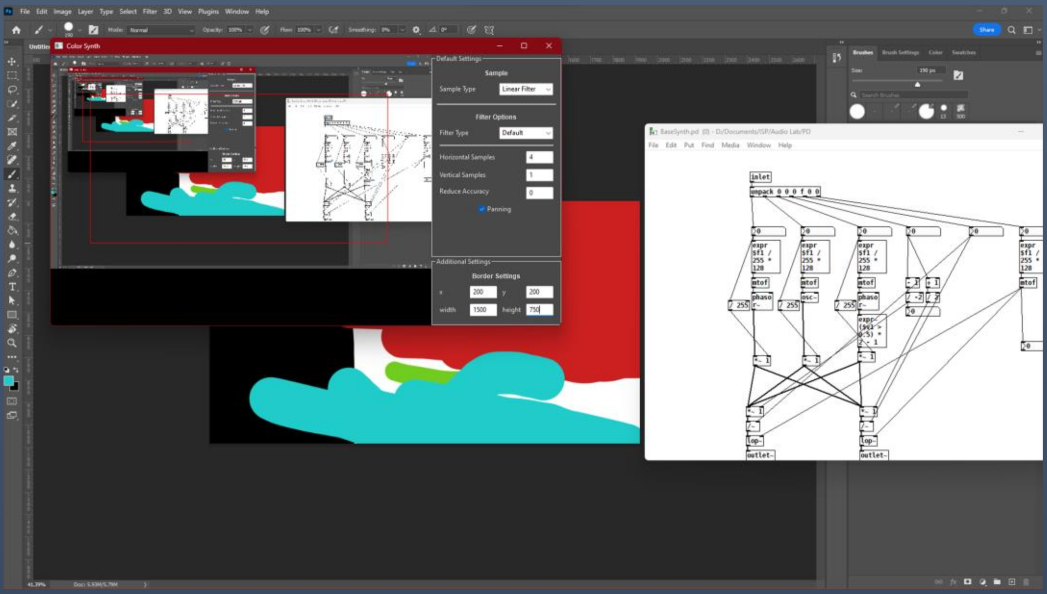
2023

How to send image data to a synthesiser and how can we use this?


File Edit Image Layer Type Select Filter 3D View Plugins Window Help
Color Synth
BaseSynth.pd - D:/Documents/ISP/Audio Lab/PD
inlet
outlet~

Default Settings

Sample

Sample Type: Linear Filter

Filter Options

Filter Type: Default

Horizontal Samples: 4

Vertical Samples: 1

Reduce Accuracy: 0

[✓ Panning]

Additional Settings

Border Settings

x: 200 y: 200

width: 1500 height: 750

CASIER ROBBE

INTERACTIVE SOUND PRODUCTION, HOWEST UNIVERSITY OF APPLIED SCIENCES, KORTRIJK, BELGIUM

ROBBE.CASIER@STUDENT.HOWEST.BE

TABLE OF CONTENTS

ABSTRACT

This paper explains the process I did to be able to send image data to a synthesiser and implemented ways to alter the results for use cases with a program. The program has two options for sending the data. One for more musical test cases, while the other was designed for videos. The first option has potential but learning how to use the program is not efficient enough.

INTRODUCTION

Having seen the GDC talk that Mick Gordon gave where he explained what he did to get an image into the soundtracks, I wanted to know if I could use an image to control a synth. With this I had some ideas of different ways the program that would handle the image. These were, first line, random line, pixel by pixel, using mouse to instruct which pixel's data should be send or a full iteration.

THEORETICAL FRAMEWORK

COMMUNICATION PROTOCOL

USER DATAGRAM PROTOCOL

In computer networking, the User Datagram Protocol (UDP) is one of the core communication protocols of the internet protocol suite used to send messages to interconnected devices.[1]

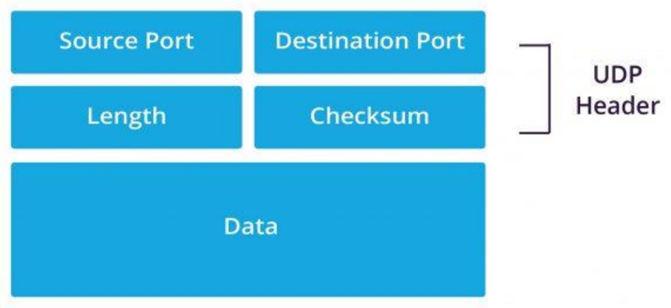


Figure 1

OPEN SOUND CONTROL

Open Sound Control (OSC) is a protocol for communication between computers, sound synthesizers and other multimedia devices.[2] OSC is a higher-level protocol built on top of UDP specifically designed for multimedia communication.

REDUCING TOTAL DATA

SPATIAL PARTITIONING

Spatial partitioning is a technique commonly used in games to divide an area into smaller areas, reducing computation for unnecessary data by limiting the specific action to the data that is available in the current area or its neighbours.

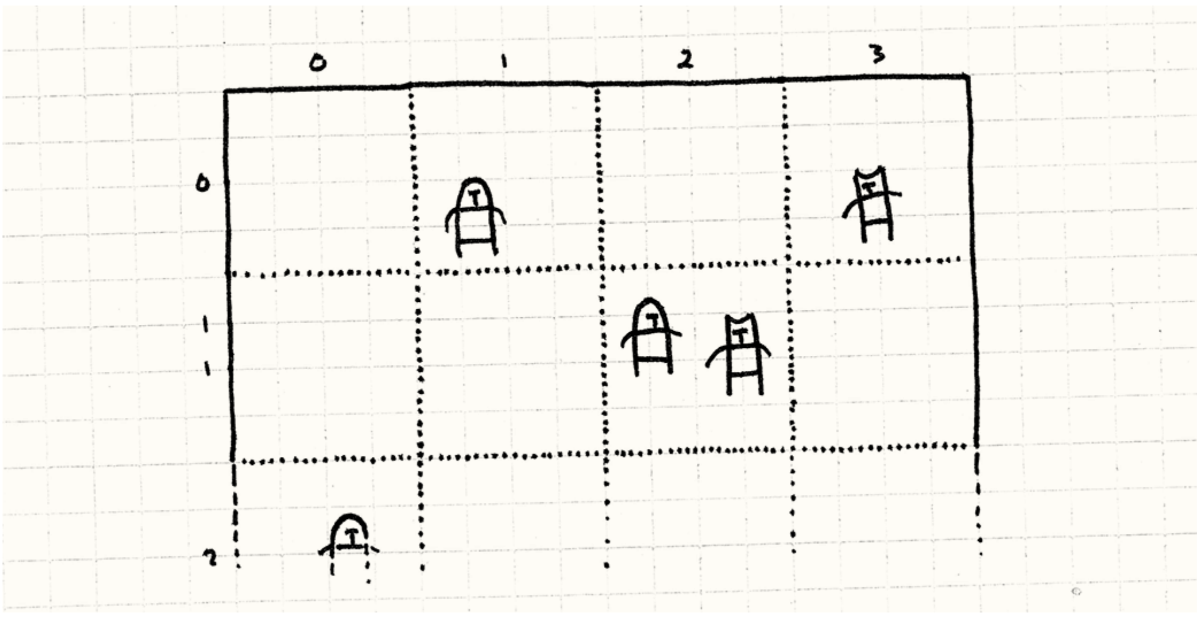


Figure 2 [3]

LINEAR FILTER

A technique used in image processing to blend the colour of a pixel with its neighbours. This can be done with a convolution operation. For example. Mean/average filters or Gaussian filter.[4]

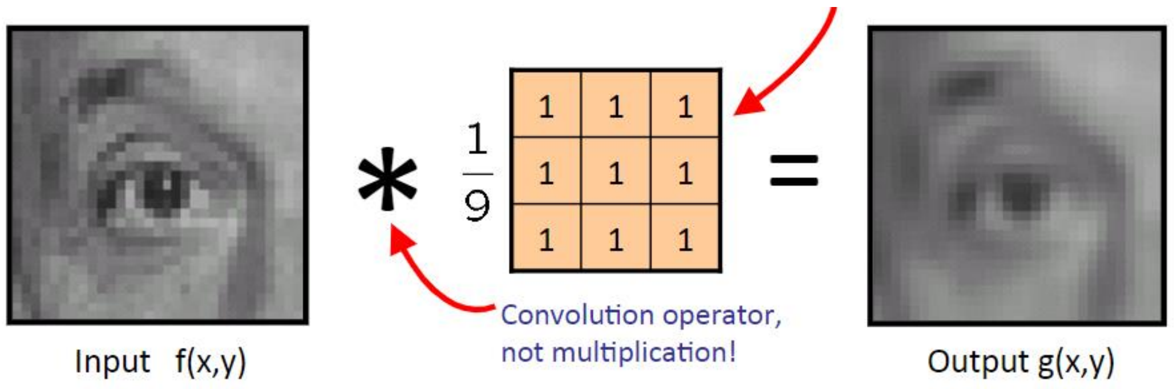


Figure 3 ([5])

By using this we can take an average colour from an area defined by the spatial partitioning end send this to the synthesiser. Reducing the amount of calls to the amount of areas.

