METASURFACE64

Dominique Blanchemain *

Documentation : http://blanchemain.info/Documents/Programmation/index.php?page=metaSurface
https://github.com/dblanchemain/metaSurface
Document translation(EN): Mary Waller
Plouguerneau, France
dblanchemain@free.fr

ABSTRACT

metaSurface64 (Figure 2) is a control surface for performing continuous real time sound transformations in a window using a mouse. It has its own loop generator, up to 64 voices, and a multi-effect FX engine. It is software developed in C ++ for Linux. It can be compiled for windows with Msys2.

"A tiling or tessellation of a flat surface is the covering of a plane using one or more geometric shapes, called tiles, with no overlaps and no gaps (Wikipedia)." In this document, we will use the terms **tiling** to designate the flat surface and **pad** for geometric shape.

Each **pad** on the surface allows direct control of the gain of the audio output with a slider here called a **mixer**. This mixer also allows you to control the parameters of pad attached plugins.

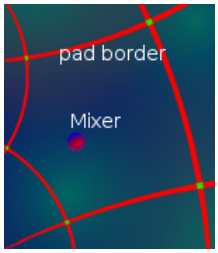

Figure 1: *Mixer.*

It is also possible to control the tracks and plugins of an external sequencer (**Ardour**[1] or **Reaper**[2]). To do this, the application uses modules controllable by **OSC** which come from the Faust language library which is embedded. It is also possible to have an external mono audio input for a pad.

You can write new plugins in the **Faust language** and integrate them into the surface. New controllers for external plugins incorporated in your DAW (digital audio workstation) can be added to extend the possibilities.

Finally, you can create new tiling for your surface.

Pads audio files can be multi-channel. The **Jack** audio output of each pad can be set independently and can be single or multi-channel.

This document is not a manual but a simple presentation of the software. I can't ignore the interface description. This is essential to correctly perceive the number of parameters transmitted to the **DSP** generator.

The documentation is available here:

http://blanchemain.info/Documents/Programmation
/index.php?page=metaSurface

1. INTRODUCTION

Creating audio objects for electroacoustic composition is a time-consuming and complicated process.

The tools and plugins integrated into the sequencers allow you to work on samples in real time, but the management of the plugins interfaces is often not adapted to the simultaneous manipulation of multiple parameters.

Audio file editors, **Audacity** for example, don't work in real-time and don't allow continuous adaptation of the settings according to what is heard. In all cases, it is not possible to simultaneously alter the settings for several plugins.

AudioMulch[3] was the first application to offer a metasurface which allowed the modification of several plugins settings at a time simply by moving the mouse. This tool became particularly relevant for creating music.

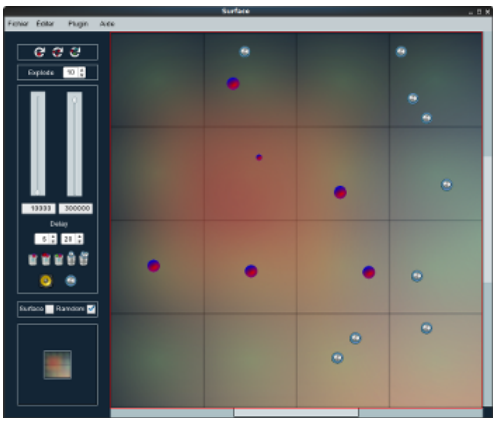

Figure 2: *metaSurface64.*

By taking the concept of **AudioMulch** and combining it with the idea of a Midi multi-pad like the **Novation Lauchpad**(Figure 3) or the **LinnStrument**, i.e. a trigger surface for playing audio files, I came to design this software **metaSurface64** (Figure 2) with a limit of 64 audios channels which corresponds, among other things, to the limit of the number of audio channels of **Reaper**.

The fact that you can integrate your own tiling into the **metaSurface64** gives the application a lot of flexibility.

* This work was supported by S.Letz

[1] https://ardour.org/

[2] https://www.reaper.fm/

[3] http://www.audiomulch.com/

Figure 3: *Example Midi Pad.*

2. PRESENTATION

Most multi pads take the form of a group of rectangular buttons which are used to start Midi events, along with settings buttons. **metaSurface64** consist of a window with a tiling and a variable number of pads with a management interface.

