x264 视频压缩教程精简版

欢迎阅读本教程！本教程精简了科普部分，若有什么不会的可以直接加群 691892901 哦(`·ω·´)ゞ

食用方法与注意事项

1. 准备一个文本文档，参考本篇，根据你的需求配置写个参数，忘了就打开文档查ヾ(●ω●)ノ
2. 到本文后面的下载栏下载喜欢的压制软件，把参数贴进去保存，有需要就导入视频压缩
3. 学会基本的软件使用后，下载急用版教程，根据情况复制编写出需要的参数＼（@￣▽￣@）／

- 在综合版看完 x264 的工作原理和操作方法，(差不多)看懂就下载急用版，马上开工 ৻(￣ ￣৻)
- 或者准备一个文本文档，参考本篇，根据你的需求配置写几个参数，忘了就打开文档查ヾ(●ω●)ノ
- 考虑到正片(参数讲解)的内容有些冗长，所以将命令行参数写法直接放于下面，不要一眼略过哦

压制软件工作流程图解（不严谨）

命令行参数 command prompt

[打开 Windows 自带的 CMD.exe]

[引用程序]　　C:\文件夹\x264.exe

[CLI 参数]　　--me esa --merange 48 --keyint 100 [...]

[导出，空格，导入]　　--output C:\文件夹\导出.mp4 C:\文件夹\导入.mp4

[完整 CLI 参数格式]　　x264.exe --me esa --merange 24 [...] --output "导出.mp4" "导入.mp4"

命令行参数注意事项

在 CLI 中，空格代表命令间的分隔符. 所以带空格的路径要加引号(ΦωΦ)

压制报错检查

压制软件中，如果视频一下子好了，且导出的视频损坏，就顺着进度窗口往上找，确认第一个出现的"error"后根据报错分析:

- unknown option 代表写错一个参数/多打一个空格，造成相邻的字符被当成了参数值
- x264 [error]: could not open output file 'NUL'代表 null.sys 坏了，先备份再替换一个新的即可. 打开 txt，保存为 bat 双击运行:

```
sc config Null start = system && sc start Null
if %errorlevel% EQU 0 (echo Success!) && pause
```

- 频繁黑屏，蓝屏等情况说明 CPU 高温。换硅脂，清理电脑积灰，增加风扇转速即可解
- clEnqueueNDRangeKernel error '-4'代表显存太小，应关闭 OpenCL

三角形定律(右图)

分辨率换算: 原本高 720，宽高比 4:3，那么宽就是 720÷3×4=960

x264 压制 log 内容

```
[info]: indexing input file [0.7%]                打开视频

ffms [info]:
 Format       : mov,mp4,m4a,3gp,3g2,mj2
 Codec        : h264
 PixFmt       : yuvj420p
 Framerate : 60/1                                 视频流信息
 Timebase     : 1000/15360000
 Duration     : 0:01:30
ffms [info]: 1920x1080p 0:1 @ 60/1 fps (vfr)
ffms [info]: color matrix: bt709
x264 [info]: using cpu capabilities: MMX2 SSE2Fast SSSE3 SSE4.2 AVX
x264 [info]: OpenCL ... GeForce GTX 970          软硬件信息
x264 [info]: AVC Encoder x264 core 148 ...xiaowan [8-bit@all X86_64]
x264 [info]: profile: High ... bit-depth: 8-bit
x264 [info]: opencl=1 cabac=1 ref=3 deblock=1:0:0 ... aq3=0
```

```
x264 [info]: started at Sat Nov 24 02:58:23 2018
[0.0%] 1/5412 frames, 0.238 fps, 17882 kb/s, 36.38 KB, eta 6:19:13, est.size 192.28 MB
.....                                    压制信息
[99.4%] 5378/5412 frames, 44.60 fps, 271.22 kb/s, 2.90 MB, eta 0:00:00, est.size 2.92 MB

x264 [info]: frame I:12      Avg QP:21.49    size: 82888
.....                                    输出视频流信息
x264 [info]: kb/s:269.65                 (此处被大幅省略)
encoded 5412 frames, 44.83 fps, 269.72 kb/s, 2.90 MB
x264 [info]: ended at .....
x264 [info]: encoding duration 0:02:01       其他信息
```

动态搜索

--me<dia, hex, umh, esa>搜索方式， 从左到右依次变得复杂， umh 之前会漏掉信息， 之后收益递减， 所以推荐 umh(˚▽˚)/

--merange<整数>越大越慢的动态搜索范围， 建议 16， 32 或 48. 由于是找当前动态向量附近有没有更优值， 所以太大会让编码器在动态信息跑不到的远处找或找错， 造成减速并降低画质

--no-fast-pskip<开关>关闭跳过编码 p 帧的功能. 建议在日常和高画质编码中使用

--direct auto<开关， 默认 spatio>指定动态搜索判断方式的参数， 除直播外建议 auto

--non-deterministic<开关， 对称多线程>让动态搜索线程得知旧线程实际搜素过的区域， 而非参数设定的区域. 理论上能帮助找到正确的动态矢量以增强画质， 增加占用， "deterministic"是好几门学科的术语， 代表完算性， 即"算完才给出结果的程度"， 反之就是"欠算性"

