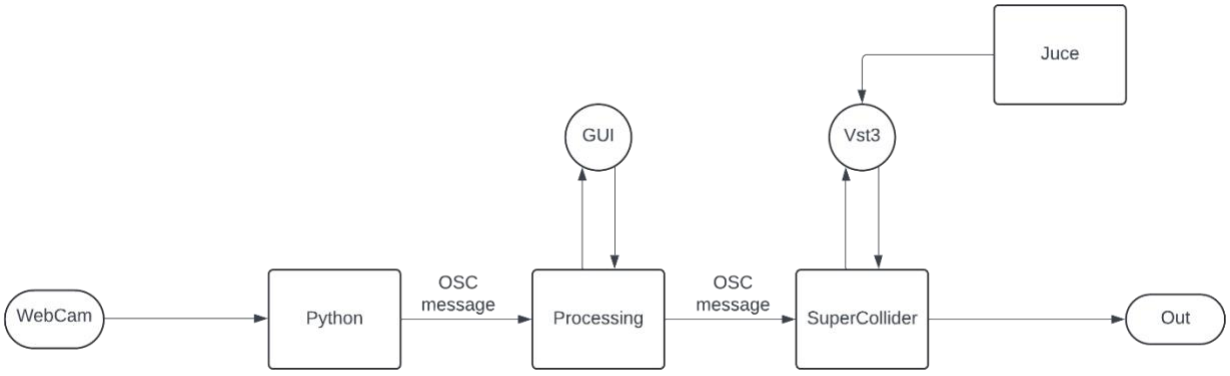
Academic Year 2023-24

052483 Computer Music: Languages and Systems

Homework

CARPS



GOAL

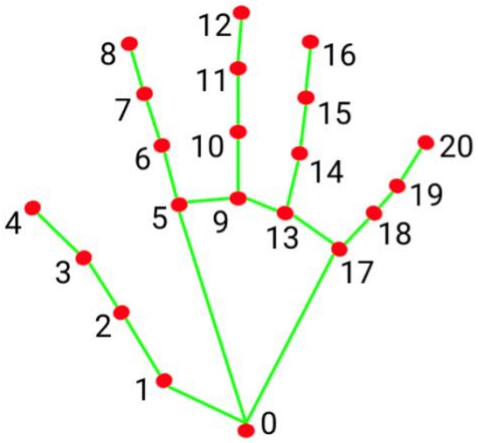
The goal is to create an instrument controlled by hand to make user understand what it feels like to "touch" by hand abstract aspect of sound such as duration in an arpeggiator or sustain and handling different effects.

INTERACTION

The interaction is based on MediaPipe Solutions, an open-source project by google that offers a variety of APIs and libraries that, thanks to Artificial intelligence and Machine learning, can be used for tracking and general image analysis and recognition.
In this project MediaPipe has been used for hand tracking and to also draw the hand and the tracked joint.
To implement this, we created a python program based on the cv2 library (that gives the possibility to open the webcam and use data gathered from it), and MediaPipe.
After calling the 2 libraries and setting up the necessary to track, the program call different hand landmarks and perform calculation to get the values that will later be used to change the parameter of the arpeggiator and the effects.
The complete set of values that the code sends to Processing consist in:

- Pitch which varies depending on the position on the X axis of the wrist (landmark 0).
- Distance_Thumb_Index which is simply the distance between the tip of the thumb (landmark 4) the tip of the index finger (landmark 8).
- Distance_Pinky_Tip which is the distance between the tip of the pinky finger (landmark 20) and the wrist.
- Middle_Bend which is obtained by confronting the distance between the tip of the middle finger (landmark 12) and the wrist and the distance between the DIP of the middle finger (landmark 11) and the wrist. The value will be 1 if the first one is bigger and 0 if the second one is bigger, meaning 1 if the finger is extended and 0 if the finger is bent.
- Distance_Middle_Tip which is the distance from the tip of the middle finger and the wrist.
- Ring_Bend which is the same as the Middle_Bend but for the ring finger.

