# x264视频压缩教程**精简版**

欢迎阅读！本教程精简了科普部分, 若有什么不会的可以直接加群[691892901](https://jq.qq.com/?_wv=1027&k=5YJFXyf)哦(`･ω･´)ゞ

### 压制软件工作流程图解（不严谨）



### 命令行参数command prompt

[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] 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](file:///C:\Windows\System32\drivers\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

# 正片 - 数据类型

## 宏块

: AVC视频标准定义为16x16大小的初始分块结构, 在此基础上x264会根据画面继续分块

## GOP

: group of pictures, 视频中代表一组由参考帧和非参考帧组成的子视频

## 关键帧

: 由(min-)keyint参数决定, scenecut参数判断. 一般作为gop开头

* I帧是图像, 给两侧B, P, b帧做参考. 节省设立IDR帧的算力
* IDR帧有创建进度条落点access point, 以及左右割为独立的子视频, 不再相互参考两个作用

## 参考帧

: 有IDR, I, P和pyramid-B帧四种. 区分P, B帧由b-adapt参数决定

* P帧含I P宏块, 上下帧不同但有压缩价值时设立. 往进度条前参考, 叫prediction frame
* pyramid B帧含I P B宏块, 上下帧几乎一样时设立. 给左右b帧参考, 实现了更长的连续b帧. 如IbBbBbBbP

## 非参考帧

: B帧, 其中的信息完全来自参考帧. 和pyramid B帧统称bi-prediction frame

## 嫁接帧:

SI和SP帧, 代表互动/多分辨率视频, 从附近的switching I帧切到另一条流上的SP帧

## 待定帧:

x264编码器, Lookahead线程内部还未处理, 记做AUTO的帧

--scenecut

<整数, 不建议修改>Lookahead中, 两帧差距达到该参数值则触发转场

--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

--min-keyint

<整数, 默认25>判断新发现的转场距离上个IDR帧是否小于该值长短. 有两种设定逻辑, 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ヘ(>\_<ヘ)
* 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

--ref

<整数1~16, 推荐fps÷100+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

--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

# 初始化与Lookahead

前瞻与分块等进程最早启动. 建立了GOP与帧类型(关键帧, 参考帧), 间接影响了mbtree, VBV等功能, 以及ABR, CRF量化模式的初始数据.

x265教程中展开关于SPS/PPS的信息. 大致流程见完整版教程

--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

--rc-lookahead

<帧数量, 范围1~250, 推荐keyint÷2>指定mbtree的检索帧数, 通常设在帧率的2.5~3倍. 高则占用内存增加延迟, 低则降低压缩率和平均画质. mbtree会自动选择--rc-lookahead和max(--keyint, max(--vbv-maxrate, --bitrate)÷--vbv-bufsize×fps)中最小的值作为检索帧数

--lookahead-threads

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

--bframes

<整数, 0~16>连续最多的B帧数量, 超出后设P帧. 一般设14, 手机压片建议设5省电

## P/B帧推演

--b-adapt 2

<所有情况, 整数0~2, 建议2 >0代表不设B帧

## VBV - 基于缓冲条件的量化控制

video buffer verifier手动指定用户的网络/设备下所允许的缓冲速度kbps, 以控制CRF/ABR模式. 与CRF模式一并使用时称为VBR双层模式

--vbv-bufsize

<整数kbps, 默认关=0, 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, 默认关=0>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

* 大: GOP大小=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: GOP大小=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: GOP大小<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: GOP大小>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

# 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( ﾟ▽ﾟ)/

--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

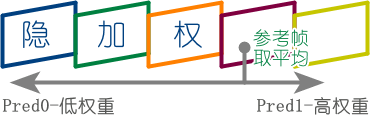
<开关>动态搜索不查色度平面, 以节省一点算力加速压制, 建议直播/录屏用

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

## 加权预测weighted prediction

解决少数淡入淡出过程中部分pu误参考, 亮度变化不同步所制造的画面问题. 分为P, B条带用的显加权, 编码器直接从原画和编码过的参考帧做差; B条带用的隐加权, 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

# 变换-量化

: 将频率从低到高的信息列出来, 用低通滤镜砍掉高频信息. x265教程中展开

# 码率控制模式



根据工况要求与妥协的变化选一种上层模式, 搭配0~全部的下层模式, 或特殊模式.一般情况默认crf, 直播时算力够用则crf~vbr, 直播时算力不够则用abr, 码率硬限下用2pass-abr, 做实验测试用cqp

### 质量呼应码率 / Constant Rate Factor / CRF上层模式

--crf

<浮点范围0~69, 默认23>据"cplxBlur, mbtree, B帧偏移"等内部参数实现每帧分配各自qp的固定目标质量模式, 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

