aMaySyn documentation

Team210/QM

20/21/2018

for self-reference mostly, here are the current key events:

1 GLOBAL

these are global functions, regardless of which of the widgets is currently active.

[ESC] quit the amazing aMaySyn.

[TAB] toggle which of the TRACK/PATTERN WIDGET is active (notice the purple frame)

[F1] randomize the aMaySyn Colors! (optics only)

[F2] rename the whole song - title.may will be the filename, and title.syn will be the synth definition file which is read (if not present, test.syn will be taken). I have to fix something here, it can lead to stupid inconsistencies at this point.

[F5] as in the point before: will reload the synth definition file from title.syn / test.syn

[F8] print some debug stuff (did I really implement this? idk)

[F11] UPCOMING: that nice curve editor I'm planning, for "global automation curves"

[CTRL]+[N] clear the whole song

[CTRL]+[L] load title.may TODO: input prompt for filename

[CTRL]+[S] save title.may TODO: input prompt for filename

[CTRL]+[B] export the GLSL code to title.glsl and copy it directly into the clipboard
→ just paste into shadertoy and execute! :)

2 TRACK WIDGET

here, you set MODULEs into TRACKs. Each MODULE is assigned a PATTERN as well as more parameters, as *time of start* and *transpose* (for now).

[SHIFT]+[↑/↓] transpose module up/down (does not affect pattern!)

HIFT]+[←/→] move module through time

[HOME/END] move module to beginning/end of track

[↑/↓] switch to the track above / below

[←/→] switch to the neighboring module to the left/right

[HOME/END] switch to the first / last module in this track

[+] add empty module to the track

[C] add module to the track and assign the current pattern

[−] delete current module [TODO: undo function! not in there yet.]

CTRL]+[←/→ : move all modules in this track through time

[C]

[Pg↑/Pg↓] assign another pattern

[A/S] switch this track's synth (Drums, GFX, None are special values). If you update via F5, this list might change!

[F3] rename track

[F4] change some track parameters in a space-separated list of values (currently, these are *track normalization factor* and *maximum note release* (more about this in the Sequencer section)

[ALT]+... Usually it's like that: during editing, you will get more precision

3 PATTERN WIDGET

here, you set NOTEs into TRACKs.

4 Sequencer

The way aMacSyn stored its song information is structured in TRACKS, MODULES, PATTERNS and NOTES. It is designed like this to allow for long repetitions without the need to store much redundancy. Each TRACK contains MODULES, each MODULE is assigned one specific PATTERN, and each PATTERN contains NOTES. E.g. If you want to repeat a certain structure of 8 notes for 4 times, write these 8 notes into a pattern, then put 4 modules inside your track and assign each module this specific pattern − voila; you just saved some storage.

Now the GPU lives in a way in which each unit of time (e.g. each sample) has no memory of the one beforehand. Therefore, each sample we have to figure out which modules are playing simultaneously, and then inside the corresponding patterns, which notes play simultaneously, and with this knowledge eventually call the synth generator.

max release:
One specific detail is the idea of *release*, e.g. a note that was played before and was defined in a way not to stop instantaneously after the note itself ended. You can define this value for each track (currently via pressing **[F4]**) so the sequencer will then look for notes that *at most* this amount of beats might have stopped before the actual point in time. Obviously, if you want to implement long-releasing synths, you have to increase this value, but in order to save computation time, try to keep it as short as possible, or just have a super-mega-overpowered machine which can live with any max release.

5 The aMaySyn language

My mission was set to find a way in which one has the option to think terms established in the synthesizer world, not having to think about the peculiarity of GLSL, but still be able to combine ideas for synths, in a resourceful and playful way.

Now I have no idea whether that worked. But I have this language for now. It works like this: there are basic forms, some compound forms, and after a while, there are main forms. The latter ones are the ones you can actually select in the aMaySyn editor, Let's first talk about non-drum-synths for now − this is the list of forms I currently implemented

- Generally, each of the following `<FORM...>` parameters can consist of a constant numerical value, a basic form, an advanced form, or also an arbitrary product of any of the preceding forms, separated with a multiplication asterisk **but not any space!** E.g. `2.0*envA*saw1` will return the product of two times some form called `envA` times some form called `saw1`. Get the idea?

- `MAIN <ID> <FORM1> <FORM2> <FORM3> ...`
  each MAIN form will be a linear combination of each term that is passed after its name, and it will be called NAME in the editor. they will be available in the track synth list in the order they appear in the synth definition (.syn) file.

- Reserved forms ('uniforms'): These are pre-defined values that you can mention, but not redefine. These are (currently)

`f`	the note frequency in Hz
`t`	the time in seconds
`t`	the time in seconds since the beginning of the current note
`B`	the current beat (which is time scaled with the beats per seconds, or BPM/60)
`vel`	the note velocity ...one day, you can modify that in the editor
`Bsyn`	the number of the synth assigned to the current track (you will not use it)
`Bproc`	the relative percentage the current note is inbetween its beginning and end
`Bprog`	the amount of beats since the beginning of the current note, i.e. just `t` scaled
`L`	the length of the track in beats
`tL`	the length of the track in seconds
`SPB`	the value of seconds per beats, i.e. just the inverse of BPS
`BPS`	the value of beats per seconds, e.g. just BPM/60
`BPM`	the value of beats per minute, e.g. the track speed
`note`	the integer value of the current note in half steps (i forgot where I defined the zero)
`Fsample`	the sample rate in Hz, most probably 44100

- `OSC <ID> [SHAPE:SIN/SAW/TRI/SQU] <FREQ> <PHASE> <PARAMS>`:

- `LFO <ID> [SHAPE:SIN/SAW/TRI/SQU] <FREQ> <PHASE> <PARAMS>`:

- `ENV <ID> [SHAPE:ADSR/ADSREXP/DOUBLESLOPE/SS/SSDROP/EXPDECAY/EXPDECAYREPEAT] <PARAMS>`:

- `GAC <ID> <PARAMS>`: yeah, I'll get to that.

- `FORM <ID> <OPERATOR> <PARAMS>` so these are compound forms. `<OPERATOR>` can be - currently - one of:

  - `MIX`
  - `DETUNE`
  - `PITCHSHIFT`
  - `QUANTIZE`
  - `OVERDRIVE`
  - `CHORUS`
  - `DELAY`
  - `WAVESHAPE`
  - `SATURATE`

  yeeeah... explain these later on...

- `DRUM <ID> ...` the DRUMs are actually similar to OSCs, but with more pre-defined component, and independent of the trigger note or its frequency (however, if you want some variation, you might use `vel`, I just have to implement that into the editor