1.1 PluginEditor.cpp

/\*

  ==============================================================================

  This is an automatically generated GUI class created by the Projucer!

  Be careful when adding custom code to these files, as only the code within

  the "//[xyz]" and "//[/xyz]" sections will be retained when the file is loaded

  and re-saved.

  Created with Projucer version: 6.0.8

  ------------------------------------------------------------------------------

  The Projucer is part of the JUCE library.

  Copyright (c) 2020 - Raw Material Software Limited.

  ==============================================================================

\*/

//[Headers] You can add your own extra header files here...

#include "PluginProcessor.h"

//[/Headers]

#include "PluginEditor.h"

//[MiscUserDefs] You can add your own user definitions and misc code here...

//[/MiscUserDefs]

//==============================================================================

PluginEditor::PluginEditor (PluginProcessor& processor)

    : juce::AudioProcessorEditor(processor), audioProcessor(processor)

{

    //[Constructor\_pre] You can add your own custom stuff here..

    //[/Constructor\_pre]

    spectrum\_canvas.reset (new SpectrumCanvas (processor.params, processor.rendering));

    addAndMakeVisible (spectrum\_canvas.get());

    spectrum\_canvas->setName ("spectrum\_canvas");

    spectrum\_canvas->setBounds (144, 32, 272, 192);

    leftCutoffSlider.reset (new juce::Slider ("leftCutoffSlider"));

    addAndMakeVisible (leftCutoffSlider.get());

    leftCutoffSlider->setRange (1, 15000, 0);

    leftCutoffSlider->setSliderStyle (juce::Slider::RotaryVerticalDrag);

    leftCutoffSlider->setTextBoxStyle (juce::Slider::NoTextBox, false, 80, 20);

    leftCutoffSlider->setBounds (216, 232, 128, 88);

    rightCutoffSlider.reset (new juce::Slider ("rightCutoffSlider"));

    addAndMakeVisible (rightCutoffSlider.get());

    rightCutoffSlider->setRange (1, 15000, 0);

    rightCutoffSlider->setSliderStyle (juce::Slider::RotaryVerticalDrag);

    rightCutoffSlider->setTextBoxStyle (juce::Slider::NoTextBox, false, 80, 20);

    rightCutoffSlider->setBounds (320, 232, 128, 88);

    strengthSlider.reset (new juce::Slider ("strengthSlider"));

    addAndMakeVisible (strengthSlider.get());

    strengthSlider->setTooltip (TRANS("Strength of stereo effect"));

    strengthSlider->setRange (0, 2, 0);

    strengthSlider->setSliderStyle (juce::Slider::RotaryVerticalDrag);

    strengthSlider->setTextBoxStyle (juce::Slider::NoTextBox, false, 80, 20);

    strengthSlider->setBounds (0, 8, 128, 88);

    frequencySpreadSlider.reset (new juce::Slider ("frequencySpreadSlider"));

    addAndMakeVisible (frequencySpreadSlider.get());

    frequencySpreadSlider->setRange (0.1, 0.999, 0);

    frequencySpreadSlider->setSliderStyle (juce::Slider::RotaryVerticalDrag);

    frequencySpreadSlider->setTextBoxStyle (juce::Slider::NoTextBox, false, 80, 20);

    frequencySpreadSlider->setBounds (0, 104, 128, 88);

    dynamicSplitEnabledButton.reset (new juce::ToggleButton ("dynamicSplitEnabledButton"));

    addAndMakeVisible (dynamicSplitEnabledButton.get());

    dynamicSplitEnabledButton->setButtonText (TRANS("Dynamic split"));

    dynamicSplitEnabledButton->setToggleState (true, juce::dontSendNotification);

    dynamicSplitEnabledButton->setBounds (8, 216, 112, 24);

    attackSlider.reset (new juce::Slider ("attackSlider"));

    addAndMakeVisible (attackSlider.get());

    attackSlider->setTooltip (TRANS("few ew few few few fwe fewf wef ewf ew"));

    attackSlider->setRange (0, 1, 0);

    attackSlider->setSliderStyle (juce::Slider::RotaryVerticalDrag);

    attackSlider->setTextBoxStyle (juce::Slider::NoTextBox, false, 80, 20);

    attackSlider->setBounds (0, 232, 128, 88);

    juce\_\_label.reset (new juce::Label ("Left cutoff",

                                        TRANS("Left cutoff")));

    addAndMakeVisible (juce\_\_label.get());

    juce\_\_label->setFont (juce::Font (15.00f, juce::Font::plain).withTypefaceStyle ("Regular"));

    juce\_\_label->setJustificationType (juce::Justification::centred);

    juce\_\_label->setEditable (false, false, false);

    juce\_\_label->setColour (juce::TextEditor::textColourId, juce::Colours::black);

    juce\_\_label->setColour (juce::TextEditor::backgroundColourId, juce::Colour (0x00000000));

    juce\_\_label->setBounds (208, 312, 144, 24);

    juce\_\_label2.reset (new juce::Label ("Right cutoff",

                                         TRANS("Right cutoff")));

    addAndMakeVisible (juce\_\_label2.get());

    juce\_\_label2->setFont (juce::Font (15.00f, juce::Font::plain).withTypefaceStyle ("Regular"));

    juce\_\_label2->setJustificationType (juce::Justification::centred);

    juce\_\_label2->setEditable (false, false, false);

    juce\_\_label2->setColour (juce::TextEditor::textColourId, juce::Colours::black);

    juce\_\_label2->setColour (juce::TextEditor::backgroundColourId, juce::Colour (0x00000000));

    juce\_\_label2->setBounds (312, 312, 150, 24);

    juce\_\_label3.reset (new juce::Label ("Strength",

                                         TRANS("Strength")));

    addAndMakeVisible (juce\_\_label3.get());

    juce\_\_label3->setFont (juce::Font (15.00f, juce::Font::plain).withTypefaceStyle ("Regular"));

    juce\_\_label3->setJustificationType (juce::Justification::centred);

    juce\_\_label3->setEditable (false, false, false);

    juce\_\_label3->setColour (juce::TextEditor::textColourId, juce::Colours::black);

    juce\_\_label3->setColour (juce::TextEditor::backgroundColourId, juce::Colour (0x00000000));

    juce\_\_label3->setBounds (0, 80, 128, 32);

    juce\_\_label4.reset (new juce::Label ("Frequency spread",

                                         TRANS("Frequency spread")));

    addAndMakeVisible (juce\_\_label4.get());

    juce\_\_label4->setFont (juce::Font (15.00f, juce::Font::plain).withTypefaceStyle ("Regular"));

    juce\_\_label4->setJustificationType (juce::Justification::centred);

    juce\_\_label4->setEditable (false, false, false);

    juce\_\_label4->setColour (juce::TextEditor::textColourId, juce::Colours::black);