MIDI

The range of MIDI notes is (0 – 127), which is more than the average amount of keys in a piano. The MIDI notes represent the following:

Table 1 (MIDI range with the applicable key and frequency)[6]

MIDI note number	Key number (Piano)	Note Names (English)	Frequency
127		G9	12543.85 Hz
…		…	…
108	88	C8	4186.01 Hz
…	…	…	…
21	1	A0	27.50 Hz
…			…
0			8.18 Hz

BASIC OPERATION

The intent for the NIME was to be able to control a synth with only using image data and the user the freedom to create an image that can be represented using this NIME. This can only happen with using the colour data a pixel has. This data contains the red, blue and green numeric value, giving the synthesiser three possible waves to change frequency with. But it requires some calculations to be able to know what value from (0 -255) would be required if the user want's a specific frequency.

When it comes to hardware, the user only needs a pc. But it would make it easier for a user, if they have a second screen. Currently the software only relies on the recording of the first screen, which makes it less practical when the program is always recording itself. Using Max/MSP or Pure Data that accepts the current established messages a user could create there way of interpreting the data.

This would mainly be used for the musical representation of a picture, although it is possible for using it on existing images and videos, for a more controlled sound, the artist would need to create an image where each pixel is deliberately placed by the artist with the vision of what the current setup would produce of sound. With digital drawing tools this makes it possible, like Adobe Photoshop. But even as a performance the NIME could be used, by drawing in the mentioned software or alternative ones, an artist could perform by just drawing randomly or similar shapes in order to produce sound live.

By only using the program, 3 main messages can be received.

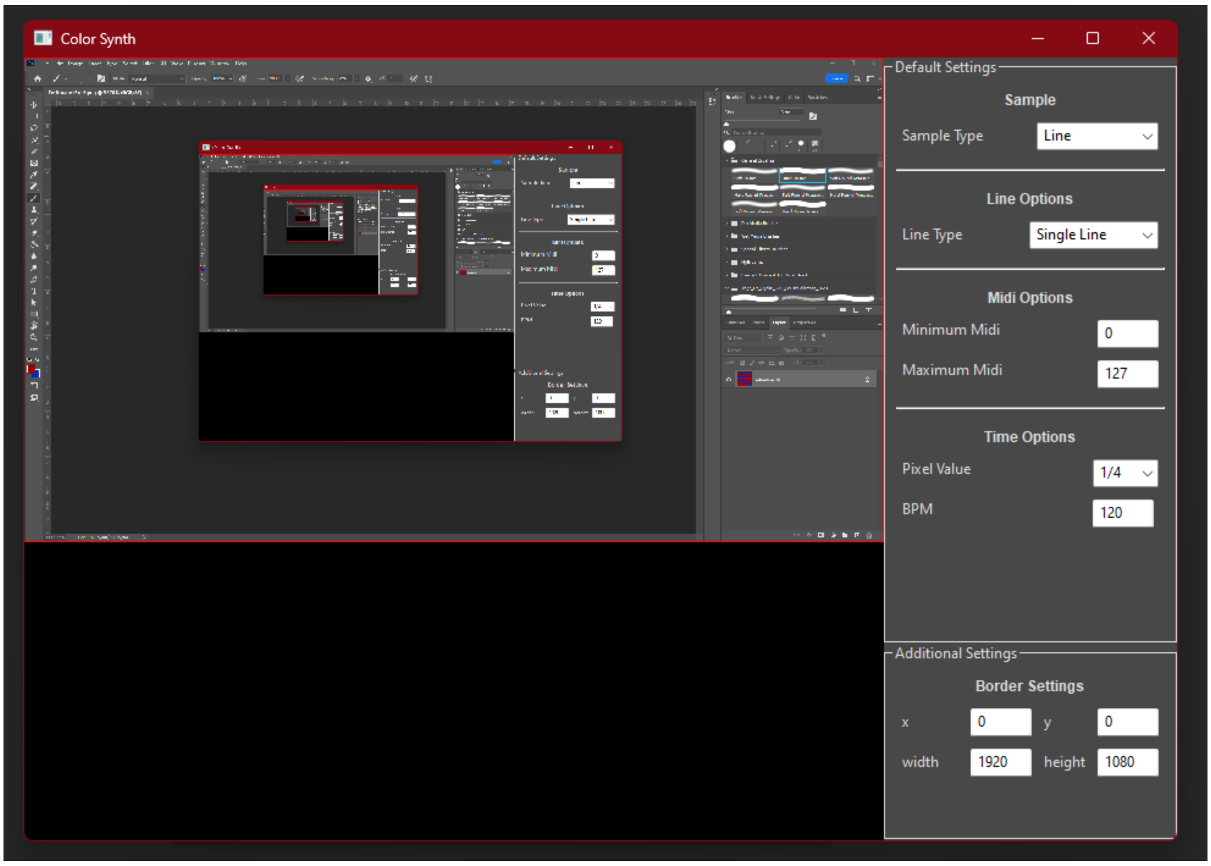
1. "/colorsynth" ("Line" option)
2. "/colorfiltersynth" ("Linear Filter" option)
3. "/synthamount"

The first 2 are the main data while the last one only is for updating potential dynamic synthesiser creation.

Table 2

"/colorsynth"	"/colorfiltersynth"
(INT) R	(INT) Index
(INT) G	(INT) R
(INT) B	(INT) G
(INT) MIDI Min	(INT) B
(INT) MIDI Max	(FLOAT) Pan position (-1 to 1)
	(INT) Total areas
	(INT) Priority (0-255)
	(INT) MIDI Min
	(INT) MIDI Max

