**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Департамент программной инженерии

|  |  |
| --- | --- |
| СОГЛАСОВАНО  Доцент Департамента программной инженерии Факультета компьютерных наук, канд. техн. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.А. Дударев  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2019 г. | УТВЕРЖДАЮ  Академический руководитель образовательной программы «Программная инженерия»  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Шилов  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2019 г. |

|  |  |
| --- | --- |
| *Подп. и дата* |  |
| *Инв. № дубл.* |  |
| *Взам. инв. №* |  |
| *Подп. и дата* |  |
| *Инв. № подл* | *RU.17701729.04.01-01 12 01-1* |

**ПРОГРАММНЫЙ СИНТЕЗАТОР В ФОРМАТЕ ИНСТРУМЕНТА VST**

**Текст программы**

**ЛИСТ УТВЕРЖДЕНИЯ**

**RU.17701729.04.01-01 12 01-1-ЛУ**

|  |  |
| --- | --- |
|  | Исполнитель:  студент группы БПИ182  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Петелин М.О. /  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2019 г. |

**2019**

|  |
| --- |
| УТВЕРЖДЕН  RU.17701729.04.01-01 12 01-1-ЛУ |

|  |  |
| --- | --- |
| *Подп. и дата* |  |
| *Инв. № дубл.* |  |
| *Взам. инв. №* |  |
| *Подп. и дата* |  |
| *Инв. № подл* | *RU.17701729.04.01-01 12 01-1* |

**ПРОГРАММНЫЙ СИНТЕЗАТОР В ФОРМАТЕ ИНСТРУМЕНТА VST**

**Текст программы**

**RU.17701729.04.01-01 12 01-1**

**Листов 183**

**2019**

# АННОТАЦИЯ

В данном программном документе представлен текст программы «Программный синтезатор в формате инструмента VST». В разделе «Текст программы» находится текст программы, распределённый по файлам.

**СОДЕРЖАНИЕ**

[1. ТЕКСТ ПРОГРАММЫ 5](#_Toc8772577)

[1.1. Components\AudioComponent.cs 5](#_Toc8772578)

[1.2. Components\AudioComponentWithParameters.cs 5](#_Toc8772579)

[1.3. Components\DelayManager.cs 7](#_Toc8772580)

[1.4. Components\DistortionManager.cs 16](#_Toc8772581)

[1.5. Envelope.cs 24](#_Toc8772582)

[1.6. Components\EnvelopesManager.cs 28](#_Toc8772583)

[1.7. Components\Filter.cs 35](#_Toc8772584)

[1.8. Components\FiltersManager.cs 38](#_Toc8772585)

[1.9. Components\Oscillator.cs 43](#_Toc8772586)

[1.10. Components\OscillatorsManager.cs 45](#_Toc8772587)

[1.11. Components\ParameterFilter.cs 49](#_Toc8772588)

[1.12. Components\Routing.cs 52](#_Toc8772589)

[1.13. Components\Voice.cs 56](#_Toc8772590)

[1.14. Components\VoicesManager.cs 62](#_Toc8772591)

[1.15. DSP\Effects\Delay\DelayLine.cs 68](#_Toc8772592)

[1.16. DSP\Effects\Delay\IDelay.cs 71](#_Toc8772593)

[1.17. DSP\Effects\Delay\PingPongDelay.cs 72](#_Toc8772594)

[1.18. DSP\Effects\Delay\StereoOffsetDelay.cs 74](#_Toc8772595)

[1.19. DSP\Effects\Delay\VariousTimeDelay.cs 77](#_Toc8772596)

[1.20. DSP\Effects\Distortion\AbsClipper.cs 79](#_Toc8772597)

[1.21. DSP\Effects\Distortion\BitCrusher.cs 80](#_Toc8772598)

[1.22. DSP\Effects\Distortion\CubicClipper.cs 81](#_Toc8772599)

[1.23. DSP\Effects\Distortion\IDistortion.cs 82](#_Toc8772600)

[1.24. DSP\Effects\Distortion\SampleRateReduction.cs 82](#_Toc8772601)

[1.25. DSP\Effects\Distortion\SoftClipper.cs 84](#_Toc8772602)

[1.26. DSP\Filters\DCBlocker.cs 85](#_Toc8772603)

[1.27. DSP\Filters\Downsampler.cs 87](#_Toc8772604)

[1.28. DSP\Filters\SvfFilter.cs 97](#_Toc8772605)

[1.29. DSP\Generators\AdsrEnvelope.cs 103](#_Toc8772606)

[1.30. DSP\Generators\SineLFO.cs 110](#_Toc8772607)

[1.31. DSP\Generators\WaveTableOscillators.cs 111](#_Toc8772608)

[1.32. Plugin Parts\AudioProcessor.cs 117](#_Toc8772609)

[1.33. Plugin Parts\MidiProcessor.cs 119](#_Toc8772610)

[1.34. Plugin Parts\ParameterFactory.cs 122](#_Toc8772611)

[1.35. Plugin Parts\Plugin.cs 124](#_Toc8772612)

[1.36. Plugin Parts\PluginCommandStub.cs 127](#_Toc8772613)

[1.37. Plugin Parts\PluginEditor.cs 127](#_Toc8772614)

[1.38. Plugin Parts\PluginPrograms.cs 131](#_Toc8772615)

[1.39. UI\BipolarKnob.cs 132](#_Toc8772616)

[1.40. UI\EditorView.xaml 133](#_Toc8772617)

[1.41. UI\EditorView.xaml.cs 150](#_Toc8772618)

[1.42. UI\Knob.xaml 159](#_Toc8772619)

[1.43. UI\Knob.xaml.cs 161](#_Toc8772620)

[1.44. Converters.cs 168](#_Toc8772621)

[1.45. Utilities.cs 178](#_Toc8772622)

# ТЕКСТ ПРОГРАММЫ

## Components\AudioComponent.cs

## Components\AudioComponentWithParameters.cs

## Components\DelayManager.cs

using Jacobi.Vst.Framework;

using System;

namespace MultimodSynth

{

/// <summary>

/// Компонент плагина, отвечающий за эффект "дилэй".

/// </summary>

class DelayManager : AudioComponentWithParameters

{

/// <summary>

/// Указывает тип эффекта дилэй.

/// </summary>

public enum StereoMode

{

None,

StereoOffset,

VariousTime,

PingPong,

}

/// <summary>

/// Максимальная амплитуда генератора низких частот.

/// </summary>

private const float MaxLfoDepth = 0.05f;

/// <summary>

/// Максимальное время задержки в секундах.

/// </summary>

private const float MaxTime = 1f;

/// <summary>

/// Текущее время задержки (в секундах).

/// </summary>

private float delay;

/// <summary>

/// Текущий коэффициент "чистого" сигнала без эффекта.

/// </summary>

private float dryCoeff = 1;

/// <summary>

/// Текущий коэффициент обратной связи.

/// </summary>

private float feedback;

/// <summary>

/// Текущая амплитуда генератора низких частот

/// (в диапазоне [0, 1]).

/// </summary>

private float lfoDepth;

/// <summary>

/// Максимальное значение времени задержки (в сэмплах).

/// </summary>

private float maxDelay;

/// <summary>

/// Текущий тип эффекта дилэй.

/// </summary>

private StereoMode mode;

/// <summary>

/// Текущее значение коэффициента, показывающего, как сильно

/// отличаются левый и правый канал выходного сигнала (значение в диапазоне [-1, 1]).

/// </summary>

private float stereoAmount;

/// <summary>

/// Текущий коффициент сигнала с применённым к нему эффектом.

/// </summary>

private float wetCoeff;

/// <summary>

/// Знак сигнала с применённым к нему эффектом.

/// </summary>

private int wetSign = 1;

/// <summary>

/// Текущий объект эффекта дилэй.

/// </summary>

private IDelay currentDelay;

/// <summary>

/// Объект класа PingPongDelay.

/// </summary>

private readonly PingPongDelay pingPongDelay;

/// <summary>

/// Объект класса VariousTimeDelay.

/// </summary>

private readonly VariousTimeDelay variousTimeDelay;

/// <summary>

/// Объект класса StereoOffsetDelay.

/// </summary>

private readonly StereoOffsetDelay stereoOffsetDelay;

/// <summary>

/// Генератор низких частот.

/// </summary>

private SineLFO lfo;

/// <summary>

/// Фильтр низких частот, используемый для сглаживания параметра времени задержки.

/// </summary>

private ParameterFilter timeFilter;

/// <summary>

/// Фильтр низких частот, используемый для сглаживания параметра стерео.

/// </summary>

private ParameterFilter stereoAmountFilter;

/// <summary>

/// Фильтр низких частот, используемый для сглаживания параметра количество входного сигнала и выходного.

/// </summary>

private ParameterFilter mixFilter;

/// <summary>

/// Объект, управляющий параметром типа эффекта.

/// </summary>

public VstParameterManager ModeManager { get; set; }

/// <summary>

/// Объект, управляющий параметром времени задержки.

/// </summary>

public VstParameterManager TimeManager { get; set; }

/// <summary>

/// Объект, управляющий параметром коэффициента обратной связи.

/// </summary>

public VstParameterManager FeedbackManager { get; set; }

/// <summary>

/// Объект, управляющий параметром стерео.

/// </summary>

public VstParameterManager StereoAmountManager { get; set; }

/// <summary>

/// Объект, управляющий параметром количества входного и выходного сигналов.

/// </summary>

public VstParameterManager MixManager { get; set; }

/// <summary>

/// Объект, управляющий параметром инвертирования выходного сигнала.

/// </summary>

public VstParameterManager InvertManager { get; set; }

/// <summary>

/// Объект, управляющий параметром частоты генератора низких частот.

/// </summary>

public VstParameterManager LfoRateManager { get; set; }

/// <summary>

/// Объект, управляющий параметром амплитуды генератора низких частот.

/// </summary>

public VstParameterManager LfoDepthManager { get; set; }

/// <summary>

/// Инициализирует новый объект класса DelayManager, принадлежащий переданному плагину

/// и имеющий переданный префикс названия параметров.

/// </summary>

/// <param name="plugin">Плагин, которому принадлежит создаваемый объект.</param>

/// <param name="parameterPrefix">Префикс названия параметров.</param>

public DelayManager(Plugin plugin, string parameterPrefix = "DL")

: base(plugin, parameterPrefix)

{

stereoOffsetDelay = new StereoOffsetDelay();

variousTimeDelay = new VariousTimeDelay();

pingPongDelay = new PingPongDelay();

lfo = new SineLFO();

InitializeParameters();

}

/// <summary>

/// Инициализирует параметры с помощью переданной фабрики параметров.

/// </summary>

/// <param name="factory">Фабрика параметров</param>

protected override void InitializeParameters(ParameterFactory factory)

{

// Параметр типа дилэя.

ModeManager = factory.CreateParameterManager(

name: "MODE",

valueChangedHandler: SetMode);

// Параметр времени задержки.

TimeManager = factory.CreateParameterManager(

name: "TIME",

defaultValue: 0.8f,

valueChangedHandler: SetTimeTarget);

timeFilter = new ParameterFilter(UpdateTime, 0, 100);

TimeManager.PropertyChanged += (sender, e) =>

{

if (e.PropertyName == "ActiveParameter")

currentDelay?.Reset();

};

// Параметр коэффициента обратной связи.

FeedbackManager = factory.CreateParameterManager(

name: "FB",

defaultValue: 0.5f,

valueChangedHandler: SetFeedback);

// Параметр стерео-эффекта.

StereoAmountManager = factory.CreateParameterManager(

name: "STER",

defaultValue: 0.5f,

valueChangedHandler: SetStereoAmountTarget);

stereoAmountFilter = new ParameterFilter(UpdateStereoAmount, 0, 100);

// Параметр количества выходного и входного сигналов.

MixManager = factory.CreateParameterManager(

name: "MIX",

defaultValue: 0.5f,

valueChangedHandler: SetMixTarget);

mixFilter = new ParameterFilter(UpdateMix, 0);

// Параметр инвертирования выходного сигнала.

InvertManager = factory.CreateParameterManager(

name: "INV",

valueChangedHandler: SetInvert);

// Параметр частоты генератора низких частот.

LfoRateManager = factory.CreateParameterManager(

name: "RATE",

valueChangedHandler: SetLfoRate);

// Параметр амплитуды генератора низких частот.

LfoDepthManager = factory.CreateParameterManager(

name: "DEPTH",

valueChangedHandler: SetLfoDepth);

}

/// <summary>

/// Обработчик изменения типа дилэя.

/// </summary>

/// <param name="value">Нормированное новое значение параметра.</param>

private void SetMode(float value)

{

var newMode = Converters.ToDelayMode(value);

if (newMode != mode)

{

mode = newMode;

switch (mode)

{

case StereoMode.StereoOffset:

ChangeDelay(stereoOffsetDelay);

break;

case StereoMode.VariousTime:

ChangeDelay(variousTimeDelay);

break;

case StereoMode.PingPong:

ChangeDelay(pingPongDelay);

break;

}

}

}

/// <summary>

/// Изменяет текущий объект дилэя на новый.

/// </summary>

/// <param name="newDelay">Новый объект дилэя.</param>

private void ChangeDelay(IDelay newDelay)

{

currentDelay = newDelay;

currentDelay?.Reset();

currentDelay?.SetFeedback(feedback);

currentDelay?.SetStereo(stereoAmount);

}

/// <summary>

/// Обработчик изменения времени задержки.

/// </summary>

/// <param name="target">Нормированное новое значение параметра.</param>

private void SetTimeTarget(float target)

{

timeFilter.SetTarget((float)Converters.ToDelayTime(target));

}

/// <summary>

/// Обработчик изменения "сглаженного" значения времени задержки.

/// </summary>

/// <param name="value">Новое значение времени задержки.</param>

private void UpdateTime(float value)

{

delay = value \* SampleRate;

}

/// <summary>

/// Обработчик изменения коэффициента обратной связи.

/// </summary>

/// <param name="value">Нормированное новое значение параметра.</param>

private void SetFeedback(float value)

{

feedback = value;

currentDelay?.SetFeedback(feedback);

}

/// <summary>

/// Обработчик изменения параметра стерео-эффекта.

/// </summary>

/// <param name="target">Нормированное новое значение параметра.</param>

private void SetStereoAmountTarget(float target)

{

stereoAmountFilter.SetTarget((float)Converters.ToStereoAmount(target));

}

/// <summary>

/// Обработчик изменения "сглаженного" значения параметра стерео-эффекта.

/// </summary>

/// <param name="value">Новое значение стерео-эффекта.</param>

private void UpdateStereoAmount(float value)

{

stereoAmount = value;

currentDelay?.SetStereo(stereoAmount);

}

/// <summary>

/// Обработчик изменения параметра количества выходного и входного сигнала.

/// </summary>

/// <param name="target">Нормированное новое значение параметра.</param>

private void SetMixTarget(float target)

{

mixFilter.SetTarget(target);

}

/// <summary>

/// Обработчик изменения "сглаженного" значения количества выходного и входного сигнала.

/// </summary>

/// <param name="value">Новое значение.</param>

private void UpdateMix(float value)

{

wetCoeff = value;

dryCoeff = 1 - value;

}

/// <summary>

/// Обработчик изменения параметра инвертирования выходного сигнала.

/// </summary>

/// <param name="target">Нормированное новое значение параметра.</param>

private void SetInvert(float value)

{

if (value < 0.5)

wetSign = 1;

else

wetSign = -1;

}

/// <summary>

/// Обработчик изменения частоты генератора низких частот.

/// </summary>

/// <param name="target">Нормированное новое значение параметра.</param>

private void SetLfoRate(float value)

{

lfo.SetFrequency((float)Converters.ToDelayLfoRate(value));

}

/// <summary>

/// Обработчик изменения амплитуды генератора низких частот.

/// </summary>

/// <param name="target">Нормированное новое значение параметра.</param>

private void SetLfoDepth(float value)

{

lfoDepth = value;

}

/// <summary>

/// Обработка новых входных данных.

/// </summary>

/// <param name="inputL">Левый канал входного сигнала</param>

/// <param name="inputR">Правый канал входного сигнала</param>

/// <param name="outputL">Левый канал выходного сигнала.</param>

/// <param name="outputR">Правый канал выходного сигнала.</param>

public void Process(float inputL, float inputR, out float outputL, out float outputR)

{

// Сглаживание значений параметров.

timeFilter.Process();

mixFilter.Process();

stereoAmountFilter.Process();

if (mode == StereoMode.None)

{

outputL = inputL;

outputR = inputR;

}

else

{

var lfoCoeff = 1 + MaxLfoDepth \* lfoDepth \* lfo.Process();

currentDelay.SetDelay(Math.Min(maxDelay, delay \* lfoCoeff));

currentDelay.Process(inputL, inputR, out var wetL, out var wetR);

outputL = dryCoeff \* inputL + wetSign \* wetCoeff \* wetL;

outputR = dryCoeff \* inputL + wetSign \* wetCoeff \* wetR;

}

}

/// <summary>

/// Обработчик изменения частоты дискретизации.

/// </summary>

/// <param name="newSampleRate">Новая частота дискретизации.</param>

protected override void OnSampleRateChanged(float newSampleRate)

{

timeFilter.SampleRate = newSampleRate;

stereoAmountFilter.SampleRate = newSampleRate;

mixFilter.SampleRate = newSampleRate;

stereoOffsetDelay.SampleRate = newSampleRate;

variousTimeDelay.SampleRate = newSampleRate;

pingPongDelay.SampleRate = newSampleRate;

lfo.SampleRate = newSampleRate;

maxDelay = MaxTime \* newSampleRate;

}

}

}

## Components\DistortionManager.cs

using Jacobi.Vst.Framework;

namespace MultimodSynth

{

/// <summary>

/// Компонент плагина, отвечающий за эффект "дисторшн".

/// </summary>

class DistortionManager : AudioComponentWithParameters

{

/// <summary>

/// Указывает тип эффекта дисторшн.

/// </summary>

public enum DistortionMode

{

None,

AbsClipping,

SoftClipping,

CubicClipping,

BitCrush,

SampleRateReduction,

}

/// <summary>

/// Текущая сила эффекта дисторшн.

/// </summary>

private float amount;

/// <summary>

/// Текущий уровень громкости входного сигнала.

/// </summary>

private float amp = 1;

/// <summary>

/// Текущее значение постоянного амплитудного смещения.

/// </summary>

private float dcOffset;

/// <summary>

/// Текущий коэффициент "чистого" сигнала без эффекта.

/// </summary>

private float dryCoeff = 1;

/// <summary>

/// Текущий тип эффекта дисторшн.

/// </summary>

private DistortionMode mode;

/// <summary>

/// Текущий коэффициент сигнала с применённым к нему эффектом.

/// </summary>

private float wetCoeff;

/// <summary>

/// Текущий объект эффекта дисторшн.