Clicking on a pad creates a mixer and starts looping an audio file or audio input. The volume is determined by the distance from the center of the pad. Moving the mixer (Figure 1) adjusts the volume in real time as a function of this distance.

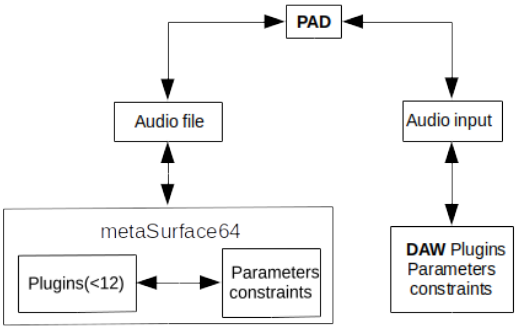

Figure 4: *Pad.*

An ordered list of plugins can be associated with each pad. Their parameters can be modified statically by the interface or dynamically if one associates them with moving the pad's mixer.(Figure 4)

When the surface is activated by clicking on the speaker button, **libfaust** allows the generation of a DSP and its interface GTK. Each tab represents a pad.

It is possible to obtain an interface for multichannel audio outputs if the multichannel option has been selected (Figure 11).

2.1. Pad

Each pad has an interface with specific parameters. Without going into detail, it is possible to specify an audio file for the automatic loop engin integrated in the surface. This loop is activated by the DSP creation if a mixer is defined for the pad. Audio files can be mono or multi-channel.

The playback can be multi-channel if this option has been chosen.

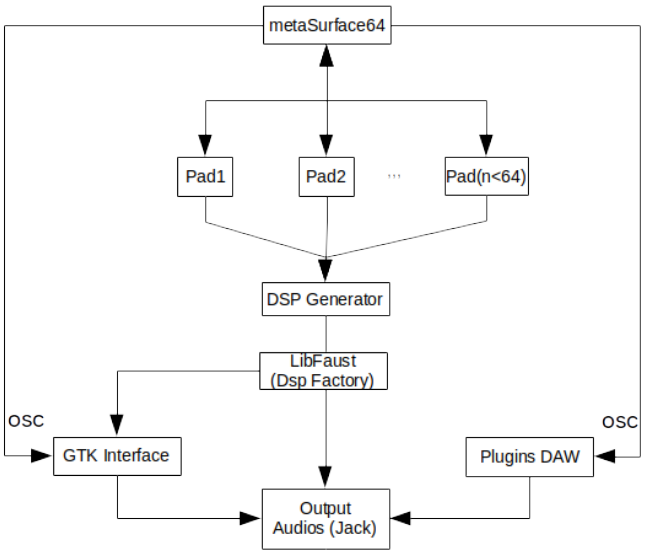

Figure 5: *Design.*

If no file has been selected, there is a mono entry visible in **Jack**(Figure 6).

It is possible to move the plugin management on to the sequencer by selecting the DAW option. It will be necessary to define the track associated with the pad in the sequencer.

Faust obviously does not intervene directly in the real-time management of sequencers' plugins, but he does allows the creation of new ones. On the other hand, it is possible to control them with the metaSurface64.

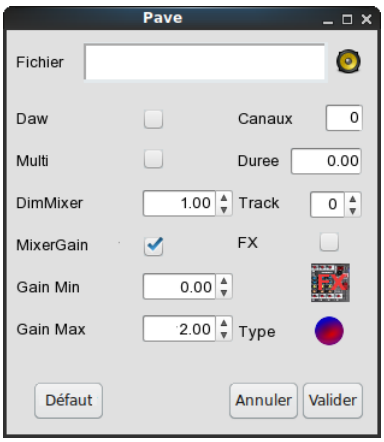

Figure 6: *Pad interface.*

By default, moving the mixer affects the gain of the output. It is possible to change this option by deselecting it. Dynamically modifiable plugins will always be modified by the mixer.

This is also where we define a chain of effects.

3. PLUGINS

3.1. Generality

Width Faust, I have made 12 plugins which are integrated in Métasurface64: Compression, Delay, Distortion, Flanger, Freeverb, Granulator, HPF/LPF, Mixer, MoogVCF, parametricEQ, Phaser, RingModulator(Figure 7).

