
UNITE
2015
EUROPE
unity

UNITY'S AUDIO SYSTEM UNDER THE HOOD

WHO AM I?

Jan Marguc

Full-time audio programmer at Unity in Copenhagen

Worked at IO Interactive on the audio tech of Glacier2 used in Hitman Absolution (2012)

Worked at TC Electronic on audio plugins for the PowerCore and TDM platforms

Ex-demoscener and hobby-musician


unity
5

AGENDA

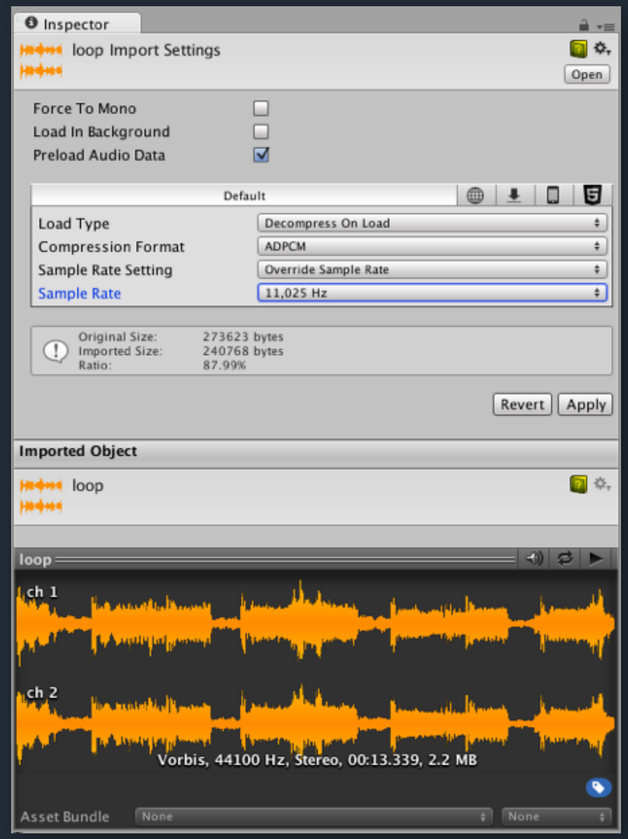
- Upgrade path from Unity 4 to 5
- Audio mixer
- Native audio plugins

THE NEW AUDIOCLIP

Audio data no longer inside AudioClip asset, but stored in a separate file.

AudioClip is now a lightweight reference object.

- No need to keep AudioClips in asset bundles.
- Fine-grained loading strategy control over audio



THE NEW AUDIOCLIP

Meta-data extracted at import time:

- About 550 bytes per AudioClip
- Length
- Channel count
- Sample rate
- Can be queried before the actual audio data has been loaded.

THE NEW AUDIOCLIP

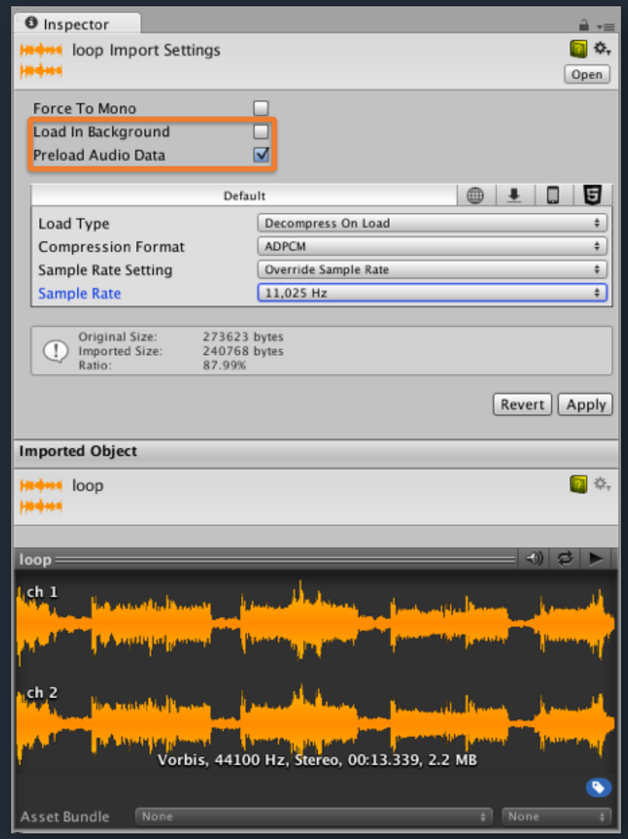
Better control over clip loading

Load on demand:

- `AudioSource.Play()/PlayOneShot()`
- `AudioClip.LoadAudioData()`
- `AudioClip.UnloadAudioData();`

Load in background:

- No stalling on main thread
- Also works for Decompress On Load

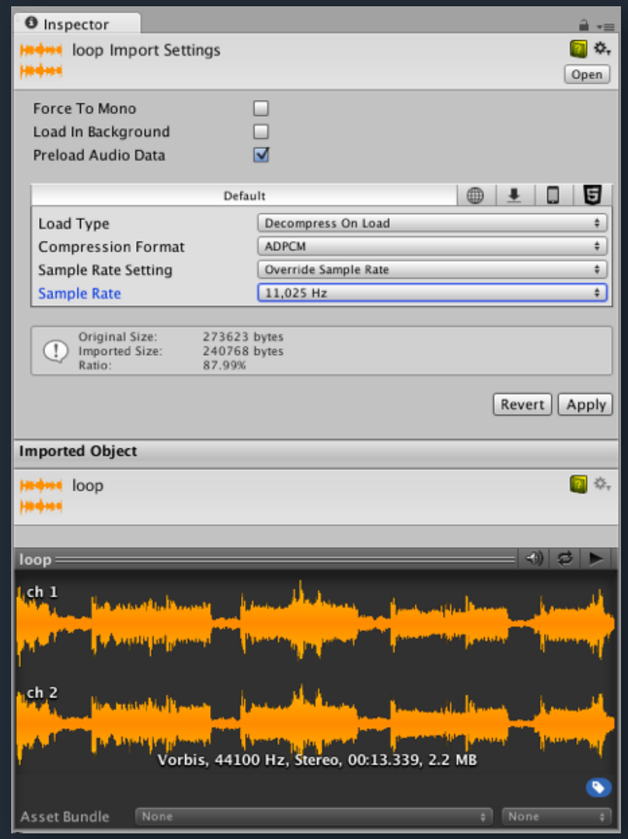


THE NEW AUDIOCLIP

Query load state via `AudioClip.loadState`:

- `AudioDataLoadState.Unloaded`
- `AudioDataLoadState.Loading`
- `AudioDataLoadState.Loaded`
- `AudioDataLoadState.Failed`

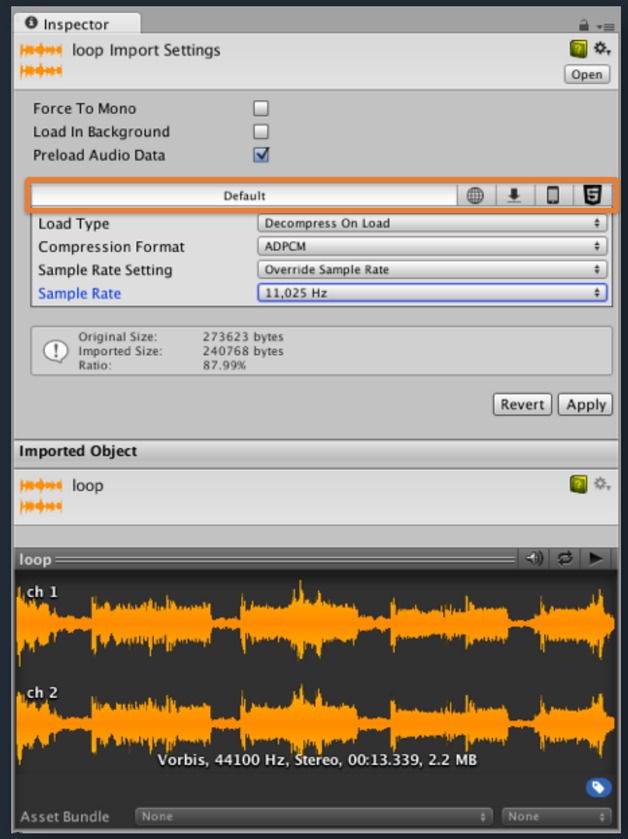
Loading behavior is now consistent between editor and players.



THE NEW AUDIOCLIP

Control over compression:

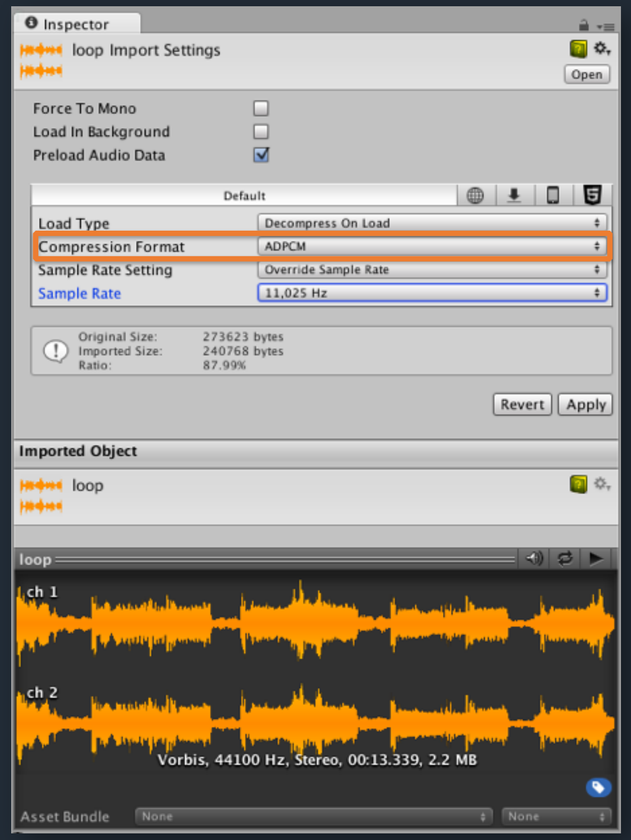
- Per platform override settings
- New codec options, especially for mobile and console.



THE NEW AUDIOCLIP

Compression types:

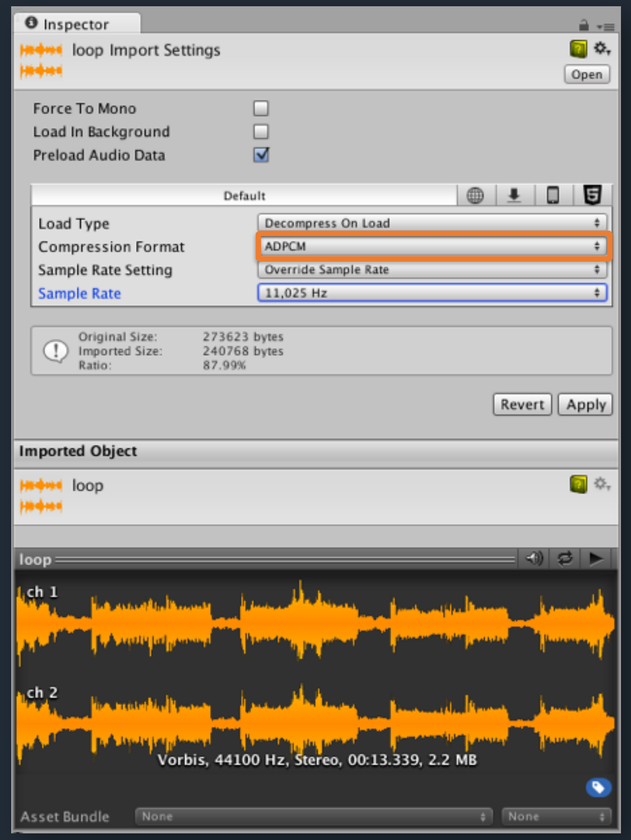
- PCM (uncompressed, previously called Native)
- Vorbis
  - Generally better than MP3 at low bit rates
- ADPCM



THE NEW AUDIOCLIP

ADPCM

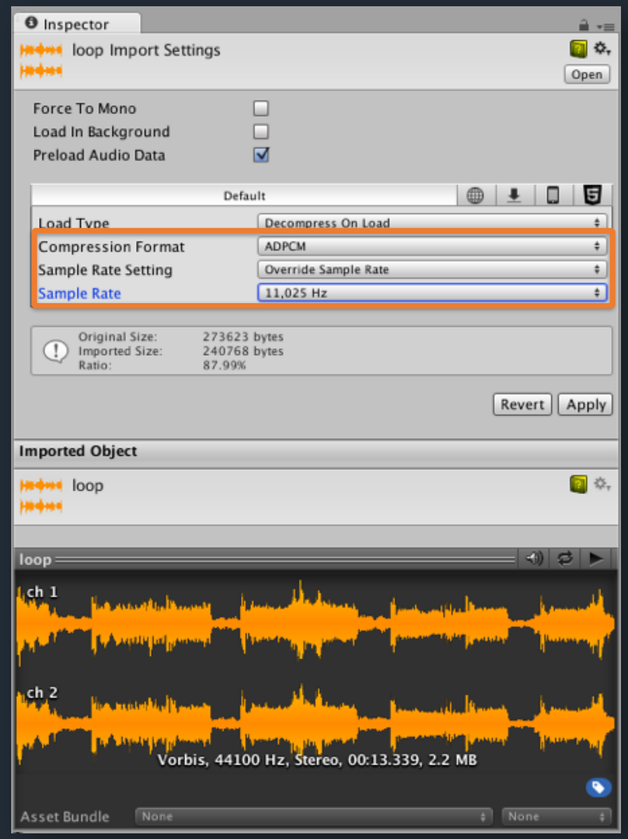
- Lightweight compression (fixed 3.5x ratio)
- Good compromise between CPU/memory usage and compression performance
- Good choice for noisy natural sounds (footsteps, weapons, collisions)