EXPERIMENTAL SETUP



Color Synth

Default Settings

Sample

Sample Type: Line

Line Options

Line Type: Single Line

Midi Options

Minimum Midi: 0

Maximum Midi: 127

Time Options

Pixel Value: 1/4

BPM: 120

Additional Settings

Border Settings

x: 0 y: 0

width: 1920 height: 1080

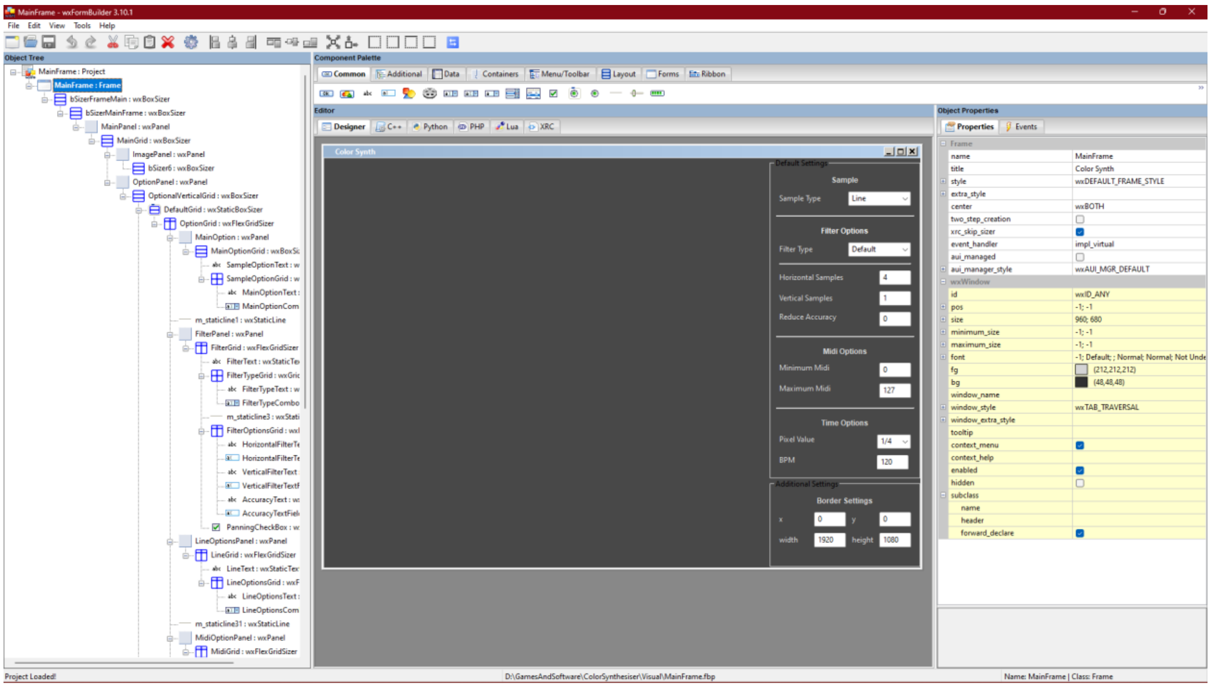
Figure 4

CHOICES BEFORE STARTING THE EXPERIMENT

In order to create the experiment a few choices had to be made before the actual programming could be done.

Before even deciding which programming language would be used, there was the question as to how the program would gets its UI. First option was to use an existing engine like Unity or Unreal Engine. But this would bind the project to an engine.

For creating the program, Python has been used. Python was the first language that a library was found that makes it possible for sending OSC messages. Although there is an option with C++ too, Python still was preferred for its easy use. And to solve the UI in Python a library called wxPython lets you create application windows this together with the software wxFormBuilder it makes it quite easy to create and design an applications UI with the desired interactions.



```
MainFrame : Project
  MainFrame : Frame
    bSizerFrameMain : wxBoxSizer
      bSizerMainFrame : wxBoxSizer
        MainPanel : wxPanel
          MainGrid : wxBoxSizer
            ImagePanel : wxPanel
              bSizer6 : wxBoxSizer
            OptionPanel : wxPanel
              OptionalVerticalGrid : wxBoxSizer
                DefaultGrid : wxStaticBoxSizer
                  OptionGrid : wxFlexGridSizer
                    MainOption : wxPanel
                      MainOptionGrid : wxBoxSi
                        SampleOptionText : w
                        SampleOptionGrid : w
                          MainOptionText :
                          MainOptionCom
                    m_staticline1 : wxStaticLine
                    FilterPanel : wxPanel
                      FilterGrid : wxFlexGridSizer
                        FilterText : wxStaticTe
                        FilterTypeGrid : wxGric
                          FilterTypeText : w
                          FilterTypeCombo
                        m_staticline3 : wxStati
                        FilterOptionsGrid : wxl
                          HorizontalFilterTe
                          HorizontalFilterTe
                          VerticalFilterText
                          VerticalFilterTextF
                          AccuracyText : w
                          AccuracyTextFiel
                        PanningCheckBox : w
                    LineOptionsPanel : wxPanel
                      LineGrid : wxFlexGridSizer
                        LineText : wxStaticTex
                        LineOptionsGrid : wxF
                          LineOptionsText :
                          LineOptionsCom
                    m_staticline31 : wxStaticLine
                    MidiOptionPanel : wxPanel
                      MidiGrid : wxFlexGridSizer
```

Default Settings

Sample

Sample Type	Line

Filter Options

Filter Type	Default
Horizontal Samples	4
Vertical Samples	1
Reduce Accuracy	0

Midi Options

Minimum Midi	0
Maximum Midi	127

Time Options

Pixel Value	1/4
BPM	120

Additional Settings

Border Settings

x	0	y	0
width	1920	height	1080

Frame	
name	MainFrame
title	Color Synth
style	wxDEFAULT_FRAME_STYLE
extra_style	
center	wxBOTH
two_step_creation	☐
xrc_skip_sizer	✓
event_handler	impl_virtual
aui_managed	☐
aui_manager_style	wxAUI_MGR_DEFAULT
wxWindow	
id	wxID_ANY
pos	-1; -1
size	960; 680
minimum_size	-1; -1
maximum_size	-1; -1
font	-1; Default; ; Normal; Normal; Not Unde
fg	(212,212,212)
bg	(48,48,48)
window_name	
window_style	wxTAB_TRAVERSAL
window_extra_style	
tooltip	
context_menu	✓
context_help	
enabled	✓
hidden	☐
subclass	
name	
header	
forward_declare	✓

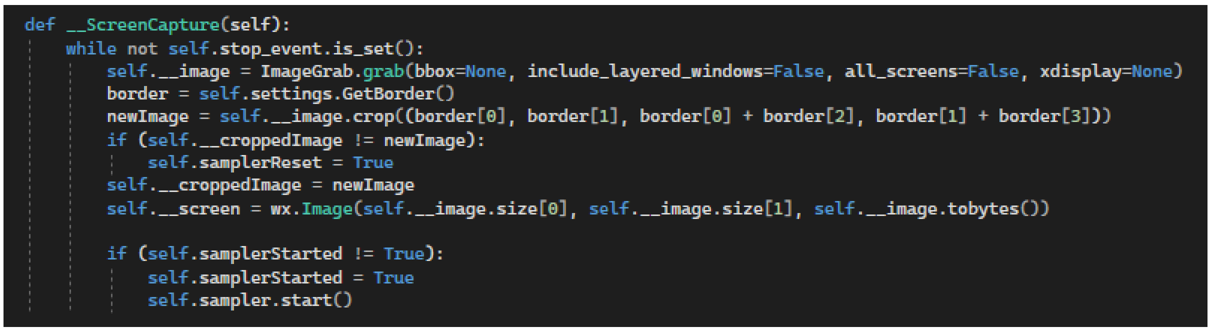
Project Loaded!   D:\GamesAndSoftware\ColorSynthesiser\Visual\MainFrame.fbp   Name: MainFrame | Class: Frame

Figure 5 wxFormBuilder designer

CORE COMPONENTS

SCREEN CAPTURE

To be able to get the information from the screen, a screen recording must be done. The full screen is stored for the UI to use and a cropped version of the screen is stored separately for the program to use before sending it to the network.