--no-chroma-me<开关>动态搜索不查色度平面，以节省一点算力加速压制，建议直播/录屏用

运动补偿

动态搜索让块间连起来，运动补偿 motion compensation 用 SAD, SATD 算法找出参考块间子像素最像的源，将动搜所得的块-帧间插值(运动矢量)细化，让块间细节精确地连起来. 跳过=大量细节损失

--subme<x264 中影响模式决策和率失真优化. 整数范围 0~10， 60fps=9， -=8， +=10>根据片源的帧率判断. 分四个范围. 由于动漫片源制于 24~30fps， 因此可节省一些算力；但同是动漫源的 60fps 虚拟主播则异. 主流 120Hz 的手机录屏目前最高也不够用. 由于性能损耗大， 所以不建议一直开满

- <1>逐块 1/4 像素 SAD 一次， <2>逐块 1/4 像素 SATD 两次
- <3>逐宏块 1/2 像素 SATD 一次， 再逐块 1/4 像素 SATD 一次
- <4>逐宏块 1/4 像素 SATD 一次， 再逐块 1/4 像素 SATD 一次
- <5>+增加双向参考 b 块的补偿
- <6>+率失真优化处理 I， P 帧；<7>+率失真优化处理 I， P， B， b 帧
- <8>+I， P 帧启用 rd-refine；<9>I， P， B， b 帧启用 rd-refine

- <10，边际效应=压缩，trellis 2，aq-strength>0>+子像素上跑 me hex 和 SATD 比找参考源
- <11，边际效应>压缩>+关闭所有 10 中的跳过加速，不推荐．原本是为了 trellis 3 设计的，但没了

加权预测 weighted prediction

avc 首发，治理了少数淡入淡出过程中部分 pu 误参考，亮度变化不同步的瑕疵．分为 P，B 条带用的显加权 explict WP<编码器直接从原画和编码过的参考帧做差>与 B 条带用的隐加权 implict WP<用参考帧的距离做加权平均插值>

--weightb<开关，默认关/只加 P 条带>启用 B 条带的显隐加权预测．条带所在 SPS 中可见 P，B 加权开关状态，及显加权模式下解码器须知的权重．光线变化和淡入淡出在低成本/旧动漫中少见

关键帧

参考帧

--keyint<整数>一般设 9×帧率(9 秒一个落点)，拖动进度条越频繁，就越应该降低(如 5×帧率)

- 短视频，不拖进度条可以设 keyint -1 稍微降低文件体积，剪辑素材就乖乖设 5 秒一个吧(⦿o⦿)

--min-keyint<整数，默认 25>当检测到转场，就看转场是否小于 min-keyint，若小于 min-keyint 就不浪费算力而插入 I 帧．在这上面，x264 给了我们两种选择，而它们给出的画质都一样：

- 设 5 或更高，省了设立一些 IDR 帧拖慢速度．快速编码/直播环境直接设=keyint へ(>_<へ)
- 设 1 来增加 IDR 帧，一帧被判做转场本来就意味着前后溯块的价值不高．而 P/B 帧内可以放置 I 宏块，x264 会倾向插 P/B 帧．好处是进度条落点在激烈的动作场面更密集，画质编码用

--bframes<整数，0~16>连续最多的 B 帧数量．一般设 14，手机压片建议设 5 省电

--b-adapt 2<所有情况，整数 0~2>0 代表不设 b 帧，1 代表快速，所有情况建议 2 精确[¬º-°]¬

嫁接帧

GOP 结构建立，参数集

给视频帧分段并最终整合成 gop 内树叉状的参考结构后，将其中的关键帧递给下一步帧内编码．一来冗余，二来防止参考错误蔓延，照顾丢包人士，三来搭建 NALU 为基础传输 ss 的网络串流架构

--rc-lookahead keyint÷2<整数>查未来开销，判断 VBV，IBP 帧以及 mbtree 策略用的帧数，让重要的场景画质有保障

--lookahead-threads<整数，线程>开 opencl 后可据显卡算力与显存速度，手动将其提高

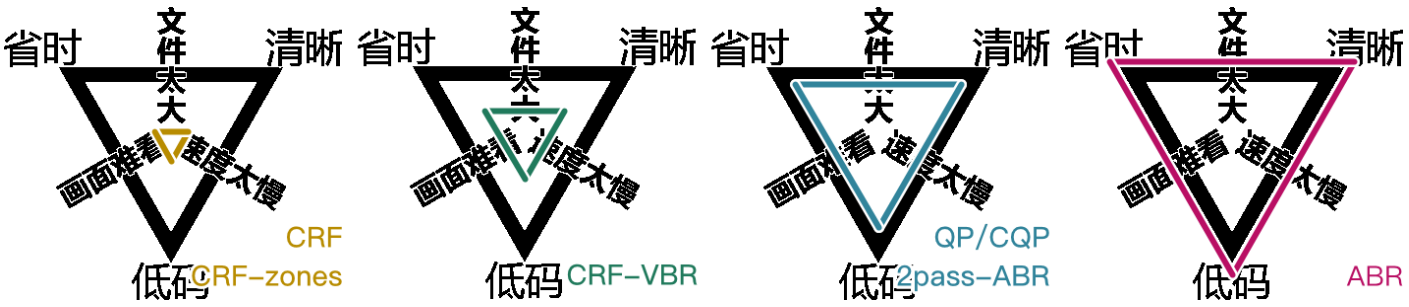
--opencl<协议，默认关>将 cpu 算不过来的 lookahead-threads 分给显卡. 自动创建 clbin 以防每次跑都要编译一次 opencl 内核

