声音设计师如何利用PD+ Heavy进行DSP插件的开发\_Part1

# Intro

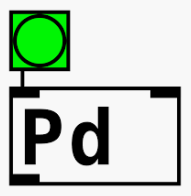
提到音频插件的开发，许多声音设计师会认为这是“程序员专属的黑魔法”，因为通常这项工作需要参与者能够编写代码，同时要具备扎实的数学、物理及数字信号处理方面的知识。而对于使用别人提供的音频工具的声音设计师来说，这都不是工作的必备技能，因此他们常常倾向于与此类“过于技术的话题”保持距离。

当声音设计师在别人的项目中听到了某种Wwise没有提供过的DSP效果时，他们非常有可能能够迅速理解其工作原理，并有能力利用熟悉的工具做出技术原型，甚至实现一模一样的交互表现。然而，要么由于性能方面的限制，要么由于此类工具的产出物无法以插件形式部署到创作工具及引擎中去，导致声音设计师的很多想法都止步于原型阶段而未能对实际游戏产生贡献。

幸运的是，通过使用PureData与Heavy，声音设计师现在已经有机会独立创造出具备良好性能、可以在创作工具中使用且能够在实即游戏中运行的效果器及音频源插件。

# 一点点科普

## 什么是PureData？



PureData（PD）是由美国软件工程师Miller Puckette于上世纪90年代开发的视觉化编程语言，主要用于计算机音乐及多媒体艺术的创作。由于PD是开源项目，自发布以来已经积累了庞大的开发者基础，并持续有新的扩展功能被添加（现在你甚至可以在PD里加载VST插件……）。