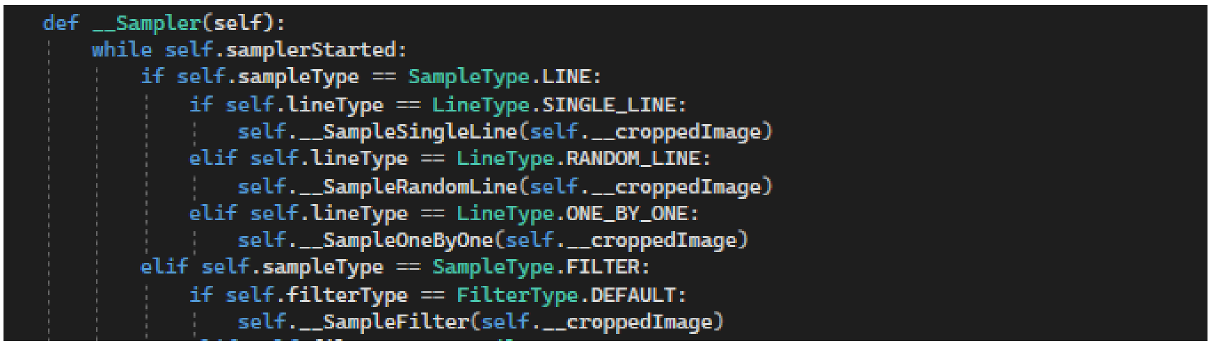


```python
def __ScreenCapture(self):
    while not self.stop_event.is_set():
        self.__image = ImageGrab.grab(bbox=None, include_layered_windows=False, all_screens=False, xdisplay=None)
        border = self.settings.GetBorder()
        newImage = self.__image.crop((border[0], border[1], border[0] + border[2], border[1] + border[3]))
        if (self.__croppedImage != newImage):
            self.samplerReset = True
        self.__croppedImage = newImage
        self.__screen = wx.Image(self.__image.size[0], self.__image.size[1], self.__image.tobytes())

        if (self.samplerStarted != True):
            self.samplerStarted = True
            self.sampler.start()
```

Figure 6

SAMPLER

The sampler executes the desired option we select. The current program has two sample types, “Line” and “Linear Filter”. With “Line” having a secondary option, “Single Line”, “Random Line” and “One By One”.



```
def __Sampler(self):
    while self.samplerStarted:
        if self.sampleType == SampleType.LINE:
            if self.lineType == LineType.SINGLE_LINE:
                self.__SampleSingleLine(self.__croppedImage)
            elif self.lineType == LineType.RANDOM_LINE:
                self.__SampleRandomLine(self.__croppedImage)
            elif self.lineType == LineType.ONE_BY_ONE:
                self.__SampleOneByOne(self.__croppedImage)
        elif self.sampleType == SampleType.FILTER:
            if self.filterType == FilterType.DEFAULT:
                self.__SampleFilter(self.__croppedImage)
```

Figure 7

Single Line option: only takes the first line of an image and sends pixel by pixel to the network.

Random Line option: Has the same functionality as “Single Line”, but with the addition that it randomly chooses a line every time it completes a line.

One By One option: This option goes through every pixel.

Linear Filter option: splits the screen into four identical parts. Each part calculates the average colour and send this to the network through a separate message for each part. On top of this an additional variable has been added, which is the panning. This will depend on the centre of the area where the data comes from.

MAX/MSP AND PURE DATA

As an experimental receiver of the data, Max/MSP was used, this was later changed to Pure Data. With Max/MSP it was the goal to make a synthesiser that could receive the messages and generate the sound through the RGB-values given. As experiment for the “Line” options only the grey values where originally used and for the “Linear Filter” a setup of 4 separate simple synthesisers where made with each colour having its own wave.

Both setups changed the values (0-255) to (0-127), in order to be able to be brought to frequencies by converting a MIDI-input to its frequency, the MIDI-input being the converted value.

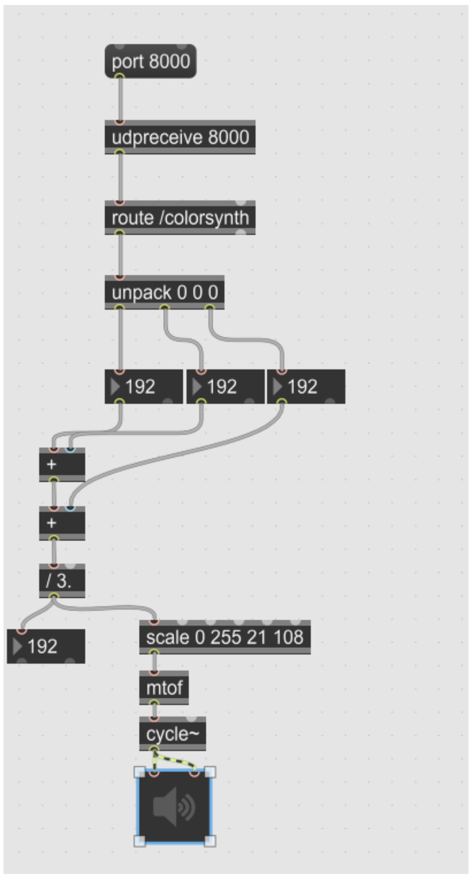


Figure 8 “Line” option synthesiser funtion

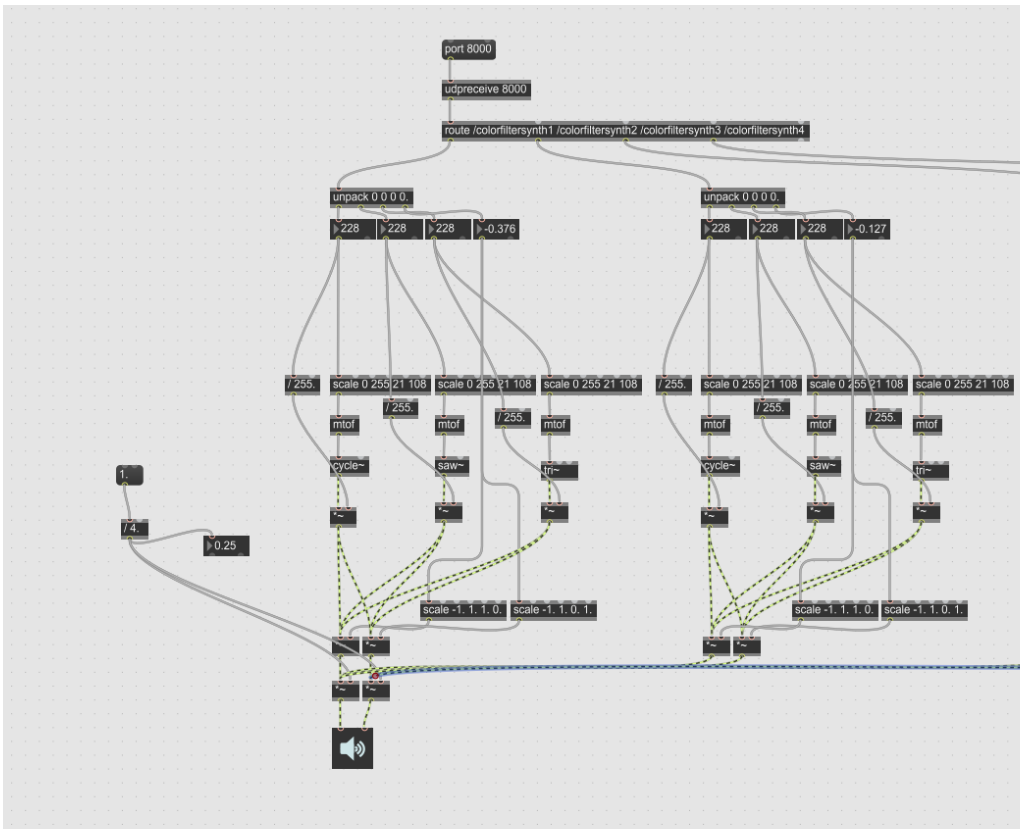
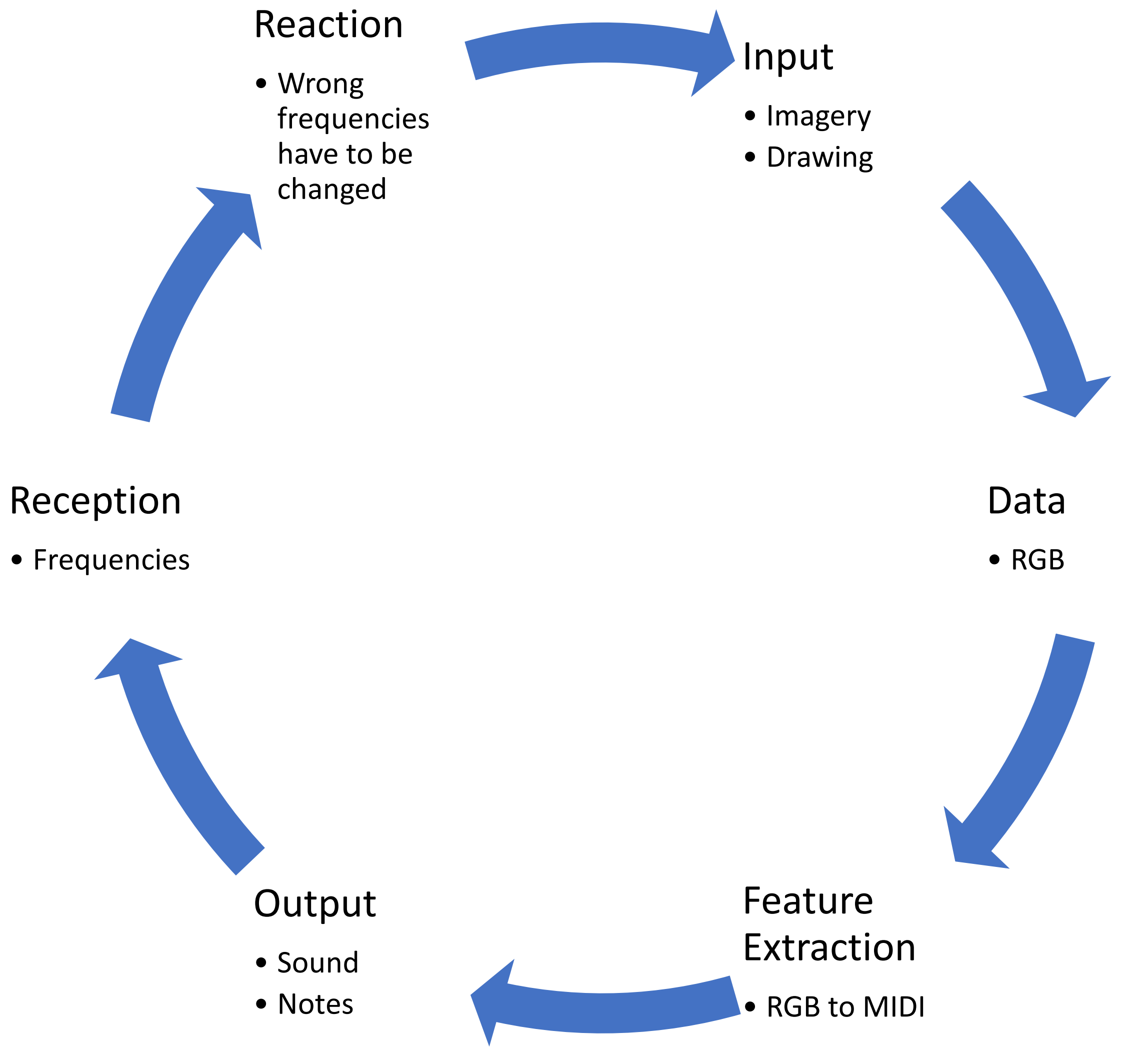