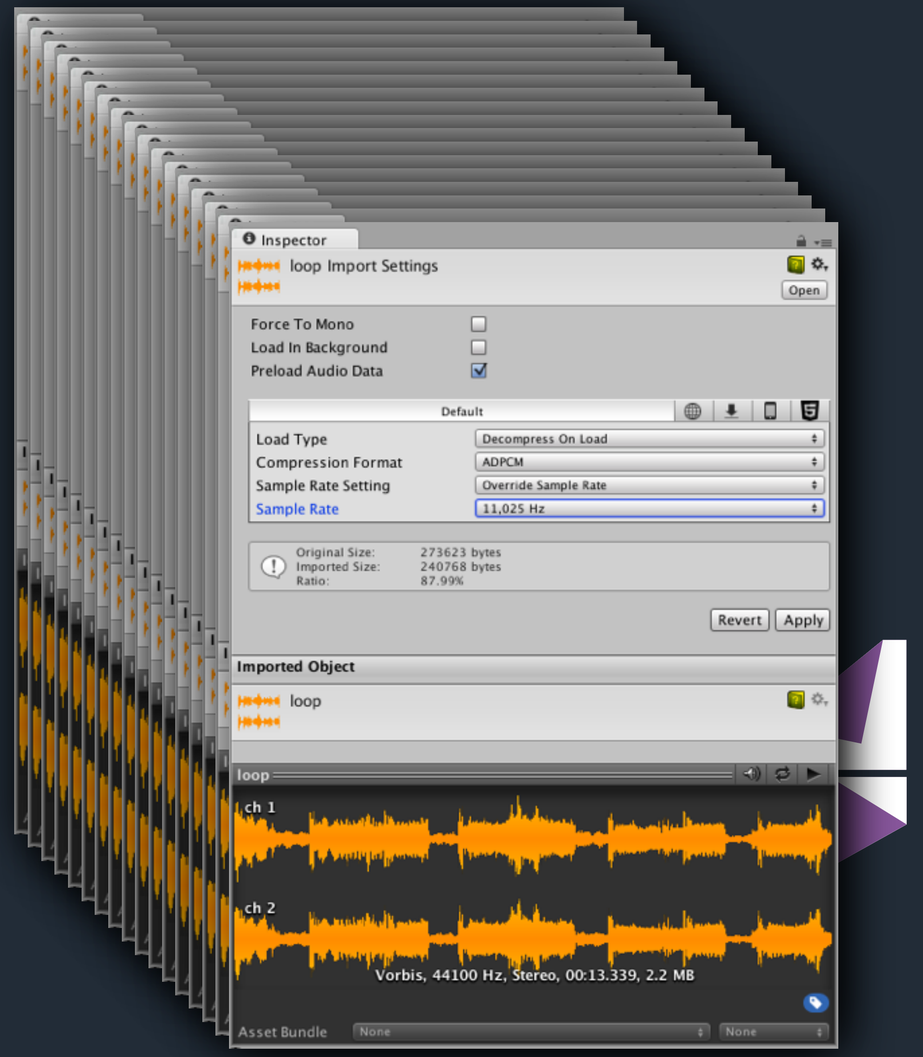
THE NEW AUDIOCLIP

Uncompressed and ADPCM allow encoding with custom sample rates

- Unity can pick the optimal sample rate
- Sample rate can also be chosen manually



THE NEW AUDIOCLIP



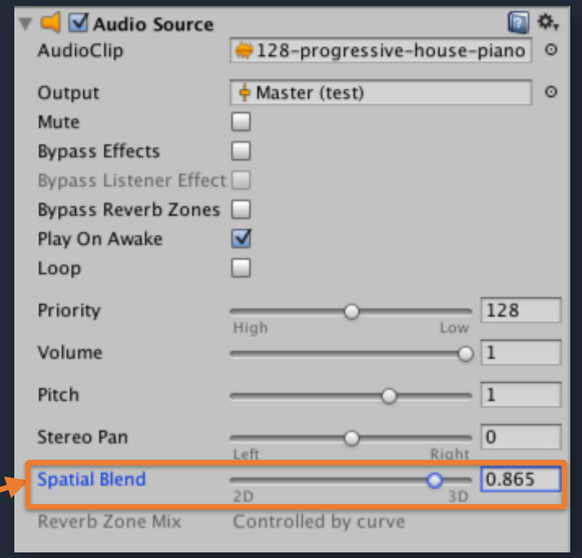
…multi-select editing!!

AUDIOSOURCE

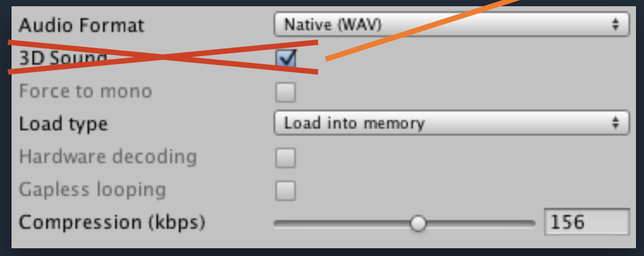
Removed 3D setting from AudioClip

- Voices can now be 3D or 2D (or a blend of both).
- Spatial Blend parameter/curve controls 2D/3D mixture

Auto update scenes with fixed AudioSource.clip assignments.



AudioSource in Unity 5



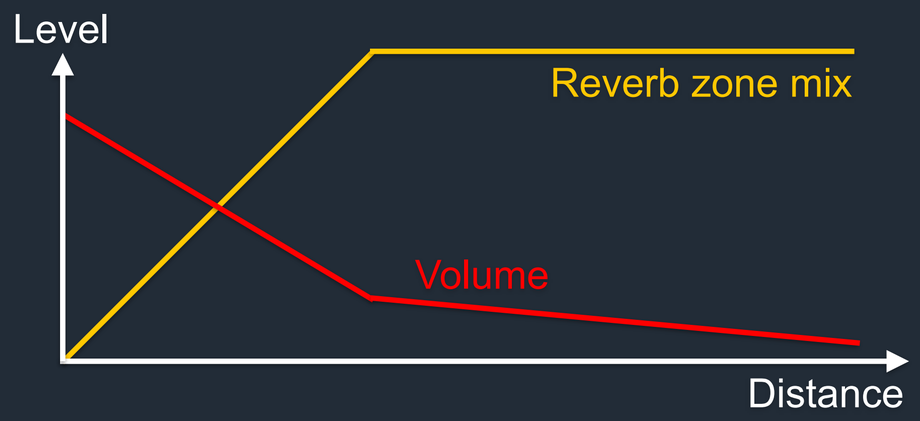
AudioClip in Unity 4

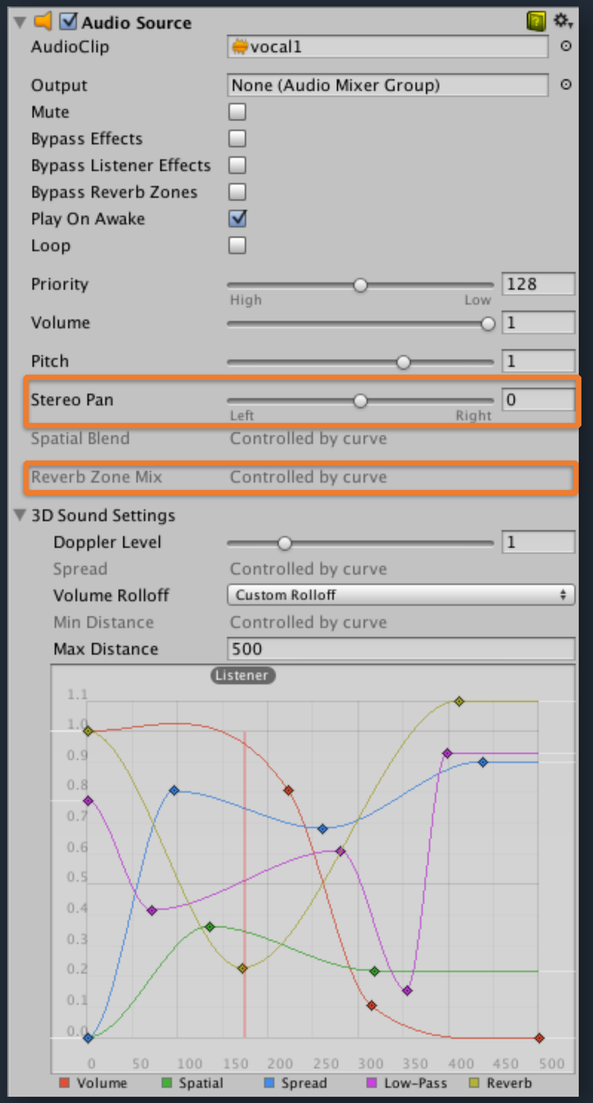
AUDIOSOURCE

Stereo Pan parameter for easy positioning of sounds in 2D games

Reverb Zone Mix parameter and curve

- Can go beyond 0 dB for near field / distant sound cross-over.
- Also works on 2D sounds.



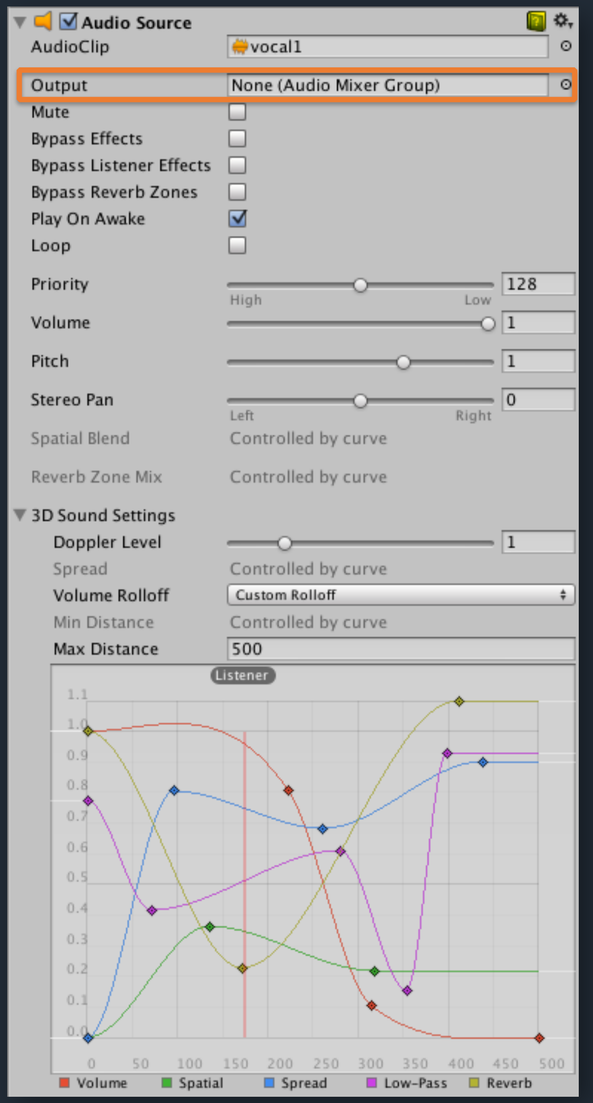


AUDIOSOURCE

Refined pause behaviour:

- One-shots are also paused on game or editor pause.
- `AudioSource.Pause()/UnPause()`

