# VCB-Studio 教程 26 YUV 和 RGB 互转(2)

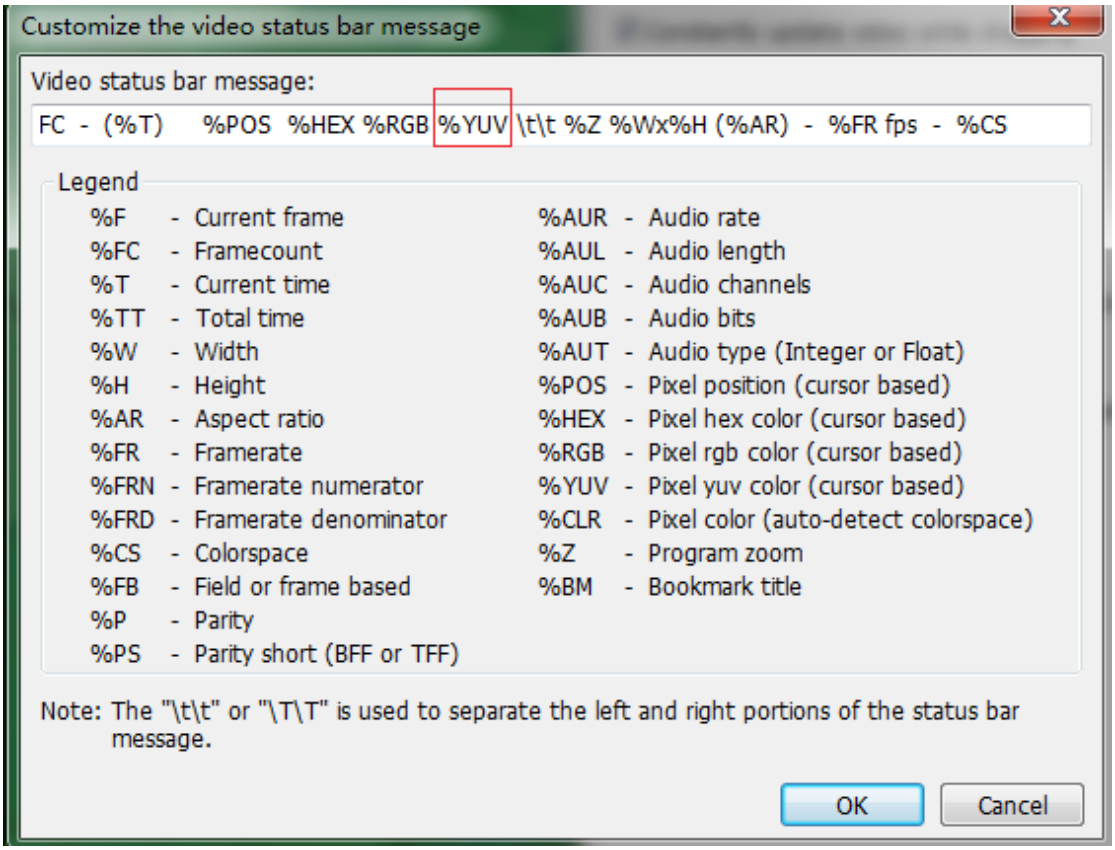
## 《快要坏掉的八音盒》制作参考

本教程旨在以《快要坏掉的八音盒》一番，来讲述动漫制作中如何判断蓝光本身的颜色错误，并做针对性的修复。同时教授大家如何批量做截图，和处理全静止的 menu 特典等。

片源下载链接: http://pan.baidu.com/s/1ePwy8 密码: v951

## 1. avspmod 中如何设置显示 RGB 和 YUV 信息

要了解视频中某个区域/像素的 YUV/RGB 信息 我们需要有个能查看视频里数值的方法。avspmod 就有这个功能。在 avspmod 的菜单栏里，Options-program settings，Video 选项卡中点击 customize video status bar...




最上面的输入地方就是给你自定义的。其实就是写各种代码来表示不同的意义，比如%YUV 就是显示 YUV 信息，把它找个合适的位置（比如%RGB 旁边）加上去。

接下来，在调试视频的时候，如果你的最终输出是 YUV (就是不主动转为 RGB 输出)，你就可以在下方的状态栏中看到 YUV 信息了。比如下图我鼠标停在红色小圈中，近似白的部分：


可以看到 YUV 信息是(219,128,128)，非常接近标准白(235,128,128)。这个信息是直接读取的。

如果输出的是 YUV 信息，avspmod 会自动计算转换为 RGB 数据并显示。matrix 和 range 可以在菜单栏 video - YUV->RGB 里面选择。注意计算的 RGB 数据，精度和准确度不是很高。所以在预览视频成品信息的时候，我们一般选择用 dither tools 做 YUV->RGB，而不是用 avspmod 自带的转换。