Figure 9 “Linear Filter” option synthesiser function

OUTPUT



Reaction
• Wrong frequencies have to be changed
Input
• Imagery
• Drawing
Data
• RGB
Feature Extraction
• RGB to MIDI
Output
• Sound
• Notes
Reception
• Frequencies

ITERATIONS

PERFORMANCE

Performance was a bottleneck that had to be dealt with in order for the application to run smoothly for larger images/screen images. Two solutions came to mind when developing the program. To give an idea, for the “Line” option, a single line took around 1-2 seconds and the “One By One”, took more than 30 minutes. The “Linear Filter” took around 20 seconds, with each segment only being started after each other, one segment took around 5 seconds.

SKIPPING EQUAL COLOURS

This iteration happened to the “Line” options. Instead of sending every pixel, only a new colour would be sent. This optimization reduced the time needed to around 15-30 seconds for sequence.

SKIPPING PIXELS

This iteration only applies to the “Linear Filter” option as this is the option that takes the whole picture into account. By setting a number of pixels to skip, we can reduce the amount of times the program has to go through a for-loop. It is less accurate than the complete average, but on the audible side, it can be a kind of a pitch shifter.

CHANGING PIXEL ITERATION

The most important change that happened to the “Linear Filter” option is the way it loops over all the pixels in an area. In the original setup, it only used for-loops, but these are computational heavy. Even with spatial partitioning, a desired 24 Frames Per Second (FPS) was not even closely reached.

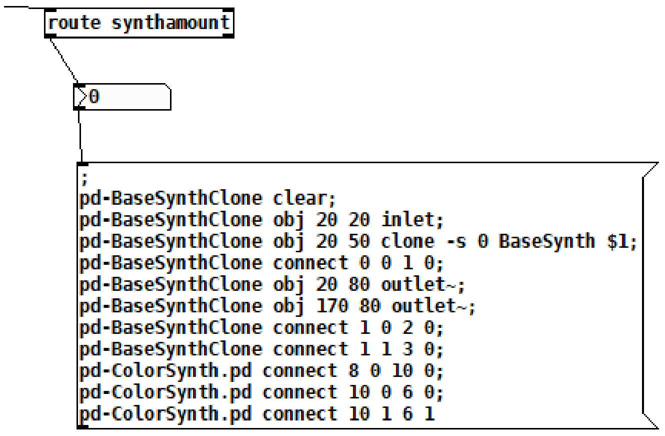
In order to replace the for-loops, the Numpy library was used for its iteration on its lists. This made it possible to reach the 24FPS, but the performance still lowers when the screen is more divided.

DYNAMIC SPATIAL PARTITIONING

With this iteration has some changes that were purely made with the Pure Data synthesiser in mind. Before, there were a hard coded spatial partitioning of 4 parts. Together with optimization and with having more points of interest on the screen, it was needed to have this iteration. With this iteration the user can dynamically change the amount of horizontal and vertical divided partitions.

For the Pure Data synthesiser, additional data had to be sent in order to dynamically create synthesisers. This happens by creating a node in Pure Data with some a parameter for the amount you want to create.

This happens by first clearing the Pure Data file and then reconstruct this with commands. The file contains a "Clone" node, since this can't be dynamically changed, it has to be recreated.



```
route synthamount
0
;
pd-BaseSynthClone clear;
pd-BaseSynthClone obj 20 20 inlet;
pd-BaseSynthClone obj 20 50 clone -s 0 BaseSynth $1;
pd-BaseSynthClone connect 0 0 1 0;
pd-BaseSynthClone obj 20 80 outlet~;
pd-BaseSynthClone obj 170 80 outlet~;
pd-BaseSynthClone connect 1 0 2 0;
pd-BaseSynthClone connect 1 1 3 0;
pd-ColorSynth.pd connect 8 0 10 0;
pd-ColorSynth.pd connect 10 0 6 0;
pd-ColorSynth.pd connect 10 1 6 1
```

Figure 10

SPATIAL PARTITIONING WITH FOCUS

The problem with having all parts equally loud is that there will be no focus on major changes in a scene. For this an iteration was created called "Linear Filter Layered", which looks at the image and decides the priority level of an area with how brighter the colour is from the previous averaged colour. In the Pure Data synthesiser, this will be used as a low-pass filter, being the lower the value the more, high frequencies, get reduced.

MIDI-RANGE OPTION

In order to give the user a more reasonable range of frequencies to work with, a MIDI-range has been added. Pure white wouldn't have to be 12k frequency all the time and will give control to the user what they want to use.

TIME OPTION

This was the last iteration and the one that has the most possibilities for future use. With this iteration each pixel is represented as a note value, similar to note values in scores. Adding the option for changing the BPM can give more options to a more musical directed approach. This iteration is only applicable for the “Line” option.

