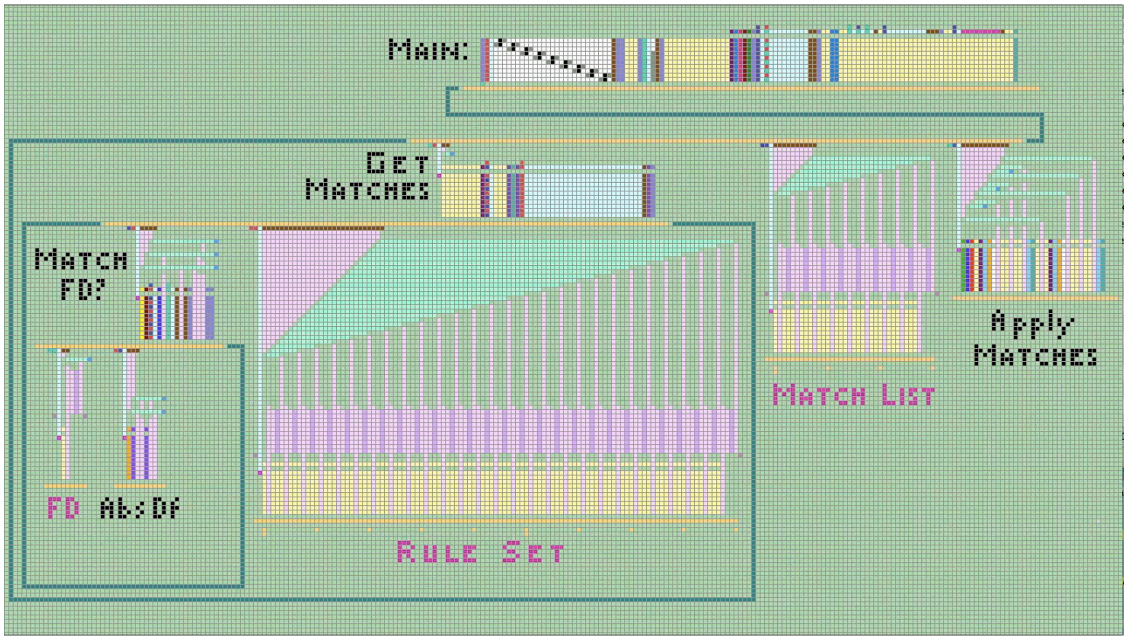
Trivia about the Bitpict-implements-Bitpict Project



- What you saw running in the video was:
  - a MacbookPro
  - running a Parallels virtual machine
  - running Windows
  - running a Java virtual machine
  - running the 2D multicolor Bitpict engine
  - running 2D multicolor Bitpict rewrites
  - running an IDR language interpreter
  - running an IDR program
  - running 1D grayscale bitpict rewrites
  - running Falling White Dots

- It was mentioned in the movie that outgoing data sent from function calls to defuns uses packet switched communications, and return data is circuit switched. If you look closely, outgoing and return data differ in another way – outgoing data is bitsliced, i.e., a block of diagrams is cut horizontally, and sent one slice at a time. Returned data is sent one diagram at a time, i.e., cut vertically.

- One motivation for this exercise was to get a sense of spatially structured computation. People sometimes now refer to physical computation – for example "molecular computation". But many such systems of "computation" are quite different from what computer scientists usually talk about. In particular, geometry plays a big part. Geometry also plays a major part in Bitpict computation. This exercise involved trying to create more standard computer science constructs in such a world, with various interesting insights. For example:

  - Distance really matters. Getting data and functionality together is a big deal. Probably a majority of all rewrites were doing this. There are various ways to solve this, including moving the data, or the functions, or make many replicas of things so one is always nearby. The separate fast/slow data movement machanisms was interesting, and maybe pretty general - special mechanisms for moving things big distances, and slower ones for where functional mechanisms can grab on.

  - Encapsulation by space and color. There is an issue of how to create functional structures that only interact with what you want them to. Here they had special smart parts (the taps on the bus) that grabbed what was meant for them. Otherwise they were isolated from one another by geographic regions containing (almost) totally inert material (the background light-gray-green). It sometimes happened that various signals used inside the defuns (e.g., those to read out from data latches, or those to trigger the restoring of the defun operators) would escape and shoot off into other defuns causing havoc. Special blocks and rules had to be put in place to trap them.
  - Control Flow. How do you manage sophisticated control of execution flow (conditionals, looping) in a spatial context? Here much was done with data-flow execution (things run when their data is all there). There is also some use of execution tokens and other sorts of special signaling protocols to coordinate events across distance.

- Writing this system took about 1hr/day for a year. I'm on medical leave with severe chronic fatigue syndrome (check out the bestselling *Seabiscuit* author Laura Hillenbrand, one of my CFS heroes in this). I can do very little, some days nothing at all. But when I could manage it, this was more fun than watching Netflix. Making the video took another couple of months.

- You may get the feeling that the system written here seems like a mix of computer programming and circuit design. Me too.

- The "fast wires" carry any data or function-name colors, and use O(logN) rules to move things a distance M in O(logM) rewrites. You cannot tap into them with Defun's though.

- The total number of rules (797) would be substantially reduced if Bitpict would allow groups of colors to be defined (e.g., the grayscale data values, the function colors), and used in the rewrites. Currently all cases have to be put in by hand. Yamamoto's VISULAN was a variant of bitpict that allows this, but the current system uses a highly efficient graphical-unification-based method to track matches so it does very little search. I never got around to working out the variation of that algorithm for color groups.

- The Bitpict system used here actually allows multiple 2D layers which can be used either for 2.5D type operations, or full 3D rewrites. This capability could be used to do the bitpict-IDR-bitpict exercise with full 2D IDR diagrams. Hard to show on paper or in a video though.

- I've been writing Bitpict rulesets to do various things on and off for over 20yrs now - first at Bellcore, then at the University of Michigan with grants from NSF and Intel. This drew on a lot of the tricks I learned.

- Previous Records: The current system had 797 rules, and used 4M rewrites. The previous record holder was 132 rules, for a rulset that combined the DNA replication and the "1-D mental-paper-folding" examples. It basically let you give it DNA from which it would construct a "protein" that would then fold itself up in 2D upon completion, using about 10K rewrites. The second biggest was 122 rules that did explicit geometric constructions to figure out how to get a virtual robot arm to bend in the right places to touch a target, bit only required 700-800 rewrites.

- Defuns in this system can call other functions. You saw several layers of that here starting with GetMatches( ). That is why defuns connect to the bus above (to receive funcalls), and below (to send out funcalls). It cannot do recursion however. Can you see why?