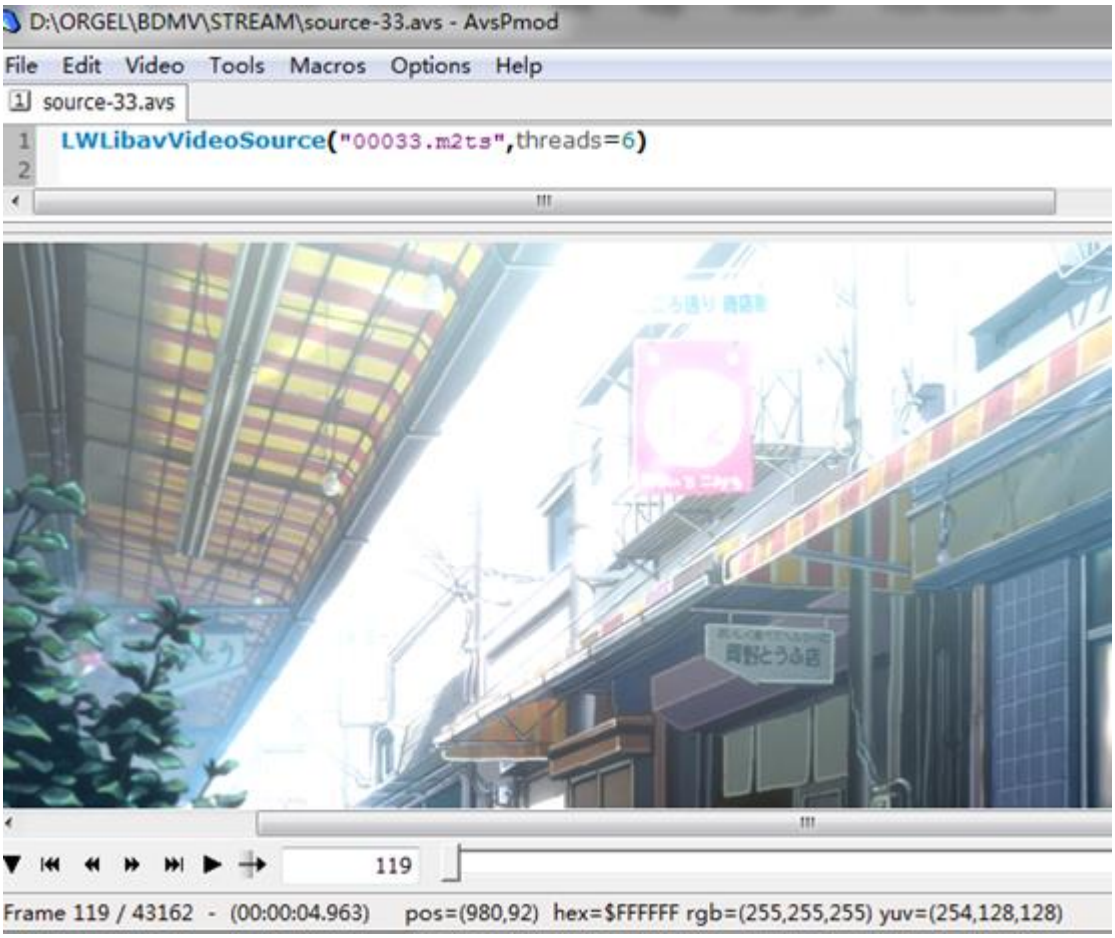
如果你最终输出的是 RGB，那么下方 RGB 信息就是原生输出的值，而 YUV 信息则是 avspmod 计算出来的。注意 avspmod 计算结果精确度并不是很高。所以如果你想看 YUV 信息，就选择 YUV 输出；想看精确的 RGB 信息，就选择 RGB 输出。

VSEdit 的预览原生自带查看 YUV/RGB 信息。如果你的输出是 YUV，那么 vsedit 预览显示 YUV；否则显示 RGB。

## 2. 观测视频中不正常的 range

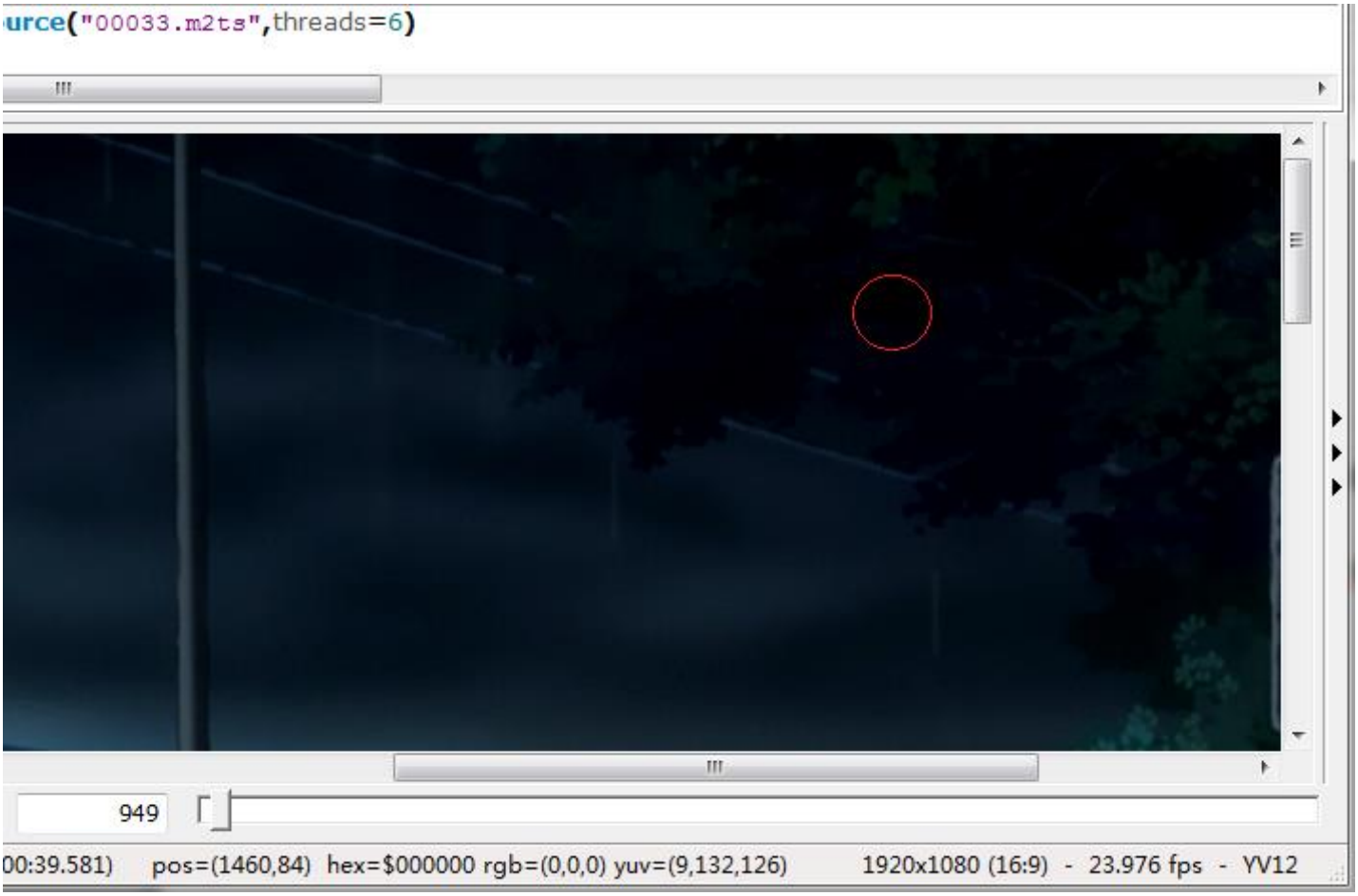
一般来说，按照工业规范制作的视频，其 YUV 格式都满足 tv_range 的标准，即 Y 的范围为[16,235], UV 的范围为[16,240]。把一个 tv_range 的视频变为 pc_range，这个过程叫做 YC 伸张，反之叫做 YC 压缩。

