```
/*

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

    This file was auto-generated!

    It contains the basic framework code for a JUCE plugin processor.


==============================================================================
*/

#include "PluginProcessor.h"
#include "PluginEditor.h"

//==============================================================================
DeJongContortionAudioProcessor::DeJongContortionAudioProcessor()
#ifndef JucePlugin_PreferredChannelConfigurations
    : AudioProcessor (BusesProperties()
                     #if ! JucePlugin_IsMidiEffect
                      #if ! JucePlugin_IsSynth
                       .withInput  ("Input",  AudioChannelSet::stereo(), true)
                      #endif
                       .withOutput ("Output", AudioChannelSet::stereo(), true)
                     #endif
                       ),

parameters (*this, nullptr, Identifier ("DejongContortion"),
          {
              std::make_unique<AudioParameterFloat>
              ("varA", "Color A", NormalisableRange<float>(1, 48, 1), 0, " ",
AudioProcessorParameter::genericParameter, [](float value, int) {return String(value) + "
db";}),
              std::make_unique<AudioParameterFloat>
              ("varB", "Color B", NormalisableRange<float>(1, 48, 1), 0, " ",
AudioProcessorParameter::genericParameter, [](float value, int) {return String(value) + "
db";}),
              std::make_unique<AudioParameterFloat>
              ("varC", "Color C", NormalisableRange<float>(0.01, 0.9, 0.01f), 0, " ",
AudioProcessorParameter::genericParameter, [](float value, int) {return String(value) + "
";}),
              std::make_unique<AudioParameterFloat>
              ("varD", "Color D", NormalisableRange<float>(0.01, 0.9, 0.01f), 0, " ",
```

```
AudioProcessorParameter::genericParameter, [](float value, int) {return String(value) + "
";}),
            std::make_unique<AudioParameterFloat>
            ("drive", "Drive", NormalisableRange<float>(0, 24, 0.5f), 0, " ",
AudioProcessorParameter::genericParameter, [](float value, int) {return String(value) + "
db";}),
            std::make_unique<AudioParameterFloat>
            ("gain", "Output Gain", NormalisableRange<float>(-12, 12, 0.1), 0, " ",
AudioProcessorParameter::genericParameter, [](float value, int) {return String(value) + "
db";}),

        })

#endif
{
    varAParameter = parameters.getRawParameterValue("varA");
    varBParameter = parameters.getRawParameterValue("varB");
    varCParameter = parameters.getRawParameterValue("varC");
    varDParameter = parameters.getRawParameterValue("varD");
    driveParameter = parameters.getRawParameterValue("drive");
    gainParameter = parameters.getRawParameterValue("gain");


}

DeJongContortionAudioProcessor::~DeJongContortionAudioProcessor()
{
}

//
==============================================================
============
const String DeJongContortionAudioProcessor::getName() const
{
    return JucePlugin_Name;
}

bool DeJongContortionAudioProcessor::acceptsMidi() const
{
   #if JucePlugin_WantsMidiInput
    return true;
   #else
    return false;
   #endif
}

bool DeJongContortionAudioProcessor::producesMidi() const
```

```
{
  #if JucePlugin_ProducesMidiOutput
   return true;
  #else
   return false;
  #endif
}

bool DeJongContortionAudioProcessor::isMidiEffect() const
{
  #if JucePlugin_IsMidiEffect
   return true;
  #else
   return false;
  #endif
}

double DeJongContortionAudioProcessor::getTailLengthSeconds() const
{
   return 0.0;
}

int DeJongContortionAudioProcessor::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 DeJongContortionAudioProcessor::getCurrentProgram()
{
   return 0;
}

void DeJongContortionAudioProcessor::setCurrentProgram (int index)
{
}

const String DeJongContortionAudioProcessor::getProgramName (int index)
{
   return {};
}

void DeJongContortionAudioProcessor::changeProgramName (int index, const String&
newName)
{
}
```

```
//
==============================================================================
void DeJongContortionAudioProcessor::prepareToPlay (double sampleRate, int
samplesPerBlock)
{

}

void DeJongContortionAudioProcessor::releaseResources()
{

}

#ifndef JucePlugin_PreferredChannelConfigurations