/// </summary>

private IDistortion currentDistortion;

/// <summary>

/// Объект класса SoftClipper.

/// </summary>

private readonly SoftClipper softClipper;

/// <summary>

/// Объект класса AbsClipper.

/// </summary>

private readonly AbsClipper absClipper;

/// <summary>

/// Объект класса CubicClipper.

/// </summary>

private readonly CubicClipper cubicClipper;

/// <summary>

/// Объект класса BitCrusher.

/// </summary>

private readonly BitCrusher bitCrusher;

/// <summary>

/// Объект класса SampleRateReductor.

/// </summary>

private readonly SampleRateReductor sampleRateReductor;

/// <summary>

/// Фильтр высоких частот для устранения постоянного амплитудного смещения.

/// </summary>

private DCBlocker dcBlocker;

/// <summary>

/// Фильтр низких частот, применяющийся к входному сигналу.

/// </summary>

private SvfFilter lowPass;

/// <summary>

/// Фильтр низких частот, используемый для сглаживания параметра

/// уровня громкости входного сигнала.

/// </summary>

private ParameterFilter ampFilter;

/// <summary>

/// Фильтр низких частот, используемый для сглаживания параметра количества входного и выходного сигналов.

/// </summary>

private ParameterFilter mixFilter;

/// <summary>

/// Фильтр низких частот, используемый для сглаживания параметра постоянного амплитудного сдвига.

/// </summary>

private ParameterFilter asymmetryFilter;

/// <summary>

/// Объект, управляющий параметром типа эффекта.

/// </summary>

public VstParameterManager ModeManager { get; private set; }

/// <summary>

/// Объект, управляющий параметром силы эффекта.

/// </summary>

public VstParameterManager AmountManager { get; private set; }

/// <summary>

/// Объект, управляющий параметром постоянного амплитудного сдвига.

/// </summary>

public VstParameterManager AsymmetryManager { get; private set; }

/// <summary>

/// Объект, управляющий параметром громкости входного сигнала.

/// </summary>

public VstParameterManager AmpManager { get; private set; }

/// <summary>

/// Объект, управляющий параметром частоты среза фильтра низких частот.

/// </summary>

public VstParameterManager LowPassCutoffManager { get; private set; }

/// <summary>

/// Объект, управляющий параметром количества выходного и входного сигналов.

/// </summary>

public VstParameterManager MixManager { get; private set; }

/// <summary>

/// Инициализирует новый объект класса DistortionManager, принадлежащий переданному плагину

/// и имеющий переданный префикс названия параметров.

/// </summary>

/// <param name="plugin">Плагин, которому принадлежит создаваемый объект.</param>

/// <param name="parameterPrefix">Префикс названия параметров.</param>

public DistortionManager(

Plugin plugin,

string parameterPrefix = "D")

: base(plugin, parameterPrefix)

{

dcBlocker = new DCBlocker(10);

lowPass = new SvfFilter(type: SvfFilter.FilterType.Low);

absClipper = new AbsClipper();

softClipper = new SoftClipper();

cubicClipper = new CubicClipper();

bitCrusher = new BitCrusher();

sampleRateReductor = new SampleRateReductor();

InitializeParameters();

}

/// <summary>

/// Инициализирует параметры с помощью переданной фабрики параметров.

/// </summary>

/// <param name="factory">Фабрика параметров</param>

protected override void InitializeParameters(ParameterFactory factory)

{

// Параметр типа дисторшна.

ModeManager = factory.CreateParameterManager(

name: "TYPE",

valueChangedHandler: SetMode);

// Параметр силы эффекта.

AmountManager = factory.CreateParameterManager(

name: "AMNT",

defaultValue: 0.5f,

valueChangedHandler: SetAmount);

// Параметр постоянного амплитудного сдвига.

AsymmetryManager = factory.CreateParameterManager(

name: "ASYM",

defaultValue: 0.5f,

valueChangedHandler: SetAsymmetryTarget);

asymmetryFilter = new ParameterFilter(UpdateAsymmetry, 0);

// Параметр уровня громкости входного сигнала.

AmpManager = factory.CreateParameterManager(

name: "AMP",

defaultValue: 0.25f,

valueChangedHandler: SetAmpTarget);

уровня

изгиба

SetDecayTime(decayTime);

SetReleaseTime(releaseTime);

}

}

}

## Components\EnvelopesManager.cs

using Jacobi.Vst.Framework;

using System.Collections.Generic;

foreach (var envelope in envelopes)

envelope.SetSustainLevel(value);

}

/// <summary>

/// Обработчик изменения длительности стадии затухания.

/// </summary>

/// <param name="value">Нормированное новое значение параметра.</param>

private void SetReleaseTime(float value)

{

}

## Components\FiltersManager.cs

using Jacobi.Vst.Framework;

using System.Collections.Generic;

namespace MultimodSynth

{

/// <summary>

/// Компонент плагина, управляющий одним фильтром многих голосов.

/// </summary>

class FiltersManager : AudioComponentWithParameters

{

/// <summary>

/// Текущий тип фильтра.

/// </summary>

private SvfFilter.FilterType filterType;

/// <summary>

/// Текущая "ширина" фильтра.

/// </summary>

private float curve;

/// <summary>

/// Текущий множитель частоты среза фильтра.

/// </summary>

private float cutoffMultiplier;

/// <summary>

/// Текущий коэффициент отслеживания частоты играющей ноты.

/// </summary>

private float trackingCoeff;

/// <summary>

/// Список фильтров, связанных с этим менеджером фильтров.

/// </summary>

private List<Filter> filters;

/// <summary>

/// Фильтр низких частот, используемый для сглаживания параметра множителя частоты среза фильтра.

/// </summary>

private ParameterFilter cutoffMultiplierFilter;

/// <summary>

/// Объект, управляющий параметром типа фильтра.

/// </summary>

public VstParameterManager FilterTypeManager { get; private set; }

/// <summary>

/// Объект, управляющий параметром множителя частоты среза фильтра.

/// </summary>

public VstParameterManager CutoffManager { get; private set; }

/// <summary>

/// Объект, управляющий параметром коффициента отслеживания частоты играющей ноты.

/// </summary>

public VstParameterManager TrackingCoeffManager { get; private set; }

/// <summary>

/// Объект, управляющий параметром "ширины" фильтра.

/// </summary>

public VstParameterManager CurveManager { get; private set; }

/// <summary>

/// Инициализирует новый объект класса FiltersManager, принадлежащий переданному плагину

/// и имеющий переданный префикс названия параметров.

/// </summary>

/// <param name="plugin">Плагин, которому принадлежит создаваемый объект.</param>

/// <param name="parameterPrefix">Префикс названия параметров.</param>

public FiltersManager(

Plugin plugin,

string parameterPrefix)

: base(plugin, parameterPrefix)

{

filters = new List<Filter>();

InitializeParameters();

}

/// <summary>

/// Инициализирует параметры с помощью переданной фабрики параметров.

/// </summary>

/// <param name="factory">Фабрика параметров</param>

protected override void InitializeParameters(ParameterFactory factory)

{

// Параметр типа фильтра.

FilterTypeManager = factory.CreateParameterManager(

name: "TYPE",

valueChangedHandler: SetFilterType);

// Параметр множителя частоты среза фильтра.

CutoffManager = factory.CreateParameterManager(

name: "CUT",

defaultValue: 1,

valueChangedHandler: SetCutoffMultiplierTarget);

cutoffMultiplierFilter = new ParameterFilter(UpdateCutoffMultiplier, 1, 10);

// Параметр коэффициента отслеживания частоты играющей ноты.

TrackingCoeffManager = factory.CreateParameterManager(

name: "TRK",

defaultValue: 1,

valueChangedHandler: SetTrackingCoeff);

// Параметр "ширины" фильтра.

CurveManager = factory.CreateParameterManager(

name: "CRV",

defaultValue: 0.5f,

valueChangedHandler: SetCurve);

}

/// <summary>

/// Обработчик изменения длительности типа фильтра.

/// </summary>

/// <param name="value">Нормированное новое значение параметра.</param>

private void SetFilterType(float value)

{

var newType = Converters.ToFilterType(value);

if (filterType != newType)

{

filterType = newType;

foreach (var filter in filters)

filter.SetFilterType(filterType);

}

}

/// <summary>

/// Обработчик изменения множителя частоты среза фильтра.

/// </summary>

/// <param name="value">Нормированное новое значение параметра.</param>

private void SetCutoffMultiplierTarget(float value)

{

var mult = (float)Converters.ToFilterCutoffMultiplier(value);

cutoffMultiplierFilter.SetTarget(mult);

}

/// <summary>

/// Обработчик изменения "сглаженного" множителя частоты среза фильтра.

/// </summary>

/// <param name="value">Новое значение множителя частоты среза фильтра.</param>

private void UpdateCutoffMultiplier(float value)

{

cutoffMultiplier = value;

foreach (var filter in filters)

filter.SetCutoffMultiplier(cutoffMultiplier);

}

/// <summary>

/// Обработчик изменения коэффициента отслеживания частоты играющей ноты.

/// </summary>

/// <param name="value">Нормированное новое значение параметра.</param>

private void SetTrackingCoeff(float value)

{

trackingCoeff = value;

foreach (var filter in filters)

filter.SetTrackingCoeff(value);

}

/// <summary>

/// Обработчик изменения "ширины" фильтра.

/// </summary>

/// <param name="value">Нормированное новое значение параметра.</param>

private void SetCurve(float value)

{

curve = value;

foreach (var filter in filters)

filter.SetCurve(curve);

}

/// <summary>

/// Возвращает новый объект фильтра, связанный с этим объектом.

/// </summary>

/// <returns>Новый фильтр.</returns>

public Filter CreateNewFilter()

{

var filter = new Filter();

filter.SetCutoffMultiplier(cutoffMultiplier);

filter.SetCurve(curve);

filter.SetTrackingCoeff(trackingCoeff);

filter.SetFilterType(filterType);

filters.Add(filter);

return filter;

}

/// <summary>

/// Метод, выполняющий обновление всех сглаживающих фильтров.

/// </summary>

public void Process()

{

cutoffMultiplierFilter.Process();

}

}

}

## Components\Oscillator.cs

using System;

namespace MultimodSynth

{

/// <summary>

/// Компонент голоса плагина, представляющий осциллятор.

/// </summary>

class Oscillator : AudioComponent

{

/// <summary>

/// Частота играемой ноты.

/// </summary>

private float noteFrequency;

/// <summary>

/// Играемая частота.

/// </summary>

private float frequency;

/// <summary>

/// Изменение фазы за 1 сэмпл.

/// </summary>

private float phaseIncrement;

/// <summary>

/// Множитель частоты.

/// </summary>

private float pitchMultiplier;

/// <summary>

/// Текущая фаза осциллятора.

/// </summary>

private float phasor;

/// <summary>

/// Ссылка на используемый объект класса WaveTable.

/// </summary>

private WaveTableOscillator waveTable;

/// <summary>

/// Устанавливает новый объект класса WaveTable.

/// </summary>

/// <param name="waveTable"></param>

public void SetWaveTable(WaveTableOscillator waveTable)

{

this.waveTable = waveTable;

this.waveTable.SetPhaseIncrement(phaseIncrement);

}

/// <summary>

/// Устанавливает новое значение частоты играемой ноты.

/// </summary>

/// <param name="value"></param>

public void SetNoteFrequency(float value)

{

noteFrequency = value;

UpdateCoefficients();

}

/// <summary>

/// Устанавливает новое значение множителя частоты.

/// </summary>

/// <param name="value"></param>

public void SetPitchMultiplier(float value)

{

pitchMultiplier = value;

UpdateCoefficients();

}

/// <summary>

/// Обновляет все коэффициенты.

/// </summary>

private void UpdateCoefficients()

{

frequency = noteFrequency \* pitchMultiplier;

phaseIncrement = frequency / SampleRate;

waveTable?.SetPhaseIncrement(phaseIncrement);

}

/// <summary>

/// Обработка новых входных данных.

/// </summary>

/// <param name="phaseModulation">Фазовая модуляция.</param>

/// <returns>Выходной сигнал.</returns>

public float Process(float phaseModulation = 0)

{

// Расчёт текущей фазы.

var phase = phasor + phaseModulation;

phase -= (float)Math.Floor(phase);

var waveTable = this.waveTable;

float result;

if (waveTable == null)

result = 0;

else

result = waveTable.Process(phase);

// Прибавление инкремента фазы текущей частоты.

phasor += phaseIncrement;

if (phasor >= 1)

phasor -= 1;

return result;

}

/// <summary>

/// Сбрасывает текущее состояние осциллятора.

/// </summary>

public void Reset() => phasor = 0;

/// <summary>

/// Обработчик изменения частоты дискретизации.

/// </summary>

/// <param name="newSampleRate">Новая частота дискретизации.</param>

protected override void OnSampleRateChanged(float newSampleRate)

{

phaseIncrement = frequency / newSampleRate;

}

}

}

## Components\OscillatorsManager.cs

using Jacobi.Vst.Framework;

using System;

using System.Collections.Generic;

namespace MultimodSynth

{

/// <summary>

/// Компонент плагина, управляющий одним осциллятором многих голосов.

/// </summary>

class OscillatorsManager : AudioComponentWithParameters

{

/// <summary>

/// Коэффициент высоты играемой ноты (изменяется параметром подстройки в полутонах).

/// </summary>

private float pitchFine;

/// <summary>

/// Другой коэффициент высоты играемой ноты (изменяется параметром подстройки в центах).

/// </summary>

private float pitchSemi;

/// <summary>

/// Общий коэффициент высоты играемой ноты.

/// </summary>

private float pitchMultiplier;

/// <summary>

/// Ссылка на текущую используемую таблицу сэмплов.

/// </summary>

private WaveTableOscillator waveTable = Utilities.WaveTables[0];

/// <summary>

/// Список осцилляторов, связанных с этим менеджером осцилляторов.

/// </summary>

private List<Oscillator> oscillators;

/// <summary>

/// Фильтр низких частот, используемый для сглаживания изменений частоты играемой ноты.

/// </summary>

private ParameterFilter pitchMultiplierFilter;

/// <summary>

/// Объект, управляющий параметром подстройки частоты (в полутонах).

/// </summary>

public VstParameterManager PitchSemiManager { get; private set; }

/// <summary>

/// Объект, управляющий параметром подстройки частоты (в центах).

/// </summary>

public VstParameterManager PitchFineManager { get; private set; }

/// <summary>

/// Объект, управляющий параметром текущей используемой таблицы семплов.

/// </summary>

public VstParameterManager WaveTableManager { get; private set; }

/// <summary>

/// Инициализирует новый объект класса OscillatorsManager, принадлежащий переданному плагину

/// и имеющий переданный префикс названия параметров.

/// </summary>

/// <param name="plugin">Плагин, которому принадлежит создаваемый объект.</param>

/// <param name="parameterPrefix">Префикс названия параметров.</param>

public OscillatorsManager(

Plugin plugin,

string parameterPrefix) :

base(plugin, parameterPrefix)

{

oscillators = new List<Oscillator>();

InitializeParameters();

}

/// <summary>

/// Инициализирует параметры с помощью переданной фабрики параметров.

/// </summary>

/// <param name="factory">Фабрика параметров</param>

protected override void InitializeParameters(ParameterFactory factory)

{

PitchSemiManager = factory.CreateParameterManager(

name: "SEMI",

defaultValue: 0.5f,

valueChangedHandler: SetPitchSemi);

PitchFineManager = factory.CreateParameterManager(

name: "FINE",

defaultValue: 0.5f,

valueChangedHandler: SetPitchFine);

pitchMultiplierFilter = new ParameterFilter(UpdatePitchMultiplier, 0);

WaveTableManager = factory.CreateParameterManager(

name: "TYPE",

defaultValue: 0,

valueChangedHandler: SetWaveTable);

}

/// <summary>

/// Обработчик изменения подстройки частоты в полутонах.

/// </summary>

/// <param name="value">Нормированное новое значение параметра.</param>

private void SetPitchSemi(float value)

{

pitchSemi = (float)Math.Pow(2, Converters.ToSemitones(value) / 12.0);

var target = pitchSemi \* pitchFine;

pitchMultiplierFilter.SetTarget(target);

}

/// <summary>

/// Обработчик изменения подстройки частоты (в центах).

/// </summary>

/// <param name="value">Нормированное новое значение параметра.</param>

private void SetPitchFine(float value)

{

pitchFine = (float)Math.Pow(2, Converters.ToCents(value) / 1200.0);

var target = pitchSemi \* pitchFine;

pitchMultiplierFilter.SetTarget(target);

}

/// <summary>

/// Обработчик изменения "сглаженного" значения общей подстройки частоты.

/// </summary>

/// <param name="value">Новое значение подстройки частоты.</param>

private void UpdatePitchMultiplier(float value)

{

pitchMultiplier = value;

foreach (var oscillator in oscillators)

oscillator.SetPitchMultiplier(pitchMultiplier);

}

/// <summary>

/// Обработчик изменения текущей таблицы сэмплов.

/// </summary>

/// <param name="value">Нормированное новое значение параметра.</param>

private void SetWaveTable(float value)

{

var newWaveTable = Converters.ToWaveTable(value);

if (waveTable != newWaveTable)

{

waveTable = newWaveTable;

foreach (var oscillator in oscillators)

oscillator.SetWaveTable(waveTable.Clone());

}

}

/// <summary>

/// Возвращает новый объект осциллятора, связанный с этим объектом.

/// </summary>

/// <returns>Новая огибающая.</returns>

public Oscillator CreateNewOscillator()

{

var res = new Oscillator();

res.SetPitchMultiplier(pitchMultiplier);

res.SetWaveTable(waveTable.Clone());

oscillators.Add(res);

return res;

}

/// <summary>

/// Метод, выполняющий обновление всех сглаживающих фильтров.

/// </summary>

public void Process()

{

pitchMultiplierFilter.Process();

}

}

}

## Components\ParameterFilter.cs

using System;

namespace MultimodSynth

{

/// <summary>

/// Фильтр низких частот, используемый для сглаживания изменения параметров.

/// </summary>

/// <seealso cref="http://www.musicdsp.org/en/latest/Filters/257-1-pole-lpf-for-smooth-parameter-changes.html"/>

class ParameterFilter : AudioComponent

{

/// <summary>

/// Частота дискретизации, для которой вычислен коэффициент DefaultExp.

/// </summary>

private const double BaseSampleRate = 44100;

/// <summary>

/// Коэффициент для генерации коэффициента скорости отклика фильтра.

/// </summary>

private const double DefaultExp = 0.0015995606308184566;

/// <summary>

/// Коэффициенты фильтра.

/// </summary>

private float a, b;

/// <summary>

/// Текущее сглаженное значение.