The associated DSP files allow generation of LV2 or VST, with for example the **Faust IDE**, to use them in the sequencers.

I propose versions in mono and 8 channels(vst for **Reaper**).

Ardour duplicates the plugins according to how many channels are in the track. **Reaper** doesn't offer this functionality and its management of multi-channel tracks is problematic for controlling parameters with **metaSurface64**. It is recommended to generate adapted plugins from the DSP files proposed.

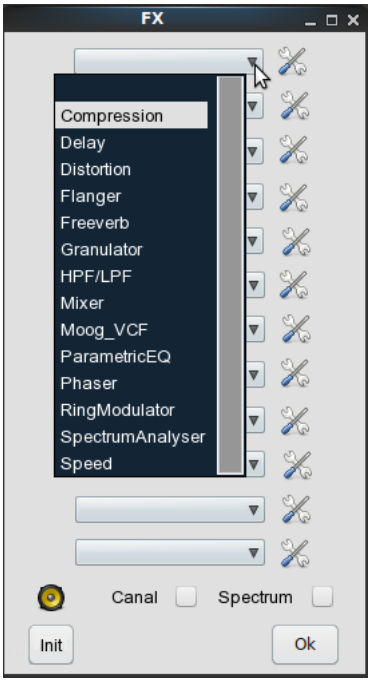

Figure 7: *Effects selection.*

If the option DAW is selected for a pad, the plugins used in the sequencer can be controlled from **metaSurface64**. Their order in the surface and the sequencer must be rigorously respected for this to work.

The configuration of the dynamically modifiable parameters is done by clicking on the icon which can be found immediately on the right of each plugin. The structure of the windows for defining the parameter limits is the same for all the plugins(Figure 8).

A parameter must be validated for it to be taken into account. It is possible to define the variation interval of a parameter. We can also specify a variation coefficient to accelerate or slow down the modification with the movement of the mixer.

3.2. Add a plugin

metaSurface64 allows the creation and integration of new plugins written in **Faust language** by editing a DSP file with either an editor like **Emacs** or using **Grame's FaustIde**[1]. The latter also

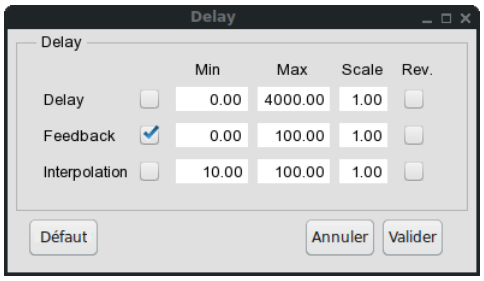

Figure 8: *Example limits table.*

allows exportation of plugins in LV2 and/or VST format. This possibility increases the potential to experiment while working on researching new sounds.

For exemple, to create a new plugin: by editing this code from

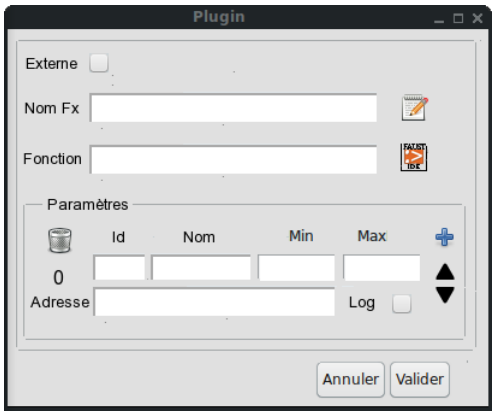

Figure 9: *Add plugin.*

R.Michon:

```
import("stdfaust.lib");
import("music.lib");
mu = library("music.lib");
myFbComb=ba.bypass1(combp,myComb)
with{
  comb_group(x)=vgroup("Comb Filter", x);
  combp=comb_group(vgroup("[0]",checkbox("
      Bypass [tooltip: When this is checked,
       Comb Filter has no effect]")));
   myComb=comb_group(hgroup("[1]",+~(mu.
       delay(2048,delLength)*(-a1))));
        a1 = vslider("a1",0,-1,0.999,0.001)
             : si.smooth(0.999);
        delLength = slider("delLength
            ",1,1,2000,1);
};
```

and saving it in the folder

```
$HOME/metaSurface/Plugins
```

with the name feedbackCombFilter.lib, this new plugin will appear in your plugins list.

