VCB-Studio 教程 07: AviSynth 基础与入门

0. 前言

这篇教程需要教程 6：VS 基础与入门作为前置

尽管 avs 早于 vs 存在了十几年，上手写一个 hello world 级别的脚本也比 vs 容易，但是 avs 混乱的参数传递机制，使得想深入学习 avs 的使用，比 vs 其实来的困难。本教程假定你通过 vs 教程，已经对变量、参数传递等有了最基本的认识，这样，当我们讨论 avs 特有的一些机制的时候，不至于从零开始。

AviSynth 主页和文档：http://avisynth.nl/index.php/Main_Page
AviSynth+主页和文档：http://avs-plus.net/
在线词典：http://dict.cn/

avs 目前最新版是 avs 2.6.0，只有 32bit；
avs+是 avs 的一个改良 mod，优势在于，有官方 64bit 版本。
尽管以 vcb-s 对于 avs+的使用，全局 64bit 化是可行的，但是因为懒，所以直到转 vs 之前，vcb-s 一直使用 32bit 版本的 avs。如果你希望使用 64bit 的 avs，建议使用 avs+。

有 vs 的存在，avs 的意义不是很大，至少作为定位高质量、复杂处理的压制，vs 优秀的内存管理机制、原生的多线程优化，和各种新科技滤镜，让它对比 avs 已然优势明显。但是 vcb-s 系列教程依旧讲述 avs，因为很多以前的教程和脚本是基于 avs 的，我们需要保证大家能理解并继承上个时代的智慧结晶。

1. 简单的 avs 脚本

以下是一个简单的脚本，以 YUV420P16 的格式读入一个 mkv，并转为 RGB32 显示：

```
SetMemoryMax(1000)
a = "00000.mkv"
LWLibavVideoSource(a,format="yuv420p16",stacked=true)
dither_convert_yuv_to_rgb(chromak="lanczos",taps=4,noring=true,lsb_in=true)
```

avs 原生不支持多线程，但是支持设置最大使用内存。这里我们用 setMemoryMax()来设置最大使用 1000MB。

除了系统设置（比如 setMemoryMax），avs 的主要内容一般由两种语句构成：赋值句和输出句。

赋值句的含义和 vs 的赋值句大致相同，表现为 变量=表达式 的结构。比如 a = "00000.mkv"就是一个赋值句。
avs 的函数不再有各种域，只要载入了，直接就可以用。avs 的函数一般来自两种地方，第一种是滤镜原生 dll，第二种是写好的库，后缀名为 avsi。这两个种文件一般放在 avs 根目录的 plugins 文件夹内，这样 avs 就可以自动载入。

输出句，表现为直接将表达式作为一行，比如：
LWLibavVideoSource(a,format="yuv420p16",stacked=true)
dither_convert_yuv_to_rgb(chromak="lanczos",taps=4,noring=true,lsb_in=true)
这两句就是两个输出句。

avs 中，随时随地维护一个叫做 last 的 clip，这个 clip 要么为空值，要么记录上一个输出类型为 clip 的输出句，输出的结果：

SetMemoryMax(1000) <- 这句是系统设置，不产生 last
a = "00000.mkv" <- 这句是赋值句，不产生 last
LWLibavVideoSource(a,format="yuv420p16",stacked=true) <- 这句结束后，last 为 lwlvs 载入的 YUV 视频
dither_convert_yuv_to_rgb(chromak="lanczos",taps=4,noring=true,lsb_in=true) <- 这句结束后，last 为 dither_convert_yuv_to_rgb 转为的 RGB24

虽然语法上，avs 允许输出句输出非 clip 的类型，但是 last 并不会去记录。一般而言，也没有必要写出非 clip 输出的输出句。

avs 结束的时候 ,输出最后一个输出句的结果 ,相当于输出 last。如果 last 为空(全程没有一个 clip 类型的输出句), 返回效果是 Not a Clip 的报错信息。

2. AVS 函数的调用和参数传递

无论是赋值句，还是输出句，avs 进行运算主要也是通过函数进行的。函数的调用，以及参数的传递，跟 vs 有类似性。比如我们看 LWLibavVideoSource 的 doc：

