VCB-Studio教程12 16bit YUV的处理

本教程旨在讲述avs/vs处理中，YUV不同bitdepth的转换，以及一些简单的高精度处理用法。

0. YUV的量化（以下部分主要来自mawen1250的讲解）

RGB模型下，三个平面都属于强度，而转换为YUV之后，只有亮度Y属于强度，UV则属于色差。在模拟信号表示（或者计算机中浮点数表示）中，用0-1来表示强度的强弱。R、G、B的范围都是0-1前提下，Y的来源是R、G、B的加权平均，权重和为1，所以Y也是0-1。在使用无符号整数的时候，8bit下常常用[16,235]来对应0-1，这就是tvrange(limited range)；也可以用[0,255]来对应，这就是pcrange(full range)。注意，一般RGB永远使用的是pcrange，只有YUV会出现pcrange和tvrange的设定。

U、V的来源是对R、G、B、Y作差。比如Cb是类似B-Y，Cr是类似R-Y。那么不做任何其他处理的话，其范围是一个奇怪的值： 现在假设Y=(R+G+B)/3，那么Cb最小的值就是当B=0、R=1、G=1，也就是-2/3；Cb最大的值是当B=1、R=0、G=0，这时候Cb就是2/3。于是Cb的取值范围就变成了[-2/3, 2/3]。

但我们显然不希望UV的取值范围是这么一个奇怪的数值，而且是和R、G、B在Y中的权重有关。所以我们就给它乘上一个修正的scale，标准化到[-0.5, 0.5]的范围，动态范围和Y保持一致，都是1。而UV之所以可以取负值是因为它是作差的产物，所以当B和Y相同的时候，代表没有色差，Cb就是0，而0就是色差通道的中点（neutral point）。

在用无符号整型表示数据的时候，因为没有负数，所以我们人为给UV规定了一个中点，在8bit下就是128。然后[0-0.5, 0+0.5]就被对应到了[128-112, 128+112]，也就是[16, 240]，这是tvrange下UV的取值范围；对于UV来说，limited range下8bit动态范围就是224。

然而full range下，8bit的动态范围是255，所以定义成了[128-255/2, 128+255/2]，得到了[0.5, 255.5]，不在[0,255]范围内。所以full range的chroma其实定义并非完美；好在实际操作中极难碰到临界的chroma取值，所以实际影响不大。

总结一下：8bit下，tvrange时候，Y的范围是[16,235]，UV的范围是[16,240]，pcrange下则都是[0,255]。UV以128为neutral point。

当扩展到更高bitdepth的整数时候，比如转换为10bit，tvrange下的操作是YUV都 \* 4，原来是(160, 60,140)，变成(640,240,560)。pcrange下，原则上是Y转换到[0,1]的浮点，UV对齐中点，再进行对应的转换：

Y：160/255\*1023 ≈ 642

U：512 + (60-128)/255\*1023 ≈ 239

V: 512 + (140-128)/255\*1023 ≈ 560

注意，UV不等于8bit的数值直接\*1023/255。虽然一个简单的近似公式是全都 \* 4，这在tvrange下是精确计算，pcrange下是近似计算。16bit下计算方法类似，只不过倍数从4变成了256，动态范围从1023变为65535。

1. avs中高精度YUV的显示方法——stacked/interleaved 16bit

avs当中，默认的精度是每个像素，每个通道8bit. 如果想使用10/16bit的精度，就必须通过8bit来伪装，更精确的说，让两个像素的位置，来记录一个像素的信息，从而提供16bit的储存空间。

考虑如下2x2的16bit数值：

5762 18329

52543 345

每个数字的前8位，我们称为MSB(Most Significant Bits,最重要的位数。好比一个数字的十位比个位重要)，后8位称为LSB(Least Significant Bits,不重要的位数)

将每个数字转换成(MSB, LSB)的格式，我们得到(MSB\*256 + LSB=16位原数)：

(22,130) (71,153)

(205,63) (1,89)

其中，每一位的范围是0~255， 128为中值。所以如果将这四个16bit的数值，通过四舍五入(rounding)到8bit，结果应该是：

23, 72

205,1

换言之，和这组16bit的显示的效果，最接近的8bit矩阵就是上文所示。

如果采用截位取整，结果和四舍五入相差并不大：

22, 71

205,1

一个简单的想法是，通过两个像素来并排显示一个像素，这样显示的后果是：

22,130,71,153

205,63,1,89

这种显示方式，叫做interleaved 16bit。interleaved是交织的意思。它的特点是，图像宽度增倍，每个数据的MSB和LSB交织显示。其中，奇数列的数值，正好是截位取整后的数值。所以从Interleaved 16bit中，还是可以稍微看出点片源痕迹的：

原图：

Interleaved 16bit:



除了Interleaved 16bit，还有一种存储方式，叫做stacked 16bit(stacked 意思是层叠)。它的规则是：上下层叠两个矩阵，第一个矩阵放MSB，第二个矩阵放LSB。

比如之前提到的

(22,130) (71,153)

(205,63) (1,89)

如果用stacked 16bit来显示：

22 71

205 1

130 153

63 89

它的特点是，图像高度增倍，而且上半部分完全是16bit通过截尾后转为8bit。所以stacked 16bit在avs中，表现是上半部分正常，下半部分很乱：

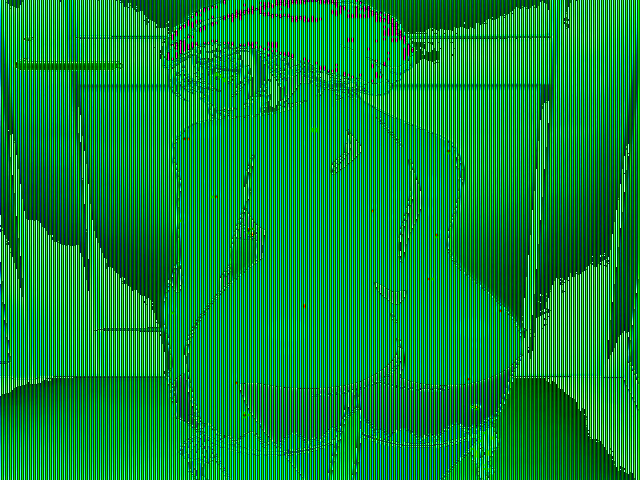
原图：

转stacked 16bit：

2. 10bit YUV的显示方法——stacked/interleaved 10bit

10bit的理解方式和16bit是完全一样的，除了10bit的MSB其实只有两位，最多是0~4的取值。加上8bit的LSB，构成10bit的精度。

因为10bit的MSB很接近0，所以只看MSB部分，效果很接近YUV=(0,0,0)的效果（深绿色），但是在stacked 16bit中，你还是略微可以在MSB部分看出轮廓：

interleaved 10bit:

stacked 10bit：

3. 日常处理中高精度的使用原则

x264/x265可以接受原生8bit输入，也可以支持interleaved 9~16bit的输入，输入的时候通过--input-depth 指定。一般来说，编码10bit AVC/HEVC，都是直接将处理好的interleaved 10数据喂给编码器。你也可以输出interleaved 16bit，然后让编码器自己抖动到10bit（x265需要加入--dither指令）。也就是说：

avs中使用interleaved 10bit输出，同时编码指令中加入--input-depth 10，或者avs中使用interleaved 16bit输出，同时编码指令中加入--input-depth 16，如果是x265，再加入--dither

avs脚本，很多滤镜支持16bit的处理，是通过stacked 16bit实现的。16bit处理对平面高精度过渡非常有利，因此很多涉及到平面部分的处理，比如resize，降噪，去色带等，都推荐在16bit下做处理，并且全程保持16bit的精度。

处理完毕后，如果需要压制8bit，就降低到8bit，否则转为interleaved 10/16bit输出。

如果需要预览avs，则转换为RGB24。

4. avs中不同精度之间相互转换的方法

如何在不同精度间相互转换，avs的Dither工具提供了很多好用的滤镜：

8bit->stacked 16bit: U16()

比如说：

LWLVS(“00001.m2ts”) # yuv420p8

U16() # yuv420p16 stacked

stacked 16bit->8bit: ditherpost()

LWLVS(“00001.mkv”,format=”yuv420p16”,stacked=true) #将一个mkv读入为stacked 16bit, 这个写法适用于读入10bit视频

ditherpost() #yuv420p8

ditherpost默认会做抖动处理，效果比直接截尾或者四舍五入来的好。

stacked 16bit->interleaved 16bit: Dither\_out()

LWLVS(“00001.m2ts”) # yuv420p8

U16() # yuv420p16 stacked

Dither\_out() # yuv420p16 interleaved

interleaved 16bit->stacked 16bit: C16()

LWLVS(“00001.mkv”,format=”yuv420p16”,stacked=false) # interleaved 16bit

C16() # stacked 16bit

stacked 16bit->interleaved/stacked 10bit: Down10(stack=false/true)

LWLVS(“00001.m2ts”) # yuv420p8

U16() # yuv420p16 stacked

Down10(stack=false) # yuv420p10 interleaved

5. avs中一些简单的16bit滤镜使用方法

很多滤镜就是原生工作在stacked 16bit下面的。比如说Dither\_resize16，就是16bit下的resizer。Dither\_RemoveGrain16()可以用于降噪。还有些滤镜通过参数来控制输入输出是否为16bit；比如f3kdb的input\_mode和output\_mode, 设置为1的时候表示使用stacked 16bit。

举一些简单的例子：

1、读入一个m2ts，在16bit下做降噪和去色带，然后转为720p输出，准备做10bit压制：

LWLVS() #yuv420p8

U16() #yuv420p16 stacked

Dither\_RemoveGrain16() #降噪

f3kdb(input\_mode=1,output\_mode=1) #去色带，输入输出指定为stacked 16bit

Dither\_resize16(1280,720) #降低为720p

dither\_out() #yuv420p16 interleaved

2、读入一个10bit MKV，转为720p，在16bit精度下加字幕，准备做8bit压制：

LWLVS(“mkv”,format=”yuv420p16”,stacked=true) #yuv420p16 stacked

Dither\_resize16(1280,720) #降低为720p

Textsub16(“subtitle.ass”) #加字幕

ditherpost() #输出为yuv420p8

3、读入一个30i的演唱会原盘，对它进行反交错，然后用SMDegrain做高精度降噪，准备给10bit压制：

LWLVS(“m2ts”)

QTGMC(preset=”slow”,border=true) #反交错。注意QTGMC不支持16bit处理

pre\_nr16 = last.U16() #反交错后，降噪之前，记录一个16bit的备份

SMDegrain(tr=2,thSAD=500,thSADC=200,refinemotion=true,pel=2,truemotion=false,hpad=16,vpad=16,lsb\_out=true) #注意输入依旧是QTGMC后的结果，为yuv420p8

#输出通过lsb\_out=true来指定为yuv420p16. SMDegrain的内部运算精度为32bit浮点数。

dither\_repair16(last, pre\_nr16, 3, 3) #在16bit下对降噪的结果，对比pre\_nr16做一些修复补偿。

dither\_out() #转为interleaved 16bit输出。

对于avs结尾，可以加上这么一段开关：

output\_depth = 10

output\_depth == 10?Down10(stack=false):dither\_convert\_yuv\_to\_rgb(lsb\_in=true,a1=0,a2=0.5)

注意，第一句的=是赋值，表示将output\_depth 赋值为10；

第二句的==是判断是否相等，是的话，执行dither\_out(),否则执行dither\_conver\_yuv\_to\_rgb(lsb\_in=true)，做stacked 16bit yuv->RGB的转换。

这段开关的作用是：

1、你可以通过修改第一句，表示你想输出10bit与否

2、如果你指定输出10bit，avs会做stacked 16bit->interleaved 16bit

3、否则，avs会做yuv420p16->RGB24，供你预览和调试avs。

调试的时候可以将output\_depth 赋值为8；编码前记得改为10。如果是输出16bit如法炮制；只不过Down10改为dither\_out()

6. vs中不同精度之间相互转换的方法

vs原生支持高精度的yuv信息，但是这不意味着vs中处理精度和输入输出都自动用最高精度进行。vs中，我们也需要用滤镜来进行处理。典型的包括fmtc和zimg所提供的滤镜。

fmtc的使用比较简单，就一个bitdepth函数:

src16 = core.fmtc.bitdepth(src8,bits=16)

down10 = core.fmtc.bitdepth(res16,bits=10)

其中，你可以用fulls和fulld表示输入输出是否是full。默认都是false，表示进行tvrange的转换。只不过fullrange下在处理边界值有一点小问题，所以强迫症可以用zimg处理涉及fullrange的转换。

zimg则可以通过mvf中的depth来实现：

src16 = mvf.Depth(src8, depth=16)

down10 = mvf.Depth(res16,depth=10)

同时它也是通过fulls和fulld来指定输入输出的range。比如说我们把一个fullrange yuv8bit转为limited 16bit:

src16 = mvf.Depth(src8, depth=16, fulls=True, fulld=False)

vs中，多数情况下，滤镜的处理精度和输出精度就是输入的精度。所以vs的高精度处理无需指定bitdepth，滤镜会自动读取精度信息，并自动选择输出精度。比如说系统自带的resize，就是输入啥精度，输出啥精度。也有强制精度的，比如说SangNomMod就只能支持8bit 输入输出，16bit的clip。使用的时候需要先降低到8bit，处理完了再升回去。也有类似f3kdb.Deband这种，可以通过output\_depth来指定输出精度的。

vs的脚本末端可以加上这个指令：

Debug = False

if Debug:

res=core.std.Interleave([src16,res])

res=mvf.ToRGB(res,full=False,depth=8)

else: res = core.fmtc.bitdepth(res,bits=10)

其作用是，当Debug是1/True的时候，表示开启调试模式，先将src16和res(一般处理完毕也是16bit)交织每一帧，然后转为RGB24显示，这样可以绕开vsedit自带的转RGB。否则当debug=0/False时候，表示关闭调试模式，将处理结果转为10bit输出，准备喂给编码器。