bool DeJongContortionAudioProcessor::isBusesLayoutSupported (const BusesLayout&
layouts) const
{
  #if JucePlugin_IsMidiEffect
    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() != AudioChannelSet::mono()
     && layouts.getMainOutputChannelSet() != 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 DeJongContortionAudioProcessor::processBlock (AudioBuffer<float>& buffer,
MidiBuffer& midiMessages)
{
    ScopedNoDenormals noDenormals;
    //auto totalNumInputChannels  = getTotalNumInputChannels();
    auto totalNumInputChannels = getMainBusNumInputChannels();
    //auto totalNumOutputChannels = getTotalNumOutputChannels();
```

```
auto totalNumOutputChannels = getMainBusNumOutputChannels();

bool stereo;
bool mono;

if (totalNumInputChannels == 1)
{
    stereo = false;
    mono = true;
}

else
{
    stereo = true;
    mono = false;
}

auto gain = pow(10, *gainParameter * 0.05f);
//auto drive = pow(10, *driveParameter * 0.05f);
auto prevX = 0.85;
auto prevY = 0.25;
auto varA = (log(*varAParameter) * 0.084f) + 1;
auto varB = (log(*varBParameter) * 0.089f) + 1;

for (auto i = totalNumInputChannels; i < totalNumOutputChannels; ++i)
    buffer.clear (i, 0, buffer.getNumSamples());

for (int channel = 0; channel < totalNumInputChannels; ++channel)
{
    auto* channelData = buffer.getWritePointer (channel);
    auto* inputX = buffer.getReadPointer(0);
    auto* inputY = buffer.getReadPointer(1);

    auto* outBufferOne = buffer.getWritePointer (0);
    auto* outBufferTwo = buffer.getWritePointer (1);

    for (int sample = 0; sample < buffer.getNumSamples(); ++sample)
    {
        if (stereo == true)
        {
        prevX = prevX + *varCParameter + (inputX[sample] * gain);
        prevY = prevY + *varDParameter + (inputY[sample] * gain);

        auto newXDeJongX = (sin(varA * prevY) - cos(varB * prevX));
        auto newYDeJongY = (sin(varB * prevX) - cos(varA * prevY));

        if (inputX[sample] == 0)
```

```
            {
                outBufferOne[sample] = 0;
            }

            else
            {
            outBufferOne[sample] = ((newXDeJongX) * 0.25f) * gain;
            }

            if (inputY[sample] == 0)
            {
                outBufferTwo[sample] = 0;
            }

            else
            {
            outBufferTwo[sample] = ((newYDeJongY) * 0.25f) * gain;
            }

            // Update prevX and prevY so that the next time this for-loop starts, they will
have the previous
            // sample value stored in them.
            prevX = newXDeJongX;
            prevY = newYDeJongY;
            }
            else
            {
                channelData[sample] = 0;
            }
        }
    }
}

void DeJongContortionAudioProcessor::clearBuffer(AudioBuffer<float>& buffer)
{

}

//
==============================================================================
============
bool DeJongContortionAudioProcessor::hasEditor() const
{
    return true; // (change this to false if you choose to not supply an editor)
}

AudioProcessorEditor* DeJongContortionAudioProcessor::createEditor()
```

```
{
    return new DeJongContortionAudioProcessorEditor (*this);
}

//
==============================================================================
void DeJongContortionAudioProcessor::getStateInformation (MemoryBlock& destData)
{
    // You should use this method to store your parameters in the memory block.
    // You could do that either as raw data, or use the XML or ValueTree classes
    // as intermediaries to make it easy to save and load complex data.
}

void DeJongContortionAudioProcessor::setStateInformation (const void* data, int
sizeInBytes)
{
    // You should use this method to restore your parameters from this memory block,
    // whose contents will have been created by the getStateInformation() call.
}

//
==============================================================================
// This creates new instances of the plugin..
AudioProcessor* JUCE_CALLTYPE createPluginFilter()
{
    return new DeJongContortionAudioProcessor();
}



/*

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

    This file was auto-generated!

    It contains the basic framework code for a JUCE plugin processor.