    juce\_\_label4->setColour (juce::TextEditor::backgroundColourId, juce::Colour (0x00000000));

    juce\_\_label4->setBounds (-8, 184, 152, 24);

    juce\_\_label5.reset (new juce::Label ("Attack",

                                         TRANS("Attack")));

    addAndMakeVisible (juce\_\_label5.get());

    juce\_\_label5->setFont (juce::Font (15.00f, juce::Font::plain).withTypefaceStyle ("Regular"));

    juce\_\_label5->setJustificationType (juce::Justification::centred);

    juce\_\_label5->setEditable (false, false, false);

    juce\_\_label5->setColour (juce::TextEditor::textColourId, juce::Colours::black);

    juce\_\_label5->setColour (juce::TextEditor::backgroundColourId, juce::Colour (0x00000000));

    juce\_\_label5->setBounds (16, 312, 96, 24);

    noiseSlider.reset (new juce::Slider ("noiseSlider"));

    addAndMakeVisible (noiseSlider.get());

    noiseSlider->setTooltip (TRANS("few ew few few few fwe fewf wef ewf ew"));

    noiseSlider->setRange (0, 1, 0);

    noiseSlider->setSliderStyle (juce::Slider::RotaryVerticalDrag);

    noiseSlider->setTextBoxStyle (juce::Slider::NoTextBox, false, 80, 20);

    noiseSlider->setBounds (112, 232, 128, 88);

    juce\_\_label6.reset (new juce::Label ("Noise",

                                         TRANS("Noise")));

    addAndMakeVisible (juce\_\_label6.get());

    juce\_\_label6->setFont (juce::Font (15.00f, juce::Font::plain).withTypefaceStyle ("Regular"));

    juce\_\_label6->setJustificationType (juce::Justification::centred);

    juce\_\_label6->setEditable (false, false, false);

    juce\_\_label6->setColour (juce::TextEditor::textColourId, juce::Colours::black);

    juce\_\_label6->setColour (juce::TextEditor::backgroundColourId, juce::Colour (0x00000000));

    juce\_\_label6->setBounds (128, 312, 96, 24);

    //[UserPreSize]

    //[/UserPreSize]

    setSize (444, 348);

    //[Constructor] You can add your own custom stuff here..

    tooltipWindow.reset(new TooltipWindow(this, 700));

  /\*  setResizable(true, true);

    setResizeLimits(100, 100, 1000, 1000);

    \*/

    setResizable(false, false);

    leftCutoffAttachment = std::make\_unique<SliderAttachment>(\*audioProcessor.params.tree, "leftCutoff", \*leftCutoffSlider);

    rightCutoffAttachment = std::make\_unique<SliderAttachment>(\*audioProcessor.params.tree, "rightCutoff", \*rightCutoffSlider);

    strengthAttachment = std::make\_unique<SliderAttachment>(\*audioProcessor.params.tree, "strength", \*strengthSlider);

    frequencySpreadAttachment = std::make\_unique<SliderAttachment>(\*audioProcessor.params.tree, "frequencySpread", \*frequencySpreadSlider);

    attackAttachment = std::make\_unique<SliderAttachment>(\*audioProcessor.params.tree, "attack", \*attackSlider);

    noiseAttachment = std::make\_unique<SliderAttachment>(\*audioProcessor.params.tree, "noise", \*noiseSlider);

    dynamicSplitEnabledAttachment = std::make\_unique<ButtonAttachment>(\*audioProcessor.params.tree, "dynamicSplitEnabled", \*dynamicSplitEnabledButton);

    leftCutoffSlider->setSkewFactorFromMidPoint(1024);

    rightCutoffSlider->setSkewFactorFromMidPoint(1024);

    audioProcessor.params.tree->addParameterListener("dynamicSplitEnabled", this);

    //[/Constructor]

}

PluginEditor::~PluginEditor()

{

    //[Destructor\_pre]. You can add your own custom destruction code here..

    tooltipWindow = nullptr;

    leftCutoffAttachment = nullptr;

    rightCutoffAttachment = nullptr;

    strengthAttachment = nullptr;

    frequencySpreadAttachment = nullptr;

    attackAttachment = nullptr;

    noiseAttachment = nullptr;

    dynamicSplitEnabledAttachment = nullptr;

    audioProcessor.params.tree->removeParameterListener("dynamicSplitEnabled", this);

    //[/Destructor\_pre]

    spectrum\_canvas = nullptr;

    leftCutoffSlider = nullptr;

    rightCutoffSlider = nullptr;

    strengthSlider = nullptr;

    frequencySpreadSlider = nullptr;

    dynamicSplitEnabledButton = nullptr;

    attackSlider = nullptr;

    juce\_\_label = nullptr;

    juce\_\_label2 = nullptr;

    juce\_\_label3 = nullptr;

    juce\_\_label4 = nullptr;

    juce\_\_label5 = nullptr;

    noiseSlider = nullptr;

    juce\_\_label6 = nullptr;

    //[Destructor]. You can add your own custom destruction code here..

    //[/Destructor]

}

//==============================================================================

void PluginEditor::paint (juce::Graphics& g)

{

    //[UserPrePaint] Add your own custom painting code here..

    //[/UserPrePaint]

    g.fillAll (juce::Colour (0xff323e44));

    {

        int x = 0, y = -1, width = 452, height = 348;

        juce::Colour fillColour = juce::Colour (0x21ffffff);

        //[UserPaintCustomArguments] Customize the painting arguments here..

        //[/UserPaintCustomArguments]

        g.setColour (fillColour);

        g.fillRect (x, y, width, height);

    }

    //[UserPaint] Add your own custom painting code here..

    //[/UserPaint]

}

void PluginEditor::resized()

{

    //[UserPreResize] Add your own custom resize code here..

    //[/UserPreResize]

    //[UserResized] Add your own custom resize handling here..

    //[/UserResized]

}

//[MiscUserCode] You can add your own definitions of your custom methods or any other code here...

void PluginEditor::parameterChanged (const String& parameterID, float newValue) {

    if (parameterID == "dynamicSplitEnabled") {

        attackSlider->setEnabled(newValue > 0.5f);

    }

}

//[/MiscUserCode]

//==============================================================================

#if 0

/\*  -- Projucer information section --

    This is where the Projucer stores the metadata that describe this GUI layout, so

    make changes in here at your peril!