包含PD代码的文件被称为Patch，PD支持创建SubPatch所以使用者可以通过抽象来创建功能复杂的PD项目。

由于本文不是PD入门教程，需要了解更多的读者请前往<https://puredata.info/>。

下面是一段演示视频：



***可以看作是PD版本的“Hello World”***

从上面的视频中可以看到PD的使用方式与模块化合成器十分类似——通过连接不同的Object（模块）来实现信号系统的设计；通过调节参数与系统进行实时交互。

我们可以将PD看作是声音设计师的“画图环境”，让他们可以使用自己熟悉的设计语言来设计工具的原型，并有机会在之后“投产”。

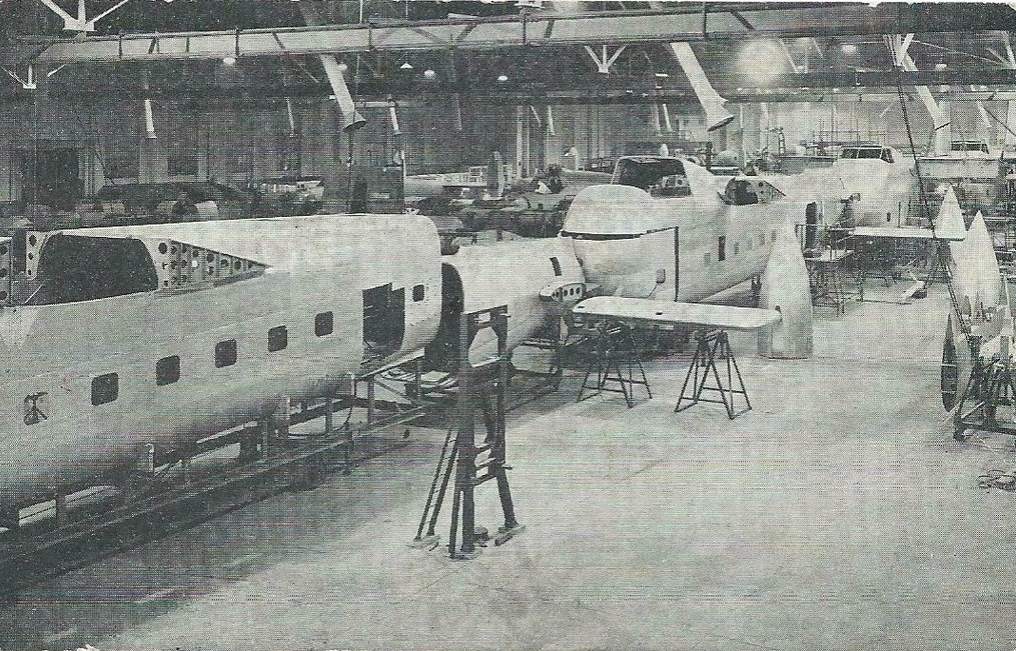


**就像用你小时候的玩具造点“能用的”东西一样**

## 什么是Heavy？

Heavy是由Enzien Audio开发的针对PD Patch的编译器，可将声音设计师制作的PD Patch转换成C/C++代码。这使得声音设计师只需在PD环境下工作就能得到可运行于目标平台的代码，而无需自己编写。

我们可以将Heavy看作是依据声音设计师提供的工具设计图纸进行生产的自动化车间。



那么我们画图和建造工厂后生产的产品能做什么呢？

# PD+Heavy生产的插件可以做什么

1，根据声音设计师在PD Patch中的设计，进行实时的DSP运算；

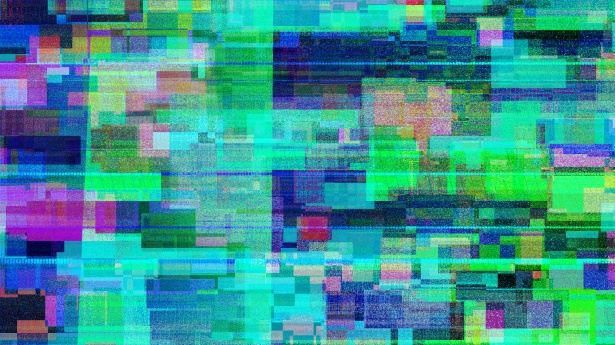
2，支持音频信号的输入输出，若源Patch中包含音频输入物件——adc~则生成效果器插件，若源patch中不包含adc~则生成源插件（如Wwise SoundSeed Wind，Wwise SynthOne）；

3，支持控制参数的输入输出，既可以接受来自Wwise SoundEngine的Game Parameter控制，也可以向Game Parameter发送数据（如作为LFO使用）；

4，支持UI事件的输入输出（仅限Unity）。

在了解了我们作为声音设计师的“新技术权力”之后，是时候设计并生产一款产品了！

# 设定目标——Sample Divider



**就像音频版本的像素化效果**

Sample Divider效果通常包含在Bitcrusher这样的失真插件当中，我将本文的目标设定为生产Sample Divider并投入实际使用。

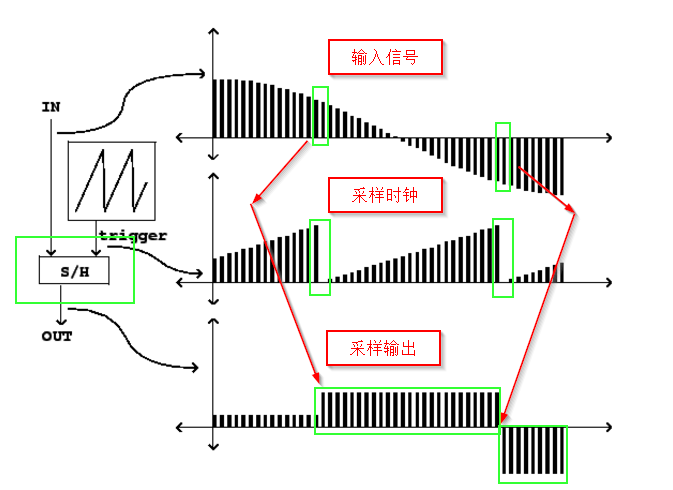
设定这样的目标的原因很简单——1，Wwise没提供Sample Divider；2，Sample Divider实际上很实用，尤其在Cyberpunk回潮的当下（想象一下如果要实现类似《Nier: Automata》中病毒入侵时以及《Titan Fall2》中Titan受创时大量出现的声音故障表现，怎么可以没有Sample Divider！）。

# 无痛DSP

在开始“绘制”作为“蓝图”的PD Patch之前，我们要先了解Sample Divider的工作原理。

Sample Divider是通过以低于输入信号采样率的频率对输入信号进行重采样，以“较低的品质进行输出”来实现对原始信号的扭曲处理的。

下图是对我们之后用到的核心元件——Sample and hold（简写为S/H）模块功能的解释，来自《电子音乐理论与技术》，作者就是PD的作者本人。



## S/H模块运作原理

如上图所示，类似正弦波的信号被送入S/H模块的左输入点，作为“Sample & Hold（采样及保持）”动作触发“时钟”的锯齿波信号被送入右边的输入点，当锯齿波的值从最高处快速下落至0点时便会触发S/H物件对信号进行采样的动作，而其余时间则保持已采集的数值。这样，当锯齿波频率越接近输入信号的采样率，则输出信号越接近输入信号，也就是“越高清”，反之则包含更多失真，也就是越“像素化”。

下面是一段Sample and hold模块在VCVRack（我以此类比真实的物理合成器）中工作的演示。



**用示波器画台阶**

# 开始画图

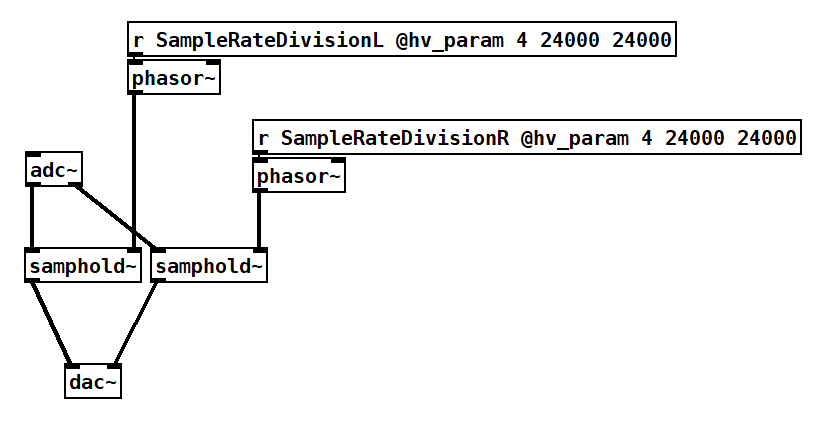
在了解了原理之后我们便可开始为生产插件准备图纸（Patch）了。

与单独使用PD情况不同的是，为了Patch能够被Heavy正常编译，我们只能使用PD当中的部分物件，否则编译时会报错。我将在本文末尾提供不可使用的PD物件列表。

为了让插件开放参数给目标平台（本文中是Wwise），我们需要按下图的格式声明参数：在PD中按Ctrl+1创建物件，填入"r”——代表该物件从外部接受信息；"SampleRateDivisionL"——在插件界面上显示的参数名称；@hv\_param——代表该物件将被Heavy编译成参数接口；4、24000、24000分别为参数的最小值、最大值和默认值。



1，为了对输入信号的左右声道进行不同的处理，增加不对称性，我在此为左右声道各创建一个参数。phasor~物件为用作S/H模块时钟的锯齿波振荡器，samphold~物件就是S/H模块，dac~物件为音频信号输出的地方。



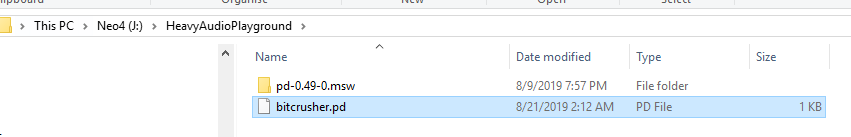
**Note：此Patch中adc~代表信号输入，当Patch当中存在adc~时，Heavy将会生成用于构建效果器插件的代码；若没有adc~物件，则Heavy会生成用于构建音频源插件的代码。**

2，接下来测试patch的运作。



**Note：我添加了控制采样频率的hslider物件，由于该物件在编译成插件后将不起作用，因此测试完之后可以删除。**

3，最后将Patch保存到**J:\HeavyAudioPlayground\bitcrusher.pd**



**Note：这个地址大家可随意选择，注意之后还会用到。**

现在“图纸”已经画好了，下一篇将介绍我们的“自动化车间（Heavy）”的建造，产品（插件）的生产以及部署！

敬请期待！