/// </summary>

private float value;

/// <summary>

/// Целевое значение фильтра.

/// </summary>

private float target;

/// <summary>

/// Коэффициент скорости отклика.

/// </summary>

private float responseTimeCoefficient;

/// <summary>

/// Показывает, активен ли фильтр в данный момент.

/// </summary>

private bool isActive;

/// <summary>

/// Действие, выполняемое при изменении текущего значения.

/// </summary>

private Action<float> valueChangedAction;

/// <summary>

/// Инициализирует новый объект класса ParameterFilter, выполняющий заданное действие

/// при каждом изменении сглаженного значения.

/// </summary>

/// <param name="valueChangedAction">Действие, выполняемое при изменении сглаженного значения.</param>

/// <param name="initialValue">Начальное значение фильтра.</param>

/// <param name="responseTimeCoefficient">Коэффициент скорости отклика фильтра</param>

public ParameterFilter(

Action<float> valueChangedAction,

float initialValue = 0,

float responseTimeCoefficient = 1)

{

a = 0.99f;

b = 1f - a;

value = initialValue;

this.valueChangedAction = valueChangedAction;

this.responseTimeCoefficient = responseTimeCoefficient;

}

/// <summary>

/// Устанавливает целевое значение фильтра.

/// </summary>

/// <param name="value"></param>

public void SetTarget(float value)

{

target = value;

isActive = true;

}

/// <summary>

/// Устанавливает коэффициент времени отклика фильтра.

/// </summary>

/// <param name="value"></param>

public void SetResponseTimeCoefficient(float value)

{

responseTimeCoefficient = value;

}

/// <summary>

/// Выполнение очередного шаг сглаживания.

/// </summary>

public void Process()

{

if (isActive)

{

var newValue = (target \* b) + (value \* a);

valueChangedAction(newValue);

if (value != newValue)

value = newValue;

else

isActive = false;

}

}

/// <summary>

/// Обработчик изменения частоты дискретизации.

/// </summary>

/// <param name="newSampleRate">Новая частота дискретизации.</param>

protected override void OnSampleRateChanged(float newSampleRate)

{

var coeff = BaseSampleRate / newSampleRate / responseTimeCoefficient;

a = (float)Math.Exp(-2 \* Math.PI \* DefaultExp \* coeff);

b = 1 - a;

}

}

}

## Components\Routing.cs

using Jacobi.Vst.Framework;

namespace MultimodSynth

{

/// <summary>

/// Компонент плагина, предстающий собой всю цепочку создания обработки звука.

/// </summary>

class Routing : AudioComponentWithParameters

{

/// <summary>

/// Буфер для хранения последних n семплов (для оверсэмплинга).

/// </summary>

private double[] samplesForOversampling = new double[8];

/// <summary>

/// Уровень громкости выходного сигнала.

/// </summary>

private float masterVolume;

/// <summary>

/// Фильтр низких частот, используемый для сглаживания параметра

/// уровня громкости выходного сигнала.

/// </summary>

private ParameterFilter masterVolumeFilter;

/// <summary>

/// Менеджер всех голосов.

/// </summary>

public VoicesManager VoicesManager { get; private set; }

/// <summary>

/// Фильтр для снижения частоты дискретизации.

/// </summary>

public Downsampler Downsampler { get; private set; }

/// <summary>

/// Эффект дисторшн.

/// </summary>

public DistortionManager DistortionManager { get; private set; }

/// <summary>

/// Эффект дилэй.

/// </summary>

public DelayManager DelayManager { get; private set; }

/// <summary>

/// Объект, управляющий параметром коэффициента повышения частоты дискретизации.

/// </summary>

public VstParameterManager OversamplingOrderManager { get; private set; }

/// <summary>

/// Объект, управляющий параметром уровня громкости выходного сигнала.

/// </summary>

public VstParameterManager MasterVolumeManager { get; private set; }

/// <summary>

/// Инициализирует новый объект класса Routing, принадлежащий переданному плагину

/// и имеющий переданный префикс названия параметров.

/// </summary>

/// <param name="plugin">Плагин, которому принадлежит создаваемый объект.</param>

/// <param name="parameterPrefix">Префикс названия параметров.</param>

public Routing(Plugin plugin, string parameterPrefix = "M\_")

: base(plugin, parameterPrefix)

{

VoicesManager = new VoicesManager(plugin, "M\_");

Downsampler = new Downsampler();

DistortionManager = new DistortionManager(plugin, "DS\_");

DelayManager = new DelayManager(plugin, "DL\_");

plugin.MidiProcessor.NoteOn += MidiProcessor\_NoteOn;

plugin.MidiProcessor.NoteOff += MidiProcessor\_NoteOff;

InitializeParameters();

}

/// <summary>

/// Инициализирует параметры с помощью переданной фабрики параметров.

/// </summary>

/// <param name="factory">Фабрика параметров</param>

protected override void InitializeParameters(ParameterFactory factory)

{

OversamplingOrderManager = factory.CreateParameterManager(

name: "OVSMP",

valueChangedHandler: SetOversamplingOrder);

MasterVolumeManager = factory.CreateParameterManager(

name: "VOL",

defaultValue: 0.5f,

valueChangedHandler: x => masterVolumeFilter.SetTarget(x));

masterVolumeFilter = new ParameterFilter(UpdateMasterVolume, 1);

}

/// <summary>

/// Обработчик изменения коэффициента повышения частоты дискретизации.

/// </summary>

/// <param name="value">Нормированное новое значение параметра.</param>

private void SetOversamplingOrder(float value)

{

int newOrder = Converters.ToOversamplingOrder(value);

if (newOrder != Downsampler.Order)

{

Downsampler.Order = newOrder;

UpdateSampleRates();

}

}

/// <summary>

/// Обработчик изменения "сглаженного" значения уровня громкости выходного сигнала.

/// </summary>

/// <param name="value">Новое значение уровня громкости.</param>

private void UpdateMasterVolume(float value) => masterVolume = value;

/// <summary>

/// Обработчик события нажатия на клавишу.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void MidiProcessor\_NoteOn(object sender, MidiNoteEventArgs e)

{

VoicesManager.PlayNote(e.Note);

}

/// <summary>

/// Обработчик события отпускания клавиши.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void MidiProcessor\_NoteOff(object sender, MidiNoteEventArgs e)

{

VoicesManager.ReleaseNote(e.Note);

}

/// <summary>

/// Генерация новых выходных данных.

/// </summary>

/// <param name="left">Левый канал выходного сигнала.</param>

/// <param name="right">Правый канал выходного сигнала.</param>

public void Process(out float left, out float right)

{

// Сглаживание значений параметров.

masterVolumeFilter.Process();

for (int i = 0; i < Downsampler.Order; ++i)

{

var voicesOutput = VoicesManager.Process();

var saturationOutput = DistortionManager.Process(voicesOutput);

samplesForOversampling[i] = saturationOutput;

}

var output = (float)Downsampler.Process(samplesForOversampling);

DelayManager.Process(output, output, out left, out right);

left \*= masterVolume;

right \*= masterVolume;

}

/// <summary>

/// Обработчик изменения частоты дискретизации.

/// </summary>

/// <param name="newSampleRate">Новая частота дискретизации.</param>

protected override void OnSampleRateChanged(float newSampleRate)

{

UpdateSampleRates();

DelayManager.SampleRate = newSampleRate;

}

/// <summary>

/// Обновляет частоты дискретизации компонентов, подверженных оверсэмплингу.

/// </summary>

private void UpdateSampleRates()

{

var scaledSampleRate = SampleRate \* Downsampler.Order;

VoicesManager.SampleRate = scaledSampleRate;

DistortionManager.SampleRate = scaledSampleRate;

}

}

}

## Components\Voice.cs

namespace MultimodSynth

{

/// <summary>

/// Компонент плагина, представляющий собой один голос.

/// </summary>

class Voice : AudioComponent

{

/// <summary>

/// Указывает тип взаимодействия двух осцилляторов.

/// </summary>

public enum ModulationType

{

None,

PhaseModulationA,

PhaseModulationB,

AmplitudeModulationA,

AmplitudeModulationB,

}

/// <summary>

/// Осциллятор A.

/// </summary>

private Oscillator oscA;

/// <summary>

/// Осциллятор B.

/// </summary>

private Oscillator oscB;

/// <summary>

/// Фильтр.

/// </summary>

private Filter filter;

/// <summary>

/// Огибающая громкости осциллятора A.

/// </summary>

private Envelope envA;

/// <summary>

/// Огибающая громкости осциллятора B.

/// </summary>

private Envelope envB;

/// <summary>

/// Огибающая частоты среза фильтра.

/// </summary>

private Envelope envFilter;

/// <summary>

/// Сила нажатия текущей играемой ноты.

/// </summary>

private float noteVelocity;

/// <summary>

/// Максимальная "сила" частотной модуляции.

/// </summary>

private float fmAmountMultiplier;

/// <summary>

/// Указывает, активен ли данный голос.

/// </summary>

public bool IsActive { get; private set; }

/// <summary>

/// Текущая играемая нота.

/// </summary>

public MidiNote Note { get; private set; }

/// <summary>

/// Текущий тип модуляции.

/// </summary>

public ModulationType Modulation { get; set; }

/// <summary>

/// Инициализирует новый объект класса Voice, имеющий переданные компоненты.

/// </summary>

/// <param name="plugin">Плагин, которому принадлежит создаваемый объект.</param>

/// <param name="oscA">Осциллятор А.</param>

/// <param name="oscB">Осциллятор B.</param>

/// <param name="filter">Фильтр.</param>

/// <param name="envA">Огибающая уровня осциллятора A.</param>

/// <param name="envB">Огибающая уровня оциллятора B.</param>

/// <param name="envFilter">Огибающая частоты среза фильтра.</param>

public Voice(

Plugin plugin,

Oscillator oscA,

Oscillator oscB,

Filter filter,

Envelope envA,

Envelope envB,

Envelope envFilter)

{

this.oscA = oscA;

this.oscB = oscB;

this.filter = filter;

this.envA = envA;

this.envB = envB;

this.envFilter = envFilter;

}

/// <summary>

/// Играет переданную ноту.

/// </summary>

/// <param name="note">Нота, которую необходимо проиграть.</param>

public void PlayNote(MidiNote note)

{

Note = note;

noteVelocity = note.Velocity / 128f;

var noteFrequency = (float)Utilities.MidiNoteToFrequency(note.NoteNo);

oscA.Reset();

oscB.Reset();

filter.Reset();

oscA.SetNoteFrequency(noteFrequency);

oscB.SetNoteFrequency(noteFrequency);

filter.SetNoteFrequency(noteFrequency);

envA.TriggerAttack();

envB.TriggerAttack();

envFilter.TriggerAttack();

IsActive = true;

}

/// <summary>

/// Прекращает проигрывание ноты.

/// </summary>

public void TriggerRelease()

{

envA.TriggerRelease();

envB.TriggerRelease();

envFilter.TriggerRelease();

}

/// <summary>

/// Генерация новых выходных данных.

/// </summary>

/// <returns>Выходной сигнал.</returns>

public float Process()

{

if (!IsActive)

return 0;

float oscMix = 0; // Здесь хранится результат

switch (Modulation)

{

case ModulationType.None:

if (envA.IsActive) // Если первый осциллятор активен, то

{

oscMix += envA.Process() \* oscA.Process(); // Прибавляем сигнал первого осциллятора

if (envB.IsActive) // Если второй осциллятор активен

oscMix += envB.Process() \* oscB.Process(); // Прибавляем сигнал второго осциллятора

}

else

{

if (envB.IsActive) // Если второй осциллятор активен, то

oscMix = envB.Process() \* oscB.Process(); // Прибавляем сигнал второго осциллятора

else

{

// Оба осциллятора неактивны, этот голос прекратил производить сигнал

IsActive = false;

return 0;

}

}

break;

case ModulationType.AmplitudeModulationA:

if (envA.IsActive) // Если несущий осциллятор активен, то

{

float mod = 0;

if (envB.IsActive) // Если модулирующий осциллятор активен, то

mod = oscB.Process() \* envB.Process();

oscMix = oscA.Process() \* envA.Process() \* (1 + mod);

}

else

{

// Несущий осциллятор не активен, этот голос прекратил производить сигнал.

IsActive = false;

return 0;

}

break;

case ModulationType.AmplitudeModulationB:

if (envB.IsActive)

{

float mod = 0;

if (envA.IsActive)

mod = oscA.Process() \* envA.Process();

oscMix = oscB.Process() \* envB.Process() \* (1 + mod);

}

else

{

IsActive = false;

return 0;

}

break;

case ModulationType.PhaseModulationA:

if (envA.IsActive) // Если несущий осциллятор активен

{

float mod = 0;

if (envB.IsActive) // Если модулирующий осциллятор активен

mod = 10 \* oscB.Process() \* envB.Process(); // Сдвиг фазы несущего сигнала.

oscMix = oscA.Process(phaseModulation: mod) \* envA.Process();

}

else

{

IsActive = false;

return 0;

}

break;

case ModulationType.PhaseModulationB:

if (envB.IsActive)

{

float mod = 0;

if (envA.IsActive)

mod = 10 \* oscA.Process() \* envA.Process();

oscMix = oscB.Process(phaseModulation: mod) \* envB.Process();

}

else

{

IsActive = false;

return 0;

}

break;

}

var filterEnvOut = envFilter.Process();

return noteVelocity \* filter.Process(oscMix, filterEnvOut);

}

/// <summary>

/// Обработчик изменения частоты дискретизации.

/// </summary>

/// <param name="newSampleRate">Новая частота дискретизации.</param>

protected override void OnSampleRateChanged(float newSampleRate)

{

fmAmountMultiplier = 5000 / SampleRate;

oscA.SampleRate = newSampleRate;

oscB.SampleRate = newSampleRate;

filter.SampleRate = newSampleRate;

envA.SampleRate = newSampleRate;

envB.SampleRate = newSampleRate;

envFilter.SampleRate = newSampleRate;

}

}

}

## Components\VoicesManager.cs

using System.Linq;

using System.Collections.Generic;

using Jacobi.Vst.Framework;

namespace MultimodSynth

{

/// <summary>

/// Компонент плагина, управляющий всем голосами.

/// </summary>

class VoicesManager : AudioComponentWithParameters

{

/// <summary>

/// Максимальное количество голосов.

/// </summary>

private const int MaxVoicesCount = 32;

/// <summary>

/// Список ссылок на все используемые и неиспользуемые голоса.

/// </summary>

private List<Voice> voicesPool;

/// <summary>

/// Список активных голосов.

/// </summary>

private List<Voice> activeVoices;

/// <summary>

/// Отображение номера ноты в список голосов, играющих эту ноту.

/// </summary>

private Dictionary<byte, List<Voice>> noteToVoicesMapping;

/// <summary>

/// Отсортированное множество номеров свободных голосов.

/// </summary>

private SortedSet<int> freeVoicesIndices;

/// <summary>

/// Текущий тип модуляции.

/// </summary>

private Voice.ModulationType modulationType;

/// <summary>

/// Менеджер всех осцилляторов A во всех голосах.

/// </summary>

public OscillatorsManager OscAManager { get; set; }

/// <summary>

/// Менеджер всех осцилляторов B во всех голосах.

/// </summary>

public OscillatorsManager OscBManager { get; set; }

/// <summary>

/// Менеджер фильра во всех голосах.

/// </summary>

public FiltersManager FilterManager { get; set; }

/// <summary>

/// Менеджер всех огибающих уровня осциллятора A во всех голосах.

/// </summary>

public EnvelopesManager OscAVolumeEnvelopeManager { get; set; }

/// <summary>

/// Менеджер всех огибающих уровня осциллятора B во всех голосах.

/// </summary>

public EnvelopesManager OscBVolumeEnvelopeManager { get; set; }

/// <summary>

/// Менеджер всех огибающих частоты среза фильтра во всех голосах.

/// </summary>

public EnvelopesManager FilterCutoffEnvelopeManager { get; set; }

/// <summary>

/// Объект, управляющий параметром типа модуляции.

/// </summary>

public VstParameterManager ModulationTypeManager { get; private set; }

/// <summary>

/// Инициализирует новый объект класса VoicesManager, принадлежащий переданному плагину

/// и имеющий переданный префикс названия параметров.

/// </summary>

/// <param name="plugin">Плагин, которому принадлежит создаваемый объект.</param>

/// <param name="parameterPrefix">Префикс названия параметров.</param>

public VoicesManager(Plugin plugin, string parameterPrefix)

: base(plugin, parameterPrefix)

{

OscAManager = new OscillatorsManager(plugin, "A\_");

OscAVolumeEnvelopeManager = new EnvelopesManager(plugin, "A\_");

OscBManager = new OscillatorsManager(plugin, "B\_");

OscBVolumeEnvelopeManager = new EnvelopesManager(plugin, "B\_");

FilterManager = new FiltersManager(plugin, "F\_");

FilterCutoffEnvelopeManager = new EnvelopesManager(plugin, "F\_");

freeVoicesIndices = new SortedSet<int>(Enumerable.Range(0, MaxVoicesCount));

noteToVoicesMapping = new Dictionary<byte, List<Voice>>();

activeVoices = new List<Voice>();

voicesPool = new List<Voice>();

for (int i = 0; i < MaxVoicesCount; ++i)

voicesPool.Add(CreateVoice());

InitializeParameters();

}

/// <summary>

/// Инициализирует параметры с помощью переданной фабрики параметров.

/// </summary>

/// <param name="factory">Фабрика параметров</param>

protected override void InitializeParameters(ParameterFactory factory)

{

ModulationTypeManager = factory.CreateParameterManager(

name: "\_MT",

valueChangedHandler: SetModulationType);

}

/// <summary>

/// Обработчик изменения типа модуляции.

/// </summary>

/// <param name="value">Нормированное новое значение параметра.</param>

private void SetModulationType(float value)

{

modulationType = Converters.ToModulationType(value);

foreach (var voice in voicesPool)

voice.Modulation = modulationType;

}

/// <summary>

/// Возвращает новый объект голоса, связанный с этим объектом.

/// </summary>

/// <returns></returns>

private Voice CreateVoice()

{

var voiceOscA = OscAManager.CreateNewOscillator();

var voiceOscB = OscBManager.CreateNewOscillator();

var voiceFilter = FilterManager.CreateNewFilter();

var oscAEnvelope = OscAVolumeEnvelopeManager.CreateNewEnvelope();

var oscBEnvelope = OscBVolumeEnvelopeManager.CreateNewEnvelope();

var filterEnvelope = FilterCutoffEnvelopeManager.CreateNewEnvelope();

filterEnvelope.SetAmplitude(0);

var voice = new Voice(Plugin, voiceOscA, voiceOscB, voiceFilter,

oscAEnvelope, oscBEnvelope, filterEnvelope);

voice.Modulation = modulationType;

return voice;

}

/// <summary>

/// Играет переданную ноту.