· swipe_x which is based on the speed of the wrist, if the wrist travel a certain distance along the x axis in a small period of time, the swipe will be 1 in a direction and -1 in the opposite, while if the wrist doesn't the value of swipe will be 0. The swipe also has a cooldown so that the same movement doesn't trigger the swipe multiple time.
· swipe_y is the same idea of swipe_x but on the y axis.
· Rotation_x which is the difference between the position of the MCP of the index finger on the x axis and the position of the MCP of the pinky finger on the x axis. The value of Rotation_x will be maximum when the hand is frontal to the camera, minimum when it's turned away from the camera and 0 when the hand is perpendicular to the camera.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

SUPERCOLLIDER

Our supercollider sound design is based on a wavetable synthesis with 5 different waves that blend together.

```
(
~buf = Buffer.allocConsecutive(5, s, 16384);
~wt = [
    Signal.sineFill(8192, 1 / (1..50), 0 ! 50),
    Signal.sineFill(8192, [1 / (1, 3..50), 0 ! 25].lace(50), 0 ! 50),
    Signal.sineFill(8192, [1/(1..40), 0!40].lace(40), 0!13),
    Signal.sineFill(8192, [1/(1, 3..40), 0!20].lace(50), 0!3),
    Signal.sineFill(8192, (10..40), 0!3)
];

~buf.do({ |buf, i| buf.loadCollection(~wt[i].asWavetable) });
)
~wt.plot;       // optional visualization
```

This code regards the creation of the different waves and their allocation in consecutive buffers. We use the sineFill method to create a Signal instance composed of sums of harmonically related sines. This method expects three arguments: the wavetable size, an array of harmonic amplitudes, and an array of initial phases specified in radians.

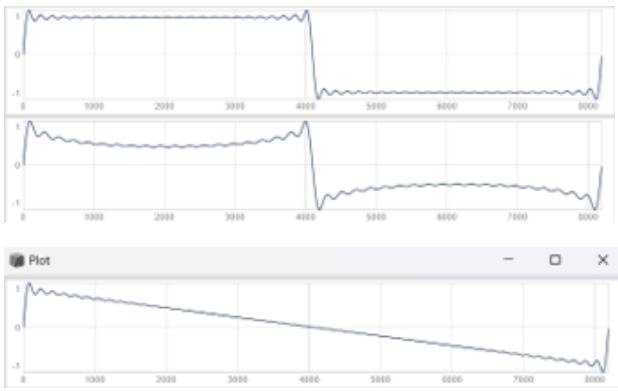
```
// Synthdef
(
SynthDef(\carps, {
    arg freq = 200, out = 0, sustain = 0.5;
    var sig, bufmod;

    bufmod = ExpRand(0.01, 3.999);
    sig = VOsc.ar(~buf[0].bufnum + bufmod, freq);
    sig = sig * EnvGen.kr(Env.perc, doneAction:2);
    sig = sig * EnvGen.kr(Env.adsr(sustainLevel:sustain), doneAction:2);
    sig = sig * 0.3 ! 2;
    sig = LeakDC.ar(sig, 0.8);
    Out.ar(out, sig);
}).add;
)
```

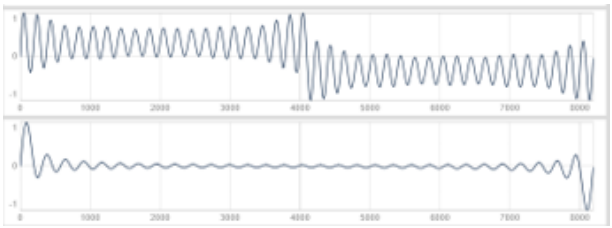
This part of code instead refers to the definition of the synth.

The most interesting part of the code is the usage of the Ugens Vosc that is specifically recommended for the wavetable synthesis; indeed, VOsc can interpolate between two wavetables, creating a blended waveshape.

The first argument of VOsc is the buffer position and we decided to add some randomness with the command “bufmod = ExpRand(0.01, 3.999)”, so every time we play the synth the sound is different because the position of the buffer, and so the waveshape, is different.

To add some detune to the synth we use LeakDC, that is a linear filter that removes DC bias from a signal. In the end for a percussive sound, that fits very well with an arpeggiator, we use Env.perc.





Here we can see the 5 different waveshape.

```
// PLUGIN SECTION
(
SynthDef(\plugin, {
    arg bus;
    var sig = In.ar(bus, 2);
    ReplaceOut.ar(bus, VSTPlugin.ar(sig, 2, id:\plugin));
}).add;
)

~vstBus = Bus.audio(s,2);

~vst = VSTPluginController(Synth(\plugin, [\in, ~vstBus, \out, 0]), id:\plugin);
~vst.open("Delay.vst3", editor:true, verbose:true);
~vst.editor;
```

In this part of code, we create the connection between supercollider and a vst3 plugin made in Juce.