For example, a pixel is the length of a quarter note, the next pixel would be sent after 0.5 seconds with 120 BMP.

$$\frac{60s}{BPM} * (4 \ * note\ length) = note\ value$$

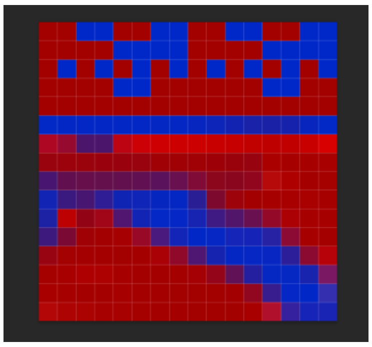


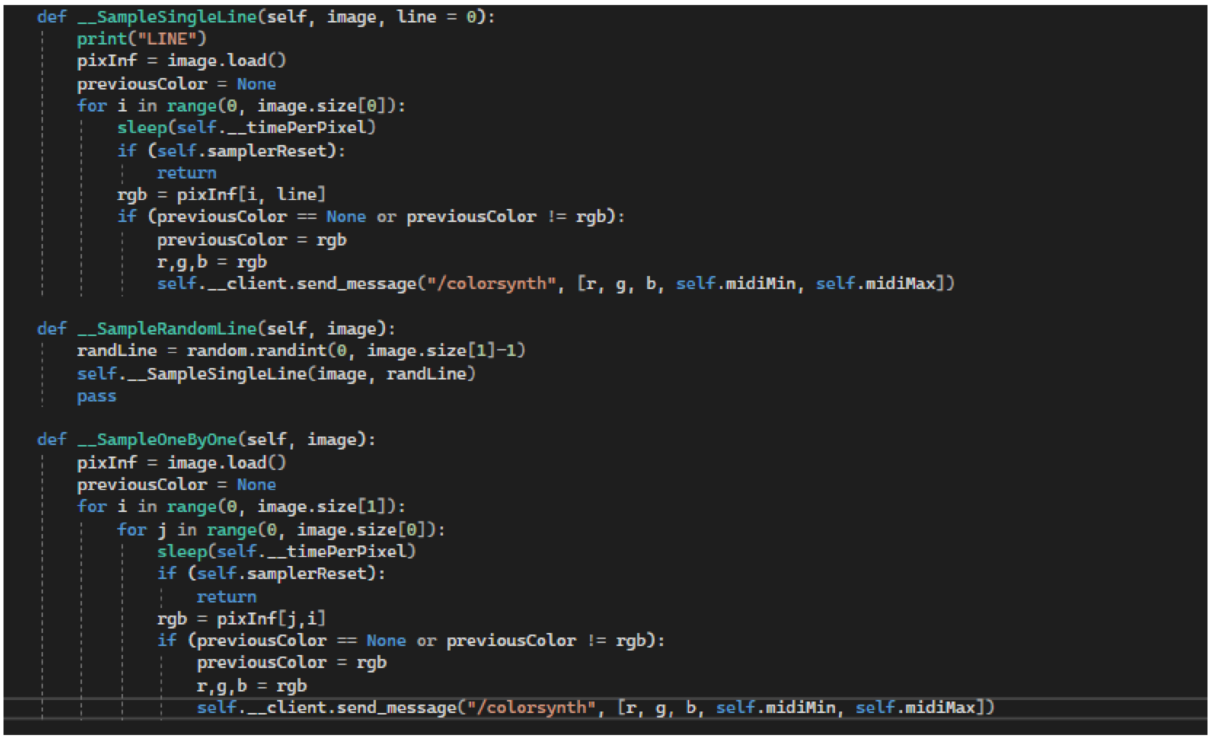
Figure 11

This was tested using a 16x16 image that contains note length switches and some random drawn parts. On a 16x16 image a line would take 4 seconds to complete with the above setup. This has given the “Line” option a more controlled way of playing sound. The first part of the image has a clear change that corelates to what we see. The second part that was drawn with a brush to make some blended changes creates a controlled change in frequency. This iteration made the synthesiser more as an instrument than something that just produces sound the moment it receives an input that is heavily dependent on the devices computational power.

Below are results of the time it would take for an image that is the size of 1920x1080 to be completed with 120 BPM.

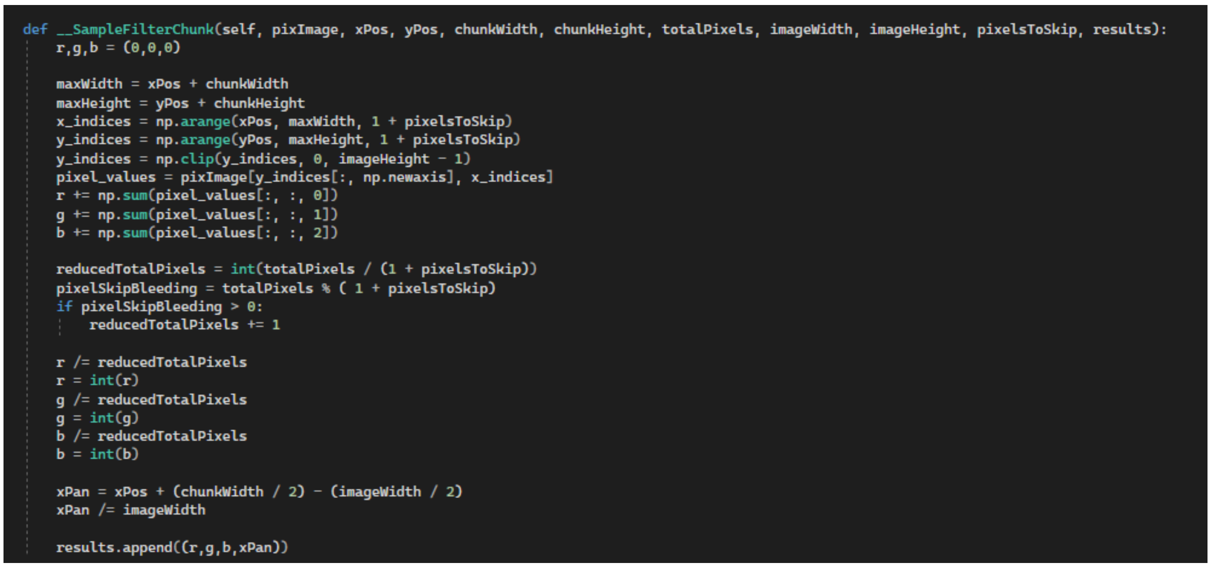
1	4,147,200 seconds	69,120 minutes	1,152 hours	48 days
$\frac{1}{2}$	2,073,600 seconds	34,560 minutes	576 hours	24 days
$\frac{1}{4}$	1,036,800 seconds	17,280 minutes	288 hours	12 days
$\frac{1}{8}$	518,400 seconds	8,640 minutes	144 hours	6 days
$\frac{1}{16}$	259,200 seconds	4,320 minutes	72 hours	3 days
$\frac{1}{32}$	129,600 seconds	2,160 minutes	36 hours	1.5 days
$\frac{1}{64}$	64,800 seconds	1,080 minutes	18 hours	0.75 days