BEGIN\_JUCER\_METADATA

<JUCER\_COMPONENT documentType="Component" className="PluginEditor" componentName=""

                 parentClasses="public juce::AudioProcessorEditor, public juce::AudioProcessorValueTreeState::Listener"

                 constructorParams="PluginProcessor&amp; processor" variableInitialisers="juce::AudioProcessorEditor(processor), audioProcessor(processor)"

                 snapPixels="8" snapActive="1" snapShown="1" overlayOpacity="0.330"

                 fixedSize="1" initialWidth="444" initialHeight="348">

  <BACKGROUND backgroundColour="ff323e44">

    <RECT pos="0 -1 452 348" fill="solid: 21ffffff" hasStroke="0"/>

  </BACKGROUND>

  <GENERICCOMPONENT name="spectrum\_canvas" id="c28ba59d3075a746" memberName="spectrum\_canvas"

                    virtualName="" explicitFocusOrder="0" pos="144 32 272 192" class="SpectrumCanvas"

                    params="processor.params, processor.rendering"/>

  <SLIDER name="leftCutoffSlider" id="110eb12e1151856" memberName="leftCutoffSlider"

          virtualName="" explicitFocusOrder="0" pos="216 232 128 88" min="1.0"

          max="15000.0" int="0.0" style="RotaryVerticalDrag" textBoxPos="NoTextBox"

          textBoxEditable="1" textBoxWidth="80" textBoxHeight="20" skewFactor="1.0"

          needsCallback="0"/>

  <SLIDER name="rightCutoffSlider" id="f5f11eb2404957fc" memberName="rightCutoffSlider"

          virtualName="" explicitFocusOrder="0" pos="320 232 128 88" min="1.0"

          max="15000.0" int="0.0" style="RotaryVerticalDrag" textBoxPos="NoTextBox"

          textBoxEditable="1" textBoxWidth="80" textBoxHeight="20" skewFactor="1.0"

          needsCallback="0"/>

  <SLIDER name="strengthSlider" id="f56ba4a089565a1e" memberName="strengthSlider"

          virtualName="" explicitFocusOrder="0" pos="0 8 128 88" tooltip="Strength of stereo effect"

          min="0.0" max="2.0" int="0.0" style="RotaryVerticalDrag" textBoxPos="NoTextBox"

          textBoxEditable="1" textBoxWidth="80" textBoxHeight="20" skewFactor="1.0"

          needsCallback="0"/>

  <SLIDER name="frequencySpreadSlider" id="ab103c844f994ea3" memberName="frequencySpreadSlider"

          virtualName="" explicitFocusOrder="0" pos="0 104 128 88" min="0.1"

          max="0.999" int="0.0" style="RotaryVerticalDrag" textBoxPos="NoTextBox"

          textBoxEditable="1" textBoxWidth="80" textBoxHeight="20" skewFactor="1.0"

          needsCallback="0"/>

  <TOGGLEBUTTON name="dynamicSplitEnabledButton" id="1d2e5d0b4ed829bd" memberName="dynamicSplitEnabledButton"

                virtualName="" explicitFocusOrder="0" pos="8 216 112 24" buttonText="Dynamic split"

                connectedEdges="0" needsCallback="0" radioGroupId="0" state="1"/>

  <SLIDER name="attackSlider" id="8781b1d7d0cf8163" memberName="attackSlider"

          virtualName="" explicitFocusOrder="0" pos="0 232 128 88" tooltip="few ew few few few fwe fewf wef ewf ew"

          min="0.0" max="1.0" int="0.0" style="RotaryVerticalDrag" textBoxPos="NoTextBox"

          textBoxEditable="1" textBoxWidth="80" textBoxHeight="20" skewFactor="1.0"

          needsCallback="0"/>

  <LABEL name="Left cutoff" id="8620c0632c450024" memberName="juce\_\_label"

         virtualName="" explicitFocusOrder="0" pos="208 312 144 24" edTextCol="ff000000"

         edBkgCol="0" labelText="Left cutoff" editableSingleClick="0"

         editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"

         fontsize="15.0" kerning="0.0" bold="0" italic="0" justification="36"/>

  <LABEL name="Right cutoff" id="40b2fd643a5430b4" memberName="juce\_\_label2"

         virtualName="" explicitFocusOrder="0" pos="312 312 150 24" edTextCol="ff000000"

         edBkgCol="0" labelText="Right cutoff" editableSingleClick="0"

         editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"

         fontsize="15.0" kerning="0.0" bold="0" italic="0" justification="36"/>

  <LABEL name="Strength" id="f0af7ab8cc3ea80f" memberName="juce\_\_label3"

         virtualName="" explicitFocusOrder="0" pos="0 80 128 32" edTextCol="ff000000"

         edBkgCol="0" labelText="Strength" editableSingleClick="0" editableDoubleClick="0"

         focusDiscardsChanges="0" fontname="Default font" fontsize="15.0"

         kerning="0.0" bold="0" italic="0" justification="36"/>

  <LABEL name="Frequency spread" id="692de3b7b1d976e" memberName="juce\_\_label4"

         virtualName="" explicitFocusOrder="0" pos="-8 184 152 24" edTextCol="ff000000"

         edBkgCol="0" labelText="Frequency spread" editableSingleClick="0"

         editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"

         fontsize="15.0" kerning="0.0" bold="0" italic="0" justification="36"/>

  <LABEL name="Attack" id="2459ebcb1b75d0a3" memberName="juce\_\_label5"

         virtualName="" explicitFocusOrder="0" pos="16 312 96 24" edTextCol="ff000000"

         edBkgCol="0" labelText="Attack" editableSingleClick="0" editableDoubleClick="0"

         focusDiscardsChanges="0" fontname="Default font" fontsize="15.0"

         kerning="0.0" bold="0" italic="0" justification="36"/>

  <SLIDER name="noiseSlider" id="cdc302d03b5071d1" memberName="noiseSlider"

          virtualName="" explicitFocusOrder="0" pos="112 232 128 88" tooltip="few ew few few few fwe fewf wef ewf ew"

          min="0.0" max="1.0" int="0.0" style="RotaryVerticalDrag" textBoxPos="NoTextBox"

          textBoxEditable="1" textBoxWidth="80" textBoxHeight="20" skewFactor="1.0"

          needsCallback="0"/>

  <LABEL name="Noise" id="8052689b5ca1ad00" memberName="juce\_\_label6"

         virtualName="" explicitFocusOrder="0" pos="128 312 96 24" edTextCol="ff000000"

         edBkgCol="0" labelText="Noise" editableSingleClick="0" editableDoubleClick="0"

         focusDiscardsChanges="0" fontname="Default font" fontsize="15.0"

         kerning="0.0" bold="0" italic="0" justification="36"/>

</JUCER\_COMPONENT>

END\_JUCER\_METADATA

\*/

#endif

//[EndFile] You can add extra defines here...

//[/EndFile]

1.2 PluginEditor.h

/\*

  ==============================================================================

  This is an automatically generated GUI class created by the Projucer!

  Be careful when adding custom code to these files, as only the code within

  the "//[xyz]" and "//[/xyz]" sections will be retained when the file is loaded

  and re-saved.

  Created with Projucer version: 6.0.8

  ------------------------------------------------------------------------------

  The Projucer is part of the JUCE library.

