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）：



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。