VISUAL STUDIO



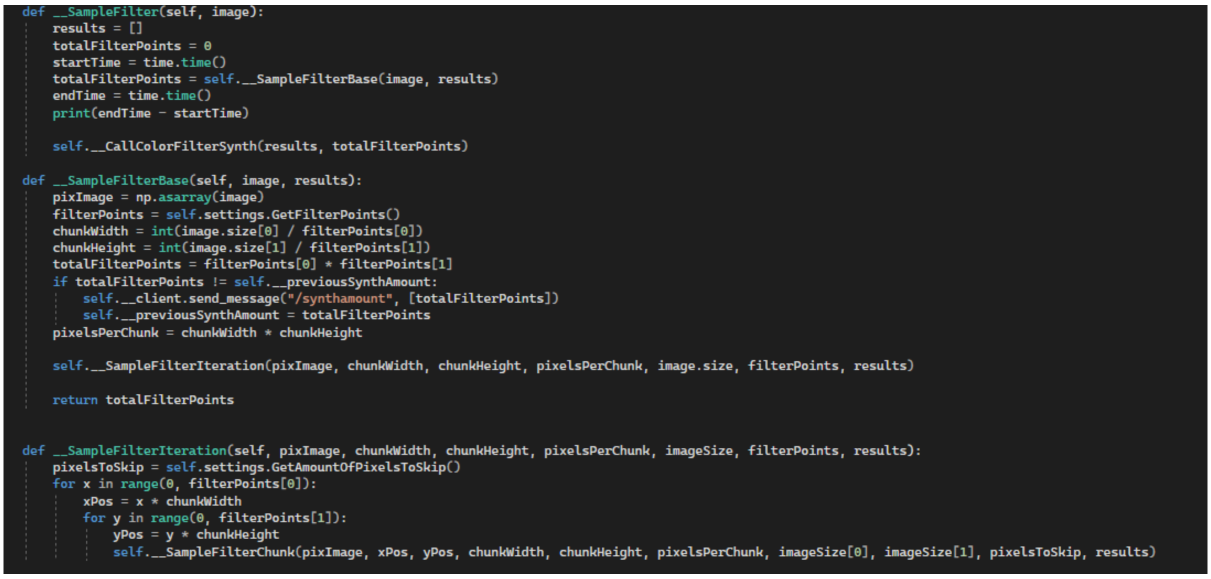
```python
def __SampleSingleLine(self, image, line = 0):
    print("LINE")
    pixInf = image.load()
    previousColor = None
    for i in range(0, image.size[0]):
        sleep(self.__timePerPixel)
        if (self.samplerReset):
            return
        rgb = pixInf[i, line]
        if (previousColor == None or previousColor != rgb):
            previousColor = rgb
            r,g,b = rgb
            self.__client.send_message("/colorsynth", [r, g, b, self.midiMin, self.midiMax])

def __SampleRandomLine(self, image):
    randLine = random.randint(0, image.size[1]-1)
    self.__SampleSingleLine(image, randLine)
    pass

def __SampleOneByOne(self, image):
    pixInf = image.load()
    previousColor = None
    for i in range(0, image.size[1]):
        for j in range(0, image.size[0]):
            sleep(self.__timePerPixel)
            if (self.samplerReset):
                return
            rgb = pixInf[j,i]
            if (previousColor == None or previousColor != rgb):
                previousColor = rgb
                r,g,b = rgb
                self.__client.send_message("/colorsynth", [r, g, b, self.midiMin, self.midiMax])
```

Figure 12 Code behind the “Line” options



```python
def __SampleFilterChunk(self, pixImage, xPos, yPos, chunkWidth, chunkHeight, totalPixels, imageWidth, imageHeight, pixelsToSkip, results):
    r,g,b = (0,0,0)

    maxWidth = xPos + chunkWidth
    maxHeight = yPos + chunkHeight
    x_indices = np.arange(xPos, maxWidth, 1 + pixelsToSkip)
    y_indices = np.arange(yPos, maxHeight, 1 + pixelsToSkip)
    y_indices = np.clip(y_indices, 0, imageHeight - 1)
    pixel_values = pixImage[y_indices[:, np.newaxis], x_indices]
    r += np.sum(pixel_values[:, :, 0])
    g += np.sum(pixel_values[:, :, 1])
    b += np.sum(pixel_values[:, :, 2])

    reducedTotalPixels = int(totalPixels / (1 + pixelsToSkip))
    pixelSkipBleeding = totalPixels % ( 1 + pixelsToSkip)
    if pixelSkipBleeding > 0:
        reducedTotalPixels += 1

    r /= reducedTotalPixels
    r = int(r)
    g /= reducedTotalPixels
    g = int(g)
    b /= reducedTotalPixels
    b = int(b)

    xPan = xPos + (chunkWidth / 2) - (imageWidth / 2)
    xPan /= imageWidth

    results.append((r,g,b,xPan))
```

Figure 13 “Linear filter” chunk



```
def __SampleFilter(self, image):
    results = []
    totalFilterPoints = 0
    startTime = time.time()
    totalFilterPoints = self.__SampleFilterBase(image, results)
    endTime = time.time()
    print(endTime - startTime)

    self.__CallColorFilterSynth(results, totalFilterPoints)

def __SampleFilterBase(self, image, results):
    pixImage = np.asarray(image)
    filterPoints = self.settings.GetFilterPoints()
    chunkWidth = int(image.size[0] / filterPoints[0])
    chunkHeight = int(image.size[1] / filterPoints[1])
    totalFilterPoints = filterPoints[0] * filterPoints[1]
    if totalFilterPoints != self.__previousSynthAmount:
        self.__client.send_message("/synthamount", [totalFilterPoints])
        self.__previousSynthAmount = totalFilterPoints
    pixelsPerChunk = chunkWidth * chunkHeight

    self.__SampleFilterIteration(pixImage, chunkWidth, chunkHeight, pixelsPerChunk, image.size, filterPoints, results)

    return totalFilterPoints


def __SampleFilterIteration(self, pixImage, chunkWidth, chunkHeight, pixelsPerChunk, imageSize, filterPoints, results):
    pixelsToSkip = self.settings.GetAmountOfPixelsToSkip()
    for x in range(0, filterPoints[0]):
        xPos = x * chunkWidth
        for y in range(0, filterPoints[1]):
            yPos = y * chunkHeight
            self.__SampleFilterChunk(pixImage, xPos, yPos, chunkWidth, chunkHeight, pixelsPerChunk, imageSize[0], imageSize[1], pixelsToSkip, results)
```