  Copyright (c) 2020 - Raw Material Software Limited.

  ==============================================================================

\*/

#pragma once

//[Headers]     -- You can add your own extra header files here --

#include <JuceHeader.h>

#include "UI/SpectrumCanvas.h"

//[/Headers]

//==============================================================================

/\*\*

                                                                    //[Comments]

    An auto-generated component, created by the Projucer.

    Describe your class and how it works here!

                                                                    //[/Comments]

\*/

class PluginEditor  : public juce::AudioProcessorEditor,

                      public juce::AudioProcessorValueTreeState::Listener

{

public:

    //==============================================================================

    PluginEditor (PluginProcessor& processor);

    ~PluginEditor() override;

    //==============================================================================

    //[UserMethods]     -- You can add your own custom methods in this section.

    void parameterChanged (const String& parameterID, float newValue) override;

    //[/UserMethods]

    void paint (juce::Graphics& g) override;

    void resized() override;

private:

    //[UserVariables]   -- You can add your own custom variables in this section.

    PluginProcessor& audioProcessor;

    typedef juce::AudioProcessorValueTreeState::SliderAttachment SliderAttachment;

    typedef juce::AudioProcessorValueTreeState::ButtonAttachment ButtonAttachment;

        std::unique\_ptr<SliderAttachment> leftCutoffAttachment, rightCutoffAttachment,

            strengthAttachment, frequencySpreadAttachment, attackAttachment, noiseAttachment;

    std::unique\_ptr<ButtonAttachment> dynamicSplitEnabledAttachment;

    std::unique\_ptr<TooltipWindow> tooltipWindow;

    //[/UserVariables]

    //==============================================================================

    std::unique\_ptr<SpectrumCanvas> spectrum\_canvas;

    std::unique\_ptr<juce::Slider> leftCutoffSlider;

    std::unique\_ptr<juce::Slider> rightCutoffSlider;

    std::unique\_ptr<juce::Slider> strengthSlider;

    std::unique\_ptr<juce::Slider> frequencySpreadSlider;

    std::unique\_ptr<juce::ToggleButton> dynamicSplitEnabledButton;

    std::unique\_ptr<juce::Slider> attackSlider;

    std::unique\_ptr<juce::Label> juce\_\_label;

    std::unique\_ptr<juce::Label> juce\_\_label2;

    std::unique\_ptr<juce::Label> juce\_\_label3;

    std::unique\_ptr<juce::Label> juce\_\_label4;

    std::unique\_ptr<juce::Label> juce\_\_label5;

    std::unique\_ptr<juce::Slider> noiseSlider;

    std::unique\_ptr<juce::Label> juce\_\_label6;

    //==============================================================================

    JUCE\_DECLARE\_NON\_COPYABLE\_WITH\_LEAK\_DETECTOR (PluginEditor)

};

//[EndFile] You can add extra defines here...

//[/EndFile]

1.3 PluginProcessor.cpp

/\*

  ==============================================================================

    This file contains the basic framework code for a JUCE plugin processor.

  ==============================================================================

\*/

#include "kfr/all.hpp"

#include "PluginProcessor.h"

#include <comcat.h>

#include "PluginEditor.h"

//==============================================================================

PluginProcessor::PluginProcessor()

     : AudioProcessor (BusesProperties()

                     #if ! JucePlugin\_IsMidiEffect

                      #if ! JucePlugin\_IsSynth

                       .withInput  ("Input",  juce::AudioChannelSet::stereo(), true)

                      #endif

                       .withOutput ("Output", juce::AudioChannelSet::stereo(), true)

                     #endif

                       ),

                       params(\*this),

                       mainProcessor(params, rendering)

{

    setLatencySamples(mainProcessor.getLatencyInSamples());

}

PluginProcessor::~PluginProcessor()

{

}

//==============================================================================

const juce::String PluginProcessor::getName() const

{

    return JucePlugin\_Name;

}

bool PluginProcessor::acceptsMidi() const

{

   #if JucePlugin\_WantsMidiInput

    return true;

   #else

    return false;

   #endif

}

bool PluginProcessor::producesMidi() const

{

   #if JucePlugin\_ProducesMidiOutput

    return true;

   #else

    return false;

   #endif

}

bool PluginProcessor::isMidiEffect() const

{

   #if JucePlugin\_IsMidiEffect

    return true;

   #else

    return false;

   #endif

}

double PluginProcessor::getTailLengthSeconds() const

{

    return 0.0;

}

int PluginProcessor::getNumPrograms()

{

    return 1;   // NB: some hosts don't cope very well if you tell them there are 0 programs,

                // so this should be at least 1, even if you're not really implementing programs.

}

int PluginProcessor::getCurrentProgram()

{

    return 0;

}

void PluginProcessor::setCurrentProgram (int index)

{

}

const juce::String PluginProcessor::getProgramName (int index)

{

    return {};

}

void PluginProcessor::changeProgramName (int index, const juce::String& newName)

{

}

std::vector<std::queue<float>> lastValues;

//==============================================================================

void PluginProcessor::prepareToPlay (double sampleRate, int samplesPerBlock)

{

    lastValues.resize(0);

    params.sampleRate = sampleRate;

    params.maxSamplesPerBlock = samplesPerBlock;

    mainProcessor.reset();

}

void PluginProcessor::releaseResources()

{

    // When playback stops, you can use this as an opportunity to free up any

    // spare memory, etc.

}

#ifndef JucePlugin\_PreferredChannelConfigurations

bool PluginProcessor::isBusesLayoutSupported (const BusesLayout& layouts) const

{

  #if JucePlugin\_IsMidiEffect

    juce::ignoreUnused (layouts);

    return true;

  #else

    // This is the place where you check if the layout is supported.

    // In this template code we only support mono or stereo.

    if (layouts.getMainOutputChannelSet() != juce::AudioChannelSet::mono()

     && layouts.getMainOutputChannelSet() != juce::AudioChannelSet::stereo())

        return false;

    // This checks if the input layout matches the output layout

   #if ! JucePlugin\_IsSynth

    if (layouts.getMainOutputChannelSet() != layouts.getMainInputChannelSet())

        return false;

   #endif

    return true;

  #endif

}

#endif

void PluginProcessor::processBlock (juce::AudioBuffer<float>& buffer, juce::MidiBuffer& midiMessages)

{

    juce::ScopedNoDenormals noDenormals;

    auto totalNumInputChannels  = getTotalNumInputChannels();

    auto totalNumOutputChannels = getTotalNumOutputChannels();

    for (auto i = totalNumInputChannels; i < totalNumOutputChannels; ++i)

        buffer.clear (i, 0, buffer.getNumSamples());

    lastValues.resize(buffer.getNumChannels());

    kfr::univector<kfr::univector<kfr::f32, 0>, 2> data({

            kfr::make\_array\_ref(buffer.getWritePointer(0),

                                buffer.getNumSamples()),

            kfr::make\_array\_ref(buffer.getWritePointer(1),

                                buffer.getNumSamples())