LWLibavVideoSource(string source, int stream_index = -1, int threads = 0, bool cache = true, int seek_mode = 0, int seek_threshold = 10, bool dr = false, int fpsnum = 0, int fpsden = 1, bool repeat = false, int dominance = 0, bool stacked = false, string format = "", string decoder = "")

其规则跟 VapourSynth 也几乎一致：除了 source，其他的都有默认值，source 在调用时候必须给定，其他的则可以缺省。

有些时候，比如 CSMod16 的 avsi 里，function header 是这么写的：
CSmod16(clip filtered, clip "source", clip "pclip", bool "lsb_in", bool "lsb", int "dither"……

规则是：没有被引号括起来的都是必须输入的（上文中仅 filtered 一个），用引号括起来的是可以缺省的（上文中剩下所有）

avs 的参数传递一般有 4 种：
1. 赋值性传递/关键字传递(keyword argument)， 在 avs 的 doc 里面被称为 named arguments。这点跟 vs 相似；
2. 直接传递/位置性传递(positional argument)，在 avs 的 doc 中被称为 argument list。这点也跟 vs 很相似；所以 LWLibavVideoSource(a,format="yuv420p16",stacked=true)，a 是直接传递，format="yuv420p16"和 stacked=true 则是赋值性传递。
3. 串联式传递，在 avs 中被称为 OOP Notation，跟 vs 的串联性传递相似，都是让前一个运算结果作为后一个函数的第一位输入，比如：AVISource("fraps.avi").dither_convert_rgb_to_yuv()。 串联式的传递在 vs 中不普遍（主要是这玩意最近才加入），avs 中却是普遍使用的，因为从 avs 最初设计这种方式就存在。
4. last 传递，是指当函数第一位输入是一个 clip，且第一位输入没有被关键字传递或串联传递，且总输入的参数不足以填满所有必须输入，且 last 不为空，那么系统将 last 作为函数的第一位输入。这是 avs 特有的一种传递方式。举个例子：

第 4 点是新手最容易弄混的地方。我们来拆开强调一次：

(1). 第一位输入是一个 clip 类型输入，其实这个绝大多数 avs 滤镜都符合条件（除了源滤镜一般第一个输入是字符串）， 一般你看 doc 都是 mt_edge (clip, string "mode", int "thY1"…) 这种上来一个第一个是 clip。
注意这里 clip 指定的时候是没有变量名称的，这意味着没有办法进行赋值性传递。而之前 CSMod16 上来是：CSmod16(clip filtered, clip "source", clip "pclip",……) 这时候你就可以用 CSMod16(filtered=dbed) 类似方式进行赋值性传递。

(2). 没有被关键字或者串联传递。比如我们看下面这个例子：
LWLibavVideoSource("00000.mkv",format="yuv420p16",stacked=true)
dither_convert_yuv_to_rgb(chromak="lanczos",taps=4,noring=true,lsb_in=true)

dither_convert_yuv_to_rgb()的 doc 如下（可以在 dither_tools 的包中找到）：

```
Dither_convert_yuv_to_rgb (
        clip   src,
        string matrix (undefined),
        bool   interlaced (false),
        bool   tv_range (true),
        string cplace ("MPEG2"),
        string chromak ("bicubic"),
        float  fh (undefined),
        float  fv (undefined),
        int    taps (undefined),
        float  a1 (undefined),
        float  a2 (undefined),
        float  a3 (undefined),
        bool   lsb_in (false),
        int    mode (undefined or 6),
        float  ampn (undefined or 0.5),
        string output ("rgb32"),
        int    ampo (undefined),
        bool   staticnoise (undefined),
        bool   noring (false)
)
```

可见这个滤镜上来强制输入一个 clip src。（有变量名 src，意味着可以进行 src=xxx 这样的赋值性传递。） 而看上文，我们只是通过赋值性传递，指定了几个可选性的参数，src 这个 input 没有被赋值性传递载入，也没有被串联输入。

(3). 总输入的参数不足以填满所有必须输入。我们输入的必须参数是 0 个，而滤镜要求的必须参数是 1 个。

(4). last 不为空。在执行 dither_convert_rgb_to_yuv 之前，last 的确不为空，记录着 lwlvs 输出的结果。

(1)+(2)+(3)+(4)同时满足，系统就会把 last 传递给函数，作为函数第一个强制性输入的参数。

last 以及 last 传递的引入，本质上是为了简化 avs 的语法和书写的。一般你看到入门级别的 avs 全是输出性语句，没有任何赋值性语句，其实就是不断地更新 last 并作为下一个函数的输入：

