VCB-Studio 教程 23 30fps 片源处理(2) Deint 与 IVTC 详解

本教程是教程 22 的后续，旨在详细的解释 Interlaced/Teleclined 视频的原理，以及该如何去处理。

1. Interlaced(交错)视频的原理

一个 interlaced 30i 1920x1080 的视频，你可以这么理解：
（下文中，帧从 0 编号，然后图像像素的行列从 1 编号）
. 源是 1920x1080 60fps 的视频；
. 将每一帧高度减半，偶数帧(0,2,4..)保留奇数行（1,3,5…1079）；奇数帧(1,3,5…)保留偶数行，变成 1920x540 60fps
. 将前后两帧合并为一帧，两个 1920x540 的图像，交织的取每一行构成 1920x1080：

白色的来自前一帧，蓝色的来自后一帧

所以大家应该不难理解为啥看 interlaced 视频会有拉丝现象：每一帧其实是由两张图像组合而来的。事实上，对于 interlaced 的视频，帧的概念并不适用，而是更多用场的概念。一个 1920x1080 30i 的视频，每一秒有近似 30x2=60 场：
每一帧 1,3,5…1079 行是上半场（Top Field）；
每一帧 2,4,6…1080 行是下半场（Bottom Field）。
所以每一帧都有两个有序半场。

avs 里面，读入一个 Interlaced 视频之后，你可以用以下的指令进行分场操作：
SeparateFields()
vs 里对应的指令是 core.std.SeparateFields(src, tff=True)

然后当你观测的视频的时候，你应该可以看到视频变成了分辨率压扁，帧率翻倍的效果。这时候你观察画面，应该是可以发现除了扁平，它跟正常的视频没什么区别。

但是如果仔细观看，还会发现前后两场，画面的中心是不重叠的。因为一张图像取奇数行和偶数行分别构成新图像 A 和 B，在原图中，A 的每一行都在 B 上面一个像素。如果需要让 A 和 B 的中心重叠，B 需要向上平移一个像素。分场之后，高度减半，新图像中一行像素相当于原图两行，所以在 1920x540 的框架下，A 比 B 高了 0.5 个像素

（有时候你会发现前后两帧似乎顺序反了，那是因为 avs 弄错了上下半场的顺序。这时候只要在 SeparateFields()前面加上 ComplementParity()。这个滤镜的作用是调换上下半场的顺序）

Interlaced 视频在编码储存的时候，其实是按场来的。只有按场来，才能保证 interlaced 视频可以合理的编码。

如果已经被分场的视频，想合并成正常的视频：
Weave()
VS 里对应的是 core.std.DoubleWeave(res, tff=True).SelectEvery(2,0)
VS 里的 DoubleWeave() ,合并后帧率不变 ,每两帧重复一帧。所以需要用 SelectEvery(2,0)来排除重复帧。

2. Deinterlace(反交错)的基本原理-BOB

既然已经知道了，Interlaced 视频每一场是一半的图像，那么一种简单的反交错思路就是，把每一场拉回两倍的宽度。这就是最基础的 deinterlace 思路——BOB

一个简单的 bob avs:

LWLibavVideoSource(“00000.m2ts”)

SeparateFields() #以上是读入并分场

A=SelectEvery(2,0) #SelectEvery(x,y)表示每 x 帧中选取第 y 帧。A 记录的就是分场后的偶数场，也就是 Top Fields

B=SelectEvery(2,1) #B 记录的就是分场后的奇数场，也就是 Bottom Fields

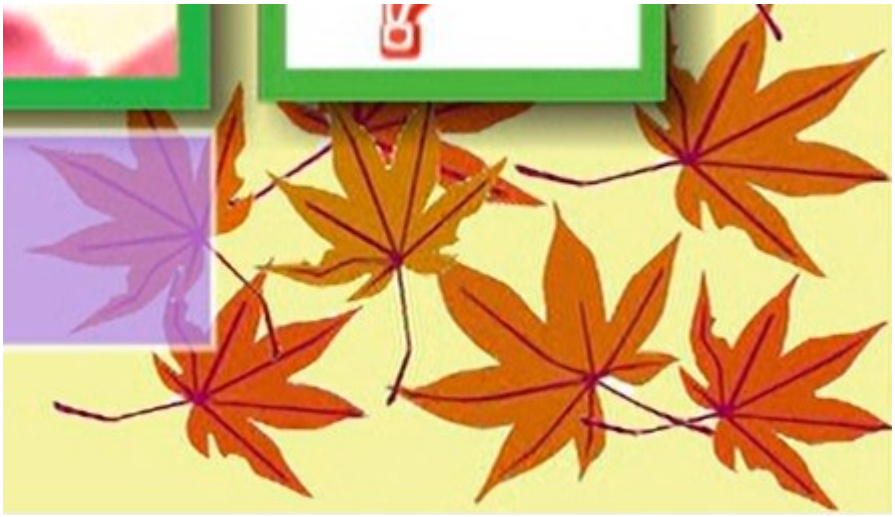
A=A.Spline64Resize(1920,1080,src_top=-0.25) #将 A 向下平移 0.25 像素后，拉到 1920x1080

B= B.Spline64Resize(1920,1080,src_top=0.25) #将 B 向下平移 0.25 像素后，拉到 1920x1080

Interleave(A,B)#将 A，B 每一帧交织的显示。

常规的 resizer(比如上文我们选择的 spline64resize)做放大，特别是针对场这种画面，容易出现大量的锯齿和 ringing：

图像处理环节，有很多算法，可以较好地，将一幅图拉升到两倍的高度。avs 里面也有这样的滤镜。这类专门的滤镜，效果就好很多（下图使用了 nnedi3）：



脚本：

```
LWLibavVideoSource("00006.m2ts",threads=1)
SeparateFields()
A=SelectEvery(2,0).nnedi3(field=1,dh=true)
B=SelectEvery(2,1).nnedi3(field=0,dh=true)
Interleave(A,B)
```

eedi2/nnedi3 等 ,都是常见的拉升插值滤镜。它们最基础的作用是给定上半场或者下半场 ,根据现有图像，插补出另一个半场。与常规 Resizer 相比，它们的算法更为复杂，效果也更好，首先对于锯齿有很好的消除效果，其次放大后的图片极少有 ringing。所以这些滤镜还在抗锯齿/放大拉升方面有很多作用，这个我们以后再说。

但是光这样纯 saptial 的拉升，还不是最好的效果。实际操作中，你会发现大量静态细节（比如字幕）出现抖动和闪烁。这是因为用上半场和下半场拉升之后的图像，没办法完全做到次像素级别的匹配。这时候就需要在时域上稳定处理，而时域处理又要涉及到动态分析。

有没有滤镜能帮我们办以上所有事情？有。最常用的 deinterlace 滤镜，QTGMC，原理就是上文所述。

3. QTGMC 滤镜的使用

QTGMC 的作用，是将一个 Interlaced 视频，做反交错后输出。它可以直接使用：

LWLVS(“”) #输入一个 30i 的视频，格式是 yuv420p8
QTGMC() #输出一个 60p 的视频，格式依旧是 yuv420p8

QTGMC()里面，有这样几个参数可以详细设置：
Preset，预设。以速度换质量的选择，一般可选的范围：
“Faster”、“Fast”、“Medium”、“Slow”、“Slower”、“Very Slow”
越慢，处理效果越好。一般选择 Medium 到 Slower 就好了。

Border，是否处理的时候加黑边。一般如果你视频没有黑边都选择 Border=true，这样对边缘的处理会更好

FPSDivisor，如果这个设置为 2，意味着输出的帧率从 60p 变为 30p。默认下就是砍掉一半的帧；因为砍掉的都是有效信息，所以不推荐开启。

ShutterBlur，如果你必须保持 30p 输出，你可以用 ShutterBlur=2 来搭配 FPSDivisor=2 来实现动态融合两帧。这样输出效果会更好。

所以一般 60p 输出：QTGMC(preset=“Slow”，Border=true)
30p 输出 QTGMC(preset=“Slow”，Border=true，FPSDivisor=2，ShuttleBlur=2)

QTGMC 其他参数用法参见 http://avisynth.nl/index.php/QTGMC
QTGMC 是一个速度黑洞。使用的时候最好单独放一个 MPP 的 Block，并且设置好 prefetch。

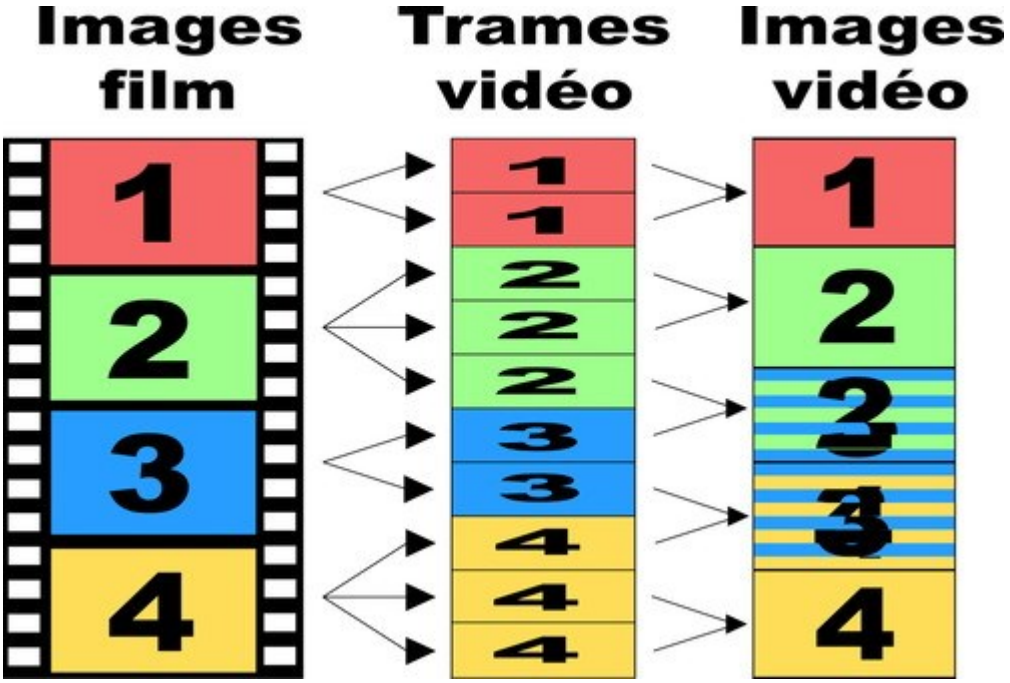
VS 里用法类似，用 haf.QTGMC()。不过必须指定 TFF=True/False.

4. Telelcine 视频的原理

欧美和日本的交流电是 60Hz，意味着电视的刷新率也是 60Hz。30i 的内容可以很好地显示（每一秒正好 60 场），但是 24p 的内容就不是那么简单了。24p 的内容，得先“转换”成 30i，才能正常播放。 Telecline 是系统性的将 24p 的内容转为 30i 的方法。常见的策略有：

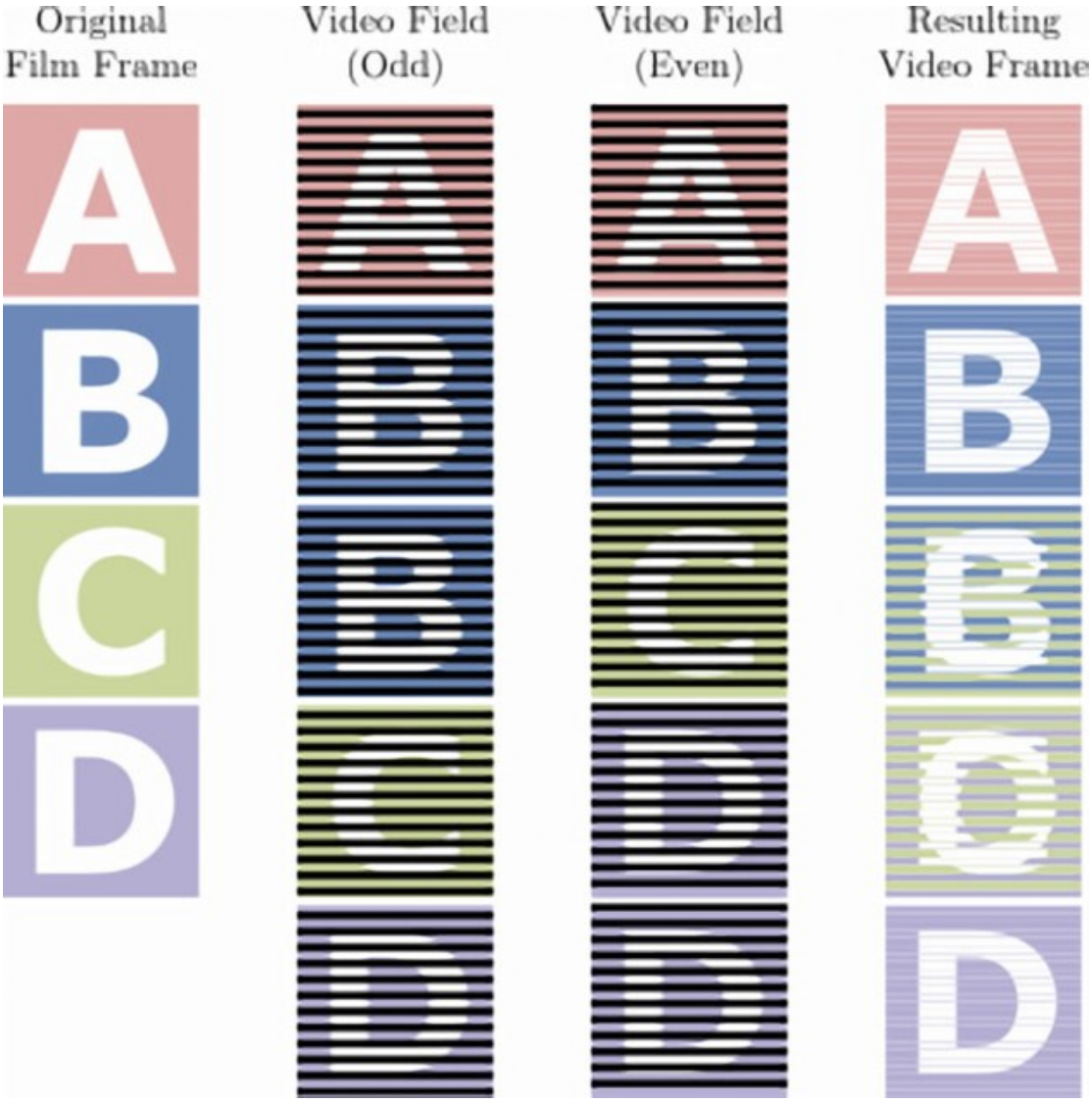
1. 加重复帧。每 4 帧周期性加一个重复帧，这样每秒钟 24 帧就变成 30 帧，然后将 30 帧看做 60 场。这样做很简单，缺点也比较明显，就是加入重复帧的地方，在播放的时候明显会动态卡顿，特别是在慢镜头下。

2. Pulldown。它是在每 8 个场中，周期性的加入两个重复场：



将每一帧分离成上下两个半场，这样 4 帧就有 8 个场；
第三帧的两个半场前，插入第二帧的重复场；
第四帧的两个半场后，插入第四帧的重复长；
这样 4 帧就有了 10 个场，帧率从 24 变为 30。

这个过程叫做 3:2 pulldown。3:2，是画面显示时间长度的比例。如图所示，在正常播放的时候，2 和 4 两帧实际占用了 1.5 帧的时间，实际播放中，相对 24d，这样有助于减少动态顿卡。
为了更好的描述的这个过程，再来看一张图：



如果只是简单分场，4 帧应该分为 8 个场：

A 上，A 下，B 上，B 下，C 上，C 下，D 上，D 下

经过 3:2 pulldown 后，4 帧分为 10 个场：

A 上，A 下，B 上，B 下，B 上，C 下，C 上，D 下，D 上，D 下。

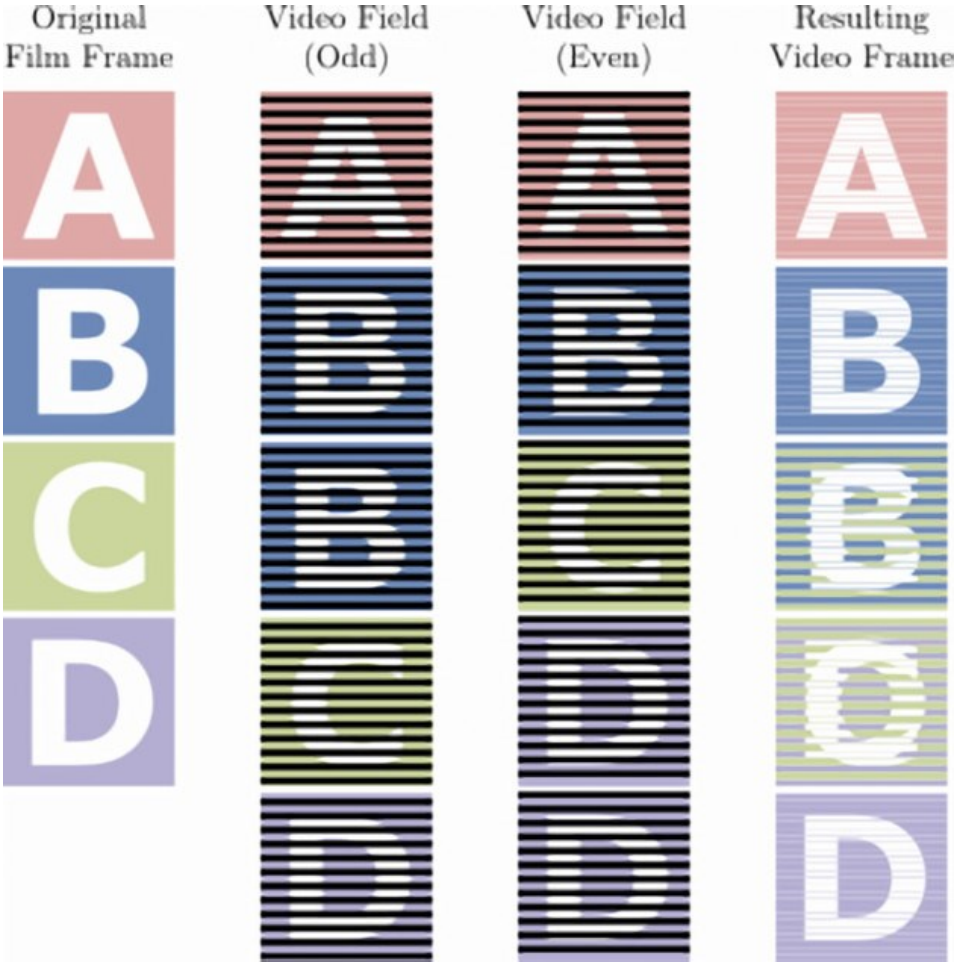
经过 pulldown 的视频，在支持 Interlaced 的显示器上播放是没问题的，但是在 pc 上显示就会有拉丝现象，表现为周期性的 5 帧烂 2 帧。这种视频就是 24t。

注意，有时候视频的 pulldown 不一定是规则的，可能出现各种花式重复场的顺序，造成各种 5 烂 1，5 烂 3 等等，且不规律不周期。但是这些视频还是属于 telecline 的范畴，解决方案也类似。

5. IVTC(Inverse-Telecine)的原理——场匹配+去重复帧。

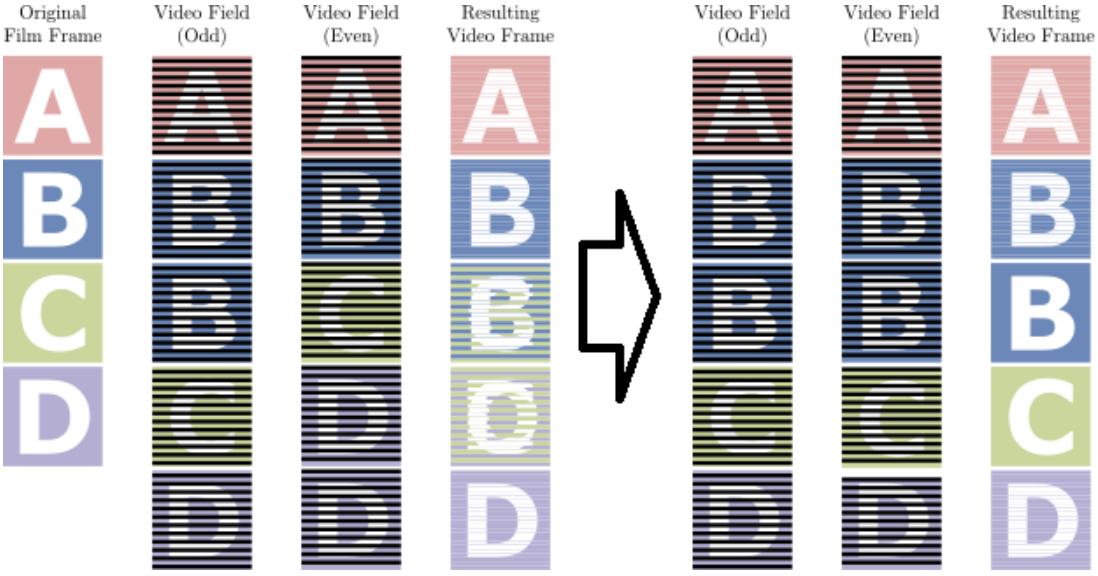
ivtc 的第一个步骤，叫做场匹配。它的目的是，通过各种换位和复制每一帧的下半场，保证每一帧的下半场，一定是跟上半场匹配的。

比如我们回来看这幅图：



成品中，三四两帧有拉丝，原因是它们的下半场跟上半场是不匹配的。

场匹配算法研究每一个出现拉丝帧的上半场，在前后的场序列中，寻找和它匹配的下半场：



经过场匹配后，出现拉丝的两帧，下半场和上半场就可以完美匹配成单一的图像。输出的图像从有拉丝的 24t，变成了没有拉丝，只有重复帧的 24d。

得到 24d 的画面后，下面只要周期性的，在 5 帧中找出重复的两帧，删掉其中一帧就好。这个步骤叫做去重复帧，是 ivtc 的第二个步骤。

6. IVTC 的 avs 实现

场匹配依赖的滤镜，叫做 tfm(http://avisynth.org.ru/docs/english/externalfilters/tivtc_tfm.htm)：

使用方式也很简单：

```
LWLibavVideoSource(...)
tfm()
```

就可以把 24t 的视频转为 24d。

有时候，视频的交错比较奇葩，简单一个 tfm()不见得能完全除掉交错，这时候就可以加强 tfm 的参数：

```
tfm(mode=3, pp=7)
```

这是加强 tfm 寻找场匹配的算法和搜索量。如果场匹配失败了，通常情况下意味着是原生 interlaced 部分，tfm 会试着做一个反交错。tfm 的反交错算法比较快，但是效果不如 qtgmc。

如果你不想让 tfm 做反交错，而是输出不能成功匹配的帧，这样方便调试，你可以用 pp=1。pp 这个参数就是决定对于不能匹配的帧，用什么方法处理。1 是不处理，6/7 是做反交错。

tfm 过后，视频变为逐行交错，但是有重复帧。这时候删除周期性的重复帧，需要用 tdecimate(http://avisynth.org.ru/docs/english/externalfilters/tivtc_tdecimate.htm):

```
LWLibavVideoSource(...)
tfm()
tdecimate(mode=1,dupThresh=3,vidThresh=3)
```

tdecimate 会在 5 帧中，找出最长连续的重复帧序列，删去一帧。如果找不到，则删掉一帧。

如果源非常规则，也可以直接 tdecimate(mode=0)。hybrid 这个参数决定是直接删帧(hybrid=0)还是 blending 融合帧(hybrid=1)。一般来说选择直接删帧，因为周期性出现的 blending 是很影响观感的。

注意 tdecimate 依赖 tfm 的信息。所以哪怕片源是 24d，依旧需要 tfm 在前面。

ivtc 后的视频变为 24p，且不存在周期性的重复帧。

VS 里对应的是 VFM()和 VDecimate(): http://www.vapoursynth.com/doc/plugins/vivtc.html

它们的功能比起 avs 简化很多，主要在于 VFM()没有做 deint 这样的后处理，而 VDecimate 相当于只有 mode=0.

因此，IVTC 后通常接一个 daa()这样的操作，来保证即便场匹配失败的帧，也能通过 blending 的方式去掉拉丝：

```
src8 = core.lsmas.LWLibavSource(a, repeat=True)
```

```
src8 = core.vivtc.VFM(src8, order=1, mode=3)
src8 = core.vivtc.VDecimate(src8)
src8 = haf.daa(src8)
```

如果想用更靠谱的 deint 方法，可以借助 QTGMC。TFM/VFM 都会在帧里标记，它是否认为这一帧在场匹配后还有拉丝，如果有，我们就使用 QTGMC 做 deint：

```
src = core.lsmas.LWLibavSource(a, repeat=True)
fm = core.vivtc.VFM(src, order=1, mode=3, cthresh=6)
deint = haf.QTGMC(src, Preset="Slow", TFF=True, FPSDivisor=2)
src8 = mvf.FilterCombed(fm, deint)
src8 = core.vivtc.VDecimate(src8)
src8 = core.std.SetFieldBased(src8, 0)
```