    });

    mainProcessor.process(data);

}

void PluginProcessor::processBlockBypassed(juce::AudioBuffer<float>& buffer, juce::MidiBuffer& midiMessages)

{

}

bool PluginProcessor::hasEditor() const

{

    return true; // (change this to false if you choose to not supply an editor)

}

juce::AudioProcessorEditor\* PluginProcessor::createEditor()

{

    return new PluginEditor(\*this);

}

//==============================================================================

void PluginProcessor::getStateInformation (juce::MemoryBlock& destData)

{

    auto state = params.tree->copyState();

    std::unique\_ptr<juce::XmlElement> xml (state.createXml());

    copyXmlToBinary (\*xml, destData);

}

void PluginProcessor::setStateInformation (const void\* data, int sizeInBytes)

{

    std::unique\_ptr<juce::XmlElement> xmlState (getXmlFromBinary (data, sizeInBytes));

    if (xmlState != nullptr)

        if (xmlState->hasTagName (params.tree->state.getType()))

            params.tree->replaceState (juce::ValueTree::fromXml (\*xmlState));

}

//==============================================================================

// This creates new instances of the plugin..

juce::AudioProcessor\* JUCE\_CALLTYPE createPluginFilter()

{

    return new PluginProcessor();

}

1.4 PluginProcessor.h

/\*

  ==============================================================================

    This file contains the basic framework code for a JUCE plugin processor.

  ==============================================================================

\*/

#pragma once

#include "JuceHeader.h"

#include "Processing/MainProcessor.h"

#include "Processing/ProcessingRendering.h"

//==============================================================================

/\*\*

\*/

class PluginProcessor  : public juce::AudioProcessor

{

public:

    //==============================================================================

    PluginProcessor();

    ~PluginProcessor() override;

    //==============================================================================

    void prepareToPlay (double sampleRate, int samplesPerBlock) override;

    void releaseResources() override;

   #ifndef JucePlugin\_PreferredChannelConfigurations

    bool isBusesLayoutSupported (const BusesLayout& layouts) const override;

   #endif

    void processBlock (juce::AudioBuffer<float>&, juce::MidiBuffer&) override;

    void processBlockBypassed (juce::AudioBuffer<float>&, juce::MidiBuffer&) override;

    //==============================================================================

    juce::AudioProcessorEditor\* createEditor() override;

    bool hasEditor() const override;

    //==============================================================================

    const juce::String getName() const override;

    bool acceptsMidi() const override;

    bool producesMidi() const override;

    bool isMidiEffect() const override;

    double getTailLengthSeconds() const override;

    //==============================================================================

    int getNumPrograms() override;

    int getCurrentProgram() override;

    void setCurrentProgram (int index) override;

    const juce::String getProgramName (int index) override;

    void changeProgramName (int index, const juce::String& newName) override;

    //==============================================================================

    void getStateInformation (juce::MemoryBlock& destData) override;

    void setStateInformation (const void\* data, int sizeInBytes) override;

    ProcessingRendering rendering;

    ProcessingParams params;

    MainProcessor mainProcessor;

private:

    //==============================================================================

    JUCE\_DECLARE\_NON\_COPYABLE\_WITH\_LEAK\_DETECTOR (PluginProcessor)

};

1.5 DataAccumulator.cpp

#include "DataAccumulator.h"

#include "JuceHeader.h"

DataAccumulator::DataAccumulator(): buffer\_pos(0), buffer\_size(-1), buffer(0) {

}

void DataAccumulator::reset(int maxSamplesPerBlock, int maxRequestLength) {

    buffer\_pos = 0;

    buffer\_size = 0;

    buffer.resize(std::max(maxSamplesPerBlock \*8, maxRequestLength \* 8));

}

void DataAccumulator::addDataSamples(const kfr::univector\_ref<kfr::f32> &data) {

    jassert(buffer.size() != 0); // not init

    if (buffer\_pos + buffer\_size + data.size() >= buffer.size()) {

        builtin\_memcpy(&buffer[0], &buffer[buffer\_pos], buffer\_size \* sizeof(kfr::f32));

        buffer\_pos = 0;

    }

    jassert(buffer\_pos + buffer\_size + data.size() < buffer.size());

    builtin\_memcpy(&buffer[buffer\_pos + buffer\_size], &data[0], data.size() \* sizeof(kfr::f32));

    buffer\_size += data.size();

}

kfr::univector\_ref<kfr::f32> DataAccumulator::getData(int left, int right) {

    jassert(left + right <= buffer\_size);

    kfr::univector\_ref<kfr::f32> res(buffer.slice(buffer\_pos, left + right));

    buffer\_pos += left;

    buffer\_size -= left;

    return res;

}

1.6 DataAccumulator.h

#pragma once

#include "kfr/all.hpp"

class DataAccumulator {

public:

    DataAccumulator();

    void addDataSamples(const kfr::univector\_ref<kfr::f32>& data) ;

    void reset(int maxSamplesPerBlock, int maxRequestLength);

    // returns [pos-left, pos+right]

    // left samples will be dropped

    kfr::univector\_ref<kfr::f32> getData(int left, int right);

    int size() { return buffer\_size; }

private:

    kfr::univector<kfr::f32> buffer;

    int buffer\_pos, buffer\_size;

};

1.7 MainProcessor.cpp

#pragma clang diagnostic push

#pragma ide diagnostic ignored "performance-unnecessary-value-param"

#include "MainProcessor.h"

using namespace kfr;

MainProcessor::MainProcessor(ProcessingParams& \_params, ProcessingRendering& \_rendering):

    params(\_params),

    rendering(\_rendering),

    dftPlan(FFT\_SZ),

    dftPlanTemp(dftPlan.temp\_size),

    dftWindow(window\_hann(FFT\_SZ)) {

}

void MainProcessor::reset() {

    univector<f32> zeros(getLatencyInSamples(), 0);

    int ovCont = OVERLAP\_COUNT;

    univector<f32> overlapZeros(OVERLAP\_COUNT, 0);

    inSumAccumulator.reset(params.maxSamplesPerBlock, FFT\_SZ);

    inSumAccumulator.addDataSamples(zeros);

    for (int i = 0; i < 2; i++) {

        inAccumulators[i].reset(params.maxSamplesPerBlock, FFT\_SZ);

        inAccumulators[i].addDataSamples(zeros);

        outAccumulators[i].reset(params.maxSamplesPerBlock, FFT\_SZ);

        fftOutAccumulators[i].reset(params.maxSamplesPerBlock, FFT\_SZ);

        fftOutAccumulators[i].addDataSamples(overlapZeros);

    }

}

void MainProcessor::process(univector<univector<f32, 0>, 2> data) {

   /\* for (int i = 0; i < 2; i++) {

        inAccumulators[i].addDataSamples(data[i]);

    }\*/

    static int iterCount = 0;

    iterCount++;

    data[0] += data[1];

    data[0] /= 2;

    inSumAccumulator.addDataSamples(data[0]);

    processFFTForAccumulator(inSumAccumulator, fftOutAccumulators[0], fftOutAccumulators[1]);

    static univector<f32, FFT\_SZ / 2> overlapAddBuffer;

    for (int i = 0; i < 2; i++) {

        // overlap-add

        while (fftOutAccumulators[i].size() >= 2 \* OVERLAP\_COUNT) {

            univector\_ref<f32> leftSegment(fftOutAccumulators[i].getData(OVERLAP\_COUNT, 0));

            univector\_ref<f32> mediumSegment(fftOutAccumulators[i].getData(OVERLAP\_COUNT,0));

            std::fill(overlapAddBuffer.begin(), overlapAddBuffer.end(), 0);

            overlapAddBuffer += leftSegment;

            overlapAddBuffer += mediumSegment;

            overlapAddBuffer /= 2.f;

            outAccumulators[i].addDataSamples(overlapAddBuffer);

        }

        jassert(fftOutAccumulators[i].size() == OVERLAP\_COUNT);

    }

    for (int i = 0; i < 2; i++) {

        univector\_ref<f32> res(outAccumulators[i].getData(data[i].size(), 0));

        builtin\_memcpy(&data[i][0], &res[0], data[i].size() \* sizeof(f32));

    }

}

void MainProcessor::processFFTForAccumulator(DataAccumulator& inAccumulator,

                                             DataAccumulator& outAccumulator1,

                                             DataAccumulator& outAccumulator2) {

    while (inAccumulator.size() >= FFT\_SZ) {

        univector\_ref<f32> in = inAccumulator.getData(FFT\_SZ - OVERLAP\_COUNT, OVERLAP\_COUNT);

        static univector<complex<f32>, FFT\_RES\_SZ> fftOut;

        dftPlan.execute(fftOut, in, dftPlanTemp); // direct; returns FFT\_SZ / 2 + 1 array

        static univector<complex<f32>, FFT\_RES\_SZ> fftOut1, fftOut2;

        applyConversionsToFFT(fftOut, fftOut1, fftOut2);

        performIFFTAndWrite(fftOut1, outAccumulator1);

        performIFFTAndWrite(fftOut2, outAccumulator2);

    }

}

void MainProcessor::performIFFTAndWrite(vecfft& in, DataAccumulator& outAccumulator) {

    static univector<f32, FFT\_SZ> ifftResult;

    dftPlan.execute(ifftResult, in , dftPlanTemp); // inverse

    ifftResult \*= dftWindow;

    ifftResult /= FFT\_SZ;

    outAccumulator.addDataSamples(ifftResult);

}

inline complex<f32> convertToCartesian(const complex<f32>& a) {

    auto x = std::polar(a.real(), a.imag());

    return {x.real(), x.imag()};

}

#define magnitudes(x) real(x)

#define phases(x) imag(x)

#define magnitude(x) x.real()

#define phase(x) x.imag()

void MainProcessor::applyConversionsToFFT(vecfft& fftData, vecfft& res1, vecfft& res2) {

    fftData = kfr::polar(fftData);

    static univector<f32, FFT\_RES\_SZ> lastSound;

    lastSound = magnitudes(fftData);

    rendering.lastSound.addValues(lastSound.data(), lastSound.size());

    res1 = fftData;

    res2 = fftData;

    processSplit(fftData, res1, res2);

    for (size\_t i = 0; i < res1.size(); i++) {

        res1[i] = convertToCartesian(res1[i]);

        res2[i] = convertToCartesian(res2[i]);

    }

}

void MainProcessor::processSplit(vecfft& data, vecfft& res1, vecfft& res2) {

    static univector<f32, FFT\_RES\_SZ> mask;

    generateMask(data, mask);

    mask += 1;

    rendering.lastMask.addValues(&mask[0], mask.size());

    for (int i = 0; i < data.size(); i++) {

        res1[i] = {magnitude(data[i]) \* mask[i], phase(data[i])};

        res2[i] = {magnitude(data[i]) \* (2 - mask[i]), phase(data[i])};

    }

}

// mask [-1, 1]

void MainProcessor::generateMask(vecfft& data, univector<f32, 0> mask) {

    static univector<f32, FFT\_RES\_SZ> mediumMagnitude(0);

    float weight = \*params.attack;

    float otherWeight = 1 - weight;

    for (int i = 0; i < data.size(); i++) {

        mediumMagnitude[i] = weight \* mediumMagnitude[i] + otherWeight \* magnitude(data[i]);

    }

    int leftFreq = freqToFFTIndex(\*params.leftCutoff);

    int rightFreq = freqToFFTIndex(\*params.rightCutoff);

    float sineX = 0.f;

    float sineStretch = 1 - \*params.frequencySpread;

    bool dynamicSpreadEnabled = \*params.dynamicSplitEnabled > 0.5f;

    if (dynamicSpreadEnabled) {

        for (int i = leftFreq; i < rightFreq; i++) {

            sineX += mediumMagnitude[i] / sineStretch / 100;

            float sineY = sin(sineX);

            mask[i] = sineY;

        }

    } else {

        for (int i = leftFreq; i < rightFreq; i++) {

            float sineY = sin(log2f(FFTIndexToFreq(i) + 1) / sineStretch / 2.5);

            mask[i] = sineY;

        }

    }

    float noiseCount = \*params.noise;

    if (noiseCount != 0) {

        noiseGenerator.nextGenerate(0.0f, 0.6f);

        float minX =  log2f(FFTIndexToFreq(0) + 1);

        float maxX = log2f(FFTIndexToFreq(data.size() - 1) + 1);

        for (int i = leftFreq; i < rightFreq; i++) {

            float x = log2f(FFTIndexToFreq(i) + 1);

            mask[i] = noiseGenerator.getValue((x - minX) / (maxX - minX) \* 2048) \* noiseCount +

                        mask[i] \* (1 - noiseCount);

        }

    }

    float effectStrength = \*params.strength;

    if (effectStrength < 1) {

        for (int i = leftFreq; i < rightFreq; i++) {

            mask[i] \*= effectStrength;

        }

    } else {

        for (int i = leftFreq; i < rightFreq; i++) {

            mask[i] \*= effectStrength;

            if (mask[i] > 1) {

                mask[i] = 1;

            }

            if (mask[i] < -1) {

                mask[i] = -1;

            }

        }

    }

    std::fill(mask.begin(), mask.begin() + leftFreq, 0);

    std::fill(mask.begin() + rightFreq, mask.end(), 0);