/// </summary>

/// <param name="note">Нота, которую необходимо проиграть.</param>

public void PlayNote(MidiNote note)

{

Voice voice;

if (freeVoicesIndices.Count == 0)

{

voice = activeVoices[0];

StopVoice(voice);

}

else

{

voice = voicesPool[freeVoicesIndices.Min];

}

byte noteNo = note.NoteNo;

if (!noteToVoicesMapping.ContainsKey(noteNo))

noteToVoicesMapping[noteNo] = new List<Voice>();

noteToVoicesMapping[noteNo].Add(voice);

activeVoices.Add(voice);

freeVoicesIndices.Remove(voicesPool.IndexOf(voice));

voice.PlayNote(note);

}

/// <summary>

/// Отпускает переданную ноту.

/// </summary>

/// <param name="note">Нота, которую необходимо отпустить.</param>

public void ReleaseNote(MidiNote note)

{

byte noteNo = note.NoteNo;

if (noteToVoicesMapping.ContainsKey(noteNo))

{

foreach (var voice in noteToVoicesMapping[noteNo])

voice.TriggerRelease();

}

}

/// <summary>

/// Останавливает переданный голос и помечает его как неиспользуемый.

/// </summary>

/// <param name="note">Голос, который необходимо остановить.</param>

private void StopVoice(Voice voice)

{

activeVoices.Remove(voice);

byte noteNo = voice.Note.NoteNo;

noteToVoicesMapping[noteNo].Remove(voice);

int voiceIndex = voicesPool.IndexOf(voice);

freeVoicesIndices.Add(voiceIndex);

}

/// <summary>

/// Генерация новых выходных данных.

/// </summary>

/// <returns>Выходной сигнал.</returns>

public float Process()

{

// Вызовы этих методов обновляют значение параметров, которые подвержены сглаживанию.

OscAManager.Process();

OscBManager.Process();

FilterManager.Process();

OscAVolumeEnvelopeManager.Process();

OscBVolumeEnvelopeManager.Process();

FilterCutoffEnvelopeManager.Process();

float sum = 0;

for (int i = 0; i < activeVoices.Count;)

{

var voice = activeVoices[i];

sum += voice.Process();

if (!voice.IsActive)

// StopVoice помечает голос как свободный

// и удаляет его из массива activeVoices.

// При этом инкрементировать значение i не нужно, потому что

// все элементы activeVoices сдвигаются влево

StopVoice(voice);

else

i += 1;

}

return sum;

}

/// <summary>

/// Обработчик изменения частоты дискретизации.

/// </summary>

/// <param name="newSampleRate">Новая частота дискретизации.</param>

protected override void OnSampleRateChanged(float newSampleRate)

{

foreach (var voice in voicesPool)

voice.SampleRate = newSampleRate;

}

}

}

## DSP\Effects\Delay\DelayLine.cs

namespace MultimodSynth

{

/// <summary>

/// Представляет собой линию задержки.

/// </summary>

/// <seealso cref="https://ccrma.stanford.edu/~jos/pasp/Variable\_Delay\_Lines.html"/>

class DelayLine : AudioComponent

{

/// <summary>

/// Максимальное время задержки (в секундах).

/// </summary>

private const float MaxTime = 1f;

/// <summary>

/// Буфер, используемый для хранения предыдущих сэмплов.

/// </summary>

private float[] buffer;

/// <summary>

/// Длина буфера.

/// </summary>

private int bufferLength;

/// <summary>

/// Указатель на ячейку буфера для записи.

/// </summary>

private int writePoint;

/// <summary>

/// Указатель на ячейку буфера для чтения.

/// </summary>

private int readPoint;

/// <summary>

/// Сдвиг точки чтения относительно указателя для чтения.

/// Используется для нецелочисленного времени задержки.

/// </summary>

private float readOffset;

/// <summary>

/// Коэффициент обратной связи.

/// </summary>

private float feedback;

/// <summary>

/// Время задержки (в сэмплах).

/// </summary>

private float delay;

/// <summary>

/// Устанавливает новое значение времени задержки (в сэмплах).

/// </summary>

/// <param name="delay">Время задержки.</param>

public void SetDelay(float delay)

{

if (this.delay == delay)

return;

this.delay = delay;

double delayedPoint = (double)writePoint - delay;

if (delayedPoint < 0)

delayedPoint += bufferLength;

readPoint = (int)delayedPoint;

if (readPoint == bufferLength)

readPoint -= 1;

readOffset = (float)(delayedPoint - readPoint);

}

/// <summary>

/// Устанавливает новое значение коэффициента обратной связи.

/// </summary>

/// <param name="value">Коэффициент обратной связи.</param>

public void SetFeedback(float value)

{

feedback = value;

}

/// <summary>

/// Возвращает значение в точке чтения (используется линейная интерполяция).

/// </summary>

/// <returns>Значение в точке чтения.</returns>

public float CalculateOutput()

{

float res = buffer[readPoint] \* (1 - readOffset);

var nextPoint = readPoint + 1;

if (nextPoint < bufferLength)

res += buffer[nextPoint] \* readOffset;

else

res += buffer[0] \* readOffset;

return res;

}

/// <summary>

/// Обработка новых входных данных.

/// </summary>

/// <param name="input">Входной сигнал.</param>

/// <returns>Выходной сигнал.</returns>

public float Process(float input)

{

buffer[writePoint] = input;

float delayedSample = CalculateOutput();

readPoint += 1;

if (readPoint == bufferLength)

readPoint = 0;

buffer[writePoint] += delayedSample \* feedback;

writePoint += 1;

if (writePoint == bufferLength)

writePoint = 0;

return delayedSample;

}

/// <summary>

/// Очищает буфер линии задержки.

/// </summary>

public void Reset()

{

buffer.Initialize();

}

/// <summary>

/// Обработчик изменения частоты дискретизации.

/// </summary>

/// <param name="newSampleRate">Новая частота дискретизации.</param>

protected override void OnSampleRateChanged(float newSampleRate)

{

bufferLength = (int)(newSampleRate \* MaxTime);

buffer = new float[bufferLength];

}

}

}

## DSP\Effects\Delay\IDelay.cs

namespace MultimodSynth

{

/// <summary>

/// Интерфейс, представляющий собой эффект дилэй.

/// </summary>

interface IDelay

{

/// <summary>

/// Обработка новых входных данных.

/// </summary>

/// <param name="inputL">Левый канал входного сигнала</param>

/// <param name="inputR">Правый канал входного сигнала</param>

/// <param name="outputL">Левый канал выходного сигнала.</param>

/// <param name="outputR">Правый канал выходного сигнала.</param>

void Process(float inputL, float inputR, out float outputL, out float outputR);

/// <summary>

/// Устанавливает новое значение времени задержки (в сэмплах).

/// </summary>

/// <param name="value">Время задержки.</param>

void SetDelay(float value);

/// <summary>

/// Устанавливает новое значение коэффициента обратной связи.

/// </summary>

/// <param name="value">Коэффициент обратной связи.</param>

void SetFeedback(float value);

/// <summary>

/// Устанавливает новое значение коэффициента стерео-эффекта.

/// </summary>

/// <param name="value">Коэффициент стерео-эффекта.</param>

void SetStereo(float value);

/// <summary>

/// Очищает буфер дилэя.

/// </summary>

void Reset();

}

}

## DSP\Effects\Delay\PingPongDelay.cs

namespace MultimodSynth

{

/// <summary>

/// Представляет собой эффект пинг-понг дилэй.

/// </summary>

class PingPongDelay : AudioComponent, IDelay

{

/// <summary>

/// Линия задержки для левого канала.

/// </summary>

private DelayLine delayL;

/// <summary>

/// Линия задержки для правого канала.

/// </summary>

private DelayLine delayR;

/// <summary>

/// Коэффициент обратной связи.

/// </summary>

private float feedback;

/// <summary>

/// Коэффициент начальной громкости левого канала.

/// </summary>

private float leftCoeff;

/// <summary>

/// Коэффициент начальной громкости правого канала.

/// </summary>

private float rightCoeff;

/// <summary>

/// Инициализирует новый объект типа PingPongDelay.

/// </summary>

public PingPongDelay()

{

delayL = new DelayLine();

delayR = new DelayLine();

}

/// <summary>

/// Обработка новых входных данных.

/// </summary>

/// <param name="inputL">Левый канал входного сигнала</param>

/// <param name="inputR">Правый канал входного сигнала</param>

/// <param name="outputL">Левый канал выходного сигнала.</param>

/// <param name="outputR">Правый канал выходного сигнала.</param>

public void Process(float inputL, float inputR, out float outputL, out float outputR)

{

outputL = delayL.CalculateOutput();

outputR = delayR.CalculateOutput();

delayL.Process(inputL \* leftCoeff + outputR \* feedback);

delayR.Process(inputR \* rightCoeff + outputL \* feedback);

}

/// <summary>

/// Устанавливает новое значение времени задержки (в сэмплах).

/// </summary>

/// <param name="value">Время задержки.</param>

public void SetDelay(float delay)

{

delayL.SetDelay(delay);

delayR.SetDelay(delay);

}

/// <summary>

/// Устанавливает новое значение коэффициента обратной связи.

/// </summary>

/// <param name="value">Коэффициент обратной связи.</param>

public void SetFeedback(float value)

{

feedback = value;

}

/// <summary>

/// Устанавливает новое значение коэффициента стерео-эффекта.

/// </summary>

/// <param name="value">Коэффициент стерео-эффекта.</param>

public void SetStereo(float value)

{

if (value < 0)

{

leftCoeff = 1;

rightCoeff = 1 + value;

}

else

{

rightCoeff = 1;

leftCoeff = 1 - value;

}

}

/// <summary>

/// Очищает буфер дилэя.

/// </summary>

public void Reset()

{

delayL.Reset();

delayR.Reset();

}

/// <summary>

/// Обработчик изменения частоты дискретизации.

/// </summary>

/// <param name="newSampleRate">Новая частота дискретизации.</param>

protected override void OnSampleRateChanged(float newSampleRate)

{

delayL.SampleRate = newSampleRate;

delayR.SampleRate = newSampleRate;

}

}

}

## DSP\Effects\Delay\StereoOffsetDelay.cs

using System;

namespace MultimodSynth

{

/// <summary>

/// Представляет собой эффект дилэй с предварительной задержкой в левом или правом канале.

/// </summary>

class StereoOffsetDelay : AudioComponent, IDelay

{

/// <summary>

/// Линия задержки для левого канала.

/// </summary>

private DelayLine delayL;

/// <summary>

/// Линия задержки для правого канала.

/// </summary>

private DelayLine delayR;

/// <summary>

/// Предварительная линия задержки для левого канала.

/// </summary>

private DelayLine offsetBufferL;

/// <summary>

/// Предварительная линия задержки для правого канала.

/// </summary>

private DelayLine offsetBufferR;

/// <summary>

/// Инициализирует новый объект типа StereoOffsetDelay.

/// </summary>

public StereoOffsetDelay()

{

delayL = new DelayLine();

delayR = new DelayLine();

offsetBufferL = new DelayLine();

offsetBufferR = new DelayLine();

}

/// <summary>

/// Обработка новых входных данных.

/// </summary>

/// <param name="inputL">Левый канал входного сигнала</param>

/// <param name="inputR">Правый канал входного сигнала</param>

/// <param name="outputL">Левый канал выходного сигнала.</param>

/// <param name="outputR">Правый канал выходного сигнала.</param>

public void Process(float inputL, float inputR, out float outputL, out float outputR)

{

inputL = offsetBufferL.Process(inputL);

inputR = offsetBufferR.Process(inputR);

outputL = delayL.Process(inputL);

outputR = delayR.Process(inputR);

}

/// <summary>

/// Устанавливает новое значение времени задержки (в сэмплах).

/// </summary>

/// <param name="value">Время задержки.</param>

public void SetDelay(float value)

{

delayL.SetDelay(value);

delayR.SetDelay(value);

}

/// <summary>

/// Устанавливает новое значение коэффициента обратной связи.

/// </summary>

/// <param name="value">Коэффициент обратной связи.</param>

public void SetFeedback(float value)

{

delayL.SetFeedback(value);

delayR.SetFeedback(value);

}

/// <summary>

/// Устанавливает новое значение коэффициента стерео-эффекта.

/// </summary>

/// <param name="value">Коэффициент стерео-эффекта.</param>

public void SetStereo(float value)

{

value \*= SampleRate;

var leftDelay = -Math.Min(value, 0);

var rightDelay = value + leftDelay;

offsetBufferL.SetDelay(leftDelay);

offsetBufferR.SetDelay(rightDelay);

}

/// <summary>

/// Очищает буфер дилэя.

/// </summary>

public void Reset()

{

offsetBufferL.Reset();

offsetBufferR.Reset();

delayL.Reset();

delayR.Reset();

}

/// <summary>

/// Обработчик изменения частоты дискретизации.

/// </summary>

/// <param name="newSampleRate">Новая частота дискретизации.</param>

protected override void OnSampleRateChanged(float newSampleRate)

{

delayL.SampleRate = newSampleRate;

delayR.SampleRate = newSampleRate;

offsetBufferL.SampleRate = newSampleRate;

offsetBufferR.SampleRate = newSampleRate;

}

}

}

## DSP\Effects\Delay\VariousTimeDelay.cs

using System;

namespace MultimodSynth

{

/// <summary>

/// Представляет собой эффект дилэй с отличающимся временем задержки в левом и правом канале.

/// </summary>

class VariousTimeDelay : AudioComponent, IDelay

{

/// <summary>

/// Линия задержки для левого канала.

/// </summary>

private DelayLine delayL;

/// <summary>

/// Линия задержки для правого канала.

/// </summary>

private DelayLine delayR;

/// <summary>

/// Базовое время задержки (в сэмплах).

/// </summary>

private float delay;

/// <summary>

/// Коэффициент времени задержки для левого канала.

/// </summary>

private float leftDelayCoeff;

/// <summary>

/// Коэффициент времени задержки для правого канала.

/// </summary>

private float rightDelayCoeff;

/// <summary>

/// Инициализирует новый объект типа VariousTimeDelay.

/// </summary>

public VariousTimeDelay()

{

delayL = new DelayLine();

delayR = new DelayLine();

}

/// <summary>

/// Обработка новых входных данных.

/// </summary>

/// <param name="inputL">Левый канал входного сигнала</param>

/// <param name="inputR">Правый канал входного сигнала</param>

/// <param name="outputL">Левый канал выходного сигнала.</param>

/// <param name="outputR">Правый канал выходного сигнала.</param>

public void Process(float inputL, float inputR, out float outputL, out float outputR)

{

outputL = delayL.Process(inputL);

outputR = delayR.Process(inputR);

}

/// <summary>

/// Устанавливает новое значение времени задержки (в сэмплах).

/// </summary>

/// <param name="value">Время задержки.</param>

public void SetDelay(float value)

{

delay = value;

delayL.SetDelay(delay \* leftDelayCoeff);

delayR.SetDelay(delay \* rightDelayCoeff);

}

/// <summary>

/// Устанавливает новое значение коэффициента обратной связи.

/// </summary>

/// <param name="value">Коэффициент обратной связи.</param>

public void SetFeedback(float value)

{

delayL.SetFeedback(value);

delayR.SetFeedback(value);

}

/// <summary>

/// Устанавливает новое значение коэффициента стерео-эффекта.

/// </summary>

/// <param name="value">Коэффициент стерео-эффекта.</param>

public void SetStereo(float value)

{

leftDelayCoeff = 1 + Math.Min(value, 0) \* 0.99f;

rightDelayCoeff = 1 - Math.Max(value, 0) \* 0.99f;

delayL.SetDelay(delay \* leftDelayCoeff);

delayR.SetDelay(delay \* rightDelayCoeff);

}

/// <summary>

/// Очищает буфер дилэя.

/// </summary>

public void Reset()

{

delayL.Reset();

delayR.Reset();

}

/// <summary>

/// Обработчик изменения частоты дискретизации.

/// </summary>

/// <param name="newSampleRate">Новая частота дискретизации.</param>

protected override void OnSampleRateChanged(float newSampleRate)

{

delayL.SampleRate = newSampleRate;

delayR.SampleRate = newSampleRate;

}

}

}

## DSP\Effects\Distortion\AbsClipper.cs

using System;

namespace MultimodSynth

{

/// <summary>

/// Представляет собой вариант эффект сатурации.

/// </summary>

/// <seealso cref=" http://www.earlevel.com/main/2017/05/26/guitar-amp-simulation/"/>

class AbsClipper : IDistortion

{

/// <summary>

/// Обработка новых входных данных.

/// </summary>

/// <param name="input">Входной сигнал.</param>

/// <returns>Выходной сигнал.</returns>

public float Process(float input)

{

if (input < -1)

return -1;

else if (input > 1)

return 1;

else

return input \* (2 - Math.Abs(input));

}

/// <summary>

/// Устанавливает новое значение "силы" эффекта (в диапазоне [0, 1]).

/// У данного варианта сатурации нет силы.

/// </summary>

/// <param name="value">Значение "силы" эффекта.</param>

public void SetAmount(float value)

{

}

}

}

## DSP\Effects\Distortion\BitCrusher.cs

using System;

namespace MultimodSynth

{

/// <summary>

/// Представляет собой эффект ухудшения качества звука

/// путём уменьшения глубины квантования.

/// </summary>

class BitCrusher : IDistortion

{

/// <summary>

/// Количество возможных уровней входного сигнала.

/// </summary>

private float steps;

/// <summary>

/// Устанавливает новое значение "силы" эффекта (в диапазоне [0, 1]).

/// Чем больше сила, тем меньше глубина квантования.

/// </summary>

/// <param name="value">Значение "силы" эффекта.</param>

public void SetAmount(float value)

{

steps = (float)Math.Pow(1 << 16, 1 - value);

}

/// <summary>

/// Обработка новых входных данных.

/// </summary>

/// <param name="input">Входной сигнал.</param>

/// <returns>Выходной сигнал.</returns>

public float Process(float input)

{

return (float)Math.Round(steps \* input) / steps;

}

}

}

## DSP\Effects\Distortion\CubicClipper.cs

namespace MultimodSynth

{

/// <summary>

/// Представляет собой вариант эффекта сатурации.

/// </summary>

/// <seealso cref="https://ccrma.stanford.edu/realsimple/faust\_strings/Cubic\_Nonlinear\_Distortion.html"/>

class CubicClipper : IDistortion

{

/// <summary>

/// Обработка новых входных данных.

/// </summary>

/// <param name="input">Входной сигнал.</param>

/// <returns>Выходной сигнал.</returns>

public float Process(float input)

{

if (input < -1)

return -1;

else if (input > 1)

return 1;

else

return 1.5f \* input - 0.5f \* input \* input \* input;

}

/// <summary>

/// Устанавливает новое значение "силы" эффекта (в диапазоне [0, 1]).