AVISource("fraps.avi") #读入 fraps 录制的 avi，RGB 格式
dither_convert_rgb_to_yuv() #转为 YUV 格式，准备压制

用 vs 写你一般得这么写（无视最近允许串联式写法，输出句改赋值句，最后手动指定输入。）：
src = core.avisource.AVISource("fraps.avi")
res = mvf.ToRGB(src)
res.set_output()

但是 avs，同样类型的写法可以玩出花，以下所有段落，都属于常见写法，效果都是一样的：

AVISource("fraps.avi").dither_convert_rgb_to_yuv() #用串联传递

dither_convert_rgb_to_yuv(AVISource("fraps.avi")) #用直接传递

AVISource("fraps.avi")
dither_convert_rgb_to_yuv(last) #用直接传递，注意 last 可以作为一个表达式参与直接传递

AVISource("fraps.avi")
dither_convert_rgb_to_yuv(src=last) #同理，last 可以作为表达式进行赋值传递

AVISource("fraps.avi")
last.dither_convert_rgb_to_yuv() #同理，last 还可以用于串联传递

src=AVISource("fraps.avi")
src.dither_convert_rgb_to_yuv() #串联传递

src=AVISource("fraps.avi")
dither_convert_rgb_to_yuv(src=src) #赋值传递， vs 教程中我们也见过，前一个 src 是变量，后一个是表达式。

src=AVISource("fraps.avi")
dither_convert_rgb_to_yuv(src) #直接传递，这里 src 就作为一个表达式。

但是以下所有红字写法都是不可行的（黑字是改正版本）：

src=AVISource("fraps.avi")
dither_convert_rgb_to_yuv()

错的原因是，第一句是赋值句，不会触发 avs 记录 last，所以到了下一句，没有 last 可以丢给滤镜作为输入。除了上文的改正方法，另一种改正写法为：

src=AVISource("fraps.avi")
src #通过这一句做一个输出语句，avs 将记录 last
dither_convert_rgb_to_yuv()

AVISource("fraps.avi")
convert=dither_convert_rgb_to_yuv()

错的原因是，第二句是赋值句，不会触发 avs 记录 last，最终 last 是 AVISource 输出的 RGB 视频，而不是转换后的。除了把第二句换为输出句，一个简单的修复是：
AVISource("fraps.avi")
convert=dither_convert_rgb_to_yuv()
convert

```
src=AVISource("fraps.avi")
convert=src.dither_convert_rgb_to_yuv()
```

同理，整个脚本是个赋值句，不会触发 last。

现在，我们来看看更复杂的。再来回顾一下 CSMod16，看看当年雯姐在帖子里说了哪种教科书式错法（https://www.nmm-hd.org/newbbs/viewtopic.php?t=781）：

```
CSmod16(clip filtered, clip "source"……)
```

无视其他参数，我们知道 CSMod16 可以只输入一个 clip filtered，这种情况下，它对 filtered 做主观锐化；CSMod16 还可以输入两个 clip，一个是 filtered，一个是 source，这种情况下，它以 source 做对比，对 filtered 进行补偿性锐化。

（主观锐化和补偿性锐化在 http://vcb-s.com/archives/4738 中有说，简单总结：主观锐化，就是把图像往锐利方向去调，往往造成画风突变；而补偿性锐化，则是对源进行降噪等处理后，拿处理后的东西进行锐化，而锐利度不会比源高，以此试图补偿降噪等处理造成的杀伤，而非意在改变画风）

假设用 dfttest 降噪，然后再用 CSmod16 做补偿性锐化：

```
LWLibavVideoSource("00001.m2ts",threads=1)
source = last
denoised = source.dfttest()
denoised.CSmod16(source)
```

最后一句，使用了串联式传递，所以第一个 clip 类型强制性输入的 input（filtered）会被设置为 denoised，而这时候我们还输入了一个 source，这个 source 将会被传递给剩下 input 中第一个，也就是 clip "source".

典型的错误写法如下:

```
LWLibavVideoSource("00001.m2ts",threads=1)
source = last
dfttest()
CSmod16(source)
```

为什么这是错的？因为 CSMod16 只需要一个强制性输入，这时候，我们已经指定了 source，这个 source 会被赋值给 filtered ,然后 ,avs 认为所有 input 传递完毕 ,并无缺少 ,所以这样的效果就是 CSMod16 只输入了一个 clip，对源执行主观锐化。

下文的写法是完全正确的：

```
LWLibavVideoSource("00001.m2ts",threads=1)
source = last
dfttest()
last.CSmod16(source)
```

这时候，通过串联赋值，filtered 会被赋值为 last(注意，dfttest()执行后，last 会被更新)，然后多输入的 source 会被传递给 clip "source"。CSMod16 收到了两个 input。同理，最后一句还可以改为 CSMod16(last, source)，一样可以正确工作。

总结一下，avs 的参数传递机制如下：

1. 如果是串联性赋值句，把上游输出的 clip 作为滤镜第一个强制性的 clip 输入；
2. 执行所有赋值性传递，如果有重复（包括：串联性传递第一个 input 的同时，赋值性传递第一个 input，比如 denoised.CSMod16(filtered=last)，等于说 filtered 同时被指定为 denoised 和 last），报错；
3. 清点所有剩下的直接传递 如果不足以满足所有的强制输入 且函数非串联 且第一个 input 没有被赋值性传递，且 last 存在，那么把 last 设置为第一个 input
4. 把所有剩下的直接传递，按照顺序，赋值给剩下的 input。
5. 如果有任何类型不匹配，或者强制输入的数量还是不够，报错。

3. 一些简单的视频编辑

在本章中，我们讲述一些 avs 中常见的用法，方便大家学习和上手

3.1 裁剪和缩放

裁剪靠的是 Crop, 缩放靠的是 Spline36Resize
doc 分别为：
http://avisynth.nl/index.php/Crop
http://avisynth.nl/index.php/Resize

假设我们读入一个原生 4:3 ,通过加黑边做成 1920x1080 的视频 ,我们先把它切割成 1440x1080(就是左右各 240 个像素), 然后缩放成 720p:

```
src = ...
Crop(src, 240, 0, -240, 0)
/*
注意这里 crop 的用法和 vs 是不同的：
如果 right 和 bottem>0，那么指定的是成品宽和高；
如果 right 和 bottem<=0，那么指定的是切割的像素。
vs 里面，CropRel 永远是切割的像素，且要求>=0。
最后说一下，你看到的这个跟 C++一般的，就是 avs 里大范围注释的方法。
所以这到这里 avs 脚本还没完呢，下面还有个缩放到 720p
*/
Spline36Resize(960,720)
```

3.2 分割与合并

分割靠的是 trim (http://avisynth.nl/index.php/Trim)
合并靠的是 Slice(http://avisynth.nl/index.php/Splice)

Trim 的用法跟 vs 里的基本一致，比如说我们要 Trim 出开头 100 帧：

```
LSMASHVideoSource("xxx.mp4")
Trim(0,100-1)#也可以用 Trim(0,-100)，-100 是什么意思自己看 doc
```

avs 里面的 trim 就没有语法糖了。

合并的用法就有点特殊了。首先，avs 里面，要想合并，必须要求视频参数一致（这点跟 vs 不同），然后 avs 是支持音频的，合并分音频同步合并（AlignedSplice）和不同步合并（UnalignedSplice），区别 doc 里有写，同步合并会把第一个视频的音频通过切割或者加静音，来保证第二个视频接上后，音轨是吻合的。

同步合并，可以通过++来实现；非同步可以用+。当你的 avs 没有载入音轨/不需要在意音轨，这俩方式没有区别。

```
v1=AVISource("fraps1.avi")
v2=AVISource("fraps2.avi")
v1++v2
Dither_convert_RGB_to_YUV() # avs 是不分大小写的
```

3.3 简单的降噪，去色带和加字幕

降噪用的是 RemoveGrain()，去色带用的是 f3kdb()，加字幕用的是 TextSub()

一样，自己去找找 doc。有的时候 doc 会附带在滤镜的下载包里。

```
src = ...
RemoveGrain(src, 11, 4) #注意这里不再跟 vs 一样用[11,4] 的数组写法
f3kdb(12,32,24,24,0,0)
TextSub("xxx.ass")
```