变换-量化

变换是将频率从低到高的信息列出来方便量化的计算，量化通过低通滤镜，砍了噪齿纹理边缘和细节等高频信息，降低画质但大幅提升了压缩

码率控制模式

根据对短板容忍度及用途的不同选择适合的方式. 短板来自处理器，及不同模式的完算性上的区别



CRF/ABR/CQP 上层模式

--crf<浮点范围 0~69，默认 23>据"cplxBlur，mbtree，B 帧偏移"给每帧分配各自 qp 的固定目标质量模式，或简称质量呼应码率模式，统称 crf. 素材级画质设在 16~18，收藏~高压画质设在 19~20.5，YouTube 是 23. 由于动画和录像的内容差距，动画比录像要给低点

--bitrate<整数 kbps，指定则关 crf>平均码率 average bitrate. 若视频易压缩且码率给高，就会得到码率更低的片子；反过来低了会强行提高量化强制码率达标. 一般推流用的"码率选项"即该参数，快但妥协了压缩+画质. 因此直播算力够用则 crf~vbr，一般情况默认 crf，码率硬限下用 2pass-abr

--qp<整数，范围 0~69，指定则关 crf>恒定量化. 每±6 可以将输出的文件大小减倍/翻倍. 影响其后的模式决策，综合画质下降或码率暴涨，所以除非 yuv4:4:4 情况下有既定目的，都不建议

--qcomp<浮点范围 0.5~1，一般建议默认>cplxBlur 迭代值每次能迭代范围的曲线缩放. 越小则复杂度迭代越符合实际状况，crf，mbtree，bframes 越有用，搭配高 crf 能使直播环境可防止码率突增. 越大则 crf，mbtree，bframes 越没用，越接近 cqp. 曲线缩放原理见 desmos 互动示例

--no-mbtree<开关>关闭少见宏块量化增强偏移. 可能只有 crf 小于 17 才用的到

常用下层模式

--qpmin<整数， 范围 0~51>最小量化值， 仅在高压环境建议设 14~16(″▽″);**--qpmax**<同上>在要用到颜色键，颜色替换等需要清晰物件边缘的滤镜时，可以设--qpmax 26 防止录屏时物件的边缘被压缩的太厉害，但其它情况永远不如 no-mbtree (*~▽~)

--chroma-qp-offset<整数， 默认 0>h.264 规定 CbCr 的码率之和应=Y 平面， 所以 x264 会拉高 CbCr 的量化. 用 psy-rd 后， x264 会自动给 qp-2 至-4. 不用 psy-rd 时， 4: 2: 0 的视频可手动设-2 至-4

--ipratio<浮点， 默认 1.4>P 帧相比 IDR/i 帧;**--pbratio**<浮点， 默认 1.3>B/b 帧相比 P 帧的偏移. 指定 IDR/I 帧 qp17， P 帧 qp20， B/b 帧 qp22 就填写″--qp 17 --ipratio 1.1765 --pbratio 1.1″

zones 下层模式

--zones<开始帧,结束帧,参数 A=?,参数 B=?•••>手动在视频中划区， 采用不同上层模式来实现如提高压制速度， 节省平均码率， 提高特定画面码率等用途(一般用来″处理″片尾滚动字幕). zones 内的 me，merange 强度/大小不能超 zones 外. 可用参数有 b=，q=，crf=，ref=，scenecut=，deblock=，psy-rd=，deadzone-intra=，deadzone-inter=，direct=，me=，merange=，subme=，trellis=

1. 参数 b=调整码率比率，可以限制 zones 内的场景使用当前 0~99999%的码率，100%相当于不变
2. 参数 q=即 QP 值，可以用来锁死 zones 内场景使用无损压缩(任何 rate factor)以做到素材用编码
3. 多个划区可以用'/'隔开: --zones 0,449,crf=32,me=dia,bframes=10/450,779,b=0.6,crf=8,trellis=1

VBR 下层模式

crf-vbv 及 abr-vbv 两种搭配，用缓冲控制 video buffer verifier 设立编解码硬限(゜▽゜)/

--vbv-bufsize<整数 kbps， 小于 maxrate>编码器解出原画后， 最多可占的缓存每秒. bufsize÷maxrate = 编码与播放时解出每 gop 原画帧数的缓冲用时秒数. 值的大小相对于编完 GOP 平均大小. 编码器用到是因为模式决策要解码出每个压缩步骤中的内容与原画作对比用

--vbv-maxrate<整数 kbps>峰值红线. 用″出缓帧码率-入缓帧码率必须≤maxrate″的要求， 让编码器在 GOP 码率超 bufsize， 即缓存用完时高压出缓帧的参数. 对画质的影响越小越好. 当入缓帧较小时， 出缓帧就算超 maxrate 也会因缓存有空而不被压缩. 所以有四种状态，需经验判断 GOP 大小