/// У этого варианта сатурации нет "силы".

/// </summary>

/// <param name="value">Значение "силы" эффекта.</param>

public void SetAmount(float value)

{

}

}

}

## DSP\Effects\Distortion\IDistortion.cs

namespace MultimodSynth

{

/// <summary>

/// Интерфейс, представляющий собой эффект дисторшн.

/// </summary>

interface IDistortion

{

/// <summary>

/// Устанавливает новое значение "силы" эффекта (в диапазоне [0, 1]).

/// </summary>

/// <param name="value">Значение "силы" эффекта.</param>

void SetAmount(float value);

/// <summary>

/// Обработка новых входных данных.

/// </summary>

/// <param name="input">Входной сигнал.</param>

/// <returns>Выходной сигнал.</returns>

float Process(float input);

}

}

## DSP\Effects\Distortion\SampleRateReduction.cs

using System;

namespace MultimodSynth

{

/// <summary>

/// Представляет собой эффект ухудшения качества звука

/// путём уменьшения частоты дискретизации.

/// </summary>

class SampleRateReductor : AudioComponent, IDistortion

{

/// <summary>

/// Время, в течении которого проигрывается записанный сэмпл.

/// </summary>

private float holdTime;

/// <summary>

/// Фазовый аккумулятор.

/// </summary>

private float phasor;

/// <summary>

/// Инкремент фазы.

/// </summary>

private float phaseIncrement;

/// <summary>

/// Записанный сэмпл, который будет повторяться некоторое время.

/// </summary>

private float sample;

/// <summary>

/// Устанавливает новое значение "силы" эффекта (в диапазоне [0, 1]).

/// Чем больше сила, тем меньше частота дискретизации.

/// </summary>

/// <param name="value">Значение "силы" эффекта.</param>

public void SetAmount(float value)

{

holdTime = (float)Math.Pow(44100, 1 - value);

UpdateCoefficients();

}

/// <summary>

/// Обновление инкремента фазы при изменении некоторых параметров.

/// </summary>

private void UpdateCoefficients()

{

phaseIncrement = holdTime / SampleRate;

}

/// <summary>

/// Обработка новых входных данных.

/// </summary>

/// <param name="input">Входной сигнал.</param>

/// <returns>Выходной сигнал.</returns>

public float Process(float input)

{

phasor += phaseIncrement;

if (phasor >= 1)

{

phasor -= 1;

sample = input;

}

return sample;

}

/// <summary>

/// Обработчик изменения частоты дискретизации.

/// </summary>

/// <param name="newSampleRate">Новая частота дискретизации.</param>

protected override void OnSampleRateChanged(float newSampleRate)

{

UpdateCoefficients();

}

}

}

## DSP\Effects\Distortion\SoftClipper.cs

namespace MultimodSynth

{

/// <summary>

/// Представляет собой эффект сатурации с изменяемым порогом.

/// </summary>

/// <seealso cref="http://www.musicdsp.org/en/latest/Effects/42-soft-saturation.html"/>

class SoftClipper : IDistortion

{

/// <summary>

/// Порог, начиная с которого начинает действовать эффект.

/// </summary>

private float treshold;

/// <summary>

/// Временная переменная, использующаяся для ускорения вычислений.

/// </summary>

private float denominator;

/// <summary>

/// Коэффициент нормализации.

/// </summary>

private float normalizationCoeff;

/// <summary>

/// Устанавливает новое значение "силы" эффекта (в диапазоне [0, 1]).

/// </summary>

/// <param name="value">Значение "силы" эффекта.</param>

public void SetAmount(float value)

{

treshold = 1 - value;

var temp = 1 - treshold;

denominator = temp \* temp;

normalizationCoeff = 1 / ((treshold + 1) / 2);

}

/// <summary>

/// Обработка новых входных данных.

/// </summary>

/// <param name="input">Входной сигнал.</param>

/// <returns>Выходной сигнал.</returns>

public float Process(float input)

{

if (input < -1)

return -1;

else if (input < -treshold)

{

var temp = input + treshold;

return normalizationCoeff \* (-treshold + temp / (1 + temp \* temp / denominator));

}

else if (input < treshold)

return normalizationCoeff \* input;

else if (input < 1)

{

var temp = input - treshold;

return normalizationCoeff \* (treshold + temp / (1 + temp \* temp / denominator));

}

else

return 1;

}

}

}

## DSP\Filters\DCBlocker.cs

using System;

namespace MultimodSynth

{

/// <summary>

/// Представляет собой фильтр, убирающий постоянное амплитудное смещение.

/// </summary>

/// <seealso cref="https://ccrma.stanford.edu/~jos/filters/DC\_Blocker\_Software\_Implementations.html"/>

class DCBlocker : AudioComponent

{

/// <summary>

/// Частота дискретизации, для которой вычислен коэффициент DefaultExp.

/// </summary>

private const float BaseSampleRate = 44100f;

/// <summary>

/// Коэффициент для генерации коэффициента скорости отклика фильтра.

/// </summary>

private double DefaultExp = 0.00079777080867193622;

/// <summary>

/// Предыдущие значения входного и выходного сигнала.

/// </summary>

private float xm1, ym1;

/// <summary>

/// Коэффициент фильтра.

/// </summary>

private float r;

/// <summary>

/// Коэффициент для нормализации выходного сигнала.

/// </summary>

private float normalizationCoeff;

/// <summary>

/// Коэффициент скорости отклика.

/// </summary>

private float responseTimeCoefficient;

/// <summary>

/// Инициализирует новый объект класса DCBlocker.

/// </summary>

/// <param name="responseTimeCoefficient">Коэффициент скорости отклика фильтра</param>

public DCBlocker(float responseTimeCoefficient = 1)

{

this.responseTimeCoefficient = responseTimeCoefficient;

}

/// <summary>

/// Обработка новых входных данных.

/// </summary>

/// <param name="x">Входной сигнал.</param>

/// <returns>Выходной сигнал.</returns>

public float Process(float x)

{

var y = normalizationCoeff \* (x - xm1) + r \* ym1;

xm1 = x;

ym1 = y;

return y;

}

/// <summary>

/// Устанавливает коэффициент времени отклика фильтра.

/// </summary>

/// <param name="value"></param>

public void SetResponseTimeCoefficient(float value)

{

responseTimeCoefficient = value;

}

/// <summary>

/// Обработчик изменения частоты дискретизации.

/// </summary>

/// <param name="newSampleRate">Новая частота дискретизации.</param>

protected override void OnSampleRateChanged(float newSampleRate)

{

var coeff = BaseSampleRate / newSampleRate / responseTimeCoefficient;

r = (float)Math.Exp(-2 \* Math.PI \* DefaultExp \* coeff);

normalizationCoeff = (1 + r) / 2;

}

}

}

## DSP\Filters\Downsampler.cs

namespace MultimodSynth

{

/// <summary>

/// Фильтр низких частот, используемый для понижения частоты дискретизации.

/// </summary>

class Downsampler : AudioComponent

{

/// <summary>

/// Коэффициенты, используемые для понижения частоты дискретизации в 2 раза.

/// </summary>

public static double[] X2OversamplingCoefficients = new[]

{

0.0000030416222567197833, 8.361952157148288e-7, -0.000005925094515842721, -0.000003024729165169841,

0.000009483783706165421, 0.00000723209777116489, -0.000013294008492742, -0.000014155214050509543,

0.000016607882158558667, 0.000024423989868674373, -0.000018310088319921986, -0.00003847376842346738,

0.000016907497464738733, 0.00005639361908773835, -0.000010563822308760637, -0.00007776107488778567,

-0.0000028102981031513762, 0.00010147838274441141, 0.000025404793602647142, -0.00012562924128648733,

-0.000059293195541806884, 0.00014737778555502898, 0.00010615545967462068, -0.0001629327263678661,

-0.0001669520856300777, 0.00016759863428288119, 0.00024156870266542192, -0.0001559330731121642,

-0.00032845768487997786, 0.00012202253220773863, 0.0004243097138152885, -0.000059881987314738115,

-0.0005237927446189893, -0.00003602718887115699, 0.0006193978212302525, 0.00017017774661556506,

-0.0007014300326804272, -0.00034528434925108943, 0.000758178232547994, 0.0005615791001342536,

-0.0007762888215179651, -0.000816105367898944, 0.0007413571604539623, 0.001102087938453264,

-0.0006387355546314502, -0.0014084413464889038, 0.00045454012826587505, 0.0017194776073926833,

-0.00017682145879295325, -0.002014868955070687, -0.00020315303365908512, 0.002269910418084365,

0.0006895717046618674, -0.0024561114384577644, -0.0012807841502350589, 0.0025421258437818978,

0.001968123259388613, -0.0024950065823285024, -0.00273492701557877, 0.0022817470579166617,

0.0035558350943221766, -0.0018710464882201244, -0.004396418860510098, 0.0012352143112775136,

0.005213177998999048, -0.00035211017965926044, -0.005953902083357598, -0.0007929967550256986,

0.006558349766125828, 0.0022057713255015233, -0.006959139407586607, -0.003881478361851405,

0.00708266710912906, 0.005804080215017983, -0.006849758826814975, -0.007945855701826782,

0.006175595973442279, 0.010267603871768961, -0.004968171703644546, -0.012719463422626474,

0.0031240124390668894, 0.015242342217107538, -0.0005188446412763827, -0.01776991293146465,

-0.003011419499901605, 0.020231093909868218, 0.007709504147400423, -0.02255290059398985,

-0.013980045763524558, 0.02466352548220426, 0.02258557190829597, -0.02649548534433017,

-0.03517923749098369, 0.02798866501937985, 0.05611521692777254, -0.029093088468055358,

-0.10149581032176695, 0.029771260001166155, 0.31676011963026557, 0.46999907783956496,

0.31676011963026557, 0.029771260001166155, -0.10149581032176695, -0.029093088468055358,

0.05611521692777254, 0.02798866501937985, -0.03517923749098369, -0.02649548534433017,

0.02258557190829597, 0.02466352548220426, -0.013980045763524558, -0.02255290059398985,

0.007709504147400423, 0.020231093909868218, -0.003011419499901605, -0.01776991293146465,

-0.0005188446412763827, 0.015242342217107538, 0.0031240124390668894, -0.012719463422626474,

-0.004968171703644546, 0.010267603871768961, 0.006175595973442279, -0.007945855701826782,

-0.006849758826814975, 0.005804080215017983, 0.00708266710912906, -0.003881478361851405,

-0.006959139407586607, 0.0022057713255015233, 0.006558349766125828, -0.0007929967550256986,

-0.005953902083357598, -0.00035211017965926044, 0.005213177998999048, 0.0012352143112775136,

-0.004396418860510098, -0.0018710464882201244, 0.0035558350943221766, 0.0022817470579166617,

-0.00273492701557877, -0.0024950065823285024, 0.001968123259388613, 0.0025421258437818978,

-0.0012807841502350589, -0.0024561114384577644, 0.0006895717046618674, 0.002269910418084365,

-0.00020315303365908512, -0.002014868955070687, -0.00017682145879295325, 0.0017194776073926833,

0.00045454012826587505, -0.0014084413464889038, -0.0006387355546314502, 0.001102087938453264,

0.0007413571604539623, -0.000816105367898944, -0.0007762888215179651, 0.0005615791001342536,

0.000758178232547994, -0.00034528434925108943, -0.0007014300326804272, 0.00017017774661556506,

0.0006193978212302525, -0.00003602718887115699, -0.0005237927446189893, -0.000059881987314738115,

0.0004243097138152885, 0.00012202253220773863, -0.00032845768487997786, -0.0001559330731121642,

0.00024156870266542192, 0.00016759863428288119, -0.0001669520856300777, -0.0001629327263678661,

0.00010615545967462068, 0.00014737778555502898, -0.000059293195541806884, -0.00012562924128648733,

0.000025404793602647142, 0.00010147838274441141, -0.0000028102981031513762, -0.00007776107488778567,

-0.000010563822308760637, 0.00005639361908773835, 0.000016907497464738733, -0.00003847376842346738,

-0.000018310088319921986, 0.000024423989868674373, 0.000016607882158558667, -0.000014155214050509543,

-0.000013294008492742, 0.00000723209777116489, 0.000009483783706165421, -0.000003024729165169841,

-0.000005925094515842721, 8.361952157148288e-7, 0.0000030416222567197833,

};

/// <summary>

/// Коэффициенты, используемые для понижения частоты дискретизации в 4 раза.

/// </summary>

public static double[] X4OversamplingCoefficients = new[]

{

-2.109927283343053e-7, -0.0000030954773334054167, -0.000006134974541370354, -0.000006824726676465002,

-0.0000029695790184545317, 0.00000543810810444409, 0.000015085719726649097, 0.00002009178690506875,

0.000014888862902276953, -0.0000018956441452944518, -0.000024787461023249072, -0.000042249293152761053,

-0.0000416318947663319, -0.000016594762254257985, 0.00002681912949281089, 0.00006964410015600254,

0.00008706318232532724, 0.00006115720033477816, -0.000006582162075097628, -0.00009042562538399676,

-0.00014823619860910388, -0.00014085719354295487, -0.00005496577117692798, 0.0000827605921760652,

0.00021037372728432093, 0.00025623571397465604, 0.00017610155088211164, -0.000016263122399402805,

-0.0002433524664945467, -0.00039176291740975356, -0.0003656295979266028, -0.00014192241084050146,

0.00020206755938300452, 0.0005095886712000344, 0.0006120209582578663, 0.00041600849812067334,

-0.00003277867660879936, -0.0005476637273956544, -0.0008731935585711394, -0.0008066909331471724,

-0.0003136134693221922, 0.00042517940824487083, 0.0010707631209802022, 0.0012756237604383829,

0.0008623284546548473, -0.00005709423002142209, -0.0010927092240135543, -0.0017338014198887903,

-0.0015927280535993687, -0.0006226341547564678, 0.000807278380377599, 0.002038853130332113,

0.002418742199839844, 0.0016321613448289393, -0.00008861584485211762, -0.002005264341221124,

-0.003177683590299736, -0.0029126840161716678, -0.00114846008412864, 0.0014290498189259149,

0.0036325993682621432, 0.004305916961712298, 0.0029101027553914928, -0.00012481940224640957,

-0.003490540750129053, -0.005544107115204153, -0.005088839378918694, -0.002030662129567573,

0.0024345818096678097, 0.006255154136374177, 0.007439466687422844, 0.005057338890021917,

-0.0001614091629296627, -0.005978610069656505, -0.009567841593976986, -0.008843251297269953,

-0.0035915417669226614, 0.004176081095884903, 0.010924281837856691, 0.013137258435477368,

0.009059066879413208, -0.00019305987809899484, -0.010767118715054515, -0.017568142274464602,

-0.01656973628096208, -0.006951764386240666, 0.007978053529804718, 0.021689601418032202,

0.027009056480791414, 0.019432545209587804, -0.00021458806269304627, -0.025044603151207545,

-0.043883194845277085, -0.04512703228917539, -0.02133375728139363, 0.027237558705049188,

0.09164363867092296, 0.15642955310486548, 0.20434870046124462, 0.22200149201850886,

0.20434870046124462, 0.15642955310486548, 0.09164363867092296, 0.027237558705049188,

-0.02133375728139363, -0.04512703228917539, -0.043883194845277085, -0.025044603151207545,

-0.00021458806269304627, 0.019432545209587804, 0.027009056480791414, 0.021689601418032202,

0.007978053529804718, -0.006951764386240666, -0.01656973628096208, -0.017568142274464602,

-0.010767118715054515, -0.00019305987809899484, 0.009059066879413208, 0.013137258435477368,

0.010924281837856691, 0.004176081095884903, -0.0035915417669226614, -0.008843251297269953,

-0.009567841593976986, -0.005978610069656505, -0.0001614091629296627, 0.005057338890021917,

0.007439466687422844, 0.006255154136374177, 0.0024345818096678097, -0.002030662129567573,

-0.005088839378918694, -0.005544107115204153, -0.003490540750129053, -0.00012481940224640957,

0.0029101027553914928, 0.004305916961712298, 0.0036325993682621432, 0.0014290498189259149,

-0.00114846008412864, -0.0029126840161716678, -0.003177683590299736, -0.002005264341221124,

-0.00008861584485211762, 0.0016321613448289393, 0.002418742199839844, 0.002038853130332113,

0.000807278380377599, -0.0006226341547564678, -0.0015927280535993687, -0.0017338014198887903,

-0.0010927092240135543, -0.00005709423002142209, 0.0008623284546548473, 0.0012756237604383829,

0.0010707631209802022, 0.00042517940824487083, -0.0003136134693221922, -0.0008066909331471724,

-0.0008731935585711394, -0.0005476637273956544, -0.00003277867660879936, 0.00041600849812067334,

0.0006120209582578663, 0.0005095886712000344, 0.00020206755938300452, -0.00014192241084050146,

-0.0003656295979266028, -0.00039176291740975356, -0.0002433524664945467, -0.000016263122399402805,

0.00017610155088211164, 0.00025623571397465604, 0.00021037372728432093, 0.0000827605921760652,

-0.00005496577117692798, -0.00014085719354295487, -0.00014823619860910388, -0.00009042562538399676,

-0.000006582162075097628, 0.00006115720033477816, 0.00008706318232532724, 0.00006964410015600254,

0.00002681912949281089, -0.000016594762254257985, -0.0000416318947663319, -0.000042249293152761053,

-0.000024787461023249072, -0.0000018956441452944518, 0.000014888862902276953, 0.00002009178690506875,

0.000015085719726649097, 0.00000543810810444409, -0.0000029695790184545317, -0.000006824726676465002,

-0.000006134974541370354, -0.0000030954773334054167, -2.109927283343053e-7,

};

/// <summary>

/// Коэффициенты, используемые для понижения частоты дискретизации в 8 раз.