Output property for routing the signal to a mixer group

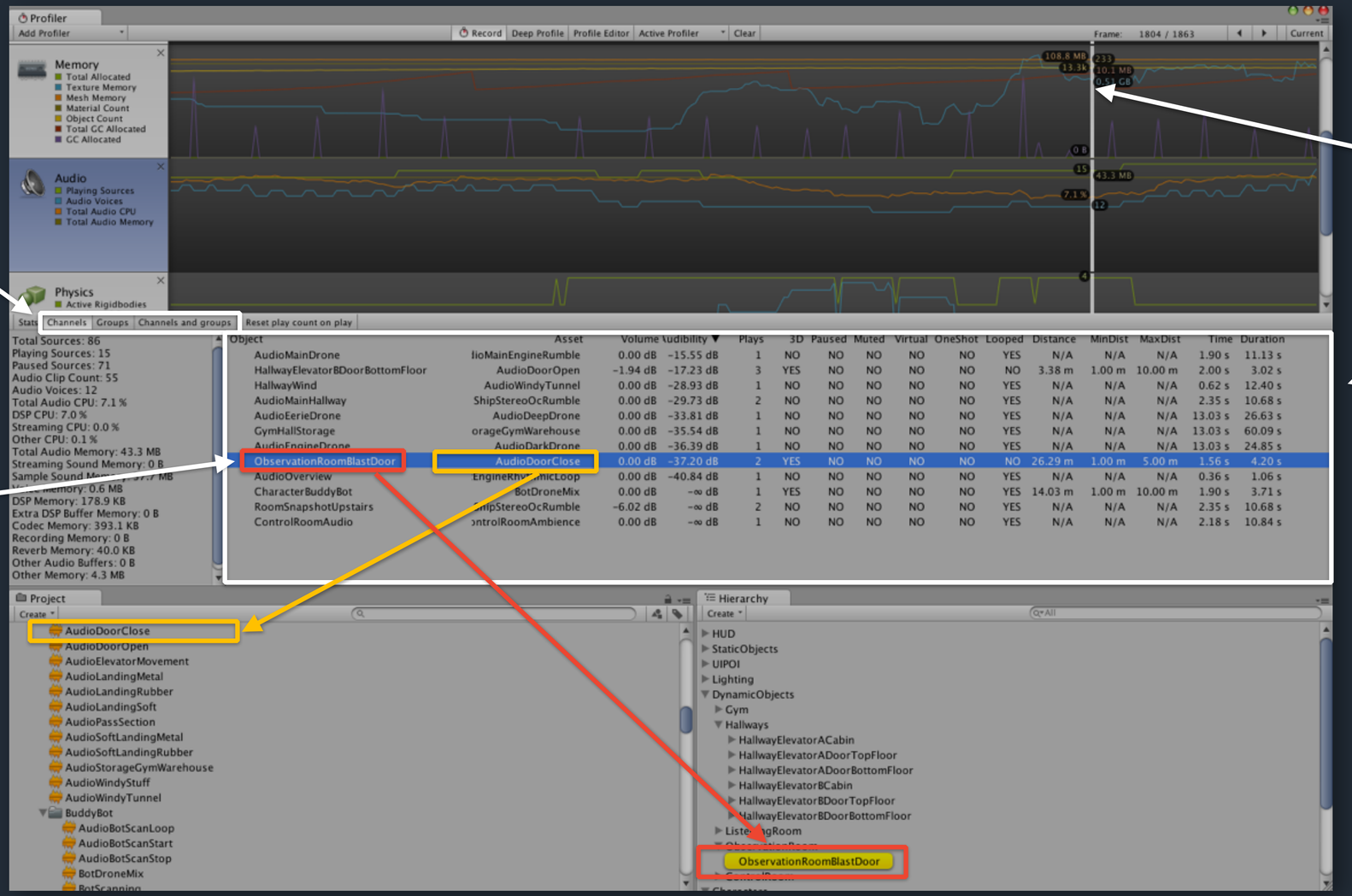


AUDIO PROFILER

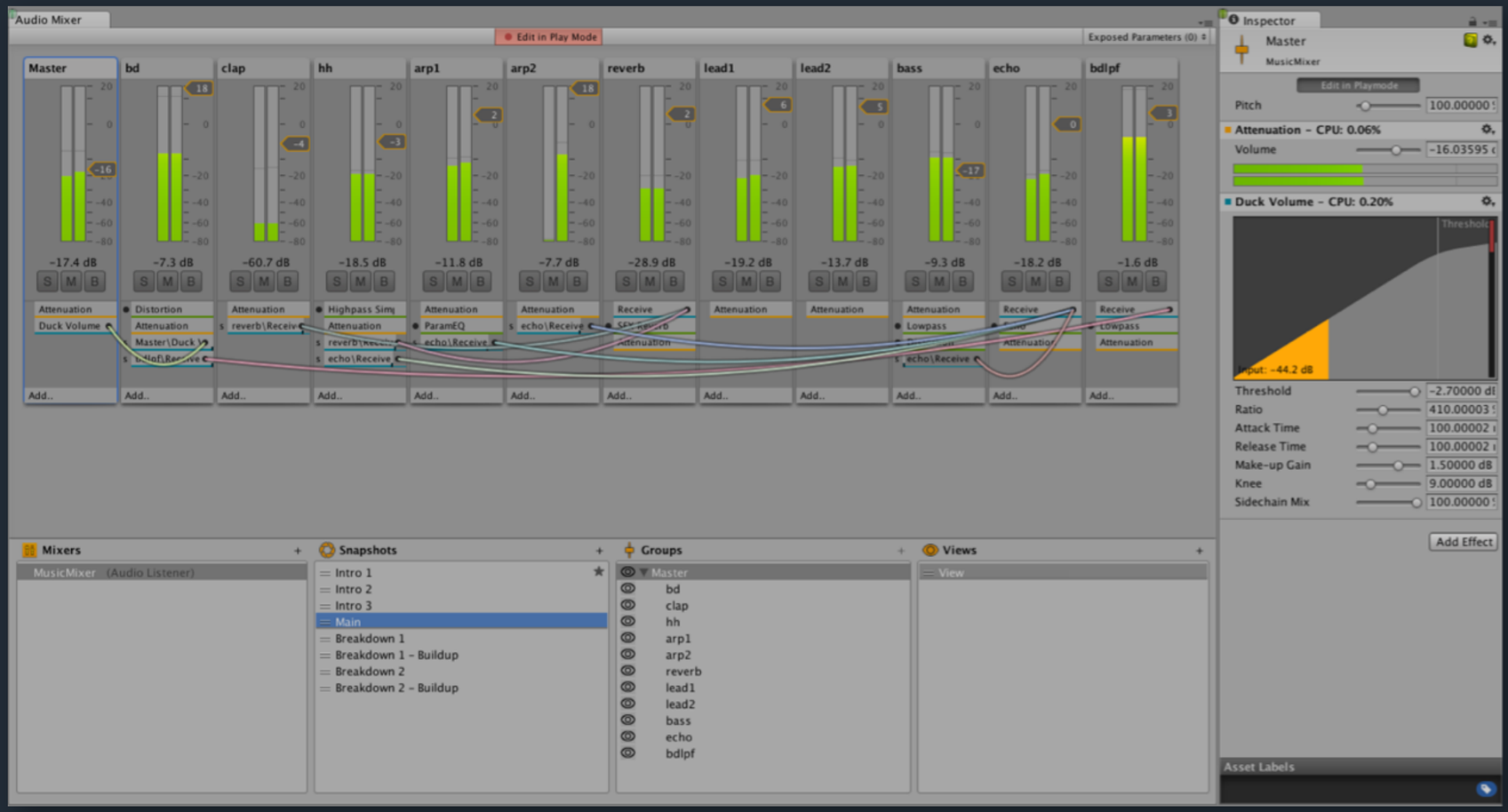
What to display:
- Channels
- Groups
- Both

Selecting a row in the frame log highlights the AudioSource and the AudioClip that it plays.

AudioSources and AudioClips playing in the selected frame

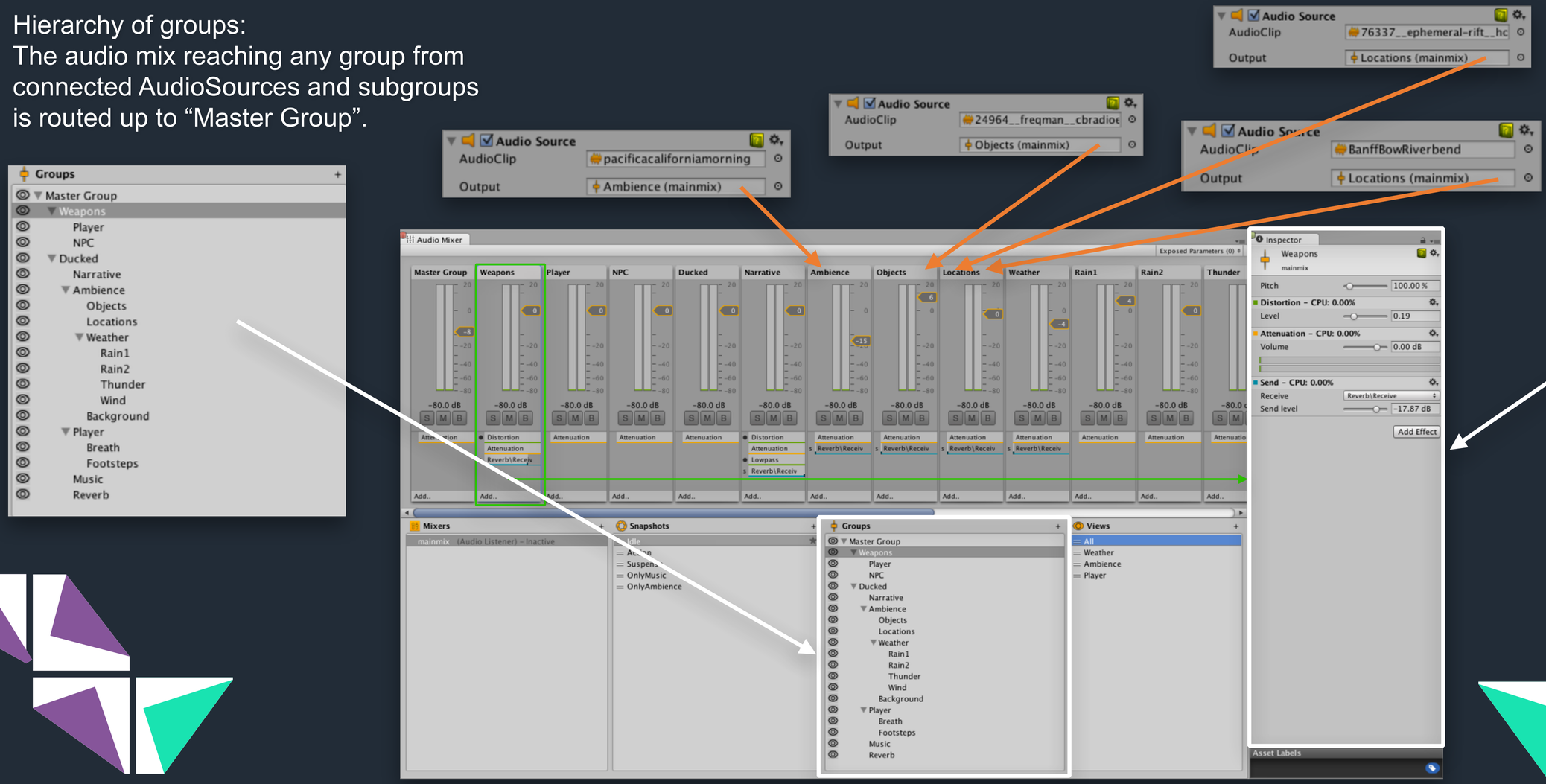


AUDIO MIXER


Audio Mixer
Edit in Play Mode
Exposed Parameters (0)
Master
bd
clap
hh
arp1
arp2
reverb
lead1
lead2
bass
echo
bdlpf
-17.4 dB
-7.3 dB
-60.7 dB
-18.5 dB
-11.8 dB
-7.7 dB
-28.9 dB
-19.2 dB
-13.7 dB
-9.3 dB
-18.2 dB
-1.6 dB
Attenuation
Duck Volume
Distortion
Highpass Sim
ParamEQ
Receive
Lowpass
Add..
Inspector
Master
MusicMixer
Edit in Playmode
Pitch
100.00000
Attenuation - CPU: 0.06%
Volume
-16.03595
Duck Volume - CPU: 0.20%
Threshold
-2.70000 dB
Ratio
410.00003
Attack Time
100.00002
Release Time
100.00002
Make-up Gain
1.50000 dB
Knee
9.00000 dB
Sidechain Mix
100.00000
Add Effect
Asset Labels
Mixers
MusicMixer (Audio Listener)
Snapshots
Intro 1
Intro 2
Intro 3
Main
Breakdown 1
Breakdown 1 - Buildup
Breakdown 2
Breakdown 2 - Buildup
Groups
Master
bd
clap
hh
arp1
arp2
reverb
lead1
lead2
bass
echo
bdlpf
Views
View

HIERARCHICAL MIXING

Hierarchy of groups:
The audio mix reaching any group from connected AudioSources and subgroups is routed up to “Master Group”.



Inspector shows selected group.

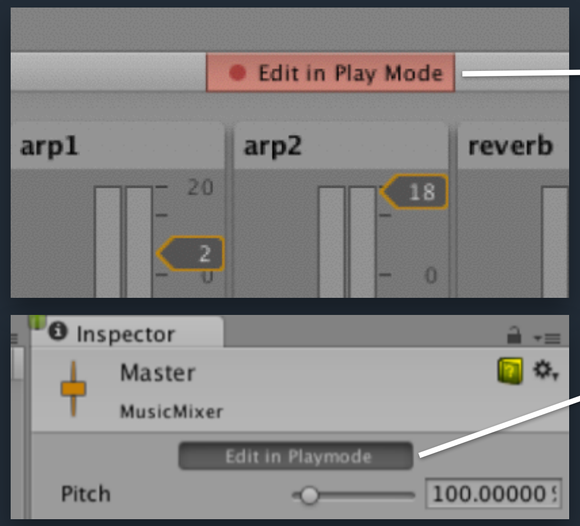
PERSISTENT EDITING

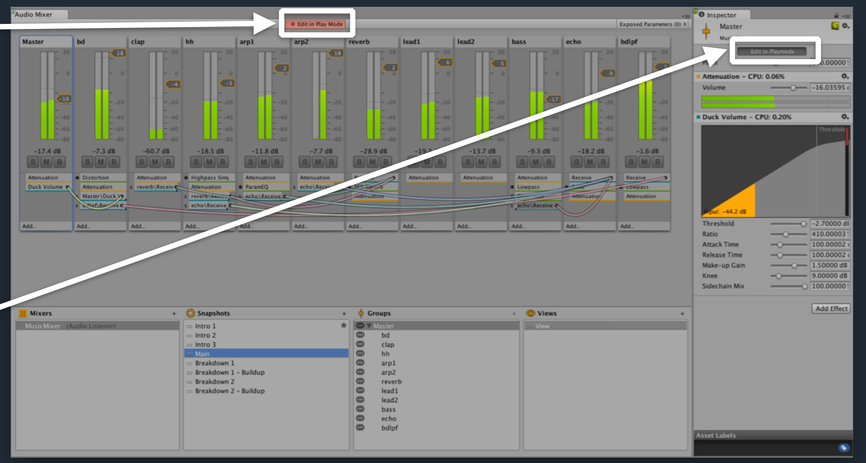
Audio mixers are assets.

Adjust levels in real-time while the game is running.

Edits made to the audio mixer during play mode are persisted.

Incredibly fast iteration time.





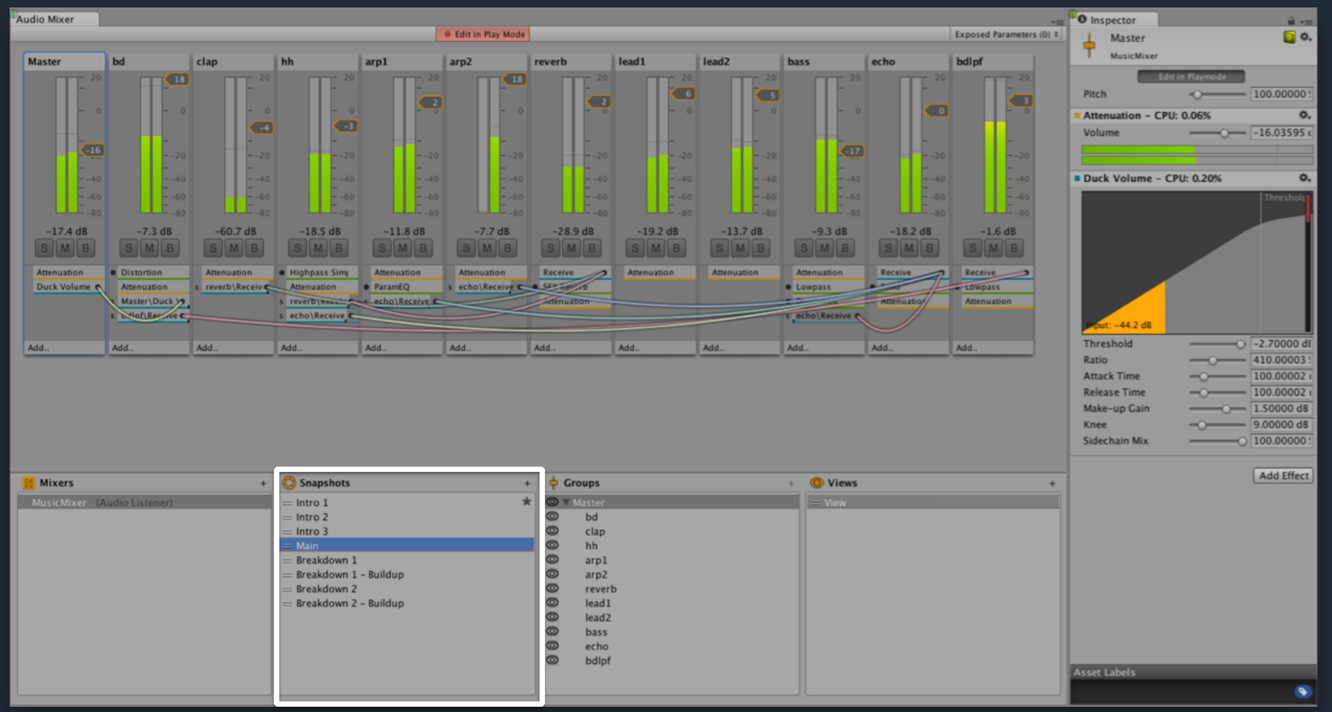
SNAPSHOTS

A snapshot captures the state all continuous parameters (volume, pitch, effect parameters).

Control themes and moods by transitioning to different snapshots.