==============================================================================
*/

#pragma once
```

```
#include "../JuceLibraryCode/JuceHeader.h"

//
==============================================================================
/**
*/
class DeJongContortionAudioProcessor  : public AudioProcessor
{
public:
    //
==============================================================================
    DeJongContortionAudioProcessor();
    ~DeJongContortionAudioProcessor();

    //
==============================================================================
    void prepareToPlay (double sampleRate, int samplesPerBlock) override;
    void releaseResources() override;

   #ifndef JucePlugin_PreferredChannelConfigurations
    bool isBusesLayoutSupported (const BusesLayout& layouts) const override;
   #endif

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

    //
==============================================================================
    AudioProcessorEditor* createEditor() override;
    bool hasEditor() const override;

    //
==============================================================================
    const 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 String getProgramName (int index) override;
    void changeProgramName (int index, const String& newName) override;

    //
==============================================================================
============
    void getStateInformation (MemoryBlock& destData) override;
    void setStateInformation (const void* data, int sizeInBytes) override;

    void clearBuffer(AudioBuffer<float>&);

    AudioProcessorValueTreeState parameters;

    float* varAParameter;
    float* varBParameter;
    float* varCParameter;
    float* varDParameter;
    float* driveParameter;
    float* gainParameter;

    double piDivisor = 2 / M_PI;



private:
    //
==============================================================================
============
    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR
(DeJongContortionAudioProcessor)
};



/*

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

    This file was auto-generated!

    It contains the basic framework code for a JUCE plugin editor.
```

```
==============================================================================
============
*/

#include "PluginProcessor.h"
#include "PluginEditor.h"

//
==============================================================================
============
DeJongContortionAudioProcessorEditor::DeJongContortionAudioProcessorEditor
(DeJongContortionAudioProcessor& p)
    : AudioProcessorEditor (&p), processor (p)
{
    // Make sure that before the constructor has finished, you've set the
    // editor's size to whatever you need it to be.
    setSize (552, 512);

    addAndMakeVisible(varASlider);
    varAAttach = new AudioProcessorValueTreeState::SliderAttachment
(processor.parameters, "varA", varASlider);
    varASlider.addListener(this);
    varASlider.setValue(*processor.varAParameter);
    varASlider.setRange(1.0, 48.0, 0.5f);
    varASlider.setSliderStyle(Slider::SliderStyle::Rotary);
    varASlider.setTextBoxStyle(Slider::TextBoxBelow, false, 64, 24);
    varASlider.setBounds(4, 24, 128, 128);

    addAndMakeVisible(varALabel);
    varALabel.setText("Color 1", dontSendNotification);
    varALabel.setBounds(42, 4, 64, 24);


    addAndMakeVisible(varBSlider);
    varBAttach = new AudioProcessorValueTreeState::SliderAttachment
(processor.parameters, "varB", varBSlider);
    varBSlider.addListener(this);
    varBSlider.setValue(*processor.varBParameter);
    varBSlider.setRange(1.0, 48.0, 0.5f);
    varBSlider.setSliderStyle(Slider::SliderStyle::Rotary);
    varBSlider.setTextBoxStyle(Slider::TextBoxBelow, false, 64, 24);
    varBSlider.setBounds(136, 24, 128, 128);

    addAndMakeVisible(varBLabel);
    varBLabel.setText("Color 2", dontSendNotification);
    varBLabel.setBounds(174, 4, 64, 24);
```

```
    addAndMakeVisible(varCSlider);
    varCAttach = new AudioProcessorValueTreeState::SliderAttachment
(processor.parameters, "varC", varCSlider);
    varCSlider.addListener(this);
    varCSlider.setValue(*processor.varCParameter);
    varCSlider.setRange(0.01, 0.9, 0.01f);
    varCSlider.setSliderStyle(Slider::SliderStyle::Rotary);
    varCSlider.setTextBoxStyle(Slider::TextBoxBelow, false, 92, 24);
    varCSlider.setBounds(268, 24, 128, 128);

    addAndMakeVisible(varCLabel);
    varCLabel.setText("Left Portal", dontSendNotification);
    varCLabel.setBounds(296, 4, 128, 24);

    addAndMakeVisible(varDSlider);
    varDAttach = new AudioProcessorValueTreeState::SliderAttachment
(processor.parameters, "varD", varDSlider);
    varDSlider.addListener(this);
    varDSlider.setValue(*processor.varDParameter);
    varDSlider.setRange(0.01, 0.9, 0.01f);