4. AVS 里面对视频性质(clip property)的读取

同 vs，avs 里面可以直接读取一些关于视频和帧本身的性质，比如说视频的总长度，帧率，一帧的长宽，类型等。这部分在 http://avisynth.nl/index.php/Clip_properties 中有详细解释，我们只列举最常用的几个：

clip. FrameCount 返回 clip 的总帧数。所以要切掉视频的首帧（第 0 帧），可以这么写：
src = ...
Trim(src, 1, src.FrameCount-1)

注意，这里的 FrameCount 其实是一个函数：
int FrameCount(clip)

之所以可以写成 last.FrameCount 这样的形式，是因为：
1. 这实际上是串联传递，将 last 传递给 clip 作为输入；
2. 如果不需要再写任何参数，avs 可以将函数括号省略。

所以你完全可以写成函数的形式：
src = ...
Trim(src, 1, FrameCount(src)-1)

clip.width, clip.height 返回 clip 的宽和高。比如我们想缩放 last 到 1/2 大小：
Spline36Resize(width/2, height/2)

这个写法，用 last 用到了滥用的地步。首先 width() 和 height() 这两个函数，没有任何输入（所以连括号都省了），那么系统把 last 拉来做输入；
然后，spline36resize 缺输入，系统再把 last 拉过来。
等效于：
last.Spline36Resize(width(last)/2, height(last)/2)

avs 里面除法是/，如果参与运算的都是整数，那么就做整数除法，比如 1080/23=46。如果参与运算的有浮点，那么就做浮点数除法，比如 1080/23.0=46.9565... 你可以用 Int() Float()这些函数做强制类型转换。比如说我们要把一个视频长宽缩小到 2/3，并且保证长宽都是 16 的倍数：

w = round(width(last)/1.5/16) * 16 #round 是将一个实数四舍五入到最近的整数
h = round(height(last)/1.5/16) * 16
Spline36Resize(w, h)

假设我们有个 1600x900 的视频：

round(width(last)/1.5/16)*16 = round(1066.666/16)*16 = round(66.66)*16=67*16=1072
round(height(last)/1.5/16)*16 = round(600.0/16)*16=round(37.5)*16=38*16=608

所以最后成品是 1072*608。

思考题，假设我们写法为：
w = round(width(last) /16/1.5) * 16
h = round(height(last)/16/1.5) * 16
Spline36Resize(w, h)

对于一个 1600x900 的视频，最后成品是多大？写个脚本试试看。

5. AVS 里面对音轨的处理

avs 里可以处理音轨。类似 AVISource 这样的滤镜读 avi，如果有音轨，是可以一并读取进来的。处理的时候，往往是视频音频各自过各自的滤镜，但是类似切割和合并，视频音频会被视为一体。

音频可以被单独的读入（比如 LWLibavAudioSource），可以被合并到一个视频轨道里（AudioDub），可以从带音频的 clip 中单独提取（KillVideo，把视频去掉，只剩下音频信息）。这些功能有时候相当好用，比如说 BDMV 有的时候是两集连在一个 m2ts 里，通过章节信息，查到第一集一共 34000 帧，那么可以这么写：

```
video = LWLibavVideoSource("00001.m2ts",threads=1)
audio =LWLibavAudioSource("00001.m2ts")
AudioDub(video,audio)
EP1=Trim(0,34000-1)
EP2=Trim(34000,0) #last=0 表示一直切到结束
EP2
```

这样就可以输出视频+音频的 EP2。音频部分，可以丢 MeGUI 压 flac，相当于用 avs 做精确的音频切割。

6. AVFS 的使用

AVFS，全程是 Avisynth Virtual File System，是可以把你的 avs 伪装成 avi，直接给任何支持 avi 的软件使用。其实 vs 也有这玩意，但是 vs 原生不支持音频，而且使用这个机制大概也只是为了让只支持 avi 的东西（比如上古转码软件，或者是非编）支持 mp4/mkv/10bit/HEVC……，不需要其他额外太多处理，所以 avs 可能写起来还更简单。

网址如下：http://turtlewar.org/avfs/
照着 readme 做就好。这玩意先要装一个 Pismo File Mount Audit Package，然后每次手动用命令行来“伪装”一个 avs。

其他一些 AVS 的高级用法，比如 runtime 机制，比如自定义函数，我们会在以后的教程中详细说。