Create a blend of any number of snapshots.

```
// Transition to snapshot
// over 3 seconds
AudioMixerSnapshot snapshot;
snapshot.TransitionTo(3.0f);
```

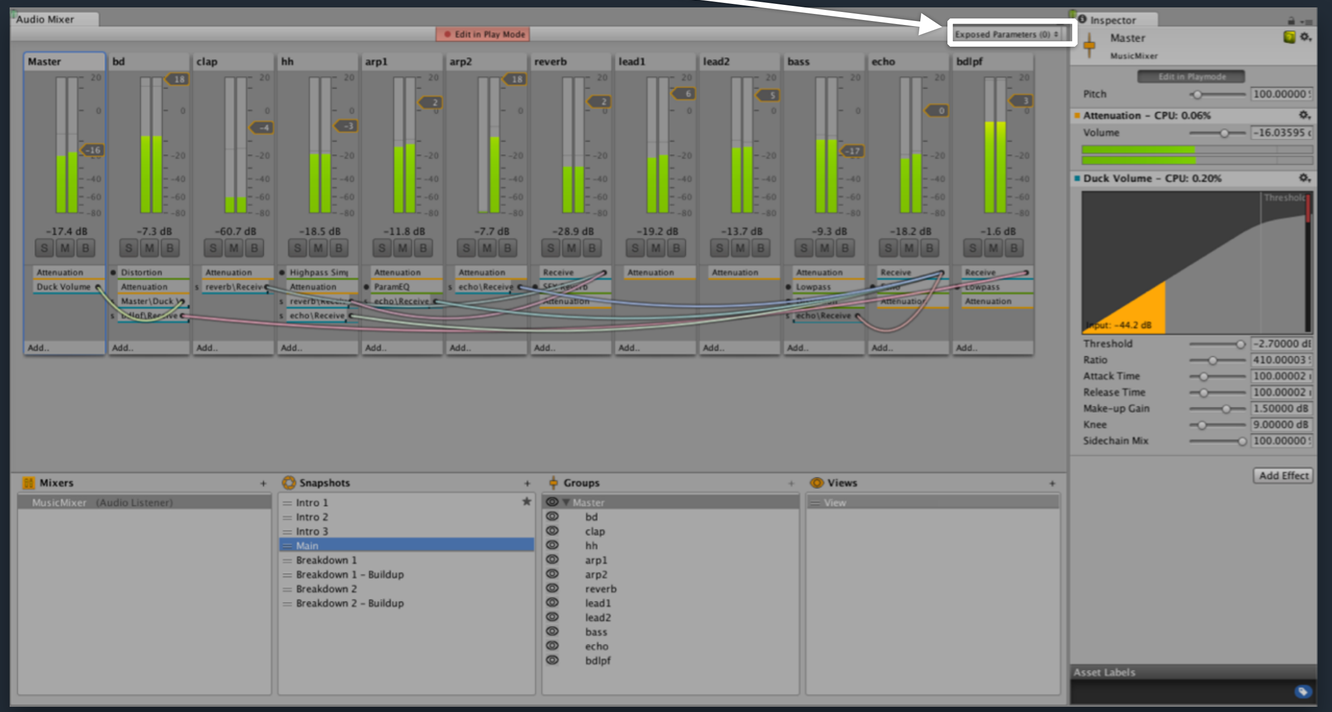


EXPOSE PARAMETERS

Expose individual parameters of an audio mixer:

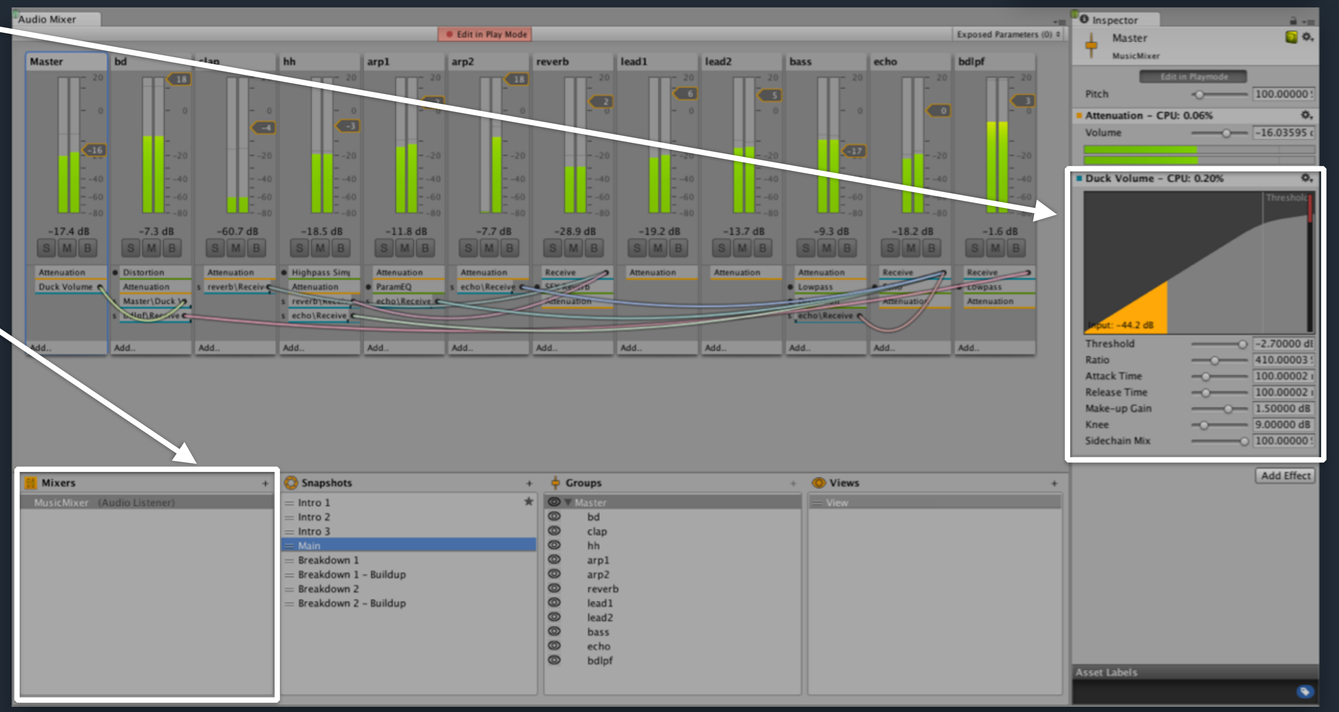
- Volume
- Pitch
- Effect parameters
- Wet levels
- Anything!

```
AudioMixer mixer = …;
float val = …;
mixer.SetFloat(“name”, val);
mixer.GetFloat(“name”, out val);
mixer.ClearFloat(“name”);
```



MIXER FEATURES

- Graphical display of loudness with VU-meters
- Solo/mute groups and bypass effects
- Sends/Receives
- Insert effects
- Side-chaining (used by Duck Volume effect)
- Hierarchies of mixers
- Support for native plugins
- Custom GUIs
- Wet/dry mixtures on all effects
- Effects automatically bypass when wet mix is zero to save CPU
- Inactive mixers are automatically suspended to save CPU



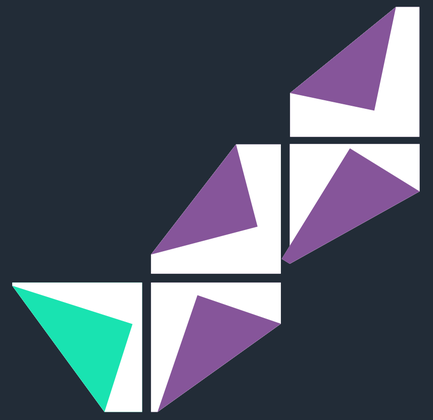
MIXER DEMOS

Both demos are linear compositions (8 fixed length loops)
Mainly changes in levels (but also a few effects)

Techno mix:
- Bass drum ducks the rest of the mix
- Resonant lowpass-filter, snapshot transitions sweep frequency
- Multiple Send effects to one Receive for reverb
- Snapshots and transitions for breakdowns

Orchestral mix:
- Snapshot transitions controlled by game state (via slider)
- Pause menu with global lowpass filter and reverb mix

https://bitbucket.org/Unity-Technologies/audiodemos



NATIVE AUDIO PLUGINS SDK

Expand Unity’s audio feature set by own or 3rd party plugins.

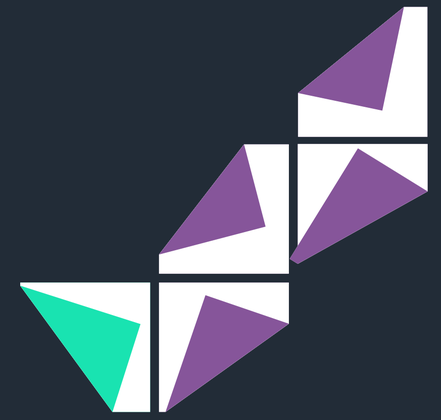
Take the best of two worlds:

- High performance native code needed for production quality
- Convenience of GUI development in C#

Builds on the well-known generic native plugin framework.

Same system used for internal mixer effects.

Many example plugins with full source code included!



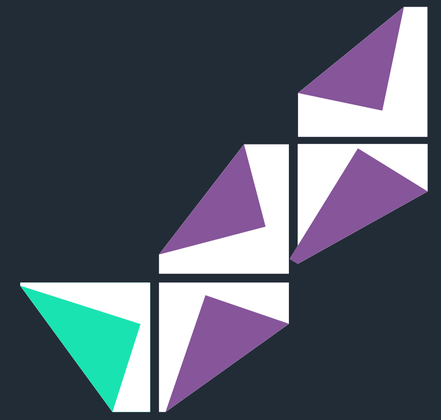
https://bitbucket.org/Unity-Technologies/nativeaudioplugins

NATIVE AUDIO PLUGINS SDK

Plugin development cycle:

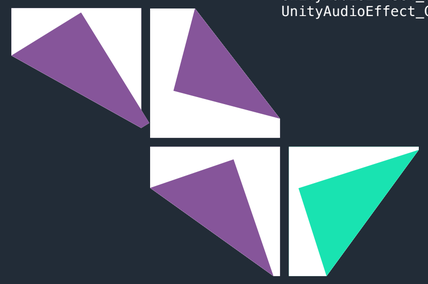
- Planning:
  - What should the plugin do?
  - How should it be operated?
  - How should the parameters be named?
- Prototype in C# using OnAudioFilterRead
- Port to C or C++ using the framework (see link below)
- (Prototype custom GUI as a .cs file in the Unity project.)
- (Compile GUI into a separate DLL.)

https://bitbucket.org/Unity-Technologies/nativeaudioplugins

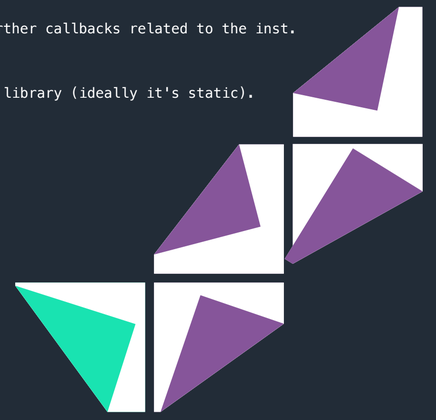


NATIVE AUDIO C INTERFACE

```
typedef UNITY_AUDIODSP_RESULT (UNITY_AUDIODSP_CALLBACK * UnityAudioEffect_CreateCallback)(UnityAudioEffectState* state);
typedef UNITY_AUDIODSP_RESULT (UNITY_AUDIODSP_CALLBACK * UnityAudioEffect_ReleaseCallback)(UnityAudioEffectState* state);
typedef UNITY_AUDIODSP_RESULT (UNITY_AUDIODSP_CALLBACK * UnityAudioEffect_ResetCallback)(UnityAudioEffectState* state);
typedef UNITY_AUDIODSP_RESULT (UNITY_AUDIODSP_CALLBACK * UnityAudioEffect_ProcessCallback)(UnityAudioEffectState* state, float* inbuffer, float* outbuffer, unsigned int length, int inchannels, int outchannels);
typedef UNITY_AUDIODSP_RESULT (UNITY_AUDIODSP_CALLBACK * UnityAudioEffect_SetPositionCallback)(UnityAudioEffectState* state, unsigned int pos);
typedef UNITY_AUDIODSP_RESULT (UNITY_AUDIODSP_CALLBACK * UnityAudioEffect_SetFloatParameterCallback)(UnityAudioEffectState* state, int index, float value);
typedef UNITY_AUDIODSP_RESULT (UNITY_AUDIODSP_CALLBACK * UnityAudioEffect_GetFloatParameterCallback)(UnityAudioEffectState* state, int index, float* value, char *valuestr);
typedef UNITY_AUDIODSP_RESULT (UNITY_AUDIODSP_CALLBACK * UnityAudioEffect_GetFloatBufferCallback)(UnityAudioEffectState* state, const char* name, float* buffer, int numsamples);

enum UnityAudioEffectDefinitionFlags
{
    UnityAudioEffectDefinitionFlags_IsSideChainTarget  = 1 << 0,   // Does this effect need a side chain buffer and can it be targeted by a Send?
    UnityAudioEffectDefinitionFlags_IsSpatializer      = 2 << 0,   // Should this effect be inserted at sources and take over panning?
};

enum UnityAudioEffectStateFlags
{
    UnityAudioEffectStateFlags_IsPlaying               = 1 << 0,   // Set when engine is in play mode. Also true while paused.
    UnityAudioEffectStateFlags_IsPaused                = 1 << 1,   // Set when engine is paused mode.
    UnityAudioEffectStateFlags_IsMuted                 = 1 << 2,   // Set when effect is being muted (only available in the editor)
    UnityAudioEffectStateFlags_IsSideChainTarget       = 1 << 3,   // Does this effect need a side chain buffer and can it be targeted by a Send?
};

struct UnityAudioParameterDefinition
{
    char                                    name[16];           // Display name on the GUI
    char                                    unit[16];           // Scientific unit of parameter to be appended after the value in textboxes
    const char*                             description;        // Description of parameter (displayed in tool tips, automatically generated documentation, etc.)
    float                                   min;                // Minimum value of the parameter
    float                                   max;                // Maximum value of the parameter
    float                                   defaultval;         // Default and initial value of the parameter
    float                                   displayscale;       // Scale factor used only for the display of parameters (i.e. 100 for a percentage value ranging from 0 to 1)
    float                                   displayexponent;    // Exponent for mapping parameters to sliders
};

struct UnityAudioEffectDefinition
{
    UInt32                                  structsize;         // Size of this struct
    UInt32                                  paramstructsize;    // Size of paramdesc fields
    UInt32                                  apiversion;         // Plugin API version
    UInt32                                  pluginversion;      // Version of this plugin
    UInt32                                  channels;           // Number of channels. Effects should set this to 0 and process any number of input/output channels they get in the process callback. Generator elements should specify a >0 value here.
    UInt32                                  numparameters;      // The number of parameters exposed by this plugin.
    UInt64                                  flags;              // Various capabilities and requirements of the plugin.
    char                                    name[32];           // Name used for registration of the effect. This name will also be displayed in the GUI.
    UnityAudioEffect_CreateCallback         create;             // The create callback is called when DSP unit is created and can be null.
    UnityAudioEffect_ReleaseCallback        release;            // The release callback is called just before the plugin is freed and should free any data associated with this specific instance of the plugin. No further callbacks related to the inst.
    UnityAudioEffect_ResetCallback          reset;              // The reset callback is called by the user to bring back the plugin instance into its initial state. Use to avoid clicks or artifacts.
    UnityAudioEffect_ProcessCallback        process;            // The processing callback is repeatedly called with a block of input audio to read from and an output block to write to.
    UnityAudioEffect_SetPositionCallback    setposition;        // The position callback can be used for implementing seek operations.
    UnityAudioParameterDefinition*          paramdefs;          // A pointer to the definitions of the parameters exposed by this plugin. This data pointed to must remain valid for the whole lifetime of the dynamic library (ideally it's static).
    UnityAudioEffect_SetFloatParameterCallback setfloatparameter;  // This is called whenever one of the exposed parameters is changed.
    UnityAudioEffect_GetFloatParameterCallback getfloatparameter;  // This is called to query parameter values.
    UnityAudioEffect_GetFloatBufferCallback    getfloatbuffer;     // Get N samples of named buffer. Used for displaying analysis data from the runtime.
```



https://bitbucket.org/Unity-Technologies/nativeaudioplugins



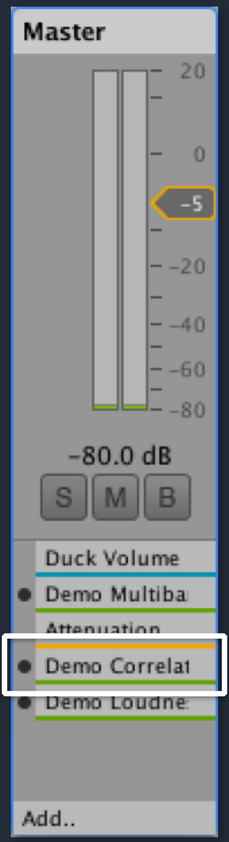
NATIVE AUDIO C INTERFACE

```
typedef UNITY_AUDIODSP_RESULT (UNITY_AUDIODSP_CALLBACK * UnityAudioEffect_CreateCallback)(UnityAudioEffectState* state);
typedef UNITY_AUDIODSP_RESULT (UNITY_AUDIODSP_CALLBACK * UnityAudioEffect_ReleaseCallback)(UnityAudioEffectState* state);
typedef UNITY_AUDIODSP_RESULT (UNITY_AUDIODSP_CALLBACK * UnityAudioEffect_ResetCallback)(UnityAudioEffectState* state);
typedef UNITY_AUDIODSP_RESULT (UNITY_AUDIODSP_CALLBACK * UnityAudioEffect_ProcessCallback)(UnityAudioEffectState* state, float* inbuffer, float* outbuffer,
typedef UNITY_AUDIODSP_RESULT (UNITY_AUDIODSP_CALLBACK * UnityAudioEffect_SetPositionCallback)(UnityAudioEffectState* state, unsigned int pos);
typedef UNITY_AUDIODSP_RESULT (UNITY_AUDIODSP_CALLBACK * UnityAudioEffect_SetFloatParameterCallback)(UnityAudioEffectState* state, int index, float value);
typedef UNITY_AUDIODSP_RESULT (UNITY_AUDIODSP_CALLBACK * UnityAudioEffect_GetFloatParameterCallback)(UnityAudioEffectState* state, int index, float* value,
typedef UNITY_AUDIODSP_RESULT (UNITY_AUDIODSP_CALLBACK * [illegible]GetFloatBufferCallback[illegible]ioEffectState* [illegible] const char* [illegible] float* [illegible]fer,

enum UnityAudioEffectDefinitionFlags
{
    UnityAudioEffectDefinitionFlags_IsSideChainTarget  = [illegible]),  // Does [illegible] effect [illegible] side chain buf[illegible] can i[illegible]ed by a
    UnityAudioEffectDefinitionFlags_IsSpatializer      = [illegible]),  // Shoul[illegible] effec[illegible]
};

enum UnityAudioEffectStateFlags
{
    UnityAudioEffectStateFlags_IsPlaying               = [illegible]),  // Set w[illegible]ine i[illegible]lay mode. Also t[illegible]ile pa[illegible]
    UnityAudioEffectStateFlags_IsPaused                = [illegible],  // Set w[illegible]gine i[illegible]d mode.
    UnityAudioEffectStateFlags_IsMuted                 = [illegible]),  // Set [illegible]fect is [illegible]muted (only [illegible]le in t[illegible]itor)
    UnityAudioEffectStateFlags_IsSideChainTarget       = [illegible]  // [illegible] effect ne[illegible] chain[illegible]nd can i[illegible]targeted b[illegible]
};

struct UnityAudioParameterDefinition
{
    char                                    name[16];         // Display name on the GUI
    char                                    unit[16];         // Scientific unit of parameter to be appended after the value in textboxes
    const char*                             description;      // Description of parameter (displayed in tool tips, automatically generated documentation, etc.)
    float                                   min;              // Minimum value of the parameter
    float                                   max;              // Maximum value of the parameter
    float                                   defaultval;       // Default and initial value of the parameter
    float                                   displayscale;     // Scale factor used only for the display of parameters (i.e. 100 for a percentage value ranging from 0 to 1)
    float                                   displayexponent;  // Exponent for mapping parameters to sliders
};

struct UnityAudioEffectDefinition
{
    UInt32                                  structsize;[illegible]  // Size [illegible]s struct[illegible]
    UInt32                                  paramstruct[illegible]  // Size [illegible]amdesc[illegible]
    UInt32                                  apiversion;[illegible]  // Plug[illegible]version[illegible]
    UInt32                                  pluginversi[illegible]  // [illegible] this p[illegible]
    UInt32                                  channels;         // [illegible] channe[illegible]fects s[illegible]set this [illegible]nd p[illegible] an[illegible]er of [illegible] outpu[illegible]nels they get in the process callback. Generator elements should specify a >0 value here.
    UInt32                                  numparamete[illegible]  // [illegible]umber of p[illegible]ers expos[illegible] this plug[illegible]
    UInt64                                  flags;            // Various capabi[illegible] of the[illegible]n.
    char                                    name[32];         // Name used for[illegible]ect. Th[illegible]e will [illegible]layed [illegible] GUI.
    UnityAudioEffect_CreateCallback         create;           // The create ca[illegible] is called w[illegible] unit [illegible]ated and [illegible]l.
    UnityAudioEffect_ReleaseCallback        release;          // The release [illegible]k is called j[illegible]ore t[illegible]gin is fr[illegible]ould f[illegible]y data a[illegible]ecific instance of the plugin. No further callbacks related to the
    UnityAudioEffect_ResetCallback          reset;            // The reset c[illegible] is called by t[illegible] to [illegible]back the p[illegible]stance [illegible]ts initia[illegible]d clicks or artifacts.
    UnityAudioEffect_ProcessCallback        process;          // The process[illegible]llback is repeate[illegible]led [illegible]a block of i[illegible]dio to[illegible]from and an [illegible]to write to.
    UnityAudioEffect_SetPositionCallback    setposition;      // The position callback can be used for implementing seek operations.
    UnityAudioParameterDefinition*          paramdefs;        // A pointer to the definitions of the parameters exposed by this plugin. This data pointed to must remain valid for the whole lifetime of the dynamic library (ideally it's static).
    UnityAudioEffect_SetFloatParameterCallback setfloatparameter;  // This is called whenever one of the exposed parameters is changed.
    UnityAudioEffect_GetFloatParameterCallback getfloatparameter;  // This is called to query parameter values.
    UnityAudioEffect_GetFloatBufferCallback getfloatbuffer;   // Get N samples of named buffer. Used for displaying analysis data from the runtime.
```

DON'T PANIC

https://bitbucket.org/Unity-Technologies/nativeaudioplugins

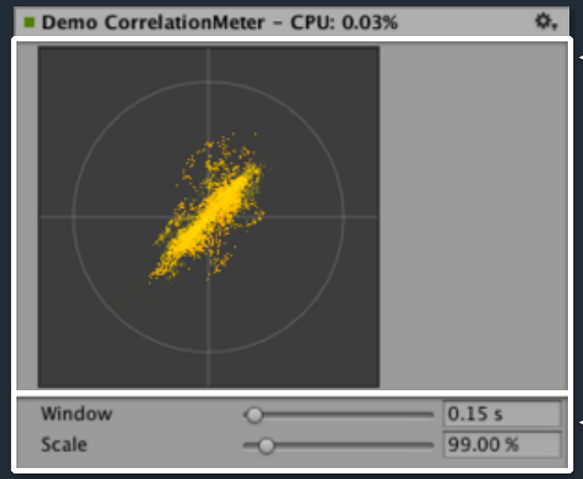
A SIMPLE PLUGIN



A simple stereo channel correlation meter

Just passes audio data through and stores it in a buffer

The buffered data is read asynchronously by the custom GUI to show the correlation between left and right channels



Custom GUI

Standard UI generated by Unity (optional)

https://bitbucket.org/Unity-Technologies/nativeaudioplugins

NATIVE AUDIO PLUGIN FRAMEWORK

Namespace allows multiple effects in a single DLL

Indexed parameters of plugin

Instance data of plugin

Min/max range

Mostly boilerplate init/destroy

Audio processing callback

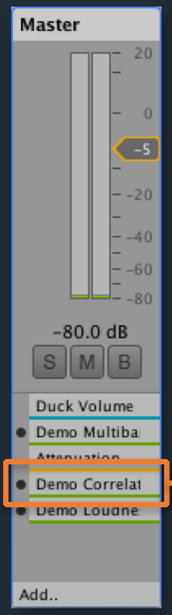
Get/set parameters

Read arrays of floating point data (from GUI)

```
#include "AudioPluginUtil.h"

namespace CorrelationMeter
{
    enum Param
    {
        P_Window,
        P_Scale,
        P_NUM
    };

    struct EffectData
    {
        float p[P_NUM];
        HistoryBuffer history[8];
        int numchannels;
    };

    int InternalRegisterEffectDefinition(UnityAudioEffectDefinition& definition)
    {
        int numparams = P_NUM;
        definition.paramdefs = new UnityAudioParameterDefinition[numparams];
        RegisterParameter(definition, "Window", "s", 0.1f, 2.0f, 0.15f,
            1.0f, 1.0f, P_Window, "Length of analysis window");
        RegisterParameter(definition, "Scale", "%", 0.01f, 10.0f, 1.0f,
            100.0f, 1.0f, P_Scale, "Amplitude scaling for monitored signal");
        return numparams;
    }

    UNITY_AUDIODSP_RESULT UNITY_AUDIODSP_CALLBACK CreateCallback(UnityAudioEffectState* state)
    {
        EffectData* data = new EffectData;
        memset(data, 0, sizeof(EffectData));
        InitParametersFromDefinitions(InternalRegisterEffectDefinition, data->p);
        state->effectdata = data;
        for (int i = 0; i < 8; i++)
            data->history[i].Init(state->samplerate * 2);
        return UNITY_AUDIODSP_OK;
    }

    UNITY_AUDIODSP_RESULT UNITY_AUDIODSP_CALLBACK ReleaseCallback(UnityAudioEffectState* state)
    {
        EffectData* data = state->GetEffectData<EffectData>();
        delete data;
        return UNITY_AUDIODSP_OK;
    }

    UNITY_AUDIODSP_RESULT UNITY_AUDIODSP_CALLBACK ProcessCallback(
        UnityAudioEffectState* state, float* inbuffer, float* outbuffer,
        unsigned int length, int inchannels, int outchannels)
    {
        EffectData* data = state->GetEffectData<EffectData>();

        memcpy(outbuffer, inbuffer, sizeof(float) * length * inchannels);

        for (unsigned int n = 0; n < length; n++)
            for (int i = 0; i < inchannels; i++)
                data->history[i].Feed(*inbuffer++);
        data->numchannels = inchannels;

        return UNITY_AUDIODSP_OK;
    }

    UNITY_AUDIODSP_RESULT UNITY_AUDIODSP_CALLBACK SetFloatParameterCallback(
        UnityAudioEffectState* state, int index, float value)
    {
        EffectData* data = state->GetEffectData<EffectData>();
        if (index >= P_NUM)
            return UNITY_AUDIODSP_ERR_UNSUPPORTED;
        data->p[index] = value;
        return UNITY_AUDIODSP_OK;
    }

    UNITY_AUDIODSP_RESULT UNITY_AUDIODSP_CALLBACK GetFloatParameterCallback(
        UnityAudioEffectState* state, int index, float* value, char *valuestr)
    {
        EffectData* data = state->GetEffectData<EffectData>();
        if (value != NULL)
            *value = data->p[index];
        if (valuestr != NULL)
            valuestr[0] = 0;
        return UNITY_AUDIODSP_OK;
    }

    int UNITY_AUDIODSP_CALLBACK GetFloatBufferCallback(
        UnityAudioEffectState* state, const char* name, float* buffer, int numsamples)
    {
        EffectData* data = state->GetEffectData<EffectData>();
        HistoryBuffer& l = data->history[0];
        HistoryBuffer& r = data->history[1];
        int w1 = l.writeindex;
        int w2 = r.writeindex;
        for (int n = 0; n < numsamples / 2; n++)
        {
            buffer[n * 2 + 0] = l.data[w1];
            if (--w1 < 0)
                w1 = l.length - 1;
            if (n * 2 + 1 < numsamples)
                buffer[n * 2 + 1] = r.data[w2];
            if (--w2 < 0)
                w2 = r.length - 1;
        }
        return UNITY_AUDIODSP_OK;
    }
}
```

https://bitbucket.org/Unity-Technologies/nativeaudioplugins

NATIVE AUDIO PLUGIN CUSTOM GUI



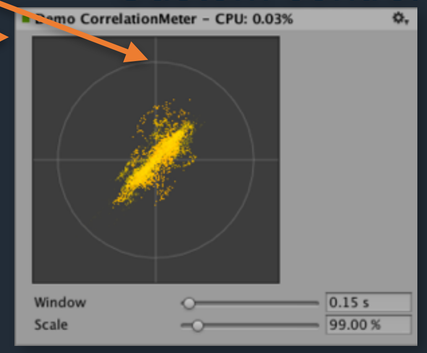
Identify as custom GUI

Name associates custom GUI with native plugin

Custom control

Custom callback for drawing GUI

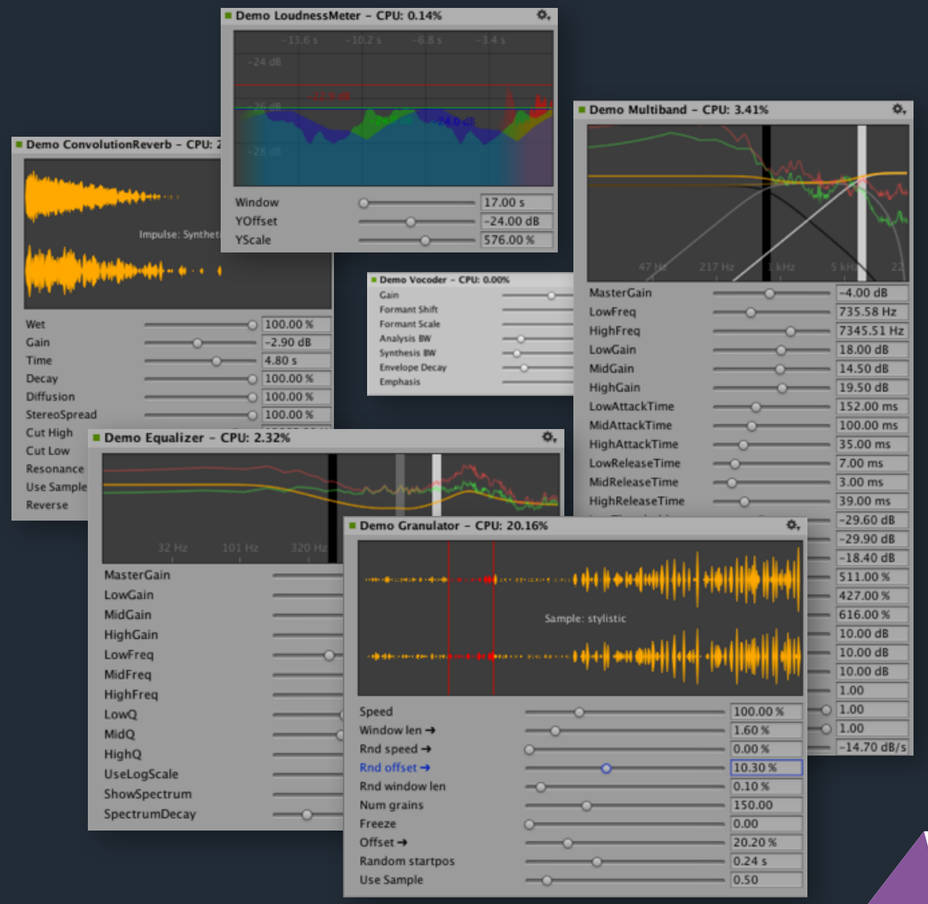
```
using UnityEditor;
using UnityEngine;

public class CorrelationMeterCustomGUI : IAudioEffectPluginGUI
{
	public override string Name
	{
		get { return "Demo CorrelationMeter"; }
	}

	public override string Description
	{
		get { return "Correlation meter demo plugin for Unity's audio plugin system"; }
	}

	public override string Vendor
	{
		get { return "Unity"; }
	}

	private static Vector3[] coord1 = new Vector3[2];
	private static Vector3[] coord2 = new Vector3[2];
	private static Vector3[] circle = new Vector3[40];

	public bool DrawControl(IAudioEffectPlugin plugin, Rect r, float samplerate)
	{
		r = AudioCurveRendering.BeginCurveFrame(r);

		if (Event.current.type == EventType.Repaint)
		{
			float blend = plugin.IsPluginEditableAndEnabled() ? 1.0f : 0.5f;

			float window;
			plugin.GetFloatParameter("Window", out window);
			window *= samplerate;
			if (window > samplerate)
				window = samplerate;

			float[] corr;
			int numsamples = (int)window;
			plugin.GetFloatBuffer("Correlation", out corr, 2 * numsamples);
			numsamples = corr.Length;

			float cx = r.x + r.width * 0.5f;
			float cy = r.y + r.height * 0.5f;

			coord1[0].Set(r.x, r.y + r.height * 0.5f, 0);
			coord1[1].Set(r.x + r.width, r.y + r.height * 0.5f, 0);
			coord2[0].Set(r.x + r.width * 0.5f, r.y, 0);
			coord2[1].Set(r.x + r.width * 0.5f, r.y + r.height, 0);

			float w = 2.0f * 3.1415926f / (float)(circle.Length - 1);
			float cr = r.height * 0.4f;
			for (int n = 0; n < circle.Length; n++)
				circle[n].Set(cx + cr * Mathf.Cos(n * w), cy + cr * Mathf.Sin(n * w), 0);

			float scale;
			plugin.GetFloatParameter("Scale", out scale);
			scale *= cr;

			float lineTint = 0.5f;
			Handles.color = new Color(lineTint, lineTint, lineTint, 0.75f);
			Handles.DrawAAPolyLine(2.0f, coord1.Length, coord1);
			Handles.DrawAAPolyLine(2.0f, coord2.Length, coord2);

			col1.a = blend;
			col2.a = 0.0f;
			float cs = 1.0f / ((numsamples / 2) - 1);
			for (int n = 0; n < numsamples / 2; n++)
			{
				float px = cx + scale * corr[n * 2];
				float py = cy - scale * corr[n * 2 + 1];
				if (px >= r.x && py >= r.y &&
					px < r.x + r.width && py < r.y + r.height)
				{
					GL.Color(Color.Lerp(col1, col2, n * cs));
					GL.Vertex3(px, py - 1.0f, 0.0f);
					GL.Vertex3(px, py, 0.0f);
				}
			}
			GL.End();
		}
		AudioCurveRendering.EndCurveFrame();
		return false;
	}

	public override bool OnGUI(IAudioEffectPlugin plugin)
	{
		float active, window, scale;
		plugin.GetFloatParameter("Active", out active);
		plugin.GetFloatParameter("Window", out window);
		plugin.GetFloatParameter("Scale", out scale);
		GUILayout.Space (5.0f);
		Rect r = GUILayoutUtility.GetRect(200, 200, GUILayout.ExpandWidth(true));
		if (r.width > r.height)
			r.width = r.height;
		else
			r.height = r.width;

		if (DrawControl(plugin, r, plugin.GetSampleRate()))
		{
			plugin.SetFloatParameter("Window", window);
			plugin.SetFloatParameter("Scale", scale);
		}
		GUILayout.Space(5.0f);
		return true;
	}
}

// Missing API for Handles.handleWireMaterial
public static class HandleUtilityWrapper
{
	static Material s_Mat;
	public static Material handleWireMaterial
	{
		get
		{
			if (s_Mat == null)
				s_Mat = (Material)EditorGUIUtility.LoadRequired(
					"SceneView/HandleLines.mat");
			return s_Mat;
		}
	}
}
```



https://bitbucket.org/Unity-Technologies/nativeaudioplugins

MANY EXAMPLE PLUGINS!

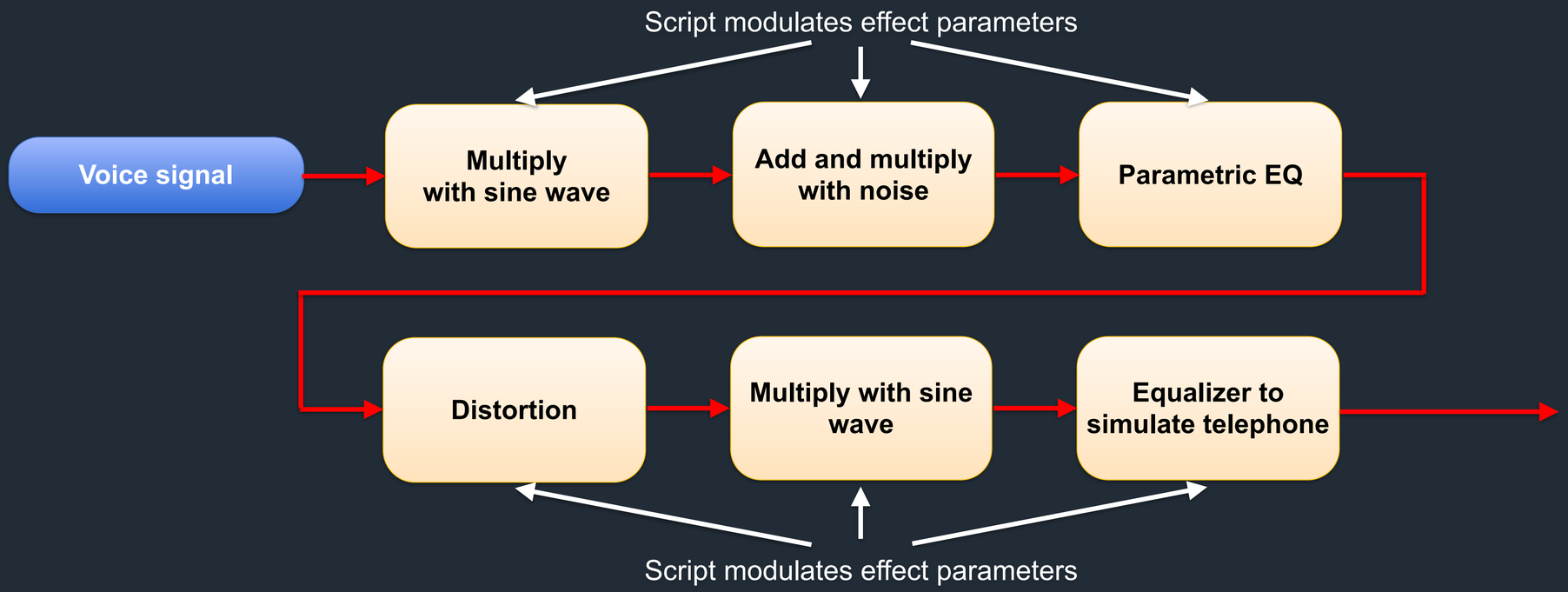
- Convolution reverb
- Level metering
- Equalizer
- Multi-band compressor
- Telephone/walkie-talkie voice processing
- Vocoder
- Granular synthesis
- Pitch detection
- Transmitting audio between external applications and Unity
- Synthesizers



https://bitbucket.org/Unity-Technologies/nativeaudioplugins

VOICE PROCESSING

Simulate communication over walkie-talkie. The amount of noise could be controlled by the distance between two players to make the signal more noisy when they are far apart.

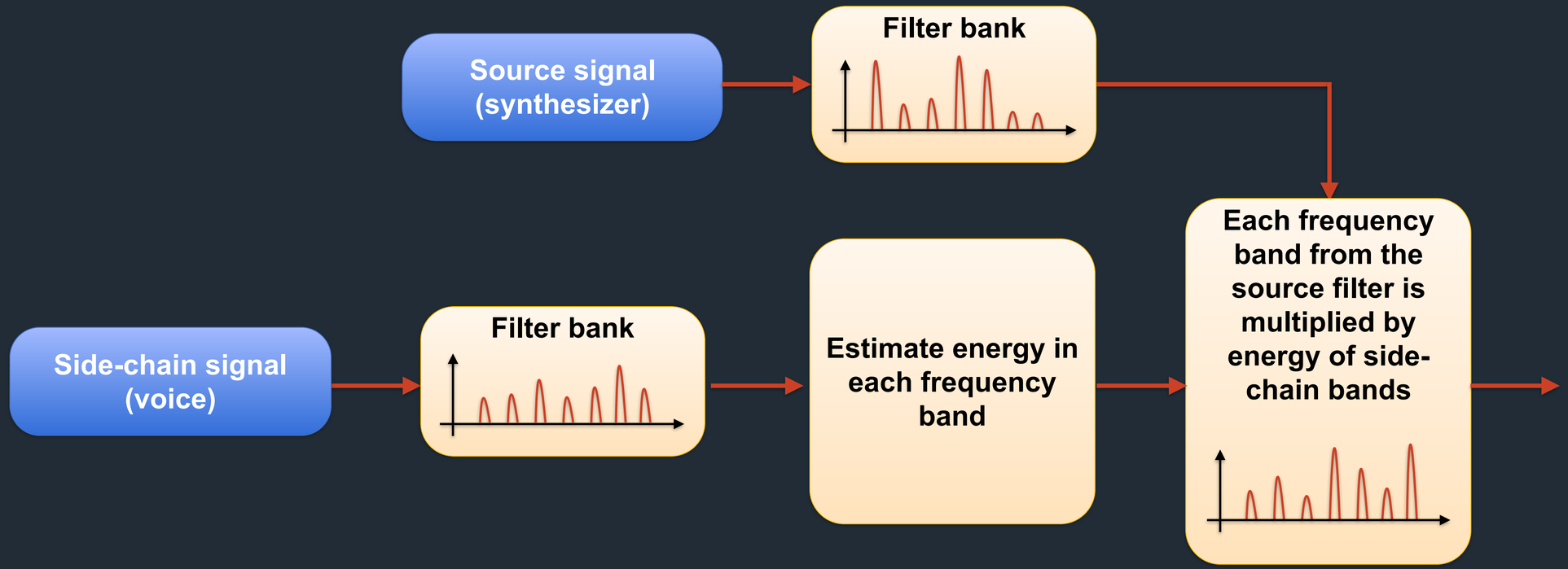


https://bitbucket.org/Unity-Technologies/nativeaudioplugins

VOCODER

The side-chain signal (typically) contains the voice signal (“modulator”), while the source signal contains the instrument signal (“carrier”).

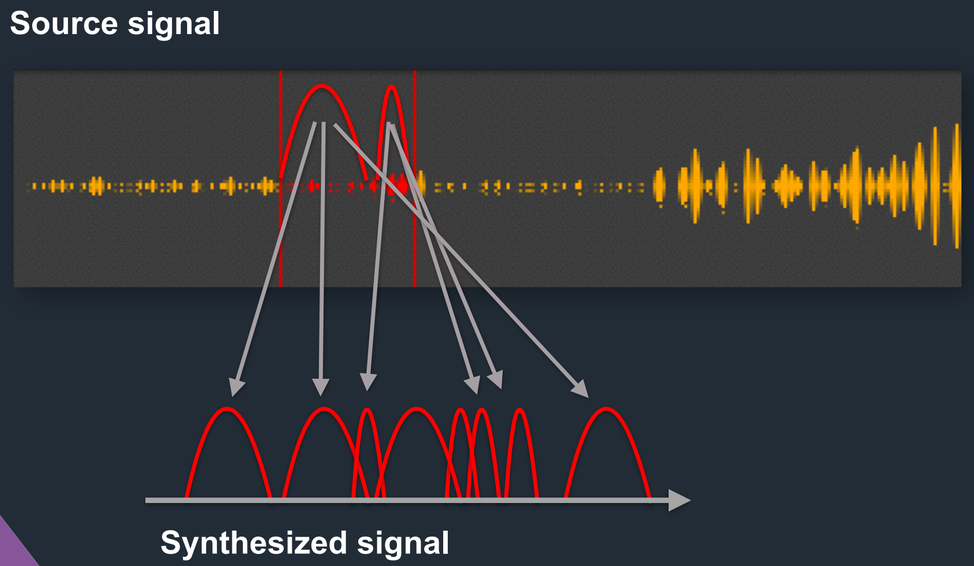
Implemented using filter banks and level detectors. Can also be implemented with FFT.

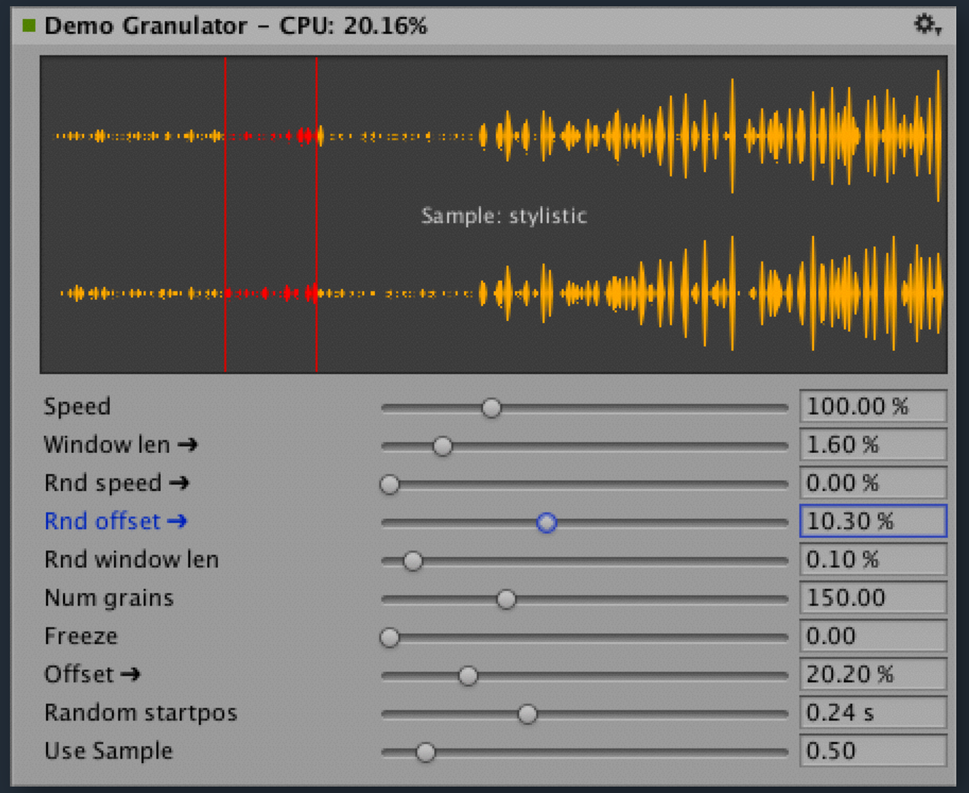


https://bitbucket.org/Unity-Technologies/nativeaudioplugins

GRANULAR SYNTHESIS

Takes grains from source signal and resynthesises output from these.





https://bitbucket.org/Unity-Technologies/nativeaudioplugins

PROCEDURAL WEATHER

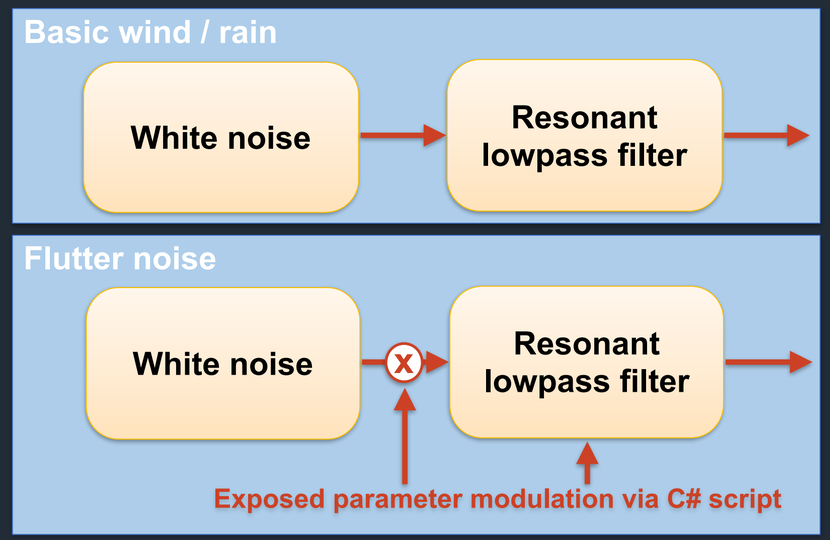
White noise basis of all sound

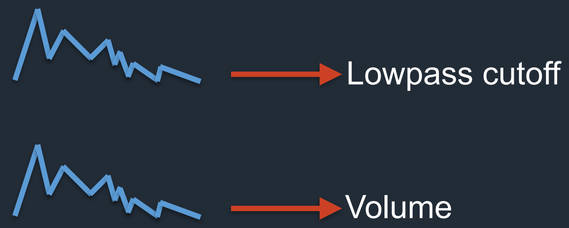
Wind (base layer, flutter, both send to howling resonance)

Rain very similar, more resonance to simulate water puddles.

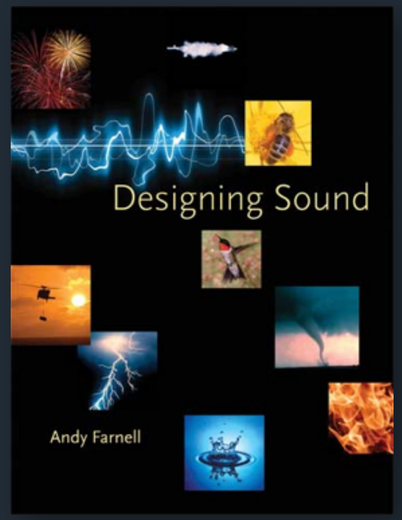
Thunder

Flash and rumble use zig-zag curves to drive volume and filter cutoff





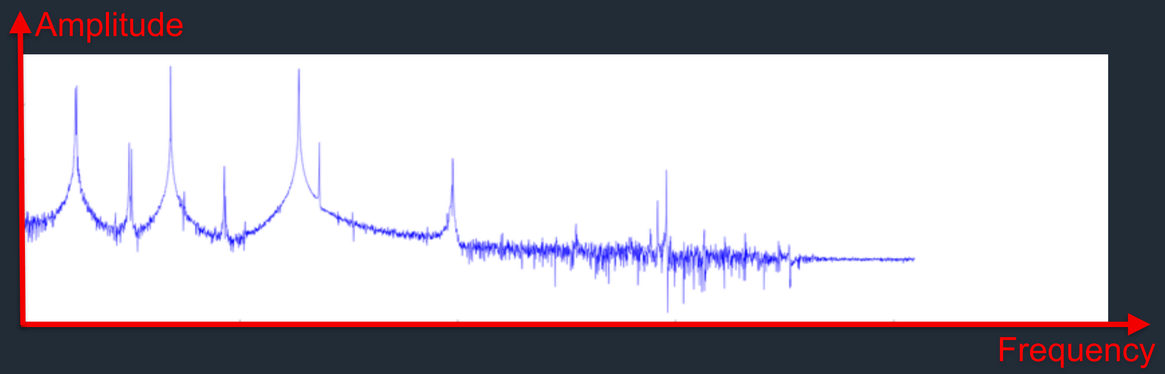
Low-frequency shaped noise (C# script)

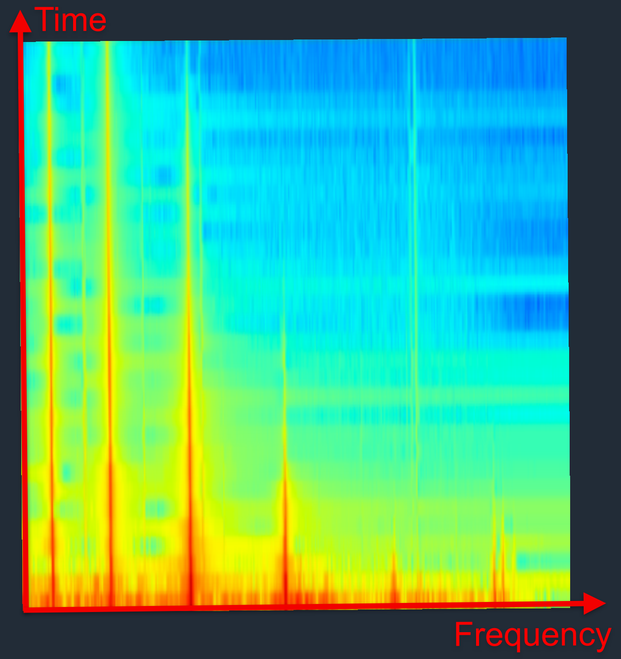


Andy Farnell: Designing Sound http://mitpress.mit.edu/books/designing-sound

MODAL SYNTHESIS

Spectrum of a bell. Notice the pronounced peaks in the spectrum.

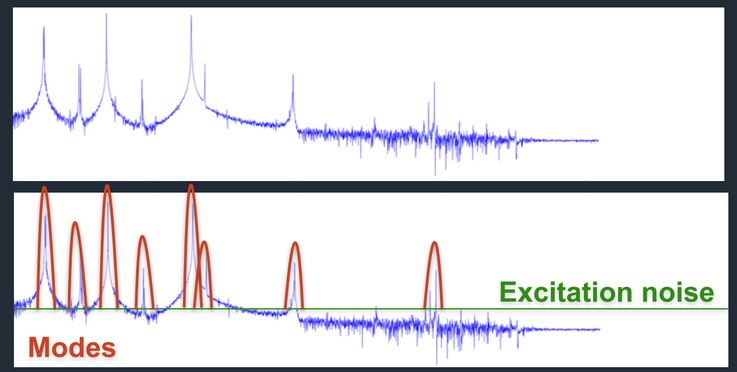


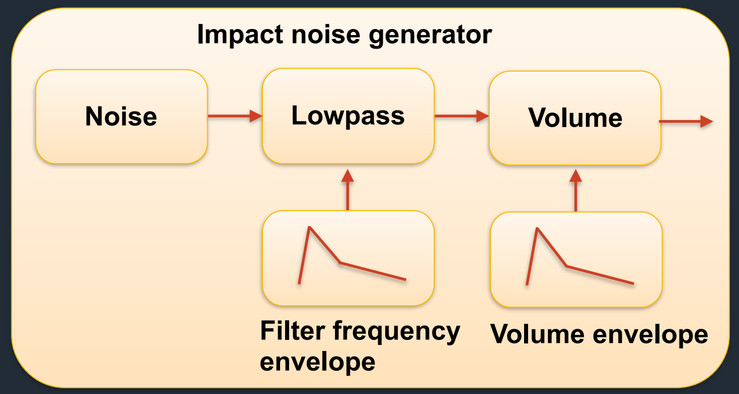


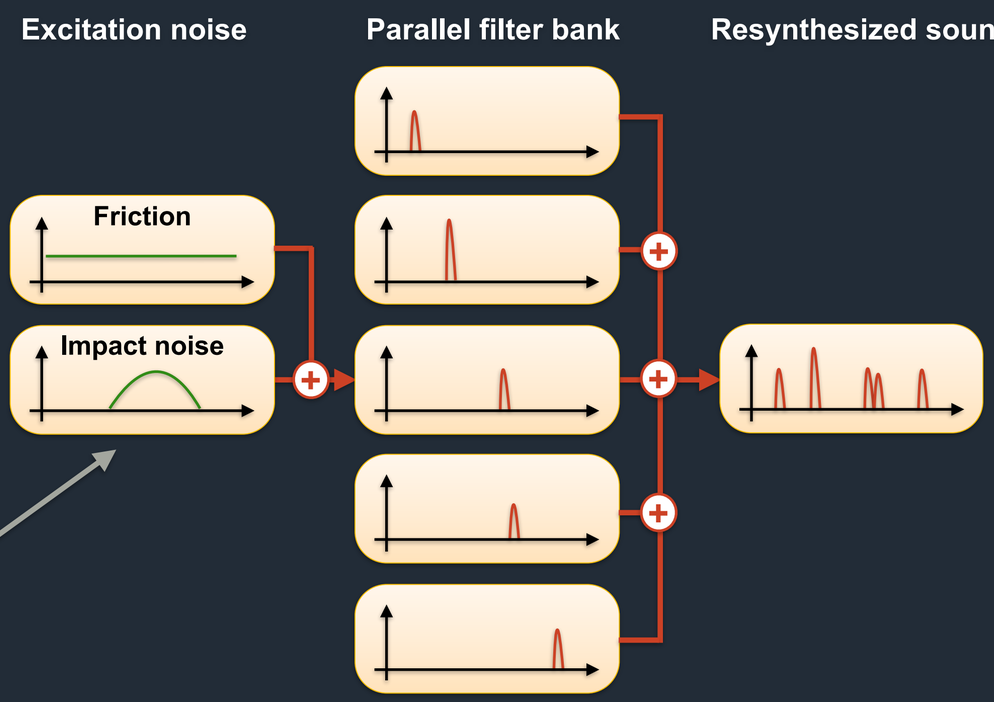
https://bitbucket.org/Unity-Technologies/nativeaudioplugins

MODAL SYNTHESIS

Re-synthesize the signal by feeding noise through a bank of parallel bandpass filters that amplify narrow bands of the spectrum (“resonators”).







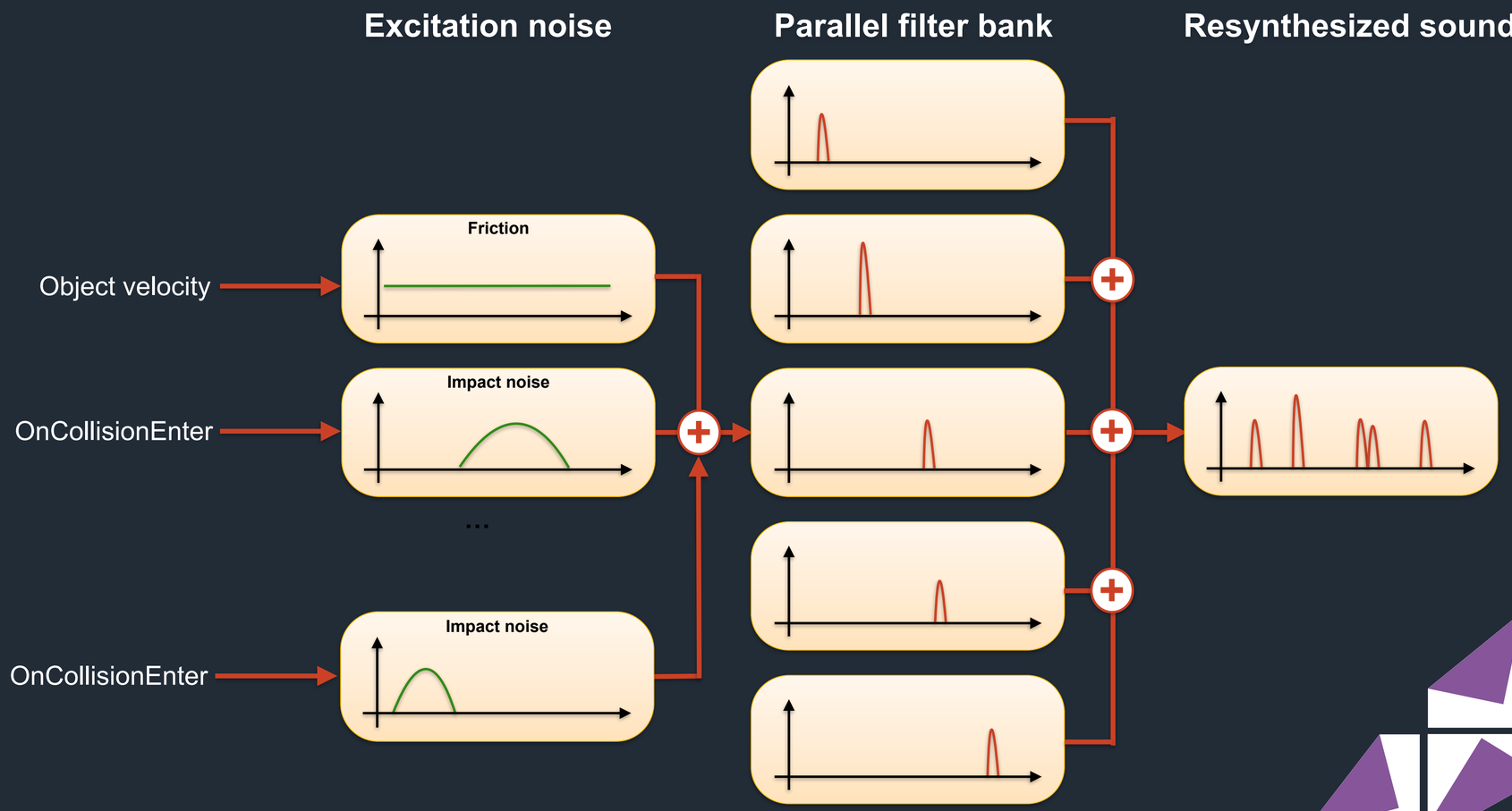
https://bitbucket.org/Unity-Technologies/nativeaudioplugins

IMPACTS/FRICTION SOUNDS

Examples:

- Impacts from physics
- Friction of sliding objects
- Metal sword sounds

Object velocity and impact force control amplitude and frequency range of the excitation source (C# script).



https://bitbucket.org/Unity-Technologies/nativeaudioplugins

TELEPORT

Transmits audio between external applications and Unity.