- 大: GOPsize=bufsize=2×maxrate，超限后等缓存满再压，避开多数涨落，适合限平均率的串流
- 小: GOPsize=bufsize=1×maxrate，超码率限制后直接压，避开部分涨落，适合限峰值的串流
- 超: GOPsize<bufsize=1~2×maxrate，超码率限制后直接压，但因视频小/crf 大所以没啥作用
- 欠: GOPsize>bufsize=1~2×maxrate，超码率限制后直接压，但因视频大/crf 小所以全都糊掉
- 由于 gop 多样， 4 种状态常会出现在同一视频中. buf~max 实际控制了这些状态的出现概率

--ratetol<浮点， 百分比， 默认 1(许 1%错误)>ABR， 2pass-ABR， VBR 的码率超限容错 tolerance

2pass-ABR 上层模式

首遍用 crf 模式分析整个视频总结可压缩信息，二遍根据 abr 模式的码率限制统一分配量化值. 除非有码率硬限，否则建议用 crf 模式. 目前所有视频网站一律二压，因此 2pass-abr 模式没有上传的用

```
--pass 1 --crf 20 --stats "D:\夹\qp.stats" [参数] --output NUL "输入.mp4"
--pass 2 --bitrate x --stats "D:\文件夹\qp.stats" [参数] --output "输出.mp4" "输入.mp4"
```

--stats<文件名>默认在 x264 所在目录下导出/入的 qp 值逐帧分配文件， 一般不用设置

--output NUL<不输出视频文件>;**--pass 1**<导出 qp.stats>;**--pass** <导入 qp.stats>

FTQP 下层模式

--qpfile<文件名>指定特定帧为 IDR， i， P， B， b 帧， 及 no-open-gop 下 K 帧的 frame type qp 下层模式. 文件内含"号位 帧类型 QP 值(换行)". 指定 qp 值为-1 时使用上层的 crf， abr， cqp 模式

多参考帧

--ref<整数-0.01×帧数+3.4， 范围 1~16>溯块参考前后帧数半径， 一图流设 1. 必须要在溯全尽可能多块的情况下降低参考长度，所以一般设 3 就不用管了ヽ(`◇´)/

--no-mixed-refs<开关>关闭混合溯块以提速，增加误参考. 混合代表 16×8, 8×8 分块的溯帧

自适应量化

adaptive quantizers 是防止对视频平面过度量化的功能. 用以根据量化程度进行画面补偿

--aq-mode<整数， 0 关， 范围 0~3>建议如下， aq 只在码率不足以还原原画时启动

4. <1>标准自适应量化(急用，简单平面)
5. <2>同时启用 aq-variance 调整 aq-strength 强度
6. <3>同时让不足以还原原画的码率多给暗场些(8-bit，或低质量 10-bit 画面)
7. <4>同时让不足以还原原画情况的码率多给纹理些(高锐多线条多暗场少平面)

--aq-strength<浮点>自适应量化强度. 据 VCB-S， 动漫的值太高浪费码率， aq-mode~strength 给<1 对 0.8>， <2 应 0.9>， <3 和 0.7>较为合理， 在真人录像上可以再增加 0.1~0.2， 画面越混乱就给的越高， 在 aq-mode 2 或更高下可以更保守的设置此参数

环路滤波

- **平滑 3**：a 或 1 皆为帧内块，但边界不在 CTU/宏块间
- **平滑 2**：a 与 1 皆非帧内块，含一参考源/已编码系子
- **平滑 1**：a 与 1 皆非帧内块，皆无参考源/已编码系子，溯异帧或动态向量相异
- **平滑 0**：a 与 1 皆非帧内块，皆无参考源/已编码系子，溯同帧或动态向量相同，滤镜关

--deblock<平滑强度:搜索精度，默认 1:0，推荐 0:0，-1:-1，-2:-1>两值于原有强度上增减

- 平滑<≥1>时用以压缩，<0~1>时略微降低锐度，适合串流
- 平滑<-2~-1>适合锐利视频源，4k 电影，游戏录屏．提高码率且会出现块失真
- 平滑<-3~-2>适合高码，高锐动画源和高画质的桌面录屏．高码率，增块失真，但高码动漫观感还是比 1 好
- 搜索<大于 2>易误判，<小于-1>会遗漏，建议保持<0~-1>，除非 qp>26 时设<1>

模式决策

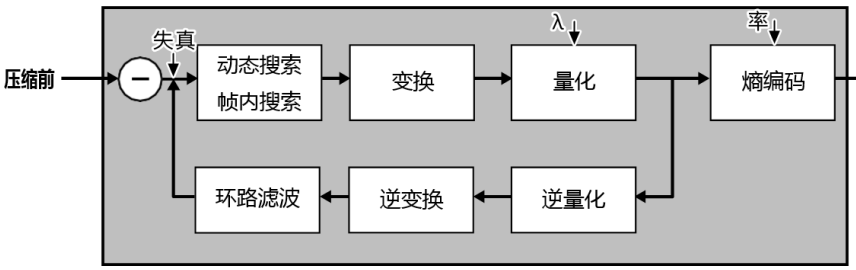
整合搜到的信息，在各种选项中给宏块定制如何分块，参考，跳过的优化

--deadzone-inter<整数 0~32，默认 21，trellis=2 时无效，小于 2 自动开启>简单省算力的帧间量化，细节面积小于死区就糊掉，大就保留．一般建议 8，高画质建议 6ヾ(≧▽≦*)o

