A mix of visual node editing and small compiled pieces has a history of being incredibly productive:

- Max/MSP
- Synthmaker
- PureData
- Game Maker
- Multimedia Fusion

- Synthmaker is now called Flowstone.
- But before that, it got high praise
- FL Studio has a version built-in now, even.

(These bullets are hyperlinked; you can click them.)

So what is it?

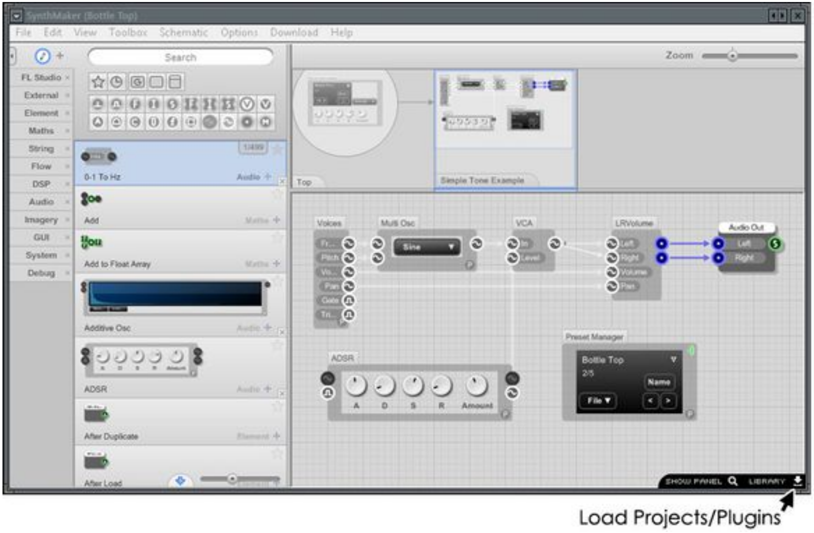

Figure 1: Basically, it was PureData/MaxMSP with a GUI editor stuck to it.

The faceplate editor will have to be written regardless.

(Unless there is just a static faceplate and widget set.)

- PureData has an embeddable runtime `libPD`
- But it generates in blocks
  - Could we patch the block size to *one*?
  - Would the performance still be tolerable?

What is PD really?

What if a custom one had to be made?

- Patch
- Object
- Message
- Pin
- Number

Patch

A patch is just a DAG with a tiny number of node types.

Khan's Algorithm to make sure looping hasn't ocurred.

Patch

A patch is just a DAG with a tiny number of node types.
Khan's Algorithm to make sure looping hasn't ocurred.

But actually…

`send~` and `receive~` immitate the block delay of cables.
So we just insert these shims wherever a loop is detected.
Usability \o/

Object

- It's literally just a name and some parameters with braindead syntax:
- `moses 5`
- Sends values below five to the left and values five or above to the right.
- And GUIs are created by typing in a named object.

- But these are actually just loaded from C modules.
- Only need a *very* basic ABI to load, ask for a table of modules, store, etc.
- `dlopen`, `dlsym`, `dlclose`, etc for live reloading (DSP compiler of choice to make individual blocks stays out of our binary!)

Messages

- The leftmost input is “hot” and any messages to it cause the node to propagate.
- Other inputs are “cold” and are acknowledged/stored but data isn’t sent further down.

Messages

- The leftmost input is “hot” and any messages to it cause the node to propagate.
- Other inputs are “cold” and are acknowledged/stored but data isn’t sent further down.
- These are basically just typed lists.
- Most of them are small; just use a fixed-size buffer and a discount slab allocator here.

Messages

- The leftmost input is “hot” and any messages to it cause the node to propagate.
- Other inputs are “cold” and are acknowledged/stored but data isn’t sent further down.
- These are basically just typed lists.
- Most of them are small; just use a fixed-size buffer and a discount slab allocator here.
- PD also lets you put messages as objects in the patch, which can be triggered when they receive a “bang.”

The GUI Problem

- Code for editing the noodle graph is actually *almost the same* as the code for moving panel widgets.
  - They are just “graph nodes” with no i/o pins!
- Panel size would be a crinkle since Rack doesn’t want you to resize your panel at runtime.

The Node Problem

- Would have to write the fundamentals (bang, mult, div, etc.)
- But if you're only dealing with control and CV, not midi, this is actually much less.
- Can still farm out workhorse DSP modules to Faust/etc (anything that compiles to C!)
  - Actually with some trickery, backends could be pluggable.

Pluggable?

- Some modules say “hey I don’t really know what I’m doing.”
- We ask them if they can handle the object creation messages.
  - This is where they run their JIT, python code, etc, . . .
- If they can’t, just move to the next adapter.

Are we winning yet?

- (Someday) convert the panel to C/C++ code.
- (Someday) convert the graph to C/C++ code.
- (Someday) code conventions so blocks can be built in static or dynamic mode
    - dynamic: gets loaded, goes through API; tells us how to gen code to call it in static builds
    - static: just splays the code out from the graph+block feedback and makes a Rack plugin you could build with `make`