/// </summary>

public static readonly double[] X8OversamplingCoefficients = new[]

{

-0.0000029200756801788613, -0.0000036908168209737206, -0.000003857741139769582, -0.000003024682318617163,

-8.350977788871221e-7, 0.000002944251445679728, 0.000008335743784466808, 0.000015073538480114391,

0.000022552302392249296, 0.000029817564493604254, 0.00003561034386583803, 0.00003847317254749848,

0.000036916791523526684, 0.000029637355104723947, 0.00001576402711344597, -0.00000489223638045258,

-0.000031624651137942116, -0.00006269018410870415, -0.00009527960921632048, -0.00012562729555959414,

-0.0001492767905220687, -0.00016150026944632451, -0.00015784871777746405, -0.00013478758600264467,

-0.00009035241826786288, -0.00002474362552954989, 0.00005922797206904976, 0.0001559306580440558,

0.00025700818217739635, 0.00035182915629416094, 0.00042829982921699437, 0.0004740073972344034,

0.0004776147040590056, 0.0004303830392349219, 0.00032766354210853185, 0.0001701751109280932,

-0.00003511748629931697, -0.0002746941741945493, -0.0005288675125380204, -0.0007729353490468208,

-0.0009790871554455078, -0.0011189645274161382, -0.0011666866444181795, -0.0011020708694801636,

-0.000913717654524201, -0.0006015971096861086, -0.00017877944581660795, 0.0003280028159593051,

0.000879186418154025, 0.001424705435366355, 0.0019077686315023064, 0.0022698752620514185,

0.0024566974362175017, 0.0024243167651252736, 0.002145200891163049, 0.0016132571309776188,

0.0008473171269436378, -0.00010750646039385977, -0.0011809941396326742, -0.002281711718559606,

-0.003303548151947555, -0.004134546766735901, -0.004667403581060008, -0.004810767603552248,

-0.004500302291959165, -0.0037083788291223272, -0.0024512878125591514, -0.0007929844732106322,

0.0011553789779305066, 0.0032405457793016875, 0.005276877963377765, 0.007060273579520812,

0.008385400398569425, 0.009064974178452884, 0.008949502590747584, 0.007945732637602508,

0.006032006505669669, 0.0032688622142878596, -0.00019648962673344639, -0.004132740183272371,

-0.008234478076064216, -0.012139348962837467, -0.01545147649481472, -0.017769637713711803,

-0.01871817944283607, -0.01797830457696428, -0.015317187990793525, -0.010612425091064472,

-0.0038695774088740088, 0.004768955023222468, 0.01502486857635727, 0.026495074986217233,

0.03867256498462638, 0.050975771551738734, 0.06278446284949787, 0.0734796155669735,

0.08248433887975852, 0.08930277412534568, 0.09355400202699099, 0.09499834226296974,

0.09355400202699099, 0.08930277412534568, 0.08248433887975852, 0.0734796155669735,

0.06278446284949787, 0.050975771551738734, 0.03867256498462638, 0.026495074986217233,

0.01502486857635727, 0.004768955023222468, -0.0038695774088740088, -0.010612425091064472,

-0.015317187990793525, -0.01797830457696428, -0.01871817944283607, -0.017769637713711803,

-0.01545147649481472, -0.012139348962837467, -0.008234478076064216, -0.004132740183272371,

-0.00019648962673344639, 0.0032688622142878596, 0.006032006505669669, 0.007945732637602508,

0.008949502590747584, 0.009064974178452884, 0.008385400398569425, 0.007060273579520812,

0.005276877963377765, 0.0032405457793016875, 0.0011553789779305066, -0.0007929844732106322,

-0.0024512878125591514, -0.0037083788291223272, -0.004500302291959165, -0.004810767603552248,

-0.004667403581060008, -0.004134546766735901, -0.003303548151947555, -0.002281711718559606,

-0.0011809941396326742, -0.00010750646039385977, 0.0008473171269436378, 0.0016132571309776188,

0.002145200891163049, 0.0024243167651252736, 0.0024566974362175017, 0.0022698752620514185,

0.0019077686315023064, 0.001424705435366355, 0.000879186418154025, 0.0003280028159593051,

-0.00017877944581660795, -0.0006015971096861086, -0.000913717654524201, -0.0011020708694801636,

-0.0011666866444181795, -0.0011189645274161382, -0.0009790871554455078, -0.0007729353490468208,

-0.0005288675125380204, -0.0002746941741945493, -0.00003511748629931697, 0.0001701751109280932,

0.00032766354210853185, 0.0004303830392349219, 0.0004776147040590056, 0.0004740073972344034,

0.00042829982921699437, 0.00035182915629416094, 0.00025700818217739635, 0.0001559306580440558,

0.00005922797206904976, -0.00002474362552954989, -0.00009035241826786288, -0.00013478758600264467,

-0.00015784871777746405, -0.00016150026944632451, -0.0001492767905220687, -0.00012562729555959414,

-0.00009527960921632048, -0.00006269018410870415, -0.000031624651137942116, -0.00000489223638045258,

0.00001576402711344597, 0.000029637355104723947, 0.000036916791523526684, 0.00003847317254749848,

0.00003561034386583803, 0.000029817564493604254, 0.000022552302392249296, 0.000015073538480114391,

0.000008335743784466808, 0.000002944251445679728, -8.350977788871221e-7, -0.000003024682318617163,

-0.000003857741139769582, -0.0000036908168209737206, -0.0000029200756801788613,

};

/// <summary>

/// Текущие коэффициенты фильтра.

/// </summary>

private double[] coefficients;

/// <summary>

/// Циклический буфер, хранящий последние n входных сэмплов.

/// </summary>

private double[] buffer;

/// <summary>

/// Указывает на индекс для записи в циклический буфер.

/// </summary>

private int currBufferIndex = 0;

/// <summary>

/// Отношение частоты дискретизации выходного сигнала к входному.

/// </summary>

private int order;

/// <summary>

/// Отношение частоты дискретизации выходного сигнала к входному.

/// </summary>

public int Order

{

get => order;

set

{

order = value;

switch (order)

{

case 1:

break;

case 2:

coefficients = X2OversamplingCoefficients;

buffer = new double[coefficients.Length];

break;

case 4:

coefficients = X4OversamplingCoefficients;

buffer = new double[coefficients.Length];

break;

case 8:

coefficients = X8OversamplingCoefficients;

buffer = new double[coefficients.Length];

break;

}

}

}

/// <summary>

/// Обработка новых входных данных.

/// </summary>

/// <param name="inputs">Массив, содержащий как минимум <see cref="Order"/> сэмплов.</param>

/// <returns>Выходной сигнал.</returns>

public double Process(params double[] inputs)

{

// order - коэффициент, во сколько раз нужно снизить частоту дискретизации

if (order == 1)

return inputs[0];

double res = 0;

for (int inputIndex = 0; inputIndex < order; ++inputIndex)

{

// Заносим сэмпл в буфер.

buffer[currBufferIndex] = inputs[inputIndex];

// Оптимизация: фильтруется только первый входной сэмпл, остальные отбрасываются

if (inputIndex == 0)

{

int index = currBufferIndex;

for (int i = 0; i < coefficients.Length; ++i)

{

res += coefficients[i] \* buffer[index];

index -= 1;

if (index < 0)

index = buffer.Length - 1;

}

}

// Указатель для записи в циклический буфер

currBufferIndex += 1;

if (currBufferIndex == buffer.Length)

currBufferIndex = 0;

}

return res;

}

}

}

## DSP\Filters\SvfFilter.cs

using System;

namespace MultimodSynth

{

/// <summary>

/// Реализация SVF (State Variable Filter).

/// </summary>

/// <seealso cref="https://cytomic.com/files/dsp/SvfLinearTrapOptimised2.pdf"/>

class SvfFilter : AudioComponent

{

/// <summary>

/// Указывает тип фильтра.

/// </summary>

public enum FilterType

{

None,

Low,

Band,

High,

Notch,

Peak,

All,

Bell,

LowShelf,

HighShelf,

};

/// <summary>

/// Текущий тип фильтра.

/// </summary>

private FilterType type;

/// <summary>

/// Частота среза фильтра.

/// </summary>

private float cutoff;

/// <summary>

/// "Ширина" фильтра.

/// </summary>

private float q = 1;

/// <summary>

/// Первый аккумулятор.

/// </summary>

private float ic1eq = 0;

/// <summary>

/// Второй аккумулятор.

/// </summary>

private float ic2eq = 0;

/// <summary>

/// Коэффициент фильтра.

/// </summary>

private float a1;

/// <summary>

/// Коэффициент фильтра.

/// </summary>

private float a2;

/// <summary>

/// Коэффициент фильтра.

/// </summary>

private float a3;

/// <summary>

/// Коэффициент фильтра.

/// </summary>

private float m0;

/// <summary>

/// Коэффициент фильтра.

/// </summary>

private float m1;

/// <summary>

/// Коэффициент фильтра.

/// </summary>

private float m2;

/// <summary>

/// Коэффициент фильтра.

/// </summary>

private float gain;

/// <summary>

/// Инициализирует новый объект типа SvfFilter c заданными параметрами.

/// </summary>

/// <param name="type"></param>

/// <param name="cutoff"></param>

/// <param name="q"></param>

/// <param name="gain"></param>

public SvfFilter(

FilterType type = FilterType.None,

float cutoff = 20000,

float q = 1,

float gain = 0)

{

SetType(type);

SetCutoff(cutoff);

SetQ(q);

SetGain(gain);

}

/// <summary>

/// Устанавливает новое значение типа фильтра.

/// </summary>

/// <param name="value"></param>

public void SetType(FilterType value)

{

type = value;

UpdateCoefficients();

Reset();

}

/// <summary>

/// Устанавливает новое значение частоты среза фильтра.

/// </summary>

/// <param name="value"></param>

public void SetCutoff(float value)

{

if (cutoff != value)

{

cutoff = value;

UpdateCoefficients();

}

}

/// <summary>

/// Устанавливает новое значение "ширины" фильтра.

/// </summary>

/// <param name="value"></param>

public void SetQ(float value)

{

q = value;

UpdateCoefficients();

}

/// <summary>

/// Устанавливает новое значение увеличения уровня громкости.

/// </summary>

/// <param name="value"></param>

public void SetGain(float value)

{

gain = value;

UpdateCoefficients();

}

/// <summary>

/// Выполняет обновление всех коэффициентов.

/// </summary>

private void UpdateCoefficients()

{

switch (type)

{

case FilterType.Low:

var g = (float)Math.Tan(Math.PI \* cutoff / SampleRate);

var k = 1 / q;

a1 = 1 / (1 + g \* (g + k));

a2 = g \* a1;

a3 = g \* a2;

m0 = 0;

m1 = 0;

m2 = 1;

break;

case FilterType.Band:

g = (float)Math.Tan(Math.PI \* cutoff / SampleRate);

k = 1 / q;

a1 = 1 / (1 + g \* (g + k));

a2 = g \* a1;

a3 = g \* a2;

m0 = 0;

m1 = 1;

m2 = 0;

break;

case FilterType.High:

g = (float)Math.Tan(Math.PI \* cutoff / SampleRate);

k = 1 / q;

a1 = 1 / (1 + g \* (g + k));

a2 = g \* a1;

a3 = g \* a2;

m0 = 1;

m1 = -k;

m2 = -1;

break;

case FilterType.Notch:

g = (float)Math.Tan(Math.PI \* cutoff / SampleRate);

k = 1 / q;

a1 = 1 / (1 + g \* (g + k));

a2 = g \* a1;

a3 = g \* a2;

m0 = 1;

m1 = -k;

m2 = 0;

break;

case FilterType.Peak:

g = (float)Math.Tan(Math.PI \* cutoff / SampleRate);

k = 1 / q;

a1 = 1 / (1 + g \* (g + k));

a2 = g \* a1;

a3 = g \* a2;

m0 = 1;

m1 = -k;

m2 = -2;

break;

case FilterType.All:

g = (float)Math.Tan(Math.PI \* cutoff / SampleRate);

k = 1 / q;

a1 = 1 / (1 + g \* (g + k));

a2 = g \* a1;

a3 = g \* a2;

m0 = 1;

m1 = -2 \* k;

m2 = 0;

break;

case FilterType.Bell:

var A = (float)Math.Pow(10, gain / 40);

g = (float)Math.Tan(Math.PI \* cutoff / SampleRate);

k = 1 / (q \* A);

a1 = 1 / (1 + g \* (g + k));

a2 = g \* a1;

a3 = g \* a2;

m0 = 1;

m1 = k \* (A \* A - 1);

m2 = 0;

break;

case FilterType.LowShelf:

A = (float)Math.Pow(10, gain / 40);

g = (float)(Math.Tan(Math.PI \* cutoff / SampleRate) / Math.Sqrt(A));

k = 1 / q;

a1 = 1 / (1 + g \* (g + k));

a2 = g \* a1;

a3 = g \* a2;

m0 = 1;

m1 = k \* (A - 1);

m2 = A \* A - 1;

break;

case FilterType.HighShelf:

A = (float)Math.Pow(10, gain / 40);

g = (float)(Math.Tan(Math.PI \* cutoff / SampleRate) \* Math.Sqrt(A));

k = 1 / q;

a1 = 1 / (1 + g \* (g + k));

a2 = g \* a1;

a3 = g \* a2;

m0 = A \* A;

m1 = k \* (1 - A) \* A;

m2 = 1 - A \* A;

break;

}

}

/// <summary>

/// Обработка новых входных данных.

/// </summary>

/// <param name="v0">Входной сигнал.</param>

/// <returns>Выходной сигнал.</returns>

public float Process(float v0)

{

float v3 = v0 - ic2eq;

float v1 = a1 \* ic1eq + a2 \* v3;

float v2 = ic2eq + a2 \* ic1eq + a3 \* v3;

ic1eq = 2 \* v1 - ic1eq;

ic2eq = 2 \* v2 - ic2eq;

return m0 \* v0 + m1 \* v1 + m2 \* v2;

}

/// <summary>

/// Сбрасывает текущее состояние фильтра.

/// </summary>

public void Reset()

{

ic1eq = 0;

ic2eq = 0;

}

/// <summary>

/// Обработчик изменения частоты дискретизации.

/// </summary>

/// <param name="newSampleRate">Новая частота дискретизации.</param>

protected override void OnSampleRateChanged(float newSampleRate)

{

UpdateCoefficients();

}

}

}

## DSP\Generators\AdsrEnvelope.cs

using System;

namespace MultimodSynth

{

/// <summary>

/// Реализация ADSR огибающей.

/// </summary>

/// <seealso cref="http://www.earlevel.com/main/2013/06/01/envelope-generators/"/>

class AdsrEnvelope

{

/// <summary>

/// Указывает стадию огибающей.

/// </summary>

public enum EnvelopeState

{

Idle,

Attack,

Decay,

Sustain,

Release

}

/// <summary>

/// Значение, которое прибавляется на стадии атаки.

/// </summary>

private double attackBase;

/// <summary>

/// Коэффициент для стадии атаки.

/// </summary>

private double attackCoef;

/// <summary>

/// Длительность стадии атаки (в сэмплах).

/// </summary>

private double attackRate;

/// <summary>

/// Разность целевого значения и максимального значения стадии атаки.

/// </summary>

private double attackTargetRatio;

/// <summary>

/// Текущее значение огибающей.

/// </summary>

private double currValue;

/// <summary>

/// Значение, которое прибавляется на стадии спада.

/// </summary>

private double decayBase;

/// <summary>

/// Коэффициент для стадии спада.

/// </summary>

private double decayCoef;

/// <summary>

/// Длительность стадии спада (в сэмплах).

/// </summary>

private double decayRate;

/// <summary>

/// Разность целевого значения и максимального значения стадий спада и затухания.

/// </summary>

private double decayReleaseTargetRatio;

/// <summary>

/// Значение, которое прибавляется на стадии затухания.

/// </summary>

private double releaseBase;

/// <summary>

/// Коэффициент для стадии затухания.

/// </summary>

private double releaseCoef;

/// <summary>

/// Длительность стадии затухания (в сэмплах).

/// </summary>

private double releaseRate;

/// <summary>

/// Текущее состояние огибающей.

/// </summary>

private EnvelopeState state;

/// <summary>

/// Уровень стадии поддержки.

/// </summary>

private double sustainLevel;

/// <summary>

/// Инициализирует новый объект типа AdsrEnvelope.

/// </summary>

public AdsrEnvelope()

{

SetAttackRate(0);

SetDecayRate(0);

SetReleaseRate(0);

SetSustainLevel(1);

SetAttackTargetRatio(0.3f);

SetDecayReleaseTargetRatio(0.0001f);

}

/// <summary>

/// Текущее состояние огибающей.

/// </summary>

public EnvelopeState State => state;

/// <summary>

/// Устанавливает новое значение "скорости" стадии атаки.

/// </summary>

/// <param name="rate"></param>

public void SetAttackRate(double rate)

{

attackRate = rate;

attackCoef = CalcCoef(rate, attackTargetRatio);

attackBase = (1 + attackTargetRatio) \* (1 - attackCoef);

}

/// <summary>

/// Устанавливает новое значение "скорости" стадии спада.

/// </summary>

/// <param name="rate"></param>

public void SetDecayRate(double rate)

{

decayRate = rate;

decayCoef = CalcCoef(decayRate, decayReleaseTargetRatio);

decayBase = (sustainLevel - decayReleaseTargetRatio) \* (1 - decayCoef);

}

/// <summary>

/// Устанавливает новое значение уровня стадии поддержки.

/// </summary>

/// <param name="level"></param>

public void SetSustainLevel(double level)

{

sustainLevel = level;

decayBase = (sustainLevel - decayReleaseTargetRatio) \* (1 - decayCoef);

}

/// <summary>

/// Устанавливает новое значение длительности стадии затухания (в сэмплах).

/// </summary>

/// <param name="rate"></param>

public void SetReleaseRate(double rate)

{

releaseRate = rate;

releaseCoef = CalcCoef(rate, decayReleaseTargetRatio);

releaseBase = -decayReleaseTargetRatio \* (1 - releaseCoef);

}

/// <summary>

/// Устанавливает разность целевого и максимального значения стадии атаки.

/// Используется для управления изгибом огибающей.

/// </summary>

/// <param name="targetRatio"></param>

public void SetAttackTargetRatio(double targetRatio)

{

if (targetRatio < 0.000000001f)

attackTargetRatio = 0.000000001f;

else

attackTargetRatio = targetRatio;

attackCoef = CalcCoef(attackRate, targetRatio);

attackBase = (1 + attackTargetRatio) \* (1 - attackCoef);

}

/// <summary>

/// Устанавливает разность целевого и максимального значения стадий спада и затухания.

/// Используется для управления изгибом огибающей.

/// </summary>

/// <param name="targetRatio"></param>

public void SetDecayReleaseTargetRatio(double targetRatio)

{

if (targetRatio < 0.000000001f)

decayReleaseTargetRatio = 0.000000001f;

else

decayReleaseTargetRatio = targetRatio;

decayCoef = CalcCoef(decayRate, targetRatio);

releaseCoef = CalcCoef(releaseRate, targetRatio);

decayBase = (sustainLevel - decayReleaseTargetRatio) \* (1 - decayCoef);

releaseBase = -decayReleaseTargetRatio \* (1 - releaseCoef);

}

/// <summary>

/// Генерация нового значения огибающей.