--deadzone-intra<整数，范围 0~32，默认 11>这个顾及帧内．一般建议 5，高画质建议 4

率失真优化 RDO 控制

心理视觉 phycological visual 就是人眼视觉清晰度的研究．如图:在真人录像上常见，但不支持动漫的高频细节



率失真优化 Rate Distortion Optimization 据多个码率下测得的失真程度点，挑出低于率失真曲线的值，再根据码率/crf 宏观地做**模式决策**．x264/5 用拉格朗日代价函数 J = D + λ·R．即"开销 = 失真+λ·码率"．x264 中**失真** Distortion 用 SSE 和 SAD 判断．SSE 比 SAD 多了个取平方计算，参考源视为 100%，将每个编码块测出的值赋予参与率失真曲线计算的权重，让跑偏的编码结果影响小

失真用 MSE, SSE, SAD, SATD, MAE 判断, x264 中有 SSE 和 Noise SSE

--trellis<整数，范围 0~2，推荐 2>一种率失真优化量化 rdoq 算法. <1>调整 md 处理完的块，快速压制用，<2>+帧内帧间参考和分块 ￣o(￣- ￣メ)

--fgo<整数默认关，推荐 15 左右>改用 NSSE，提高画质，libx264 不支持的 Film Grain Opt.

--psy-rd<a:b 浮点，默认 1:0>心理学优化设置. a 保留画面纹理，b 在 a 的基础上保留噪点细节. 影片的复杂度越高，相应的 ab 值就越高. 压制动漫时建议选择<0.4~.6:0.1~.15>，--no-mbtree 时可尝试将 b 设为 0；压制真人选择<0.7~1.3:0.12~.2>(^ ◡ ^)

--no-psy<开关>若视频量化很低纹理很清楚，右图毛刺对画质不好就关. 录像中这些毛刺很重要

游程编码-霍夫曼树

色彩信息

光强/光压的单位是 candela. 1 candela=1 nit. 这些参数主要用于标记 x264 无法识别的片源

--master-display<G(x,y)B(,)R(,)WP(,)L(最大,最小)，与 x265 的格式不一样>写进 SEI 信息里，告诉解码端色彩空间/色域信息用，搞得这么麻烦大概是因为业内公司太多. 默认未指定. 绿蓝红 GBR 和白点 WP 指马蹄形色域的三角+白点 4 个位置的值×50000. 光强 L 单位是 candela×10000

SDR 视频的 L 是 1000,1. 压 HDR 视频前一定要看视频信息再设 L，见下

- DCI-P3 电影业内: G(13250,34500)B(7500,3000)R(34000,16000)WP(15635,16450)L(?,1)
- bt709: G(15000,30000)B(7500,3000)R(32000,16500)WP(15635,16450)L(?,1)
- bt2020 超清: G(8500,39850)B(6550,2300)R(35400,14600)WP(15635,16450)L(?,1)

RGB 原信息(对照小数格式的视频信息，然后选择上面对应的参数)：

- *DCI-P3: G(x0.265, y0.690), B(x0.150, y0.060), R(x0.680, y0.320), WP(x0.3127, y0.329)*
- *bt709: G(x0.30, y0.60), B(x0.150, y0.060), R(x0.640, y0.330), WP(x0.3127,y0.329)*
- bt2020: *G(x0.170, y0.797), B(x0.131, y0.046), R(x0.708, y0.292), WP(x0.3127,y0.329)*

--cll<最大内容光强，最大平均光强>压 HDR 一定照源视频信息设，找不到不要用，例子见下

```
Bit depth                : 10 bits
Bits/(Pixel*Frame)       : 0.120
Stream size              : 21.3 GiB (84%)
Default                  : Yes
Forced                   : No
Color range              : Limited
Color primaries          : BT.2020
Transfer characteristics : PQ
Matrix coefficients      : BT.2020 non-constant
Mastering display color primaries: R: x=0.680000 y=0.320000,
G: x=0.265000 y=0.690000, B: x=0.150000 y=0.060000, White point: x=0.312700 y=0.329000
Mastering display luminance: min: 0.0000 cd/m2, max: 1000.0000 cd/m2
Maximum Content Light Level: 1000 cd/m2
Maximum Frame-Average Light Level: 640 cd/m2
```

图：cll 1000,640. master-display 由 G(13250…开头，L(10000000,1)结尾

图：cll 1655,117/L(40000000,50)/colorprim bt2020/colormatrix bt2020nc/transfer smpte2084

--colorprim<字符>播放用基色，指定给和播放器默认所不同的源，查看视频信息可知：bt470m，bt470bg，smpte170m，smpte240m，film，bt2020，smpte428，smpte431，smpte432. 如图→为 bt.2020