To do this operation we use the SC extension “VSTPlugin” and with the command VSTPluginController we can load the vst3 and apply it on the sound generated before.

```
NetAddr("127.0.0.1", 57120);
// Effetti
// Panner
(
OSCdef('OSCReceiver1',
    {
        arg msg;
        var param;
        param = msg[1];
        ~vst.set(0, param);
    },
    "/pan");
)
```

To conclude, here is an example of communication between SC and processing, by using the OSC protocol, on the address 127.0.0.1 and the port 57120.

We use the command ~vst.set to handle the plugin parameters.

```
// Arpeggio
(
~arp = Pbindef(\cARPs,
    \instrument, \carps,
    \dur, 0.20,
    \degree, Pseq([0, 4, 6, 8], inf),
    \sustain, 0.5
);
)
```

Finally the arpeggio section is managed by a simple Pbindef with the sound taken by the previous synth.

PROCESSING

```
if (theEvent.isFrom("panner")) {
  OscMessage msg4 = new OscMessage("/pan");
  msg4.add(panner);
  oscP5.send(msg4, myPc);
  msg4.print();
}
```

This part of the code represents an example of osc communication in processing.

The event “panner” is a knob, and every time the knob value is changed, we send a message to supercollider.

Processing also receives OSC messages sent from python at different addresses depending on the “Window” we are in.

We also build the GUI in processing, with standard cp5 knob and buttons.

JUCE

In our homework, we used JUCE for the development of effects applied to arpeggiator.
Among the effects we developed, we included a panner, a distortion and a delay.

Panner: the panner was developed by applying the dsp module “Panner” already present in JUCE. It is presented as a oneknob effect, which as soon as it is opened is positioned in the center and moves the signal into the left and right channels consistent with the current position of the knob.
To achieve this effect without having artifacts when changing the parameter in real time, we use the “setRule” function with argument “squareRoot4p5dB.”

Distortion: the distortion is also designed as a one knob effect, where basically the dry/wet ratio between input and effect is managed (0% dry, 100% wet).
The effect for convenience was developed in a special source called “distortion” and named within the processblock of the PluginProcessor.
The distortion effect is implemented still using JUCE's dsp modules, specifically, for the final output, a chain of effects is created that include a pre-gain, a waveshaper and a post-gain.
The pre-gain amplifies the signal before it is distorted, then follows the waveshaper that applies the actual distortion based on the value indicated by the knob, and finally the post-gain that adjusts the signal after distortion so that there are always controlled volume levels.
Plugin state is managed via XmlElement, which allows saving or loading plugin parameters via instances that are then converted to a binary format accessible to DAWs.

Delay: this is the only effect that has more than 1 parameter that can be controlled in real time, since it is a simple delay, in addition to dry/wet we also have parameters that handle feedback and tempo.
This effect also uses a JUCE internal dsp, specifically DelayLine. The delay logic basically clears all unused output channels from the input, updates the delay time if it has changed, recalculating it according to the sample rate, processes each audio channel individually, mixing the dry and wet signals according to the DRYWET parameter, and uses a feedback mechanism in which the delayed sample is fed back into the delay line with a feedback factor controlled by the FEEDBACK parameter.
To avoid glitch audio when changing the TIME parameter in real time, each time it is changed, there is a reset of the buffer.

FLOW

We saw how the information is obtained from the webcam using python, how the GUI is created from Processing, how the plugin are created on Juce, how SuperCollider create the synth and the arp and how the plugin are loaded on SuperCollider.
Now we can explore how all this programs communicate.
The flow of information starts with Python. There a cycle sends out OSC messages with all the information described in the Python chapter. The interesting part about this is that based on the “swipe_x” and “swipe_y” the address varies. This idea gives us the possibility to create a sort of “windows” so that Processing will do different action based on the “window” we are in. The windows are note, arp, plugin1, plugin2, plugin3, the user can scroll between adjacent windows using their hand. A vertical swipe will give the possibility to scroll between note arp and the effect, a horizontal swipe will give the possibility to scroll between panner, distortion, delay, but only if the user is in the effect window.
Once the OSC messages are sent, Processing gets the message and will change only parameters related to the window the user it’s in based on the OSC message address. Processing changes the GUI knobs based on the message received and sends the values to SuperCollider.
Finally SuperCollider gets the OSC message from processing and set the values.