/// </summary>

/// <returns></returns>

public double Process()

{

switch (state)

{

case EnvelopeState.Idle:

break;

case EnvelopeState.Attack:

currValue = attackBase + currValue \* attackCoef;

if (currValue >= 1f)

{

currValue = 1f;

if (decayCoef != 0)

state = EnvelopeState.Decay;

else

state = EnvelopeState.Sustain;

}

break;

case EnvelopeState.Decay:

currValue = decayBase + currValue \* decayCoef;

if (currValue <= sustainLevel)

{

currValue = sustainLevel;

state = EnvelopeState.Sustain;

}

break;

case EnvelopeState.Sustain:

currValue = sustainLevel;

break;

case EnvelopeState.Release:

currValue = releaseBase + currValue \* releaseCoef;

if (currValue <= 0)

{

currValue = 0;

state = EnvelopeState.Idle;

OnSoundStop();

}

break;

}

return currValue;

}

/// <summary>

/// Метод, вызываемый при остановке генератора огибающей.

/// </summary>

private void OnSoundStop()

{

SoundStop?.Invoke(this, new EventArgs());

}

/// <summary>

/// Событие, возникающее при остановке генератора огибающей.

/// </summary>

public event EventHandler SoundStop;

/// <summary>

/// Возвращает коэффициент для вычисления значений огибающей.

/// </summary>

/// <param name="rate"></param>

/// <param name="targetRatio"></param>

/// <returns></returns>

private double CalcCoef(double rate, double targetRatio) =>

(rate <= 0f) ? 0f : Math.Exp(-Math.Log((1.0 + targetRatio) / targetRatio) / rate);

/// <summary>

/// Метод, начинающий стадию атаки.

/// </summary>

public void TriggerAttack()

{

if (attackCoef != 0)

{

currValue = 0;

state = EnvelopeState.Attack;

}

else if (decayCoef != 0)

{

currValue = 1;

state = EnvelopeState.Decay;

}

else

state = EnvelopeState.Sustain;

}

/// <summary>

/// Метод, начинающий стадию затухания.

/// </summary>

public void TriggerRelease()

{

state = EnvelopeState.Release;

}

}

}

## DSP\Generators\SineLFO.cs

using System;

namespace MultimodSynth

{

/// <summary>

/// Реализация низкочастотного генератора синусоиды.

/// </summary>

class SineLFO : AudioComponent

{

/// <summary>

/// Частота генератора.

/// </summary>

private float frequency;

/// <summary>

/// Коэффициент для генерации.

/// </summary>

private float coeff;

/// <summary>

/// Текущее значение синусоиды.

/// </summary>

private float sin;

/// <summary>

/// Текущее значение косинусоиды.

/// </summary>

private float cos;

/// <summary>

/// Инициализирует новый объект типа SineLFO.

/// </summary>

public SineLFO()

{

sin = 0;

cos = 1;

}

/// <summary>

/// Устанавливает новое значение частоты.

/// </summary>

/// <param name="value"></param>

public void SetFrequency(float value)

{

frequency = value;

UpdateCoefficients();

}

/// <summary>

/// Выполняет обновление коэффициента.

/// </summary>

private void UpdateCoefficients()

{

coeff = (float)(2 \* Math.PI \* frequency / SampleRate);

}

/// <summary>

/// Генерация нового значения.

/// </summary>

/// <returns></returns>

public float Process()

{

sin = sin + cos \* coeff;

cos = cos - sin \* coeff;

return sin;

}

/// <summary>

/// Обработчик изменения частоты дискретизации.

/// </summary>

/// <param name="newSampleRate">Новая частота дискретизации.</param>

protected override void OnSampleRateChanged(float newSampleRate)

{

UpdateCoefficients();

}

}

}

## DSP\Generators\WaveTableOscillators.cs

using System;

using System.IO;

namespace MultimodSynth

{

/// <summary>

/// Реализация осциллятора, основанного на таблицах сэмплов.

/// </summary>

public class WaveTableOscillator

{

/// <summary>

/// Стандартная частота дискретизации.

/// </summary>

private const double DefaultSampleRate = 44100;

/// <summary>

/// Представляет собой одну таблицу сэмплов, записанную с заданным инкрементом фазы.

/// </summary>

private class WaveTable

{

/// <summary>

/// Длина таблицы.

/// </summary>

public int Length;

/// <summary>

/// Сэмплы.

/// </summary>

public float[] Samples;

/// <summary>

/// Инкремент фазы, с которым эта таблица была сгенерирована.

/// </summary>

public float PhaseIncrement;

}

/// <summary>

/// Список всех таблиц.

/// </summary>

private WaveTable[] waveTables;

/// <summary>

/// Текущая таблица.

/// </summary>

private WaveTable waveTable;

/// <summary>

/// Общее количество таблиц.

/// </summary>

private int waveTablesAmount;

/// <summary>

/// Текущий инкремент фазы.

/// </summary>

private float phaseIncrement;

/// <summary>

/// Представляет собой функцию-генератор для создания таблицы.

/// </summary>

/// <param name="phase">Текущая фаза.</param>

/// <param name="freq">Минимальная частота, на которой будет проигрываться создаваемая таблица.</param>

/// <param name="maxFreq">Максимальная частота, на которой будет проигрываться создаваемая таблица.</param>

/// <returns>Значение функции-генератора при заданных параметрах.</returns>

public delegate double GeneratorFunction(double phase, double freq, double maxFreq);

/// <summary>

/// Инициализирует новый объект типа WaveTableOscillator с заданными параметрами.

/// </summary>

/// <param name="generator">Функция-генератор.</param>

/// <param name="startFrequency">Минимальная частота, на которой планируется генерировать звук.</param>

/// <param name="endFrequency">Максимальная частота, на которой планируется генерировать звук.</param>

public WaveTableOscillator(

GeneratorFunction generator,

double startFrequency,

double endFrequency)

{

waveTablesAmount = (int)(Math.Floor(Math.Log(endFrequency / startFrequency, 2)) + 1);

waveTables = new WaveTable[waveTablesAmount];

for (int i = 0; i < waveTables.Length; ++i)

{

var frequency = startFrequency \* (1 << i);

var samples = PrepareSamples(generator, frequency, DefaultSampleRate / 2);

waveTables[i] = new WaveTable

{

Length = samples.Length,

PhaseIncrement = (float)(frequency / DefaultSampleRate),

Samples = samples,

};

}

Normalize();

}

/// <summary>

/// Метод, подготавливающий массив сэмплов.

/// </summary>

/// <param name="generator">Функция-генератор.</param>

/// <param name="startFrequency">Частота, на которой генерируется этот массив сэмплов.</param>

/// <param name="endFrequency">Максимальная частота, на которой планируется генерировать звук.</param>

/// <returns>Массив сэмплов.</returns>

private static float[] PrepareSamples(GeneratorFunction generator, double freq, double maxFreq)

{

var length = (int)DefaultSampleRate;

var result = new float[length];

for (int i = 0; i < result.Length; ++i)

result[i] = (float)generator((double)i / result.Length, freq, maxFreq);

return result;

}

/// <summary>

/// Метод, нормализующий все таблицы этого осциллятора.

/// </summary>

private void Normalize()

{

float maxAbs = 0;

foreach (var wt in waveTables)

foreach (var sample in wt.Samples)

if (Math.Abs(sample) > maxAbs)

maxAbs = Math.Abs(sample);

foreach (var wt in waveTables)

for (int i = 0; i < wt.Samples.Length; ++i)

wt.Samples[i] /= maxAbs;

}

/// <summary>

/// Инициализирует пустой объект типа WaveTableOscillator.

/// </summary>

private WaveTableOscillator()

{

}

/// <summary>

/// Клонирует текущую таблицу сэмплов.

/// </summary>

/// <returns>Новая таблица сэмплов.</returns>

public WaveTableOscillator Clone()

{

return new WaveTableOscillator

{

waveTables = waveTables,

waveTablesAmount = waveTables.Length

};

}

/// <summary>

/// Устанавливает новое значение инкремента фазы.

/// </summary>

/// <param name="phaseIncrement">Инкремент фазы.</param>

public void SetPhaseIncrement(double phaseIncrement)

{

this.phaseIncrement = (float)phaseIncrement;

int wtIndex = 0;

while (wtIndex < waveTablesAmount - 1 && phaseIncrement > waveTables[wtIndex].PhaseIncrement)

wtIndex += 1;

waveTable = waveTables[wtIndex];

}

/// <summary>

/// Метод, возвращающий сэмпл по заданной фазе.

/// </summary>

/// <param name="phase">Фаза.</param>

/// <returns>Выходной сэмпл.</returns>

public float Process(float phase)

{

if (phase >= 1)

phase -= 1;

var waveTable = this.waveTable;

if (waveTable == null)

return 0;

var tableLength = waveTable.Length;

float temp = phase \* tableLength;

int leftIndex = (int)temp;

int rightIndex = leftIndex + 1;

if (rightIndex == waveTable.Length)

rightIndex = 0;

float rightCoeff = temp - leftIndex;

float leftCoeff = 1 - rightCoeff;

return leftCoeff \* waveTable.Samples[leftIndex] + rightCoeff \* waveTable.Samples[rightIndex];

}

/// <summary>

/// Метод, записывающий объект WaveTableOscillator в поток.

/// </summary>

/// <param name="stream"></param>

/// <param name="obj"></param>

public static void Serialize(Stream stream, WaveTableOscillator obj)

{

using (var writer = new BinaryWriter(stream, System.Text.Encoding.Default, true))

{

writer.Write(obj.waveTablesAmount);

foreach (var waveTable in obj.waveTables)

{

writer.Write(waveTable.Length);

writer.Write(waveTable.PhaseIncrement);

foreach (var sample in waveTable.Samples)

writer.Write(sample);

}

}

}

/// <summary>

/// Метод, восстанавливает объект WaveTableOscillator из потока.

/// </summary>

/// <param name="stream"></param>

/// <returns></returns>

public static WaveTableOscillator Deserialize(Stream stream)

{

using (var reader = new BinaryReader(stream, System.Text.Encoding.Default, true))

{

var waveTablesAmount = reader.ReadInt32();

var waveTables = new WaveTable[waveTablesAmount];

for (int i = 0; i < waveTables.Length; ++i)

{

var length = reader.ReadInt32();

var phaseIncrement = reader.ReadSingle();

var samples = new float[length];

for (int j = 0; j < samples.Length; ++j)

samples[j] = reader.ReadSingle();

waveTables[i] = new WaveTable

{

Length = length,

PhaseIncrement = phaseIncrement,

Samples = samples,

};

}

return new WaveTableOscillator

{

waveTablesAmount = waveTablesAmount,

waveTables = waveTables,

};

}

}

}

}

## Plugin Parts\AudioProcessor.cs

using Jacobi.Vst.Core;

using Jacobi.Vst.Framework.Plugin;

using System.Threading;

namespace MultimodSynth

{

/// <summary>

/// Реализация интерфейса IVstPluginAudioProcessor.

/// </summary>

class AudioProcessor : VstPluginAudioProcessorBase

{

/// <summary>

/// Ссылка на плагин, которому принадлежит этот компонент.

/// </summary>

private Plugin plugin;

/// <summary>

/// Ссылка на объект, предстающий собой всю цепочку создания и обработки звука.

/// </summary>

public Routing Routing { get; private set; }

/// <summary>

/// Мьютекс, используемый для синхронизации UI-потока и основного потока.

/// </summary>

public Mutex ProcessingMutex { get; set; } = new Mutex();

/// <summary>

/// Текущая частота дискретизации.

/// </summary>

public override float SampleRate

{

get => Routing.SampleRate;

set => Routing.SampleRate = value;

}

/// <summary>

/// Инициализирует новых объект типа AudioProcessor, принадлежащий заданному плагину.

/// </summary>

/// <param name="plugin"></param>

public AudioProcessor(Plugin plugin) : base(0, 2, 0)

{

this.plugin = plugin;

Routing = new Routing(plugin);

}

/// <summary>

/// Метод, обрабатывающий входные данные, поступающие от плагина, и генерирующий новые выходные данные.

/// </summary>

/// <param name="inChannels">Входные каналы.</param>

/// <param name="outChannels">Выходные каналы.</param>

public override void Process(VstAudioBuffer[] inChannels, VstAudioBuffer[] outChannels)

{

ProcessingMutex.WaitOne();

var outputLeft = outChannels[0];

var outputRight = outChannels[1];

for (int i = 0; i < outputLeft.SampleCount; ++i)

{

Routing.Process(out var left, out var right);

outputLeft[i] = left;

outputRight[i] = right;

}

ProcessingMutex.ReleaseMutex();

}

}

}

## Plugin Parts\MidiProcessor.cs

using Jacobi.Vst.Core;

using Jacobi.Vst.Framework;

using System;

namespace MultimodSynth

{

/// <summary>

/// Реализация интерефейса IVstMidiProcessor.

/// </summary>

class MidiProcessor : IVstMidiProcessor

{

/// <summary>

/// Ссылка на плагин, которому принадлежит этот компонент.

/// </summary>

private Plugin plugin;

/// <summary>

/// Инициализирует новых объект типа MidiProcessor, принадлежащий заданному плагину.

/// </summary>

/// <param name="plugin"></param>

public MidiProcessor(Plugin plugin)

{

this.plugin = plugin;

}

/// <summary>

/// Количество каналов.

/// </summary>

public int ChannelCount => 16;

/// <summary>

/// Обработка новых входных данных.

/// </summary>

/// <param name="events"></param>

public void Process(VstEventCollection events)

{

foreach (var e in events)

{

if (e.EventType == VstEventTypes.MidiEvent)

{

var midiEvent = (VstMidiEvent)e;

byte firstByte = midiEvent.Data[0];

if (0x80 <= firstByte && firstByte < 0x90) // Note off

{

pressedNotesCount -= 1;

OnNoteOff(midiEvent.Data[1], midiEvent.Data[2]);

}

else if (0x90 <= firstByte && firstByte < 0xA0) // Note on

{

pressedNotesCount += 1;

OnNoteOn(midiEvent.Data[1], midiEvent.Data[2]);

}

}

}

}

/// <summary>

/// Событие, возникающее при отпускании клавиши.

/// </summary>

public event EventHandler<MidiNoteEventArgs> NoteOff;

/// <summary>

/// Событие, возникающее при нажатии клавиши.

/// </summary>

public event EventHandler<MidiNoteEventArgs> NoteOn;

/// <summary>

/// Текущее количество нажатых клавиш.

/// </summary>

private int pressedNotesCount = 0;

/// <summary>

/// Метод, вызывающий событие NoteOff с заданными параметрами.

/// </summary>

/// <param name="noteNo"></param>

/// <param name="velocity"></param>

private void OnNoteOff(byte noteNo, byte velocity)

{

NoteOff?.Invoke(this, new MidiNoteEventArgs(noteNo, velocity, pressedNotesCount));

}

/// <summary>

/// Метод, вызывающий событие NoteOn с заданными параметрами.

/// </summary>

/// <param name="noteNo"></param>

/// <param name="velocity"></param>

private void OnNoteOn(byte noteNo, byte velocity)

{

NoteOn?.Invoke(this, new MidiNoteEventArgs(noteNo, velocity, pressedNotesCount));

}

/// <summary>

/// Метод, "нажимающий" клавишу из UI.

/// </summary>

/// <param name="noteNo"></param>

/// <param name="velocity"></param>

public void PressNoteFromUI(byte noteNo, byte velocity)

{

plugin.AudioProcessor.ProcessingMutex.WaitOne();

OnNoteOn(noteNo, velocity);

plugin.AudioProcessor.ProcessingMutex.ReleaseMutex();

}

/// <summary>

/// Метод, "отпускающий" клавишу из UI.

/// </summary>

/// <param name="noteNo"></param>

/// <param name="velocity"></param>

public void ReleaseNoteFromUI(byte noteNo, byte velocity)

{

plugin.AudioProcessor.ProcessingMutex.WaitOne();

OnNoteOff(noteNo, velocity);

plugin.AudioProcessor.ProcessingMutex.ReleaseMutex();

}

}

/// <summary>

/// Аргумент событий MidiProcessor.NoteOn и MidiProcessor.NoteOff.

/// </summary>

public class MidiNoteEventArgs : EventArgs

{

/// <summary>

/// Инициализирует новый объект типа MidiNoteEventArgs с заданными параметрами.

/// </summary>

/// <param name="noteNo"></param>

/// <param name="velocity"></param>

/// <param name="pressedNotesCount"></param>

public MidiNoteEventArgs(byte noteNo, byte velocity, int pressedNotesCount)

{

Note = new MidiNote { NoteNo = noteNo, Velocity = velocity };

PressedNotesCount = pressedNotesCount;

}

/// <summary>

/// Количество нажатых нот.

/// </summary>

public int PressedNotesCount { get; protected set; }

/// <summary>

/// Играемая нота.

/// </summary>

public MidiNote Note { get; protected set; }

}

/// <summary>

/// Представляет собой одну ноту.

/// </summary>

public struct MidiNote

{

/// <summary>

/// Номер ноты.

/// </summary>

public byte NoteNo { get; set; }

/// <summary>

/// Сила нажатия клавиши.

/// </summary>

public byte Velocity { get; set; }

}

}

## Plugin Parts\ParameterFactory.cs

using Jacobi.Vst.Framework;

using System;

using System.Linq;

namespace MultimodSynth

{

/// <summary>

/// Представляет собой фабрику для удобного создания параметров.

/// </summary>

class ParameterFactory

{

/// <summary>

/// Ссылка на плагин, которому принадлежит этот компонент.

/// </summary>

private Plugin plugin;

/// <summary>

/// Ссылка на компонент PluginPrograms данного плагина.

/// </summary>

private PluginPrograms programs;

/// <summary>

/// Категория параметров.

/// </summary>

private VstParameterCategory parameterCategory;

/// <summary>

/// Префикс названия параметра.

/// </summary>

private string namePrefix;

/// <summary>

/// Инициализирует новый объект типа ParameterFactory, принадлежащий заданному

/// плагину и имеющий переданную категорию и префикс имени параметра.

/// </summary>

/// <param name="plugin"></param>

/// <param name="category"></param>

/// <param name="namePrefix"></param>

public ParameterFactory(Plugin plugin, string category = "", string namePrefix = "")

{

this.plugin = plugin;

programs = plugin.Programs;

this.namePrefix = namePrefix;

parameterCategory = new VstParameterCategory() { Name = category };

if (programs.ParameterCategories.All(x => x.Name != category))

programs.ParameterCategories.Add(parameterCategory);

}

/// <summary>

/// Создаёт новый параметр и регистрирует его.