}

//           N:  0, 1, 2,   3    ..  FFT\_RES\_SZ - 1

// frequencies:  0, x, 2\*x  3\*x  ..  sample\_rate / 2 (Nyquist frequency)

int MainProcessor::freqToFFTIndex(int frequency) const {

    return (int)((float)frequency / params.sampleRate \* 2 \* (FFT\_RES\_SZ - 1));

}

int MainProcessor::FFTIndexToFreq(int index) const {

    return (int)((float)index \* params.sampleRate / 2 / (FFT\_RES\_SZ - 1));

}

int MainProcessor::getLatencyInSamples() {

    return FFT\_SZ;

}

#pragma clang diagnostic pop

1.8 MainProcessor.h

#pragma once

#include "kfr/all.hpp"

#include "ProcessingParams.h"

#include "DataAccumulator.h"

#include "ProcessingRendering.h"

#include "NoiseGenerator.h"

#define FFT\_SZ 2048

#define FFT\_RES\_SZ (FFT\_SZ / 2 + 1)

#define OVERLAP\_COUNT (FFT\_SZ / 2)

class MainProcessor {

public:

    explicit MainProcessor(ProcessingParams& params, ProcessingRendering& \_rendering);

    void reset();

    void process(kfr::univector<kfr::univector<kfr::f32, 0>, 2> data);

    int getLatencyInSamples();

private:

    ProcessingParams& params;

    ProcessingRendering& rendering;

    DataAccumulator inAccumulators[2];

    DataAccumulator inSumAccumulator;

    DataAccumulator fftOutAccumulators[2];

    DataAccumulator outAccumulators[2];

    kfr::dft\_plan\_real<kfr::f32> dftPlan;

    kfr::univector<kfr::u8> dftPlanTemp;

    kfr::univector<kfr::f32> dftWindow;

    NoiseGenerator<float, 2048> noiseGenerator;

    typedef kfr::univector<kfr::complex<kfr::f32>, FFT\_RES\_SZ> vecfft;

    typedef kfr::univector<kfr::complex<kfr::f32>, 0> vecfft\_ref;

    void processFFTForAccumulator(DataAccumulator& inAccumulator, DataAccumulator& outAccumulator1,

                                  DataAccumulator& outAccumulator2);

    void performIFFTAndWrite(vecfft& in, DataAccumulator& outAccumulator);

    void applyConversionsToFFT(vecfft& fftData,

                               vecfft& res1,

                               vecfft& res2);

    void processSplit(vecfft& data, vecfft& res1, vecfft& res2);

    void generateMask(vecfft& data,

                      kfr::univector<kfr::f32, 0> mask);

    int freqToFFTIndex(int frequency) const;

    int FFTIndexToFreq(int index) const;

};

1.9 NoiseGenerator.h

#pragma once

template<typename T, int size>

class NoiseGenerator {

public:

    NoiseGenerator() : gen(1), distribution(-1, 1) {

        std::fill(values.begin(), values.end(), 0);

    }

    void nextGenerate(float deltaX, float deltaY) {

        float randomWeight = 1 - deltaY - deltaX;

        tmpValues[0] = values[0] \* deltaY + distribution(gen) \* (1 - deltaY);

        tmpValues[size-1] = values[size-1] \* deltaY + distribution(gen) \* (1 - deltaY);

        deltaX /= 2;

        float minValue=1, maxValue = -1.f;

        for (int i = 1; i < size - 1; i++) {

            tmpValues[i] = values[i - 1] \* deltaX + values[i + 1] \* deltaX

                    + values[i] \* deltaY +

                    randomWeight \* distribution(gen);

            if (tmpValues[i] > 1) {

                tmpValues[i] = 1;

            }

            if (tmpValues[i] < -1) {

                tmpValues[i] = -1;

            }

            minValue = std::fmin(tmpValues[i], minValue);

            maxValue = std::fmax(tmpValues[i], maxValue);

        }

        for (int i = 0; i < size; i++) {

            values[i] = (tmpValues[i] - minValue) / (maxValue - minValue) \* 2 - 1;

        }

       /\* for (int i = 1; i < size; i++) {

            values[i] = tmpValues[i - 1]  + tmpValues[i] \* deltaX;

        }\*/

    }

    inline T getValue(const size\_t& x) {

        return values[x];

    }

private:

    std::mt19937 gen;

    std::uniform\_real\_distribution<T> distribution;

    std::array<T, size> values, tmpValues;

};

1.10 ProcessingParams.cpp

#include "ProcessingParams.h"

#include <memory>

ProcessingParams::ProcessingParams(AudioProcessor& processor)

{

    NormalisableRange<float> cutoffNormalizeRange(1, 15000);

    cutoffNormalizeRange.setSkewForCentre(1024);

    tree.reset(new juce::AudioProcessorValueTreeState(

            processor, nullptr, juce::Identifier("Stereo\_plugin"),

         {

                 std::make\_unique<juce::AudioParameterFloat>("leftCutoff", // parameterID

                                                             "Left cutoff", // parameter name

                                                             cutoffNormalizeRange,

                                                             1),

                 std::make\_unique<juce::AudioParameterFloat>("rightCutoff", // parameterID

                                                             "Right cutoff", // parameter name

                                                             cutoffNormalizeRange,

                                                             15000),

                 std::make\_unique<juce::AudioParameterFloat>("strength", // parameterID

                                                             "Strength", // parameter name

                                                             0,   // minimum value

                                                             2,   // maximum value

                                                             1),

                 std::make\_unique<juce::AudioParameterFloat>("frequencySpread", // parameterID

                                                             "Frequency spread", // parameter name

                                                             0.1,   // minimum value

                                                             0.999,   // maximum value

                                                             0.3),

                 std::make\_unique<juce::AudioParameterFloat>("attack", // parameterID

                                                             "Attack", // parameter name

                                                             0,   // minimum value

                                                             0.999,   // maximum value

                                                             0.95),

                 std::make\_unique<juce::AudioParameterFloat>("noise", // parameterID

                                                             "Noise", // parameter name

                                                             0,   // minimum value

                                                             1,   // maximum value

                                                             0),

                 std::make\_unique<juce::AudioParameterFloat>("dynamicSplitEnabled", // parameterID

                                                             "Dynamic split", // parameter name

                                                             0,   // minimum value

                                                             1,   // maximum value

                                                             1),

         }));

    leftCutoff = tree->getRawParameterValue("leftCutoff");

    rightCutoff = tree->getRawParameterValue("rightCutoff");

    strength = tree->getRawParameterValue("strength");

    frequencySpread = tree->getRawParameterValue("frequencySpread");

    attack = tree->getRawParameterValue("attack");

    noise = tree->getRawParameterValue("noise");

    dynamicSplitEnabled = tree->getRawParameterValue("dynamicSplitEnabled");

}

1.11 ProcessingParams.h

#pragma once

#include <JuceHeader.h>

class ProcessingParams {

public:

    ProcessingParams(AudioProcessor& processor);

    std::unique\_ptr<juce::AudioProcessorValueTreeState> tree;

    std::atomic<float>\* leftCutoff;

    std::atomic<float>\* rightCutoff;

    std::atomic<float>\* strength;

    std::atomic<float>\* frequencySpread;

    std::atomic<float>\* attack;

    std::atomic<float>\* noise;

    std::atomic<float>\* dynamicSplitEnabled;

    float sampleRate;

    int maxSamplesPerBlock;

};

1.12 ProcessingRendering.cpp

#include "ProcessingRendering.h"

1.13 ProcessingRendering.h

#pragma once

#include "SyncedArray.h"

class ProcessingRendering {

public:

    SyncedArray<float> lastMask;

    SyncedArray<float> lastSound;

    ProcessingRendering() : lastMask(0.7f), lastSound(0.9f) {

    }

};

1.14 SyncedArray.cpp

#include "SyncedArray.h"

1.15 SyncedArray.h

#pragma once

#include "JuceHeader.h"

template<typename T>

class SyncedArray {

public:

    explicit SyncedArray(float \_smoothFactor) :

            valuesLock(),

            values(),

            smoothFactor(\_smoothFactor) {

        jassert(0.f <= smoothFactor && smoothFactor < 1);

    }

    void addValues(T\* newValues, int valuesCount) {

        juce::CriticalSection::ScopedLockType locker(valuesLock);

        if (values.size() == valuesCount && smoothFactor != 0)  {

            float otherSmoothFactor = 1 - smoothFactor;

            for (int i = 0; i < valuesCount; i++) {

                values[i] = smoothFactor \* values[i] + otherSmoothFactor \* newValues[i];

            }

        } else {

            values.assign(newValues, newValues + valuesCount);

        }

    }

    void getValues(std::vector<T>& array) {

        juce::CriticalSection::ScopedLockType locker(valuesLock);

        array.assign(values.begin(), values.end());

    }

    void reset() {

        juce::CriticalSection::ScopedLockType locker(valuesLock);

        values.resize(0);

    }

    size\_t size() {

        juce::CriticalSection::ScopedLockType locker(valuesLock);

        return values.size();

    }

private:

    juce::CriticalSection valuesLock;

    std::vector<T> values;

    float smoothFactor;

};

1.16 SpectrumCanvas.cpp

/\*

  ==============================================================================

    SpectrumCanvas.cpp

    Created: 25 Apr 2021 7:37:03pm

    Author:  user

  ==============================================================================

\*/

#include <JuceHeader.h>

#include "SpectrumCanvas.h"

//==============================================================================

SpectrumCanvas::SpectrumCanvas(ProcessingParams& \_params, ProcessingRendering& \_rendering) :

    params(\_params),

    rendering(\_rendering)

{

    setOpaque(false);

    startTimerHz(24);

  //  renderer.reset(new DynamicTextureOpenGL::DynamicTextureRenderer(\*this, 128, 128));

}

SpectrumCanvas::~SpectrumCanvas() {

    stopTimer();

}

void SpectrumCanvas::timerCallback() {

    createMaskPath();

    createSoundPath();

    getParentComponent()->repaint();

}

void SpectrumCanvas::paint (juce::Graphics& g) {

    g.fillAll(Colour::fromRGB(0, 0, 0));

    g.setColour(Colour::fromRGBA(255, 255, 255, 100));

    g.fillPath(soundPath);

    g.setColour(Colour::fromRGB(255, 0, 0));

    g.strokePath(maskPath[0],  juce::PathStrokeType (2.0) );

    g.setColour(Colour::fromRGB(0, 0, 255));

    g.strokePath(maskPath[1],  juce::PathStrokeType (2.0) );

}

void SpectrumCanvas::createMaskPath() {

    if (rendering.lastMask.size() != 0) {

        rendering.lastMask.getValues(mask);

    }

    if (mask.empty()) {

        return;

    }

    for (int maskSide = 0; maskSide < 2; maskSide++) {

        Path& path = maskPath[maskSide];

        path.clear();

        auto width = (float) getWidth();

        auto height = (float) getHeight();

        int realCnt = mask.size();

        int cnt = 20;

        float perSegment = width / (cnt - 1);

        float f = powf(realCnt, 1.f / cnt); //  j \*\* cnt = realCount111

        float j = 1;

        for (int i = 0; i < cnt; i++) {

            float x = i \* perSegment;

            float y;

            if (maskSide == 0) {

                y = mask[j] / 2.f \* height;

            } else {

                y = (2 - mask[j]) / 2.f \* height;

            }

            if (i == 0) {

                path.startNewSubPath(x, y);

            } else {

                path.lineTo(x, y);

            }

            j \*= f;

        }

        path = path.createPathWithRoundedCorners(20.f);

    }

}

void SpectrumCanvas::createSoundPath() {

    if (rendering.lastSound.size() != 0) {

        rendering.lastSound.getValues(sound);

    }

    if (sound.empty()) {

        return;

    }

    Path& path = soundPath;

    path.clear();

    auto width = (float) getWidth();

    auto height = (float) getHeight();

    int realCnt = mask.size();

    int cnt = 20;

    float perSegment = width / (cnt - 1);

    float f = powf(realCnt, 1.f / cnt); //  j \*\* cnt = realCount111

    float j = 1;

    path.startNewSubPath(-50, height + 50);

    for (int i = 0; i < cnt; i++) {

        float x = i \* perSegment;

        float y = juce::Decibels::gainToDecibels(sound[j] / 150.f, -60.f) / -60.f \* height;

      //  if (i == 0) {

            path.lineTo(x, y);

     /\*   } else {

            float startX = 0, startY = 0, joinX = 0, joinY = 0, endX, endY;

        }\*/

        j \*= f;

    }

    path.lineTo(width + 50, height + 50);

    path.closeSubPath();

    path = path.createPathWithRoundedCorners(20.f);

}

1.17 SpectrumCanvas.h

#pragma once

#include <JuceHeader.h>

#include "../Processing/ProcessingRendering.h"

#include "../Processing/ProcessingParams.h"

class SpectrumCanvas : public juce::Component, private juce::Timer

{

public:

    SpectrumCanvas(ProcessingParams& \_params, ProcessingRendering& \_rendering);

    ~SpectrumCanvas() override;

    void timerCallback() override;

    void paint (juce::Graphics&) override;

private:

    ProcessingParams& params;

    ProcessingRendering& rendering;

    Path maskPath[2];

    Path soundPath;

    std::vector<float> mask;

    std::vector<float> sound;

    void createMaskPath();

    void createSoundPath();

    JUCE\_DECLARE\_NON\_COPYABLE\_WITH\_LEAK\_DETECTOR (SpectrumCanvas)

};