Figure 14 "Linear Filter" base code

PURE DATA

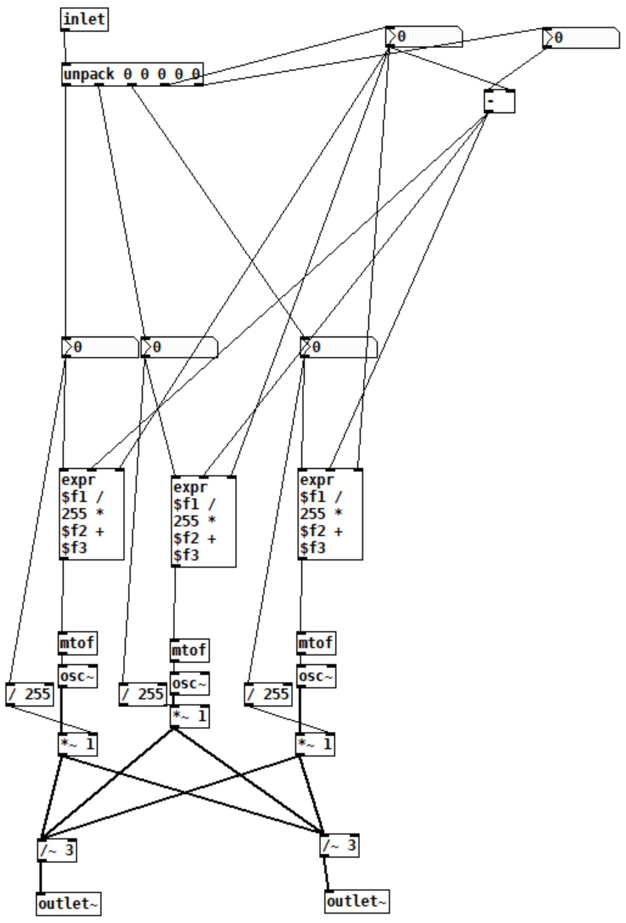


Figure 15 Pure Data "Line" option setup

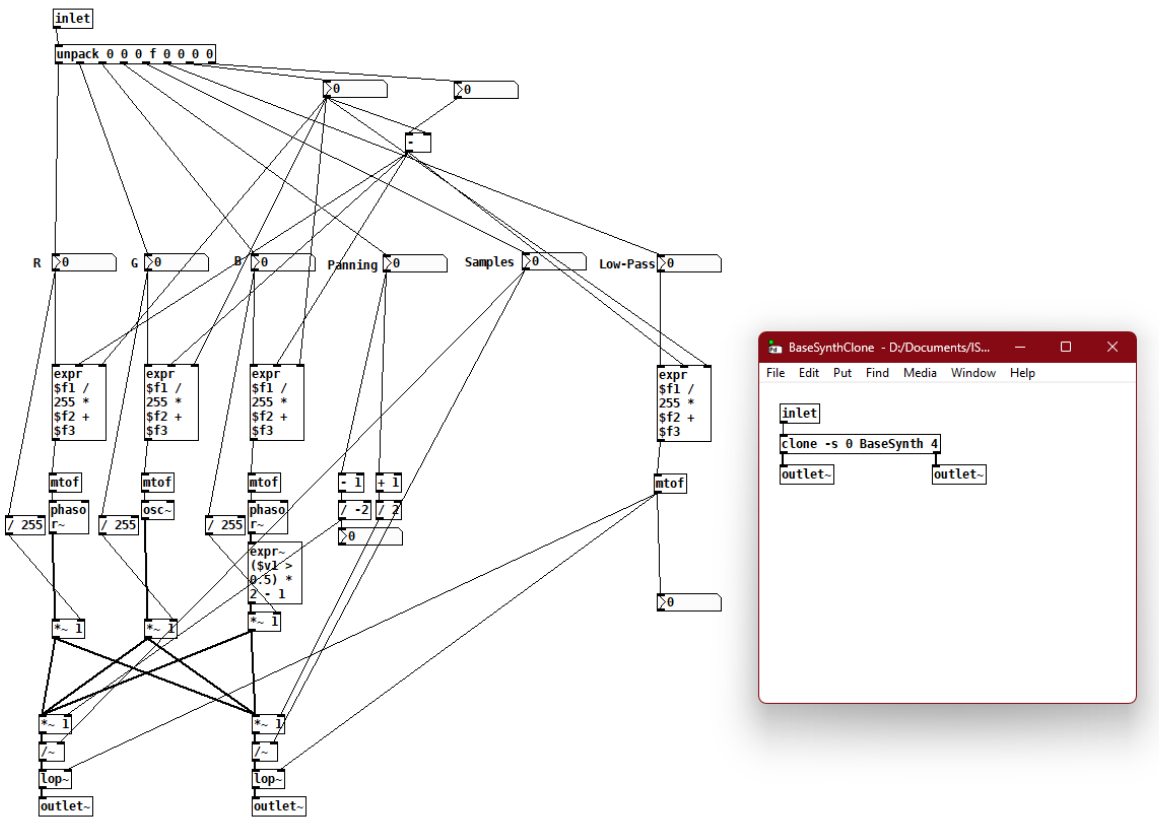
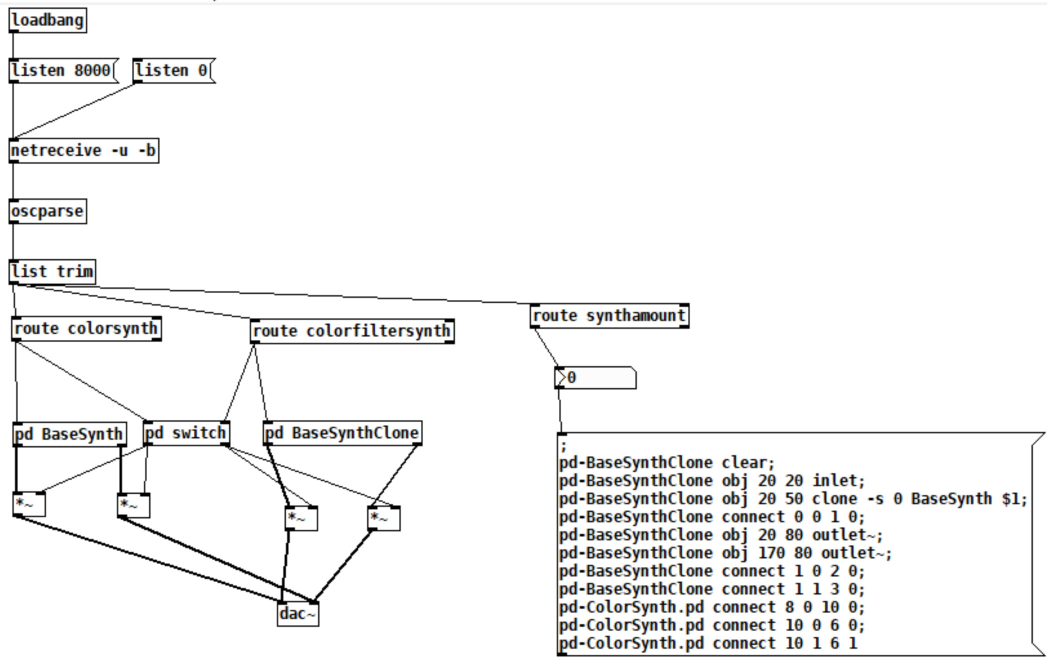


Figure 16 Pure Data “Linear Filter” option setup



```
;
pd-BaseSynthClone clear;
pd-BaseSynthClone obj 20 20 inlet;
pd-BaseSynthClone obj 20 50 clone -s 0 BaseSynth $1;
pd-BaseSynthClone connect 0 0 1 0;
pd-BaseSynthClone obj 20 80 outlet~;
pd-BaseSynthClone obj 170 80 outlet~;
pd-BaseSynthClone connect 1 0 2 0;
pd-BaseSynthClone connect 1 1 3 0;
pd-ColorSynth.pd connect 8 0 10 0;
pd-ColorSynth.pd connect 10 0 6 0;
pd-ColorSynth.pd connect 10 1 6 1
```

Figure 17 Pure Data main setup

CONLUSION

The NIME produced some good results, although it is quite clear that the way the data can be interpreted will heavily depend on the user.

The “Linear Filter” is an interesting option, but it currently does not have any practical use. It maybe could be used for designing sound that works with a company’s logo or logo animation. In which the low-pass filter parameter that was added could give some more interesting results as it currently doesn’t work as intended. Every change is being detected, but this is not dependent on an object, making it not viable for object location also frames with less change would give you the full range again.

With the last iteration on the “Line” option, the NIME created the best results and is closest to the envisioned result. But the current application can’t efficiently reproduce the setup without prior knowledge. For the image to be played correctly, the user needs to exactly know where on the screen the image.

EVALUATION & FUTUREWORK

As an experimental program this project worked as intended, but for creating synthesisers or applications that use the data produced by the application, it is not viable.

For the setup the synth had to convert the values to MIDI. This could have been done by the application, removing the two MIDI parameters. Internally this would make it possible to give the user a list of notes it can use and give the user the corresponding colour for that output. With only three MIDI outputs, three separate synths or other instruments could be controlled. This together with an application that only uses imported images or let the user draw them in the program themselves, removes the need of a boundary and the need to know where the image is located on the screen. This would also introduce the possibility of using the alpha value as a fourth MIDI output.

BIBLIOGRAPHY

[1] ‘User Datagram Protocol’, Internet Engineering Task Force, Request for Comments RFC 768, Aug. 1980. doi: 10.17487/RFC0768.

[2] M. Wright and A. Freed, ‘Open SoundControl: A New Protocol for Communicating with Sound Synthesizers’.

[3] ‘Spatial Partition · Optimization Patterns · Game Programming Patterns’. https://gameprogrammingpatterns.com/spatial-partition.html (accessed Jun. 10, 2023).

[4] R. Lini, ‘Different Filters for Image processing’, *Medium*, Oct. 17, 2021. https://medium.com/@rajilini/different-filters-for-image-processing-698e72924101 (accessed Jun. 10, 2023).

[5] M. Boels, ‘Introduction to Image Processing: Filters’, *Medium*, Oct. 26, 2019. https://medium.com/@boelsmaxence/introduction-to-image-processing-filters-179607f9824a (accessed Jun. 10, 2023).

[6] ‘MIDI note numbers and center frequencies | Inspired Acoustics’. https://www.inspiredacoustics.com/en/MIDI_note_numbers_and_center_frequencies (accessed Jun. 10, 2023).