--colormatrix<字符>播放用矩阵格式/系数：fcc，bt470bg，smpte170m，smpte240m，GBR，YCgCo，bt2020nc，bt2020c，smpte2085，chroma-derived-nc，chroma-derived-c，ICtCp，不支持图↑的 bt2020nc

--transfer<字符>传输特质：bt470m，bt470bg，smpte170m，smpte240m，linear，log100，log316，iec61966-2-4，bt1361e，iec61966-2-1，bt2020-10，bt2020-12，smpte2084，smpte428，arib-std-b67，上图 PQ 即 st.2084 的标准，所以参数值为 smpte2084

灰度

--fullrange<开关，7mod x264 自动>启用范围更广的显示器 0~255 色彩范围，而不是默认的旧电视色彩范围 16~235，如果源的视频流属性写着 bt709/full range 而不是 bt601/limited. 由于一般人不知道有这回事，所以视频来源一般/录屏软件不好的话应该检查一下，再决定用吧

其他命令行参数

开始/结束位置

--seek<整数，默认 0>从第 x 帧开始压缩**--frames**<整数，默认全部>一共压缩 x 帧

--fps<整数，特殊情况>告诉 x264 帧数

线程

--threads<整数，建议默认 1.5 倍线程>参考帧步骤要等其之前的步骤算完才开始，所以远超默认的值会因为处理器随机算的特性而降低参考帧的计算时间，使码率增加，画质降低，速度变慢

另外一种方案是让 threads 低于 cpu 实际线程， 而是尽可能接近 4 以增加参考帧的优先级， 但这就不如 sliced-threads 了. 而且 24 线程~threads=36 下， 画面的损失应该不会超过 0.5%

--sliced-threads<开关，默认关>x264 默认每帧逐线程，速度更快但代价是有的线程没法在确定的时间内吐出结果， 参考帧很容易等不到所以忽略掉本可以压缩的内容， 打开后降低处理器占用， 但压缩率可能会提高， 建议 16+线程的处理器，或高压缩用へ(￣ω ￣へ)

--slices<整数，默认自动但不可靠>手动指定"分片逐线程"下可以有多少分片，建议等于线程数

裁剪，加边，缩放/更改分辨率，删除/保留视频帧，降噪，色彩空间转换

--vfcrop：左,上,右,下/resize：宽,高,变宽比,装盒,色度采样,缩放算法/select_every：步,帧,帧…

/crop: 左,上,右,下指定左上右下各裁剪多少， 最终必须得出偶数行才能压制

/resize: 宽,高,,,更改输出视频的宽高， 建议搭配缩放算法后使用

/resize: ,,变宽比,,,减少宽度上的像素， 剩下的伸成长方形来达到压缩的参数. 任何视频网站都不支持，但网盘/商用的视频可以用这种压缩方法. 格式为源宽度：输出宽度

宽从 1920 到 1060，就是 96: 53(约分后)，就是 resize: 1060,高,96: 53,,,缩放算法

/resize: ,,,,色度采样,有 i420， i422， i444 和 rgb 四种，默认 i420. 在缩小视频分辨率，或者处理无损源视频时可以尝试使用已获得更好的大屏幕体验. 注意， 被压缩掉色彩空间的视频就不能再还原了

/resize: ,,,,,缩放算法

/select_every: 步,帧 1,帧 2...通过少输出一些帧以加速压制，用于快速预览压制结果，比如：

8 帧为一步，输出其中第 0, 1, 3, 6, 8 号帧: --vf select_every：8,0,1,3,6,8

90 帧为一步，输出其中第 0~25 号帧(最大 100 帧/步): --vf select_every：
90,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25

(仅 7mod-降噪) /hqdn3d：空域亮度降噪,空域色度降噪,时域亮度降噪,时域色度降噪

默认值是 4,3,6,4.5 若是编码画面模糊的源可以尝试默认值 1 到 1.7 倍. 若一定要用此参数来降低码率，可以考虑使用视频编辑软件的模糊滤镜

(仅 7mod-加边) /pad：←,↑,→,↓,改宽(不和加边混用),改高(不和加边混用)

用例(帧率减半，降噪)： --vf select_every:2,0/hqdn3d:0,0,3,1.5

分场扫描/隔行扫描

清晰度优先	别用	均衡
缩小/放大动漫图像：lanczos 缩小/放大录像图像：spline	bilinear	缩小任何图像 ：bicublin 其它：lanczos

--tff<开关>上场优先. **--bff**<开关>下场优先. -

-nal-hrd<开关， 默认关， 开 vbv>开启假想对照解码参数 hypothetical ref. decoder param. 零丢包零延迟环境中判断解码器额外所需信息， 写进每段序列参数集 sps 及辅助优化信息 sei 里， 适合串流

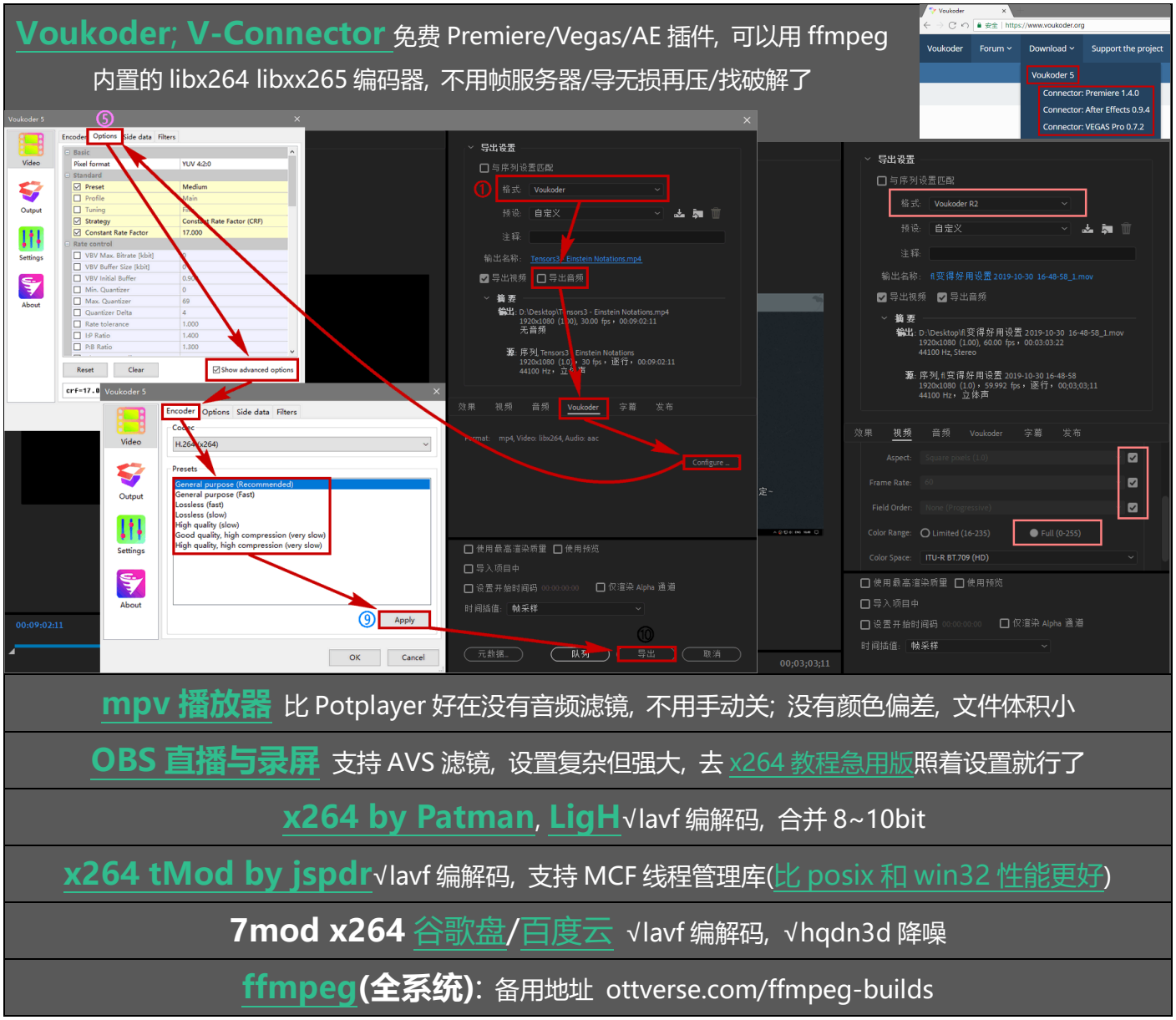
软件下载(((￣へ￣井)

ShanaEncoder ffmpeg-CLI 或 GUI 控制少量选项，高级功能(水印，高级字幕)用 ffmpeg 参数控制，上手需要时间. ffmpeg 内嵌编码器，不能替换文件

Simple x264 Launcher 英文软件，适合批量压制，需要自行封装音频

下图是画的汉化！只能压视频，但处理封装与压制音频，查看媒体元数据不如小丸工具箱，上手速度快.

小丸工具箱 提取码"crhu" 中文软件，压缩音视频，渲染字幕等，操作简单. 导入视频，点击自定义，将参数拷入，选好输出格式与滤镜就可以压制了，网上能搜到详细教程. 内嵌 MediaInfo, mp4box, Mkvtoolnix 可查看媒体元数据，封装/解封装，最适合新手



去可变帧率

ffmpeg 去 VFR：防止录像抽干手机电池的技术但差在兼容，编辑时需转换．手机先进则建议不用

pipe 的用法：″1.exe［输入］［参数］- | 2.exe［参数］［输出］-″．其中 "- | -"就是传递参数，第一个"-"代表输出，第二个"-"代表输入．而"- | - | - | -"就可以让同一个文件经过多个程序＼(=^･ω･^=)ノ

```
(仅 pipe) ffmpeg -i 输入 -f yuv4mpegpipe - | x264 [参数] - --demuxer y4m --output ″输出″
(去 vfr) ffmpeg -i 输入-y -vf fps=60 -hide_banner -f yuv4mpegpipe - | x264 [参数] - --demuxer y4m --output ″输出″
```

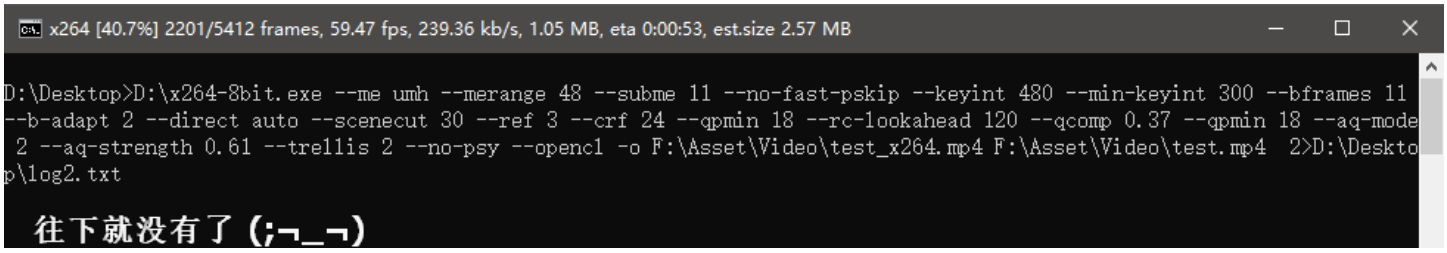
-f yuv4mpegpipe<预设字符>ffmpeg pipe 输出封装格式，此处设为 yuv for mpeg

-i<字符>ffmpeg 输入参数， **-y**<开关>不询问，直接覆盖掉同名的文件

-hide_banner<开关>ffmpeg 不显示 banner 信息，减少 cmd 窗口阅读量

--demuxer y4m<预设字符>x264 pipe 解封装格式，此处设为 yuv for mpeg

Win-CMD/Linux-bash 输出 log 日志



- Windows CMD: x264.exe[参数] 2>C: \文件夹\日志.txt [参数还可以写在右边]
- Linux Bash(或其它): x264.exe[参数] 2>&1 | tee C: \文件夹\日志.txt

附录

Potplayer播放器音量忽大忽小 ~~右键→声音/音讯→声音处理→反勾选标准化/规格化~~即可，但建议用mpv

CMD 窗口操作技巧%~dp0 "%~"是填充字的命令(不能直接用于 CMD). d/p/0 分别表示 drive 盘/path 路径/当前的第 n 号文件/盘符/路径，数字范围是 0~9 所以即使输入"%~dp01.mp4" 也会被理解为命令 dp0 和 1.mp4

这个填充展开后可能是"C: \"+"...\"+1.mp4，路径取决于当前.bat 所处的位置，这样只要.bat 和视频在同一目录下就可以省去写路径的功夫了

若懒得改文件名参数，可以用%~dpn0，然后直接重命名这个.bat，n 会将输出的视频，例子: 文件名=S.bat → 命令=--output %~dpn01.mp4 → 结果=1.mp4 转输出"S.mp4" (/·ω·)ﾉ

.bat 文件操作技巧 .bat 中，命令之后加回车写上 pause 可以不直接关闭 CMD，可以看到原本一闪而过的报错(┌■_■)

.bat 文件存不了 UTF-8 字符 在另存为窗口底部选择 UTF-8 格式

UTF-8 .bat 文件中文乱码开头加上 chcp 65001，打开 cmd--右键标题栏--属性--选择

.bat 文件莫名其妙报错 Windows 记事本会将所有保存的文件开头加上 0xefbbbf，要留空行避开

压制图像序列 不写命令，直接将视频导出成图像序列(一组图)加滤镜，在多软件协作上很省事，转回视频时需要指定帧数--fps<整数/浮点/分数>，搭配%xxd 即可 / (v x v) \

CMD 操作技巧%xxd 多文件编号规则，x 代表编号位数．比如"h%02d.png"就代表从"h00.png 到"h99.png"．由于%<整数>d 不能在.bat 文件里用，所以搭配%~dp0 使用就需要在 CMD(PowerShell, terminal 等)命令窗口中碰到一块去，用例: x264.exe[参数] [输出] --fps 30 F: \图像_%04d.png

CMD 操作技巧: 换色，试试这些命令: `color B0; color E0; color 3F; color 6F; color 8F; color B1; color F1; color F6; color 6; color 17; color 27; color 30; color 37; color 67`

命令行报错直达桌面，无错则照常运行: `[命令行] 2> [桌面]\报错.txt`

生成透明 rawvideo: `ffmpeg -f lavfi -i "color=c=0x000000@0x00:s=sntsc:r=1:d=1,format=rgba" -c:v copy output.avi`

PowerShell 内实现 UNIX pipe: 由于 PowerShell 内部跑完整个 pipe 再导出结果的机制不适用于程序间的 pipe 操作，因此需要用 cmd /s /c --%以在 PS 内部调用 CMD，例:

```
cmd /s /c --% "D:\ffmpeg.exe -loglevel 16 -hwaccel auto -y -hide_banner -i `".\导入.mp4`" -an -f yuv4mpegpipe -strict unofficial -pix_fmt yuv420p - | D:\x265.exe --preset slow --me umh --subme 5 --merange 48 --weightb --aq-mode 4 --bframes 5 --ref 3 --hash 2 --allow-non-conformance --y4m - --output `".\输出.hevc`""
```