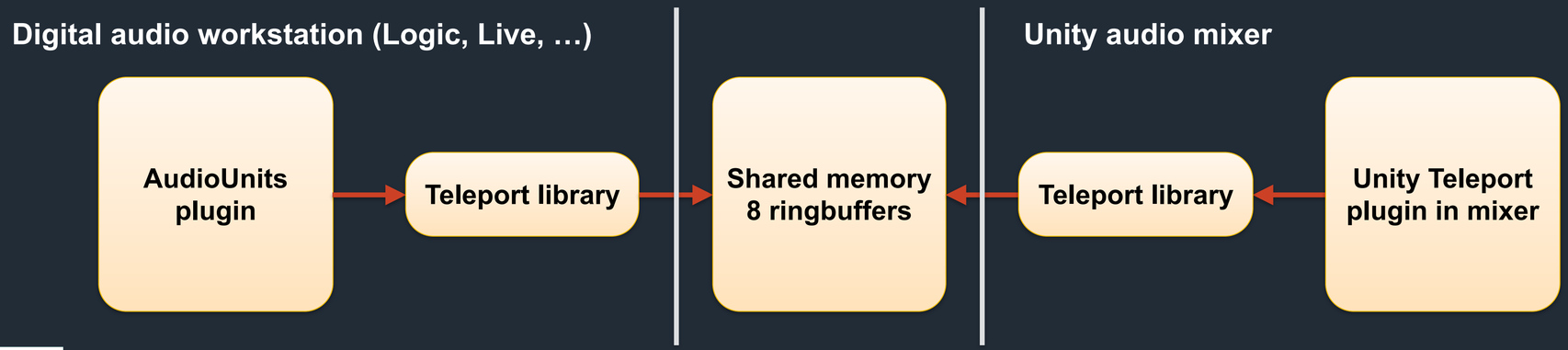
Small library “TeleportLib.cpp” manages a block of shared memory.

8 ring-buffers that may either be read from or written to by Unity or the external application.

Small console host application to send a test signal to Unity’s Teleport.

Simple AudioUnits plugin to send audio from OSX based audio apps to Unity’s Teleport.

Preview sounds live in the 3D environment while authoring them in your DAW!



https://bitbucket.org/Unity-Technologies/nativeaudioplugins

CONCLUSION

Large refactoring of the whole audio system

Many small improvements in AudioClip / AudioSource

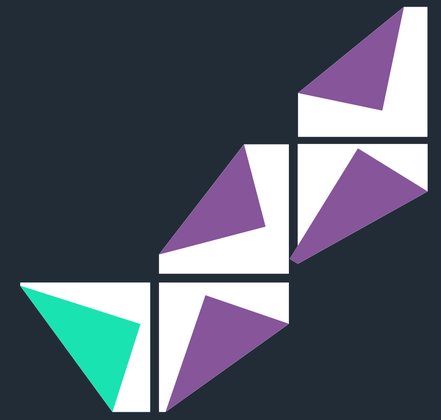
New signal-flow oriented audio mixer asset, complements existing scene setups.

Custom audio effects for future extensions

Reconsider which effects to apply offline and realtime

Create a more dynamic audio experience

Procedural sounds

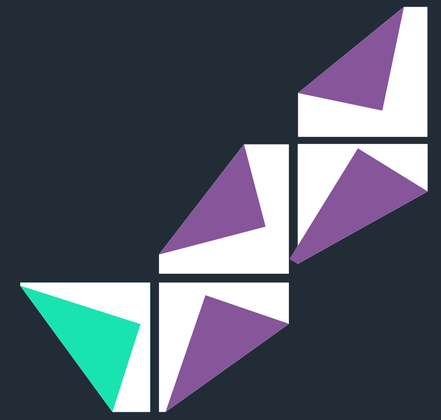


QUESTIONS?

Linear/non-linear music example projects:
https://bitbucket.org/Unity-Technologies/audiodemos

Native Audio Plugin SDK + example project (will be updated soon):
https://bitbucket.org/Unity-Technologies/nativeaudioplugins

This presentation:
files.unity3d.com/janm/UniteEurope2015.pdf



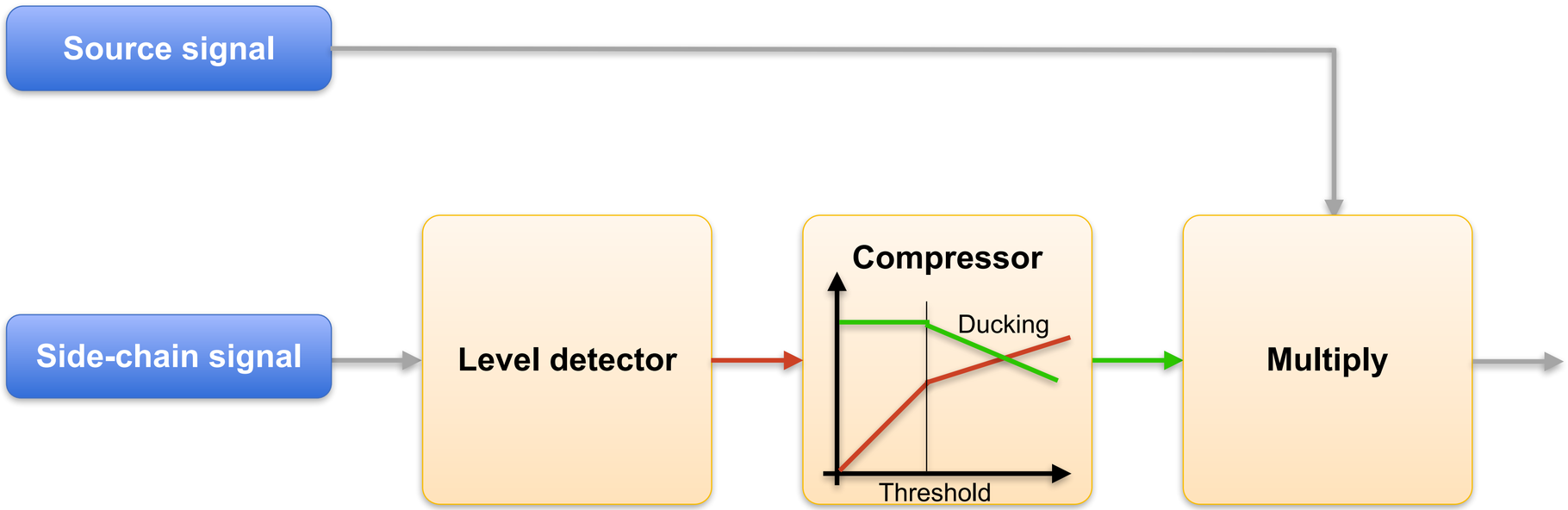
THANK YOU

FOR YOUR ATTENTION

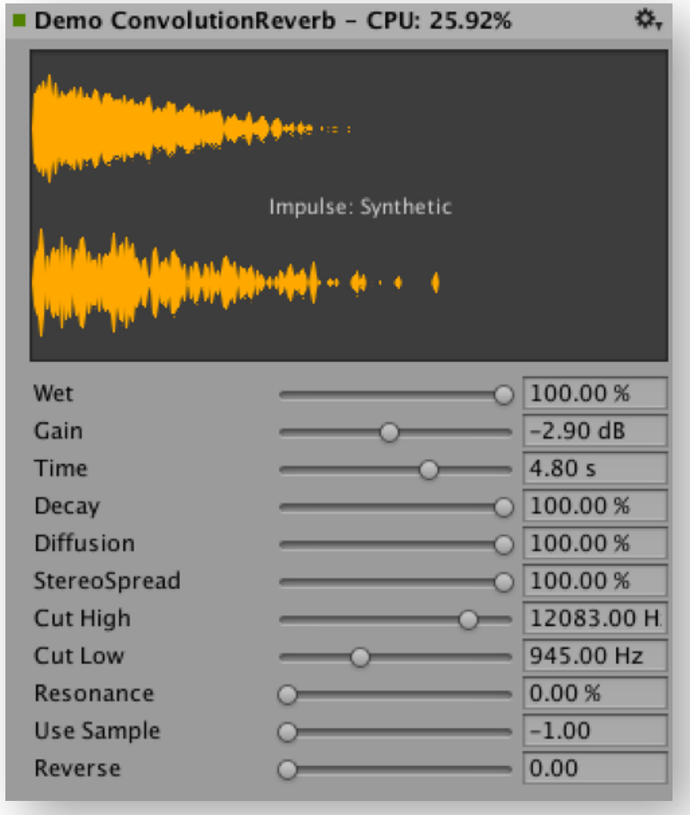
Optional slides for Q&A

Side-chain compression

Estimated side-chain level controls gain applied to source signal.
When side-chain level is above the threshold, the source is ducked.



Convolution reverb



Apply impulse h(n) on source x(n).
Length of h(n) is N samples.

Convolution process:

$$y(n) = \sum_{i=0}^{N-1} h(i)x(n-i)$$

For an impulse of 5 seconds @ 44.1 kHz that means 220.500 multiplies and adds per processed sample!

Convolution reverb

Split h(n) into K partitions of length L, where L is a power of 2.

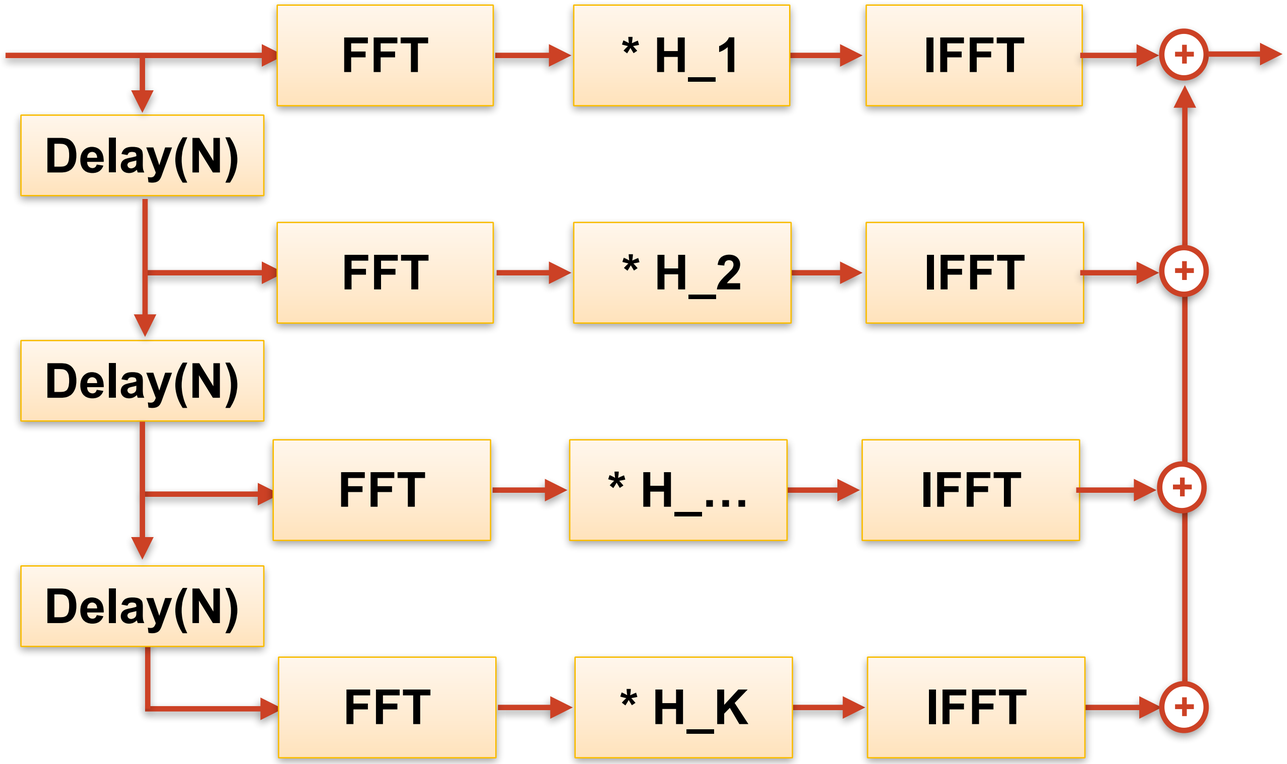
$$y(n) = \sum_{i=0}^{N-1} h(i)x(n-i) = \sum_{k=0}^{K-1} \sum_{i=0}^{L-1} h_k(i)x(n - kL - i)$$

$$y_k(n) = \sum_{i=0}^{L-1} h_k(i)x(n-kL-i) = \mathfrak{F}^{-1}\left\{\mathfrak{F}(h_k)\mathfrak{F}(x_{n-kL})\right\}$$

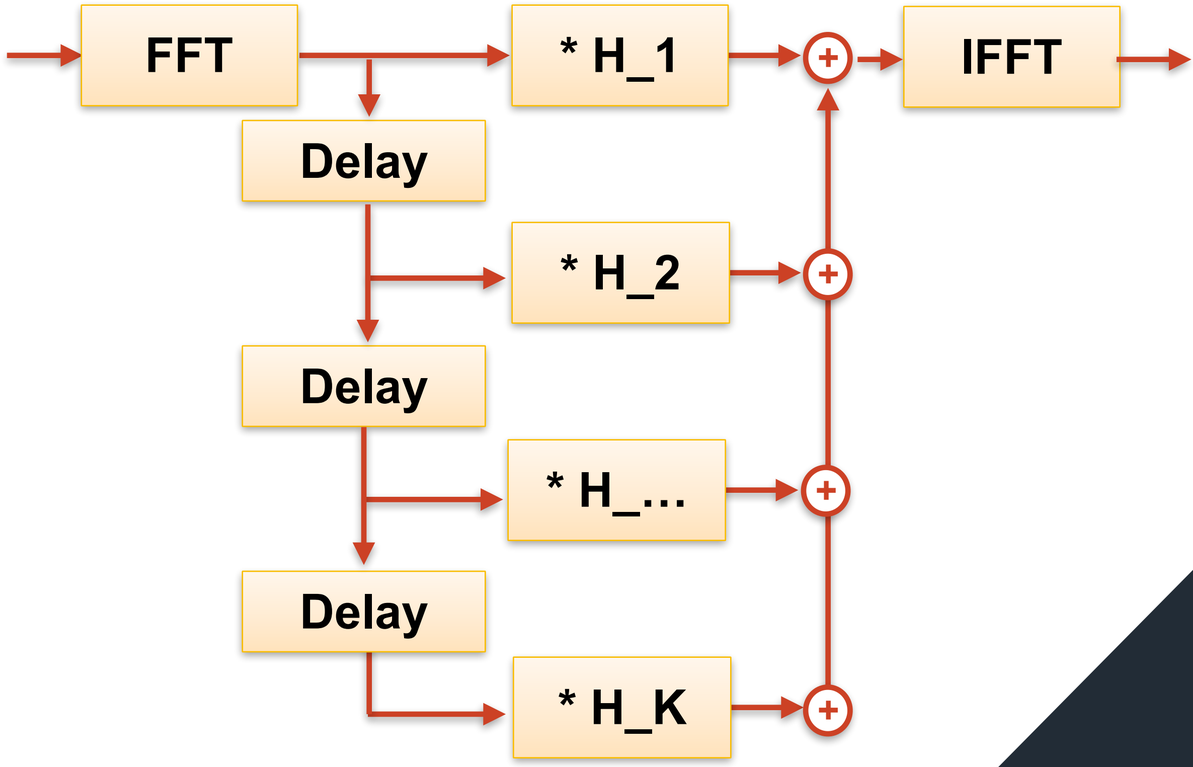
$$y(n) = \sum_{k=0}^{K-1} \mathfrak{F}^{-1}\left\{\mathfrak{F}(h_k)\mathfrak{F}(x_{n-kL})\right\} = \mathfrak{F}^{-1}\left\{\sum_{k=0}^{K-1} \mathfrak{F}(h_k)\mathfrak{F}(x_{n-kL})\right\}$$

Convolution reverb

Straightforward fast convolution of partitioned impulse:



Optimized implementation:



unity