### 平均码率 / Average Bitrate / ABR上层模式

--bitrate

<整数kbps, 指定则关crf>若视频易压缩且码率给高, 就会得到码率更低的片子; 反过来码率给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ABR, 快但妥协了压缩与画质

### 恒定量化值 / Constant Quantizer Parameter / CQP上层模式

--qp

<整数, 范围0~69, 指定则关crf, 非实验不建议>恒定量化. 每±6可以将输出的文件大小减倍/翻倍, 同速度下不如ABR, 同码率下不如CRF

### 常用下层模式

--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](https://www.desmos.com/calculator/aa7rsjuxkr)

--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"

--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

### 不常用: zones下层模式 **-** crf-zones及abr-zones两种搭配

--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=

* 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
* 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

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

### 不常用: **2pass-CRF-ABR特殊模式**

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 2**

<导入qp.stats>

### 不常用: FTQP手动模式

--qpfile

<文件名>手动指定特定帧为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

# 自适应量化

CRF/ABR设定每帧量化/qp后, 方差自适应量化variance adaptive quantizer再根据复杂度判断高低频信号, 来实现精确到宏块的qp分配过程. 讨论时注意aq与vaq的混淆

高压缩下, aq强度不足则纹理边缘的码率不足; 过高则平面/暗处的码率不足, 造成涂抹失真; 无损/快速编码时, aq强度低则好, 总的说强度要随CRF/ABR而动. 由于aq不计算帧间关系, 所以aq给出的结果往后还要用mbtree-lookahead, 率失真优化量化rdoq来宏观地重分配

--aq-mode

<整数0~3>据原画和crf/abr设定, 以及码率不足时(crf<18/低码abr)如何分配qp

* <1>标准自适应量化(急用, 简单平面)
* <2>+启用aq-variance, 自动调整aq-strength强度(录像-电影以及crf<17推荐)
* <3>+码率不够用时倾向保暗场(接受更明显的涂抹失真, 慎用)
* <4>+码率不够用时更加倾向保纹理(接受平面上的涂抹失真, 实验性, 慎用)

--aq-strength

<浮点>自适应量化强度. 搭配aq-mode, 如动漫1:0.8, 2:0.9, 3:0.7用. 录像上可加0.1~0.2, 画面混乱/观众难以注意平面时可再增加. 注意低成本动漫的平面居多, 因此码率不足时反而要妥协纹理

# 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. 图:在真人录像上常见, 但不支持动漫的高频细节

--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

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



**--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

图2: cll 1655,117/L(40000000,50)/colorprim bt2020/colormatrix bt2020nc/transfer smpte2084

**--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/色深, x264压制log



--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, 由于16~235的颜色管理更准确且码率更小, 所以能不用Fullrange就不用

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, 或高压缩用ヘ(￣ω￣ヘ)

--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

--vf

crop: 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

--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](https://shana.pe.kr/shanaencoder_download)** 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 |
| D:\Desktop\2017-08-30_181840.png[**Simple x264 Launcher**](https://bitbucket.org/muldersoft/simple-x264-launcher/downloads/) 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](https://pan.baidu.com/s/1VHonGHoZ0DmQBNZaRjML2A" \l "/)** 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 |
| [**Voukoder**; **V-Connector**](https://www.voukoder.org/)免费Premiere/Vegas/AE插件, 可以用ffmpeg内置的libx264 libxx265编码器, 不用帧服务器/导无损再压/找破解了 |
| [**mpv播放器**](https://mpv.io/installation/)比Potplayer好在没有音频滤镜, 不用手动关; 没有颜色偏差, 文件体积小 |
| **[OBS直播与录屏](https://obsproject.com/zh-cn/download)** 支持AVS滤镜, 设置复杂但强大, 去[x264教程急用版](https://nazorip.site/archives/334)照着设置就行了 |
| X2[**x264 by Patman**](https://www.mediafire.com/folder/arv5xmdqyiczc), [**LigH**](https://www.mediafire.com/?bxvu1vvld31k1)√lavf编解码, 合并8~10bit |
| [**x264 tMod by jspdr**](https://github.com/jpsdr/x264/releases/)√lavf编解码, 支持MCF线程管理库([比posix和win32性能更好](https://forum.doom9.org/showthread.php?p=1902336#post1902336)) |
| **7mod x264**[谷歌盘](https://drive.google.com/drive/folders/1kFCeNGA_wiiLt-DSeI3cyY8vxlffgQcy?usp=sharing)/[百度云](https://pan.baidu.com/s/1sbz8WztGTz3lcLzirHW_2w) √lavf编解码, √hqdn3d降噪 |
| [**ffmpeg**](http://ffmpeg.org/download.html)**(全系统)**: 备用地址 ottverse.com/ffmpeg-builds |

# 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ヽ(=^･ω･^=)丿

|  |
| --- |
| (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" (ﾉ･ω･)ﾉﾞ

.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

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`""

-----