如果一个视频的 range 不正常，比如说，本该是 pc_range 的标注成 tv_range，导致的后果就是视频对比度很大，很容易出现过亮、过暗等场景。仔细观看正片 00033.m2ts 不难发现：



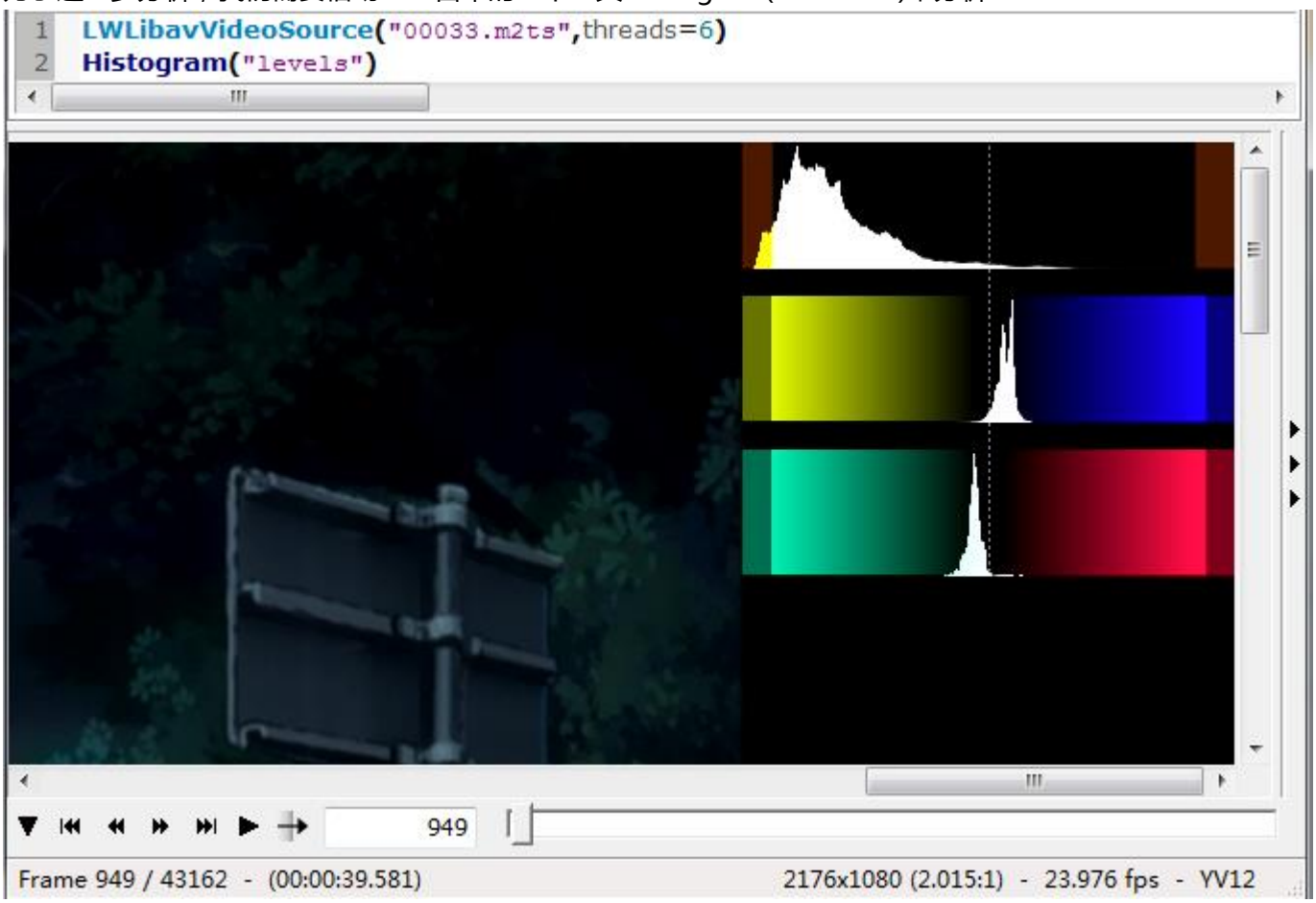

天空的亮度亮的刺眼，观察其 YUV 数值，Y 值早已超过 tv_range 下的最大值 235，已经是接近 pcrange 下(255,128,128)的标准白。所以按照 tv_range 转换后，自然出现了大片的纯白色。
如果看 mediainfo，会发现 m2ts 标注的信息还是 BT709 tv_range。而实际数值早已违背 tv_range 的范围。也就是说，原盘的色彩信息一定有问题。

同理，不难找到极黑的部分：



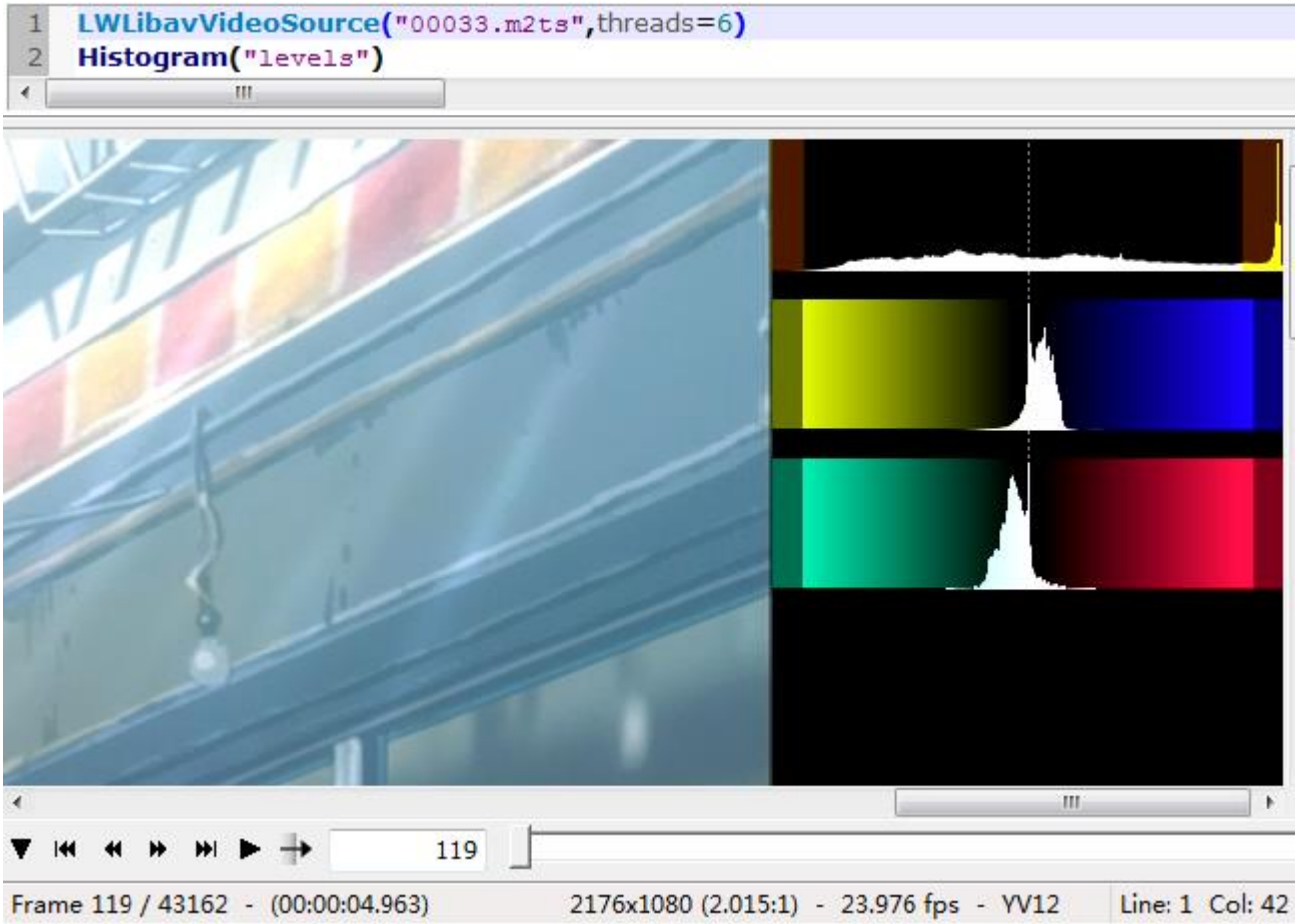

可以看到叶子部分，Y 的数值只有 9，低于 16 的下限。整个暗场区域的细节基本完全不可见。以上数值表明，正片原盘的 YUV 信息一定不是按照 tv_range 的规范来的，而是错误的。

为了进一步分析，我们需要借助 avs 自带的一个工具 Histogram（“levels”)来分析：



Histogram(“levels”)可以让你查看这一帧里面，YUV 数值的分布情况。在画面最右边会有三个图，分别表示 Y，Cb 和 Cr 的分布情况。每个图表左右各有一块扩展区域，不带拓展区域表示 tvrange 的范围，而带了扩展区域则是表示 pcrange 的全范围(0~255)。由上图可见，相当一部分像素，Y 的范围低于 tvrange 的最低值 16。

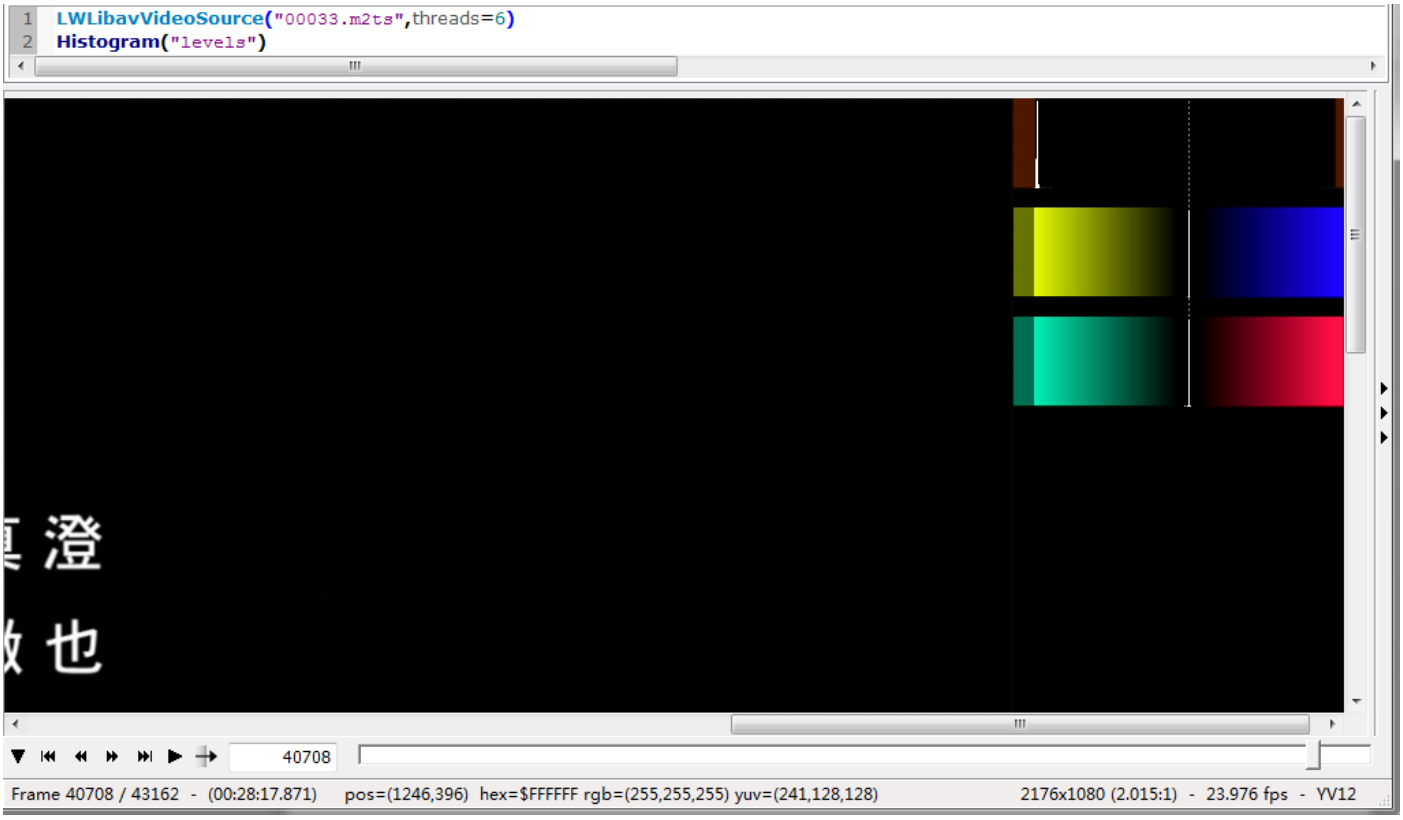
同理，我们来看一下过亮的那张图:



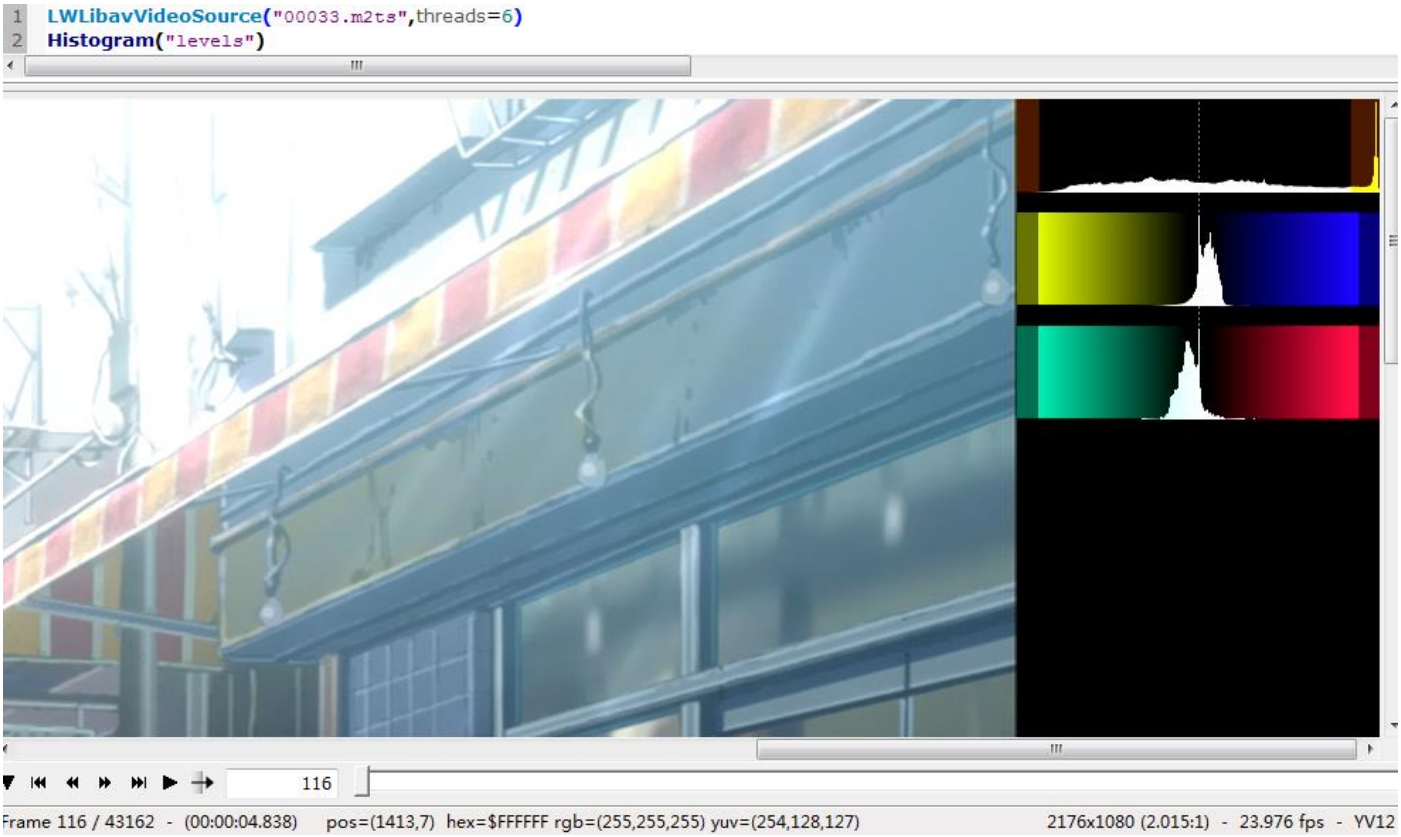

图表显示大量的像素，Y 值极高，超过了 235 的上限。

然而，观察 UV 平面，会发现 UV 的数值相对保守（一般视频中 UV 本身出现极大极小值的概率就比较低），甚至有 UV 相对于 tvrange 再次做了 YC 压缩的嫌疑。这里不同人的观点可能不同，这里我的判断是：Y 平面是 pcrange，误标作 tvrange，需要做 YC 压缩；而 UV 平面可以保持不变（也可以做 YC 伸张）

然而，观察 OP/ED，不难发现情况略不同，OP/ED 当中都有出现黑色，而原盘中的黑色是 tvrange 下的标准黑(16,128,128)。下图是 ED 的某个截取，ED 中黑色背景是标准的(16,128,128),而白色文字的 Y 有溢出，但是溢出量不大。图中是 241，最高大概为 245：




而 OP 也有类似的现象。Y 最低保持 16，最高则可上溢出到 254，非常接近 255：




综上，我们把正片分成 OP(0~749 帧)，Movie(750~40063)，ED(40064~0)三个部分：

OP: 将 Y 的范围由(16,255)缩小到(16,235)

Movie: 将 Y 做 YC 压缩，由(0,255)缩小到(16,235)

ED: 将 Y 的范围由(16,245)缩小到(16,235)

# 3. 正片的修正方法：用 Bitdepth 做 YC 伸张/缩减

Bitdepth 是 O16mod 中的一个函数（O16mod 中还有其他常用的函数，比如 U16(), Down10()等），作用是任意 bitdepth/range 的输入输出。更多信息可以在 O16mod.avsi 里面查看，这里我们介绍它的最基础用法：

Bitdepth(input_depth, output_depth, input_range, output_range, ..., stack)

分别是输入输出的 bitdepth，range。stack 是一个 true/false 变量，规定是否使用 stacked 输出。（注意，输入的高精度只能是 stacked 格式）。
所以正片部分的修正，我们可以这么做：

```
a="00033.m2ts"
LWLibavVideoSource(a,threads=1,format="yuv420p16",stacked=true)
Y       = last.Bitdepth(16,16,false,true,stack=true)
src16   = YtoUV(last.UtoY(), last.VtoY(), Y)
src8    = src16.ditherpost(mode=7)
```

逻辑为：
将片源读入为 stacked 16bit
Y 用于计算调整后的 Luma ,用 Bitdepth ,16bit 输入输出 ,但是输入的 range 假定为 pc ,输出则调整为 tvrange。这样等于是 pcrange 输入，调整为 tvrange 输出，相当于做了一次 YC 压缩。
由于 O16mod 中的函数不能设置单独处理哪个平面，所以处理完了，需要把源的 UV 平面放回去。所以 src16 为调整后的 Y 和调整前的 UV。src8 则是 src16 做了 16->8 的处理。

由于涉及到平面操作计算，而且源虽然读入为 16bit，有效精度只有 8bit。处理后最好再加一个 deband（可以用 f3kdb 或者 gradfun3）。处理完毕后的图像对比如下（左为处理前，右为处理后）：

可见，本来暗场有很多细节，但是因为太暗，或者 Luma 低于 16，显示不出。处理后，暗场细节能正常显示了。

亮场的效果也是相似的（上图为修正前，下图为修正后）：

可以看到，天空，路面和墙壁的亮场细节得以更明显的显示。

vs 中没有对应的滤镜，但是我们可以自己写一个：

fix16 = core.std.Expr(src16, ["x 65535 / 56064 * 4096 +"，""]

就是对 luma 做 pcrange->tvrange

# 4. OP/ED 的修正方法：用 Dither_lut16 做 16bit 的 YUV 运算

以 OP 为例，要把 Y 的范围从[16,255]缩小到[16,235]，我们不能借助 YC 压缩。我们只能自己设计这么个数学公式：

$$(x-16)\times\frac{235-16}{255-16}+16$$

这样是等于把 Y 线性的缩小到所要的范围。

设计好了运算式之后，我们一般要把它 16bit 化（以求在高精度下运算），言下之意，如果 x 是一个 YUV16bit 下的数值，这个运算该如何改变呢？

按照 tvrange 的 16bit 化，一切都很简单：* 256 就行了。16bit 下，Y 的范围为[16*256, 235*256]。顺道一提，10bit 是*4。这只是 tvrange；如果是 pcrange 下，则要复杂很多。
所以将原来公式 16bit 化的结果为：

$$(x-16\times256)\times\frac{(235-16)\times256}{(255-16)\times256}+16\times256=(x-4096)\times219/239+4096$$

所以我们只需要用 Dither_lut16 对 Y 做这个操作：

```
LWLibavVideoSource("00033.m2ts",threads=1,format="yuv420p16",stacked=true)
trim(0,749)
Dither_lut16("x 4096 - 239 / 219 * 4096 +",u=2,v=2)
```

vs 里类似，用 Expr/Lut 都可以，这里不再赘述。

这样，既保证了原盘本身正确的黑位不变，又把白位从 255 降低到 235。

ED 的处理方法类似。只不过折算比例从 219/239 变为 219/229。

## 5. 其他一些特典的判断

00038 和 39，如果载入 avs，会发现是完全正常的 tvrange。不用管。
00040 则是完全的 pcrange，并且在里面可以找到类似(1,128,128)和(254,128,128)的极度近似标准黑白。所以如正片做 YC 压缩。

而 00001~00031 这些 menu 和 Gallery(BD 里自带的图册)，则出现颜色和对比度十分不饱和的现象。这里我们可以选择做一个 YC 伸张。伸张后的结果依旧是 tvrange 范围内，效果却好了不少（第一张为处理前，第二张为处理后）：


# 6. 纯静态视频的无损压制

00005.m2ts，是画册的背景。它表现为纯静态的视频，配上 BGM。我们可以选择截图抽音轨，但是既然它是视频，我们也应该以视频的形式给出。首先，我们先对这个视频做一个 YC 伸张，并输出 Interleaved 10bit:

```
LWLibavVideoSource("00005.m2ts",threads=1)
Bitdepth(8,10,true,false,HQ=true,stack=false)
```

然后，因为源本身每一帧之间有微妙的不同，我们需要让源静止下来。仔细分析 BDMV 最可能的编码模式：

1. 母带是一张静止的图；
2. m2ts 里，第一个 I 帧与母带有一定的劣化偏差；
3. 后续 P 帧和 B 帧不断地去为残差做编码，试图修补这个偏差。

所以，选第一帧的 I 帧不是个好选择；选后面的帧可能会画质更好。一般而言，我们选最后一帧。我们可以用 FreezeFrame() 这个指令：

```
FreezeFrame(起始帧，终止帧，选择帧)
```

就是把视频中[起始帧，终止帧]的区间，用视频中 选择帧 那一帧代替。
所以，要把视频开始到结尾的帧，固定为最后一帧：

```
FreezeFrame(0,last.frameCount()-1,last.frameCount()-1)
```

所以最终的 avs:

```
LWLibavVideoSource("00005.m2ts",threads=1)
Bitdepth(8,10,true,false,HQ=true,stack=false)
FreezeFrame(0,last.frameCount()-1,last.frameCount()-1)
```

压制的时候，用 x264-10bit-full:

```
--qp 0 --keyint 1000 --min-keyint 1 --preset ultrafast --no-chroma-me --merange 4 --partitions all
--colormatrix bt709 --input-depth 10
```

qp0 是使用无损；
--keyint 和 --min-keyint 是定义 IDR 区间的。通常情况下，静止视频解码压力极低，可以给高
--preset ultrafast --no-chroma-me 是尽量降低时域压制的运算，因为是纯静止画面，没有必要用时域运算，包括 B 帧什么的也没必要。视频保持 IPPPPPP....就可以了。
--merange 4 同上。最低只能给到 4 了，不然其实很想给 0
--partitions all 则是开启所有 Intra 编码的选项。尽管我们将时域编码削减到最低，空域编码（一帧以内编码）的还是不能削减。

编码完成后，就可以抽取 flac 封 mkv 了。视频部分仅有 3.25MB，大致相当于两张 png 的大小（因为按照我们设置，有两个 IDR 帧）

# 7. 用 avspmod 截取图像

00007.m2ts 开始，都是相册的照片。我们的对策是截取 png 图片：

```
LWLibavVideoSource("D:\ORGEL\BDMV\STREAM\00007.m2ts",threads=1)
Bitdepth(8,16,true,false,HQ=true,stack=true)
nnedi3_resize16(output="RGB24",lsb_in=true)
```

有的时候，当 m2ts 只有一帧的时候，avs 会在识别帧率上出问题，这时候可以手动加一个 assumefps(1):

```
LWLibavVideoSource("D:\ORGEL\BDMV\STREAM\00007.m2ts",threads=1)
assumefps(1)
Bitdepth(8,16,true,false,HQ=true,stack=true)
nnedi3_resize16(output="RGB24",lsb_in=true)
```

随后就可以用 avspmod 的 video - save image as 保存截图了。

如果图像没有暗场，可以在 nnedi3_resize16 中加一个 dither=-1，让最终 RGB 不使用抖动，这样可以有效缩小截图的体积。

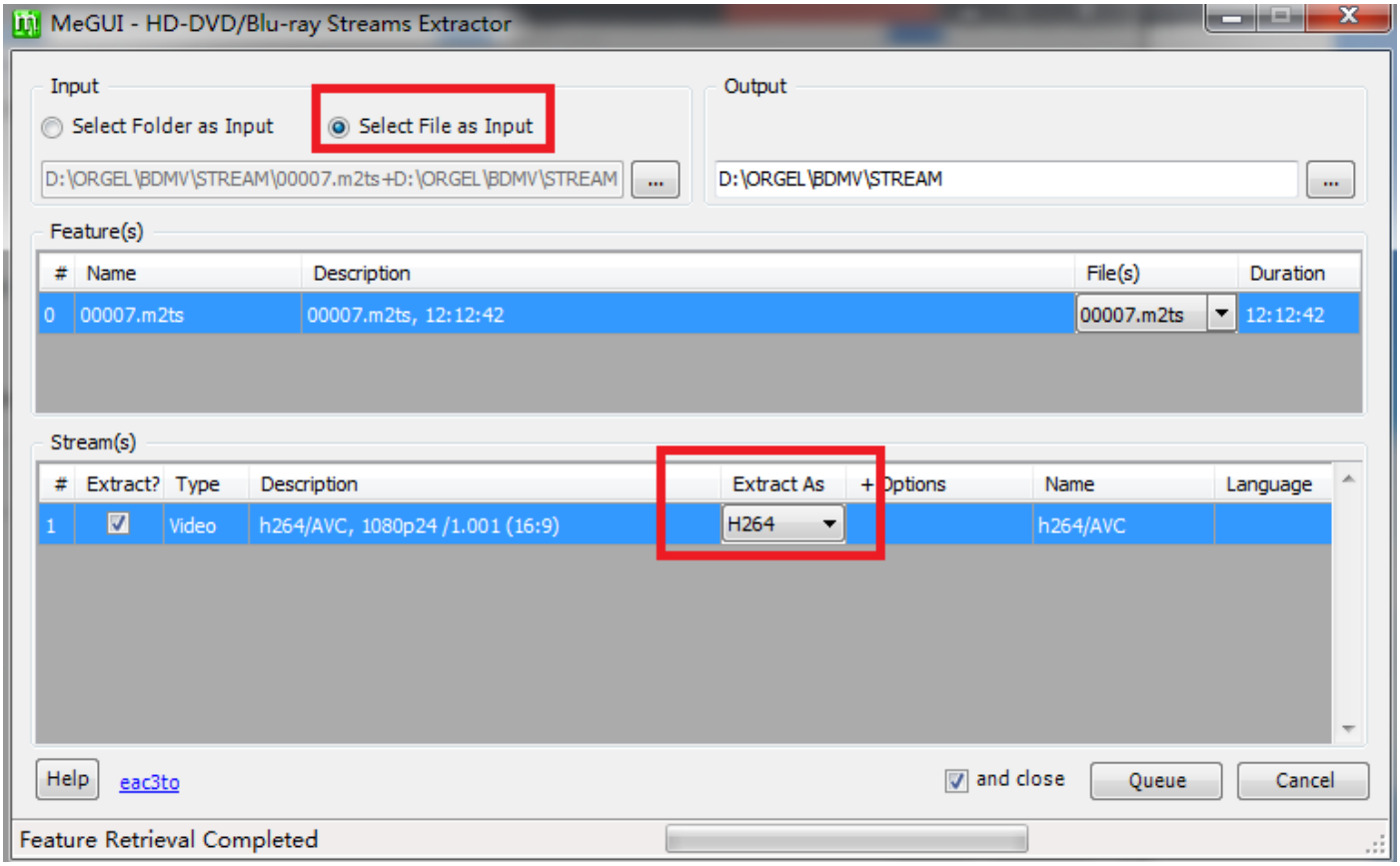
相反，如果有暗场，一般需要做个去色带/色块：

```
LWLibavVideoSource("D:\ORGEL\BDMV\STREAM\00013.m2ts",threads=1)
Bitdepth(8,16,true,false,HQ=true,stack=true)
src16 = last
nr16 = src16.Dither_removeGrain16(20,11)
noise16 = Dither_sub16(src16, nr16, dif=true)
nr16
f3kdb(8, 48, 48, 48, 0, 0, input_mode=1, output_mode=1)
f3kdb(16, 32, 32, 32, 0, 0, input_mode=1, output_mode=1)
Dither_limit_dif16(nr16, thr=0.40, elast=2.0, y=3, u=3, v=3)
dither_add16(Last,noise16,dif=true)
nnedi3_resize16(output="RGB24",lsb_in=true)
```

问题是，这么多图，一张张截取未免太累了。有没有一个批量的方法呢？有：

1. 用 megui HD Stream Extractor 将多个 m2ts 合并并抽取 h.264 视频
2. 用 avs 载入视频转换为一帧一张图的 RGB 视频
3. 用 avspmod 的 Macros-save image sequence 来保存 png

1 的话我们在教程 11 中讲过，这里抽取的时候最好选 H.264（如果选 mkv，需要你的 mkv/m2ts 的分离器设置为 haali）：



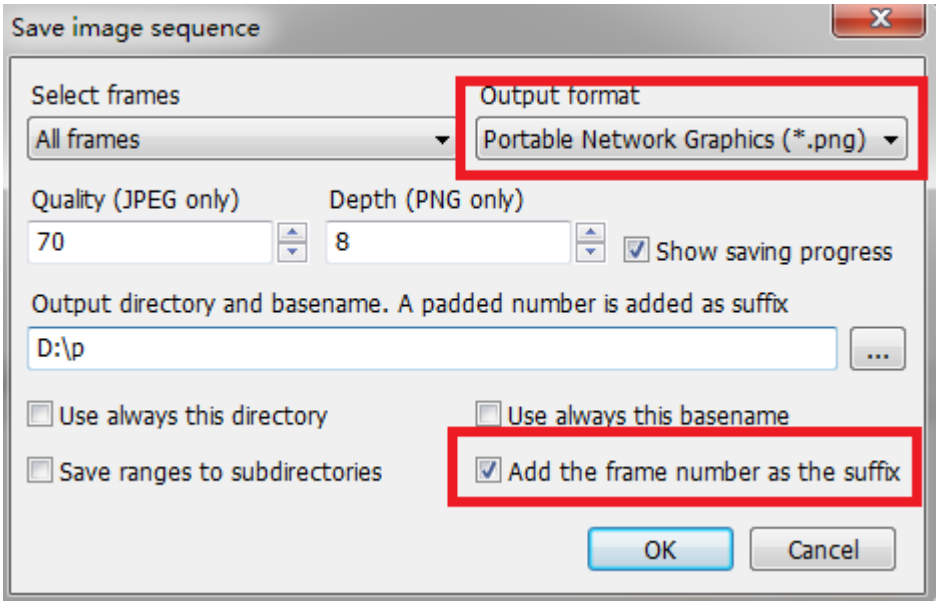

抽取后会在目录下有个 T1_Video - .h264 文件。就是这些 m2ts 的视频轨道。

下面，我们用 avs 读入它。可以发现视频是 24 帧一循环，每个循环一张图。但是最后一张图只有 23 帧（原因未知，你可以当做一个 bug）。我们可以用“每 24 帧里，抽取第 23 帧的方法“来获取一帧一张图的效果：

```
LWLibavVideoSource("D:\ORGEL\BDMV\STREAM\T1_Video - .h264",threads=1)
Bitdepth(8,16,true,false,HQ=true,stack=true)
nnedi3_resize16(output="RGB24",lsb_in=true)
SelectEvery(24,22)
```

这里有两点要注意的，1 是 LWLVS 里加载的必须是绝对路径（不然后续 Macros 会出错的）；2 是 SelectEvery(x,y) 表示每 x 帧里，抽取从 0 开始数第 y 帧。上述“第 23 帧”，我们是从 1 开始，如果从 0 开始标号，自然需要 23-1=22。所以抽取每个循环开头和最末一帧的写法是 SelectEvery(24,0)和 SelectEvery(24,23)。用 SelectEvery(24,23)会导致最后一张图无法获取，因为最后一张图仅有 23 帧，因此每 24 帧抽取最末一帧的做法，会导致最后一张图无法被抽取。

最后，我们就可以用 avspmod 自带的 Macros - save image sequence 来保存截图：




选择输出 D:\p， 输出的图像会在 D 盘根目录下，以 p000.png, p001.png, ...存在。下标号从 000 开始，其实跟图片上标号从 1 开始并不搭配。如果有心想调整，可以在 avs 里再加一帧：

```
LWLibavVideoSource("D:\ORGEL\BDMV\STREAM\T1_Video - .h264",threads=1)
Bitdepth(8,16,true,false,HQ=true,stack=true)
src16 = last
nr16 = src16.Dither_removeGrain16(20,11)
noise16 = Dither_sub16(src16, nr16, dif=true)
nr16
f3kdb(8, 48, 48, 48, 0, 0, input_mode=1, output_mode=1)
f3kdb(16, 32, 32, 32, 0, 0, input_mode=1, output_mode=1)
Dither_limit_dif16(nr16, thr=0.40, elast=2.0, y=3, u=3, v=3)
dither_add16(Last,noise16,dif=true)
nnedi3_resize16(output="RGB24",lsb_in=true)
SelectEvery(24,22)
trim(0,-1)+last
```

这样第 0 帧相当于是第 1 张图的重复。如此输出图像序列后，p000.png 是第一张图，p001 是第一张，p002 是第二张……最后只需要删掉 p000.png 就可以了。整理完毕的图像如下：