    varDSlider.setSliderStyle(Slider::SliderStyle::Rotary);
    varDSlider.setTextBoxStyle(Slider::TextBoxBelow, false, 92, 24);
    varDSlider.setBounds(400, 24, 128, 128);

    addAndMakeVisible(varDLabel);
    varDLabel.setText("Right Portal", dontSendNotification);
    varDLabel.setBounds(424, 4, 128, 24);

//    addAndMakeVisible(driveSlider);
//    driveAttach = new AudioProcessorValueTreeState::SliderAttachment
(processor.parameters, "drive", driveSlider);
//    driveSlider.addListener(this);
//    driveSlider.setValue(*processor.driveParameter);
//    driveSlider.setRange(0, 24, 0.5f);
//    driveSlider.setSliderStyle(Slider::SliderStyle::LinearHorizontal);
//    driveSlider.setBounds(24, 264, 352, 24);
//
//    addAndMakeVisible(driveLabel);
//    driveLabel.setText("Drive", dontSendNotification);
//    driveLabel.attachToComponent(&driveSlider, false);
//
//    addAndMakeVisible(gainSlider);
//    gainAttach = new AudioProcessorValueTreeState::SliderAttachment
(processor.parameters, "gain", gainSlider);
```

```
//    gainSlider.addListener(this);
//    gainSlider.setValue(*processor.gainParameter);
//    gainSlider.setRange(-12, 12, 0.1);
//    gainSlider.setSliderStyle(Slider::SliderStyle::Rotary);
//    gainSlider.setTextBoxStyle(Slider::TextBoxBelow, false, 92, 24);
//    gainSlider.setBounds(4, 192, 128, 128);
//
//    addAndMakeVisible(gainLabel);
//    gainLabel.setText("In Gain", dontSendNotification);
//    gainLabel.setBounds(42, 172, 64, 24);


}

DeJongContortionAudioProcessorEditor::~DeJongContortionAudioProcessorEditor()
{
    varAAttach.reset();
    varBAttach.reset();
    varCAttach.reset();
    varDAttach.reset();
    driveAttach.reset();
    gainAttach.reset();
}

//
==============================================================================
============
void DeJongContortionAudioProcessorEditor::paint (Graphics& g)
{
    // (Our component is opaque, so we must completely fill the background with a solid
colour)
    g.fillAll (getLookAndFeel().findColour (ResizableWindow::backgroundColourId));
}

void DeJongContortionAudioProcessorEditor::resized()
{
    // This is generally where you'll want to lay out the positions of any
    // subcomponents in your editor..
}

void DeJongContortionAudioProcessorEditor::sliderValueChanged(Slider *slider)
{
    if (slider == &varASlider)
    {
        *processor.varAParameter = varASlider.getValue();
    }
```

```
    if (slider == &varBSlider)
    {
        *processor.varBParameter = varBSlider.getValue();
    }

    if (slider == &varCSlider)
    {
        *processor.varCParameter = varCSlider.getValue();
    }

    if (slider == &varDSlider)
    {
        *processor.varDParameter = varDSlider.getValue();
    }

    if (slider == &driveSlider)
    {
        *processor.driveParameter = driveSlider.getValue();
    }

    if (slider == &gainSlider)
    {
        *processor.gainParameter = gainSlider.getValue();
    }
}



/*

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

    This file was auto-generated!

    It contains the basic framework code for a JUCE plugin editor.


==============================================================================
*/

#pragma once

#include "../JuceLibraryCode/JuceHeader.h"
#include "PluginProcessor.h"
```

```
//
==============================================================================
/**
*/
class DeJongContortionAudioProcessorEditor  : public AudioProcessorEditor,
Slider::Listener
{
public:
    DeJongContortionAudioProcessorEditor (DeJongContortionAudioProcessor&);
    ~DeJongContortionAudioProcessorEditor();

    //
==============================================================================
    void paint (Graphics&) override;
    void resized() override;
    void sliderValueChanged(Slider *slider) override;

    ScopedPointer<AudioProcessorValueTreeState::SliderAttachment> varAAttach;
    ScopedPointer<AudioProcessorValueTreeState::SliderAttachment> varBAttach;
    ScopedPointer<AudioProcessorValueTreeState::SliderAttachment> varCAttach;
    ScopedPointer<AudioProcessorValueTreeState::SliderAttachment> varDAttach;
    ScopedPointer<AudioProcessorValueTreeState::SliderAttachment> driveAttach;
    ScopedPointer<AudioProcessorValueTreeState::SliderAttachment> gainAttach;

    Slider varASlider;
    Slider varBSlider;
    Slider varCSlider;
    Slider varDSlider;
    Slider driveSlider;
    Slider gainSlider;

    Label varALabel;
    Label varBLabel;
    Label varCLabel;
    Label varDLabel;
    Label driveLabel;
    Label gainLabel;

private:
    // This reference is provided as a quick way for your editor to
    // access the processor object that created it.
    DeJongContortionAudioProcessor& processor;

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR
(DeJongContortionAudioProcessorEditor)
```

```
};
```