By adding this line:

```
process = _:myFbComb:_;
```

you can generate the LV2 and VST which will be available for your sequencer. Then, it will be necessary to define the modifiable parameters with their limits, like for the other plugins, to be able to control them dynamically from the metaSurface64.

It's also from this window that you can define the controllers of the external plugins used in your DAW (for example the Calf-plugins for Ardour).

4. CONCLUSION

The **metaSurface64** is a surface destined for the creation of complex audio objects. It offers a realtime environment to facilitate research with independant chains of effects for each audio channel. I hope it will be useful for creators and will give them new perspectives.

The code is not perfect, but it should get better quickly.

The next improvement will bring a better interface for integrating new plugins.

By elsewhere, it will be necessary to develop a Debian package to make installation easier on **Linux**.

5. ACKNOWLEDGMENTS

Many thanks to S.Letz for his patience and support!

6. REFERENCES

[1] Stephane Letz, "Online faust ide," in *Faust programming language*, GRAME, France, May 2020.

[2] Ross Bencina, "Audiomulch metasurface," Melbourne, Australia, May 2016.

[3] Julius O. Smith, "Signal processing libraries for Faust," in *Proceedings of Linux Audio Conference (LAC-12)*, Stanford, USA, May 2012.

[4] Yann Orlarey, "Version librairie du compilateur faust," in *HAL Id: hal-00965271*, GRAME, France, Mars 2014.

[5] Yann Orlarey Stéphane Letz, Dominique Fober, "Comment embarquer le compilateur faust dans vos applications ?," in *JIM*, GRAME, France, Mars 2013.

[6] Romain Michon and Julius O. Smith, "Faust-STK: a set of linear and nonlinear physical models for the Faust programming language," in *Proceedings of the 14th International Conference on Digital Audio Effects (DAFx-11)*, Paris, France, September 2011.

7. APPENDIX: EXEMPLE INTERFACE DSP

Example interface for a surface with 16 pads:

Here is an example interface for managing plugins for a tiling in multi-channel mode(Figure 11). Sorry, a better resolution is not possible due to the size of the window.

8. APPENDIX: EXEMPLE DSP READ FILE

DSP generator in C++ for simple playback of an audio file

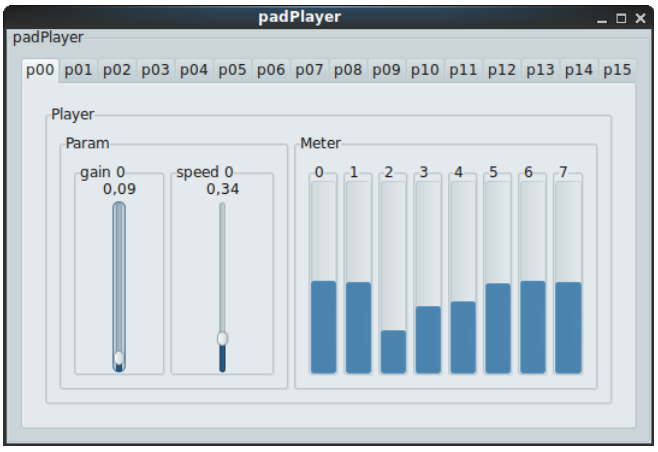

Figure 10: *Interface GTK for a surface.*

```
void metaSurface::testFileDSP(string st1){
  SF_INFO sfinfo;
  string sFileName=tabPave[selectPad-1].
      getPath()+"/"+tabPave[selectPad-1].
      getFile();
  const char *path=sFileName.c_str();

  SNDFILE* sndfile=sf_open(path, SFM_READ,
      &sfinfo) ;
  sf_close(sndfile) ;

  string nameFile="simplePlayer.dsp";
  if(sfinfo.channels>1){
  string prog;
  prog=prog+"import(\"stdfaust.lib\");";
  prog=prog+"import(\"soundfiles.lib\");";
  prog=prog+"ds=soundfile(\"[url:{\'"+path
      +"\'}]\","+to_string(sfinfo.channels)
      +");";
  prog=prog+"vmeter(x)= attach(x, envelop(x
      ) : vbargraph(\"[2][unit:dB]\", -70,
      +5));";
  prog=prog+"envelop = abs : max ~ -(1.0/ma
      .SR) : max(ba.db2linear(-70)) : ba.
      linear2db;";
  prog=prog+"sample1 = so.sound(ds, 0);";
  prog=prog+"gain = vslider(\"[0]gain
      \",0.1,0,2,0.01) : si.smoo;";
  prog=prog+"gate = button(\"[1]gate\");";
  prog=prog+"gdec(x) = hgroup(\"Player\",x)
      ;";
  prog=prog+"tdec(x) = gdec(vgroup(\"[1]\",
      x));";
  prog=prog+"lect(x) = gdec(hgroup(\"[2]\",
      x));";
  prog=prog+"lgain=tdec(gain);";
  prog=prog+"lgate=tdec(gate);";
  prog=prog+"lmet=lect(par(j,"+to_string(
      sfinfo.channels)+",hgroup(\"c%2j\",
      vmeter)));";
```

Figure 11: *Dsp multi-channel*

```
  prog=prog+"smp1 = display_progress(
      sample1.play_progress(lgain,lgate)):
      lmet;";
  prog=prog+"process = smp1;";
  createDSP(prog, st1);
  }
}
```

9. APPENDIX: EXEMPLE DSP CREATE

(In a string for DSP Factory)
Pad 0 audio file(stereo) with 1 plugin: Delay
Pad 1 audio file(stereo) with 1 plugin: flangerMono
Pad 2 to 15: input (mono link by Jack)

```
import("./faust/stdfaust.lib");
import("./faust/soundfiles.lib");
import("./faust/metaSurfaceFaust.lib");
ds=soundfile("[url:{'../Musique/Ardour/
    workInProgress/export/b_Audio 4-2.14ps1.
    wav';'../Musique/Ardour/workInProgress/
    export/b_Audio 4-2.14ps2.wav';'./
    metaSurface4/sound/vide.wav';'./
    metaSurface4/sound/vide.wav';'./
    metaSurface4/sound/vide.wav';'./
    metaSurface4/sound/vide.wav';'./
    metaSurface4/sound/vide.wav';'./
    metaSurface4/sound/vide.wav';'./
    metaSurface4/sound/vide.wav';'./
    metaSurface4/sound/vide.wav';'./
    metaSurface4/sound/vide.wav';'./
    metaSurface4/sound/vide.wav';'./
    metaSurface4/sound/vide.wav';'./
    metaSurface4/sound/vide.wav';'./
    metaSurface4/sound/vide.wav'}]",2);

vmeter(i,x)= attach(x, envelop(x) :
    vbargraph("%2i[unit:dB]", -70, +5));
envelop = abs : max ~ -(1.0/ma.SR) : max(ba
    .db2linear(-70)) : ba.linear2db;
sample(x) = so.sound(ds, x);
sgain(i) = vslider("[0]gain%2i
    ",0.1,0,2,0.01) : si.smoo;
ingain(i) = vslider("[0]gain%2i
    ",0.1,0,2,0.01) : si.smoo;
sspeed(i)=vslider("[1]speed%2i
    ",1.0,0,2,0.01) : si.smoo;
lect2(x)=hgroup("Player",x);
tdec(x)=lect2(hgroup("[0]Param",x));
lect(x)=lect2(hgroup("[1]Meter",x));
lgain(i)=tdec(sgain(i));
lspeed(i)=tdec(sspeed(i));
lmet=lect(par(j,2,vmeter(j)));
base=tgroup("Pad",hgroup("p00",hgroup
    ("[0]",sample(0).loop_speed_level(lspeed
    (0),lgain(0)):lmet:par(j,2,hgroup("[0]",
    delay))):>_),
hgroup("p01",hgroup("[0]",sample(1).
    loop_speed_level(lspeed(1),lgain(1)):
    lmet:par(j,2,hgroup("[0]",flangerMono)))
    :>_),
hgroup("p02",hgroup("[0]",_:*(ingain(2))<:
    lmet):>_),
hgroup("p03",hgroup("[0]",_:*(ingain(3))<:
    lmet):>_),
hgroup("p04",hgroup("[0]",_:*(ingain(4))<:
    lmet):>_),
hgroup("p05",hgroup("[0]",_:*(ingain(5))<:
    lmet):>_),
hgroup("p06",hgroup("[0]",_:*(ingain(6))<:
    lmet):>_),
hgroup("p07",hgroup("[0]",_:*(ingain(7))<:
    lmet):>_),
hgroup("p08",hgroup("[0]",_:*(ingain(8))<:
    lmet):>_),
hgroup("p09",hgroup("[0]",_:*(ingain(9))<:
    lmet):>_),
hgroup("p10",hgroup("[0]",_:*(ingain(10))<:
    lmet):>_),
hgroup("p11",hgroup("[0]",_:*(ingain(11))<:
    lmet):>_),
hgroup("p12",hgroup("[0]",_:*(ingain(12))<:
    lmet):>_),
hgroup("p13",hgroup("[0]",_:*(ingain(13))<:
    lmet):>_),
hgroup("p14",hgroup("[0]",_:*(ingain(14))<:
    lmet):>_),
```

```
hgroup("p15",hgroup("[0]",_:*(ingain(15))<:
    lmet):>_));
smp1=vgroup("[0]",base);
process = smp1;
```

10. APPENDIX: CODE DSP FACTORY

localDsp is a string which corresponds to code DSP and st1 allows to specify if the option OSC is desired. The file .Padplayer puts in the user's directory saves the parameters for a subsequent use when the GTK window is closed.

```
void metaSurface::createDSP(std::string
    localDsp, string st1){
  char name[256];
  char nameAudio[256];
  char rcfilename[256];
  char* home = getenv("HOME");
  string s = "padPlayer.dsp";
  char filename[s.length() + 1];
  strcpy(filename, s.c_str());
  snprintf(name, 255, "%s","Padplayer");
  snprintf(rcfilename, 255, "%s/.%s-rc",
      home, name);

  bool is_osc;
  if(st1=="-osc"){
    is_osc = 1;
  }else{
     is_osc = 0;
  }

  dsp_factory* factory = nullptr;
  dsp* DSP = nullptr;
  MidiUI* midiinterface = nullptr;
  jackaudio_midi audio;
  GUI* oscinterface = nullptr;
  string error_msg;

  cout << "Libfaust version : " <<
      getCLibFaustVersion () << endl;

  factory = createDSPFactoryFromString(
      filename,localDsp, 0,NULL, "",
      error_msg, -1);
  if (!factory) {
     cerr << "Cannot create factory : " <<
         error_msg;
     exit(EXIT_FAILURE);
  }
  std::cout << "Factory : " << factory<<
      std::endl;
  cout << "getCompileOptions " << factory->
      getCompileOptions() << endl;

  DSP = factory->createDSPInstance();
  if (!DSP) {
     cerr << "Cannot create instance "<<
         endl;
     exit(EXIT_FAILURE);
  }
  GUI* interface = new GTKUI(filename,0,
      NULL);
  DSP->buildUserInterface(interface);
  FUI* finterface = new FUI();
  DSP->buildUserInterface(finterface);
  SoundUI* soundinterface = new SoundUI();
  DSP->buildUserInterface(soundinterface);

  if (is_osc) {
     int argc1=5;
     char* argv1[64];
     argv1[0]=filename;
     argv1[1]=(char*)"-xmit";
     argv1[2]=(char*)"0";
     //cout <<argv1[0]<<argv1[1] << " : "
         << argv1[2]<< endl;
     argv1[3]=(char*)"-port";
     string s=to_string(ref->getOSC());
     argv1[4]=(char*)s.c_str();//"5540"
     cout << "osc " << argv1[4] << endl;
    oscinterface = new OSCUI(filename,argc1
        ,argv1);
    DSP->buildUserInterface(oscinterface);
  }

  if (!audio.init(basename(filename), DSP))
      {
     cout << "audio.init : " << 0<< endl;
  }
  finterface->recallState(rcfilename);
  audio.start();

  if (is_osc) {
        oscinterface->run();
  }

  interface->run();

  audio.stop();

  finterface->saveState(rcfilename);

  delete DSP;
  delete interface;
  delete finterface;
  delete oscinterface;
  delete soundinterface;

  deleteDSPFactory(static_cast<
      llvm_dsp_factory*>(factory));

}
```