/// </summary>

/// <param name="name">Имя параметра.</param>

/// <param name="defaultValue">Начальное значение.</param>

/// <param name="valueChangedHandler">Обработчик изменения значения параметра.</param>

/// <returns>Объект, управляющий созданным параметром.</returns>

public VstParameterManager CreateParameterManager(

string name = null,

float defaultValue = 0,

Action<float> valueChangedHandler = null)

{

var parameterInfo = new VstParameterInfo()

{

Category = parameterCategory,

Name = namePrefix + name,

DefaultValue = defaultValue,

};

VstParameterNormalizationInfo.AttachTo(parameterInfo);

programs.ParameterInfos.Add(parameterInfo);

var manager = new VstParameterManager(parameterInfo);

plugin.Opened += (sender, e) =>

{

manager.HostAutomation = plugin.Host.GetInstance<IVstHostAutomation>();

};

if (valueChangedHandler != null)

manager.PropertyChanged += (sender, e) =>

{

if (e.PropertyName == "CurrentValue" || e.PropertyName == "ActiveParameter")

{

plugin.AudioProcessor.ProcessingMutex.WaitOne();

valueChangedHandler(manager.CurrentValue);

plugin.AudioProcessor.ProcessingMutex.ReleaseMutex();

}

};

return manager;

}

}

}

## Plugin Parts\Plugin.cs

using Jacobi.Vst.Core;

using Jacobi.Vst.Framework;

using Jacobi.Vst.Framework.Plugin;

namespace MultimodSynth

{

/// <summary>

/// Представляет собой плагин.

/// </summary>

class Plugin : VstPluginWithInterfaceManagerBase

{

/// <summary>

/// Инициализирует новый объект типа Plugin.

/// </summary>

public Plugin() : base(

"multimod synth",

new VstProductInfo("multimod synth", "pema4", 1000),

VstPluginCategory.Synth,

VstPluginCapabilities.None,

0,

new FourCharacterCode("MSYN").ToInt32())

{

}

/// <summary>

/// Ссылка на компонент AudioProcessor плагина.

/// </summary>

public AudioProcessor AudioProcessor => GetInstance<AudioProcessor>();

/// <summary>

/// Ссылка на компонент MidiProcessor плагина.

/// </summary>

public MidiProcessor MidiProcessor => GetInstance<MidiProcessor>();

/// <summary>

/// Ссылка на компонент PluginPrograms плагина.

/// </summary>

public PluginPrograms Programs => GetInstance<PluginPrograms>();

/// <summary>

/// Метод, возвращающий реализацию интерфейса IVstPluginAudioProcessor.

/// </summary>

/// <param name="instance"></param>

/// <returns></returns>

protected override IVstPluginAudioProcessor CreateAudioProcessor(IVstPluginAudioProcessor instance)

{

if (instance == null)

{

return new AudioProcessor(this);

}

return base.CreateAudioProcessor(instance);

}

/// <summary>

/// Метод, возвращающий реализацию интерфейса IVstNidiProcessor.

/// </summary>

/// <param name="instance"></param>

/// <returns></returns>

protected override IVstMidiProcessor CreateMidiProcessor(IVstMidiProcessor instance)

{

if (instance == null)

{

return new MidiProcessor(this);

}

return base.CreateMidiProcessor(instance);

}

/// <summary>

/// Метод, возвращающий реализацию интерфейса IVstPluginPrograms.

/// </summary>

/// <param name="instance"></param>

/// <returns></returns>

protected override IVstPluginPrograms CreatePrograms(IVstPluginPrograms instance)

{

if (instance == null)

{

return new PluginPrograms(this);

}

return base.CreatePrograms(instance);

}

/// <summary>

/// Метод, возвращающий реализацию интерфейса IVstPluginEditor.

/// </summary>

/// <param name="instance"></param>

/// <returns></returns>

protected override IVstPluginEditor CreateEditor(IVstPluginEditor instance)

{

if (instance == null)

{

return new PluginEditor(this);

}

return base.CreateEditor(instance);

}

}

}

## Plugin Parts\PluginCommandStub.cs

using Jacobi.Vst.Framework;

using Jacobi.Vst.Framework.Plugin;

namespace MultimodSynth

{

/// <summary>

/// Публичный класс, являющийся точкой связывания managed и unmanaged кода.

/// </summary>

public class PluginCommandStub : StdPluginCommandStub

{

/// <summary>

/// Возвращает новый объект типа IVstPlugin.

/// </summary>

/// <returns></returns>

protected override IVstPlugin CreatePluginInstance()

{

return new Plugin();

}

}

}

## Plugin Parts\PluginEditor.cs

using MultimodSynth.UI;

using Jacobi.Vst.Core;

using Jacobi.Vst.Framework;

using System;

using System.Drawing;

using Brushes = System.Windows.Media.Brushes;

using System.Windows;

using System.Windows.Interop;

using System.Windows.Media;

namespace MultimodSynth

{

/// <summary>

/// Реализация интерфейса IVstPluginEditor, отвечающего за создание пользовательского интерфейса.

/// </summary>

class PluginEditor : IVstPluginEditor

{

/// <summary>

/// Ссылка на плагин, которому принадлежит этот компонент.

/// </summary>

private Plugin plugin;

/// <summary>

/// Объект типа HwndSource.

/// </summary>

private HwndSource hwndSource;

/// <summary>

/// Текущие границы окна редактора.

/// </summary>

private Rectangle? bounds;

/// <summary>

/// Ссылка на окно редактора.

/// </summary>

private EditorView instance;

/// <summary>

/// Инициализирует новый объект типа PluginEditor, принадлежащий переданному плагину.

/// </summary>

/// <param name="plugin"></param>

public PluginEditor(Plugin plugin)

{

this.plugin = plugin;

}

/// <summary>

/// Тип переключателей редактора.

/// </summary>

public VstKnobMode KnobMode { get; set; }

/// <summary>

/// Обрабатывает нажатия клавиш (не используется).

/// </summary>

/// <param name="ascii"></param>

/// <param name="virtualKey"></param>

/// <param name="modifers"></param>

/// <returns></returns>

public bool KeyDown(byte ascii, VstVirtualKey virtualKey, VstModifierKeys modifers)

{

return false;

}

/// <summary>

/// Обрабатывает отпускания клавиш клавиатуры (не используется).

/// </summary>

/// <param name="ascii"></param>

/// <param name="virtualKey"></param>

/// <param name="modifers"></param>

/// <returns></returns>

public bool KeyUp(byte ascii, VstVirtualKey virtualKey, VstModifierKeys modifers)

{

return false;

}

/// <summary>

/// Метод, вызываемый хостом, когда процессор бездействует (не используется).

/// </summary>

public void ProcessIdle()

{

}

/// <summary>

/// Возвращает размеры текущего окна.

/// </summary>

public Rectangle Bounds

{

get

{

if (bounds != null)

return bounds.Value;

if (instance == null)

{

instance = new EditorView();

instance.BindToPlugin(plugin);

}

var size = GetElementPixelSize(instance);

bounds = new Rectangle(0, 0, (int)size.Width, (int)size.Height);

return bounds.Value;

}

}

/// <summary>

/// Открывает и прикрепляет элемент управления к переданному hWnd.

/// </summary>

public void Open(IntPtr hWnd)

{

if (instance == null)

{

instance = new EditorView();

instance.BindToPlugin(plugin);

}

HwndSourceParameters hwndParams = new HwndSourceParameters("Better Synth");

hwndParams.ParentWindow = hWnd;

hwndParams.Height = Bounds.Height;

hwndParams.Width = Bounds.Width;

hwndParams.WindowStyle = 0x10000000 | 0x40000000; // WS\_VISIBLE|WS\_CHILD

hwndSource = new HwndSource(hwndParams);

hwndSource.RootVisual = instance;

}

/// <summary>

/// Возвращает размер элемента в экранных пикселях.

/// </summary>

/// <param name="element"></param>

/// <returns></returns>

/// <seealso cref="https://stackoverflow.com/questions/3286175/how-do-i-convert-a-wpf-size-to-physical-pixels"/>

private System.Windows.Size GetElementPixelSize(UIElement element)

{

Matrix transformToDevice;

var source = PresentationSource.FromVisual(element);

if (source != null)

transformToDevice = source.CompositionTarget.TransformToDevice;

else

using (var sauce = new HwndSource(new HwndSourceParameters()))

transformToDevice = sauce.CompositionTarget.TransformToDevice;

if (element.DesiredSize == new System.Windows.Size())

element.Measure(new System.Windows.Size(double.PositiveInfinity, double.PositiveInfinity));

return (System.Windows.Size)transformToDevice.Transform((Vector)element.DesiredSize);

}

/// <summary>

/// Закрывает редактор.

/// </summary>

public void Close()

{

if (hwndSource != null)

{

hwndSource.Dispose();

hwndSource = null;

}

instance = null;

}

}

}

## Plugin Parts\PluginPrograms.cs

using Jacobi.Vst.Framework;

using Jacobi.Vst.Framework.Plugin;

using System.Linq;

namespace MultimodSynth

{

/// <summary>

/// Реализация интерфейса IVstPluginPrograms, отвечающего за работу с программами.

/// </summary>

class PluginPrograms : VstPluginProgramsBase

{

/// <summary>

/// Ссылка на плагин, которому принадлежит этот компонент.

/// </summary>

private Plugin plugin;

/// <summary>

/// Инициализирует новых объект типа PluginPrograms, принадлежащий заданному плагину.

/// </summary>

/// <param name="plugin"></param>

public PluginPrograms(Plugin plugin)

{

this.plugin = plugin;

}

/// <summary>

/// Ссылка на коллекцию данных о параметрах.

/// </summary>

public VstParameterInfoCollection ParameterInfos { get; protected set; } = new VstParameterInfoCollection();

/// <summary>

/// Ссылка на коллекцию категорий параметров.

/// </summary>

public VstParameterCategoryCollection ParameterCategories { get; protected set; } = new VstParameterCategoryCollection();

/// <summary>

/// Создает коллекцию программ.

/// </summary>

/// <returns></returns>

protected override VstProgramCollection CreateProgramCollection()

{

VstProgramCollection programs = new VstProgramCollection();

VstProgram defaultProgram = new VstProgram(ParameterCategories);

defaultProgram.Parameters.AddRange(ParameterInfos.Select(x => new VstParameter(x)));

defaultProgram.Name = "Default";

programs.Add(defaultProgram);

return programs;

}

}

}

## UI\BipolarKnob.cs

using System;

using System.Windows;

namespace MultimodSynth.UI

{

/// <summary>

/// Представляет собой вариант ручки, стандартное значение которой находится на середине.

/// </summary>

public class BipolarKnob : Knob

{

/// <summary>

/// Инициализирует новый объект типа BipolarKnob.

/// </summary>

public BipolarKnob()

{

coloredArcFigure.StartPoint = new System.Windows.Point(20, 4);

}

/// <summary>

/// Обрабатывает вращение элементов.

/// </summary>

/// <param name="normalizedValue"></param>

protected override void Rotate(double normalizedValue)

{

var angle = Math.PI \* (1.25 - 1.5 \* normalizedValue);

var newPoint = new Point

{

X = 20 + 16 \* Math.Cos(angle),

Y = 20 - 16 \* Math.Sin(angle),

};

if (angle < Math.PI / 2)

coloredArc.SweepDirection = System.Windows.Media.SweepDirection.Clockwise;

else

coloredArc.SweepDirection = System.Windows.Media.SweepDirection.Counterclockwise;

coloredArc.Point = newPoint;

rotateTransform.Angle = 270 \* normalizedValue;

}

}

}

## UI\EditorView.xaml

<UserControl x:Class="MultimodSynth.UI.EditorView"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:MultimodSynth.UI"

mc:Ignorable="d"

Width="Auto" Height="Auto"

x:Name="viewControl">

<UserControl.Resources>

<SolidColorBrush x:Key="oscABackground" Color="#FFFFE7E7"/>

<SolidColorBrush x:Key="oscAKnobColor" Color="Red"/>

<SolidColorBrush x:Key="oscBBackground" Color="#FFFEFEFE"/>

<SolidColorBrush x:Key="oscBKnobColor" Color="YellowGreen"/>

<SolidColorBrush x:Key="filterBackground" Color="#FFE0E0FF"/>

<SolidColorBrush x:Key="filterKnobColor" Color="Blue"/>

<SolidColorBrush x:Key="otherBorderColor" Color="Gray"/>

<SolidColorBrush x:Key="otherBackground" Color="#EFEFEF"/>

<SolidColorBrush x:Key="masterKnobColor" Color="#47BAEB"/>

<SolidColorBrush x:Key="distortionBackground" Color="#FFFFDEEE"/>

<SolidColorBrush x:Key="distortionKnobColor" Color="HotPink"/>

<SolidColorBrush x:Key="delayBackground" Color="#FFFFF9EE"/>

<SolidColorBrush x:Key="delayKnobColor" Color="Orange"/>

<SolidColorBrush x:Key="keyOctaveStartColor" Color="Gray"/>

<SolidColorBrush x:Key="whiteKeyColor" Color="LightGray"/>

<LinearGradientBrush x:Key="blackKeyColor">

<GradientStop Color="#303030" Offset="0"/>

<GradientStop Color="Black" Offset="1"/>

</LinearGradientBrush>

<SolidColorBrush x:Key="buttonBorder" Color="Gray"/>

<SolidColorBrush x:Key="buttonBackground" Color="#EFEFEF"/>

<SolidColorBrush x:Key="buttonPressedBackground" Color="#BFBFBF"/>

<Style x:Key="buttonStyle" TargetType="Button">

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="Button">

<Border Margin="3"

BorderBrush="{StaticResource otherBorderColor}"

BorderThickness="3"

Background="{StaticResource buttonBackground}"

SnapsToDevicePixels="true"

CornerRadius="3">

<ContentPresenter

HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"

SnapsToDevicePixels="{TemplateBinding SnapsToDevicePixels}"

VerticalAlignment="{TemplateBinding VerticalContentAlignment}"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

</UserControl.Resources>

<Grid Background="White">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="Auto"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Grid.Resources>

<Style TargetType="local:Knob">

<Setter Property="Effect">

<Setter.Value>

<DropShadowEffect Direction="315" BlurRadius="10" Opacity="0.5"/>

</Setter.Value>

</Setter>

</Style>

<Style TargetType="local:BipolarKnob">

<Setter Property="Effect">

<Setter.Value>

<DropShadowEffect Direction="315" BlurRadius="10" Opacity="0.5"/>

</Setter.Value>

</Setter>

</Style>

</Grid.Resources>

<!-- Osc A section -->

<Border Grid.Row="0" Grid.Column="0" Margin="3"

CornerRadius="3" BorderThickness="3"

BorderBrush="{StaticResource oscAKnobColor}"

Background="{StaticResource oscABackground}">

<Grid Width="200" Margin="0,0,0,5">

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<TextBlock Grid.Row="0" Grid.Column="0" Margin="3,0,0,0">

<Run Foreground="Red" FontWeight="DemiBold">OSC A</Run>

</TextBlock>

<!-- Pitch semi -->

<StackPanel Grid.Row="1" Grid.Column="0">

<local:BipolarKnob x:Name="APitchSemi" Margin="0,2,0,0"/>

<TextBlock Margin="0,-5,0,0"

TextAlignment="Center" Text="Semi"/>

</StackPanel>

<!-- Pitch fine -->

<StackPanel Grid.Row="1" Grid.Column="1">

<local:BipolarKnob x:Name="APitchFine" Margin="0,2,0,0"/>

<TextBlock Margin="0,-5,0,0"

TextAlignment="Center" Text="Fine"/>

</StackPanel>

<!-- Tembre -->

<StackPanel Grid.Row="1" Grid.Column="2">

<local:Knob x:Name="ATembre" Margin="0,2,0,0"/>

<TextBlock Margin="0,-5,0,0"

TextAlignment="Center" Text="Tembre"/>

</StackPanel>

<!-- Envelope amp -->

<StackPanel Grid.Row="1" Grid.Column="3">

<local:Knob x:Name="AAmp" Margin="0,2,0,0"/>

<TextBlock Margin="0,-5,0,0"

TextAlignment="Center" Text="Level"/>

</StackPanel>

<!-- Envelope attack time -->

<StackPanel Grid.Row="2" Grid.Column="0">

<local:Knob x:Name="AAttack" Margin="0,2,0,0"/>

<TextBlock Margin="0,-5,0,0"

TextAlignment="Center" Text="Attack"/>

</StackPanel>

<!-- Envelope decay time -->

<StackPanel Grid.Row="2" Grid.Column="1">

<local:Knob x:Name="ADecay" Margin="0,2,0,0"/>

<TextBlock Margin="0,-5,0,0"

TextAlignment="Center" Text="Decay"/>

</StackPanel>

<!-- Envelope sustain level -->

<StackPanel Grid.Row="2" Grid.Column="2">

<local:Knob x:Name="ASustain" Margin="0,2,0,0"/>

<TextBlock Margin="0,-5,0,0"

TextAlignment="Center" Text="Sustain"/>

</StackPanel>

<!-- Envelope adaptive release time -->

<StackPanel Grid.Row="2" Grid.Column="3">

<local:Knob x:Name="ARelease" Margin="0,2,0,0"/>

<TextBlock Margin="0,-5,0,0"

TextAlignment="Center" Text="Release"/>

</StackPanel>

<!-- Envelope attack curve -->

<StackPanel Grid.Row="3" Grid.Column="0" Grid.ColumnSpan="2">

<local:Knob x:Name="AAttackCurve" Margin="0,2,0,0">

<local:Knob.LayoutTransform>

<ScaleTransform ScaleX="0.8" ScaleY="0.8"/>

</local:Knob.LayoutTransform>

</local:Knob>

<TextBlock Margin="0,-5,0,0"

TextAlignment="Center" Text="Attack curve"/>

</StackPanel>

<!-- Envelope decay-release curve -->

<StackPanel Grid.Row="3" Grid.Column="2" Grid.ColumnSpan="2">

<local:Knob x:Name="ADecayReleaseCurve" Margin="0,2,0,0">

<local:Knob.LayoutTransform>

<ScaleTransform ScaleX="0.8" ScaleY="0.8"/>

</local:Knob.LayoutTransform>

</local:Knob>

<TextBlock Margin="0,-5,0,0"

TextAlignment="Center" Text="Dec-Rel curve"/>

</StackPanel>

</Grid>

</Border>

<!-- Osc B section -->

<Border Grid.Row="1" Grid.Column="0" Margin="3"

CornerRadius="3" BorderThickness="3"

BorderBrush="{StaticResource oscBKnobColor}"

Background="{StaticResource oscBBackground}">

<Grid Width="200" Margin="0,0,0,5">

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<TextBlock Grid.Row="0" Margin="3,0,0,0">

<Run Foreground="Green" FontWeight="DemiBold">OSC B</Run>

</TextBlock>

## UI\EditorView.xaml.cs

## UI\Knob.xaml.cs

# ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Номера листов (страниц) | | | | | Всего листов (страниц в докум.) | № документа | Входящий № сопроводительного докум. и дата | Подп. | Дата |
| Изм. | Измененных | Замененных | Новых | Аннулированных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |