x264视频压缩教程**完整版**

欢迎阅读本教程！本教程的难度一般, 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

# 部分1a: 常识啊常识（｀＾´）ノ

### 封装文件

* MP4是最通用的格式, 适合网站, 电脑, 移动端和网站播放, 对字幕的支持性一般(O・・O)/
* MKV可封装几乎所有的已知视频/音频/字幕格式, 就是不能直接导入视频编辑软件( •ω• )
* FLV 是目前最轻量的, 更适合网站, 电脑, 移动端和网站播放
* MOV由苹果公司研发, 是Adobe Premiere的最佳支持格式ヾﾉ. ÒㅅÓ)ﾉｼ
* M4A是AAC, ALAC等音频的封装, 因为可以多写点元数据＼（＠￣∇￣＠）／
* M3U由苹果公司研发在线播放音频的封装, 用http协议所以延迟低, M3U8指utf-8封装

编码-解码叫encoding, decoding. 编码即压制, 解码是编码逆运算, 即播放. 硬解代表将高能低耗用途少的专用电路嵌入x86/arm处理器替代软解, 专治手机看视频烫的优化. 方案有NVDEC, libmxv, OpenCL, MMAL, D3D9/11, VDPAU, Media Foundation等, 硬件编码方案有NVENC, 高通MMF, 高通Venus, 英特尔QSV, Conexant, Elgato等, 专治手机录像烫的问题

码率就是文件体积每秒, 单位Kbps或Mbps. 10MB, 1分钟的视频, 平均是(10÷1×60)×8bit = 1333.3kbps. 10MB, 下载5分钟, 网速即10÷(5×60)=33.3kBps, ×8比特=266.7kbps

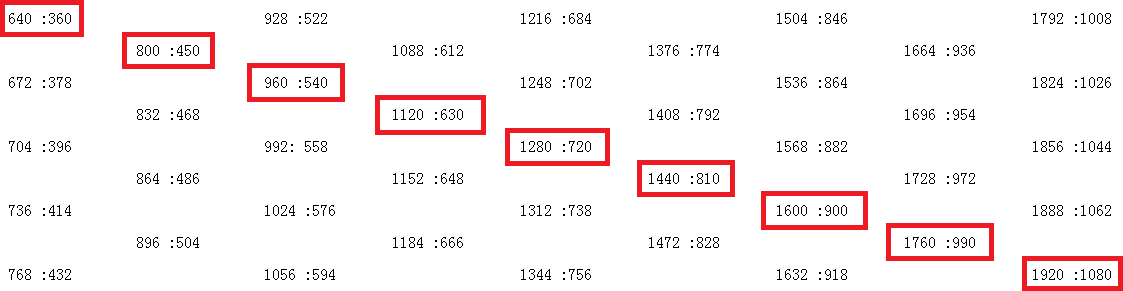


滤镜有降噪, 缩放, 锐化, 收线, 字幕等等. , 滤镜工具有Premiere, After Effects, Audition, DaVinci Resolve, AviSynth, VapourSynth, ffmpeg, FL Studio, Reaper, Ableton等等

三角形定律是不可兼得的视频压缩准则, 全都要的话只有升级处理器了

帧数就是图片数量每秒, 电影, 纪录片的帧数通常是24, 25帧, 网络视频的帧数应该是30, 60帧. 而小数帧率如23.976, 29.97帧不是为节省码率, 而是方便电视台对抗高压电的同频率干扰用, 网络视频用了会丢帧(( ;°Д°)

向量是物理概念, 指瞬间的速度+方向. 无论视频怎么动, 在每帧里都算直走. 所以虽然超纲, 但视频编码和插帧就是要算大量的插值得到这些动态信息, 也就是向量o(-`д´- . )



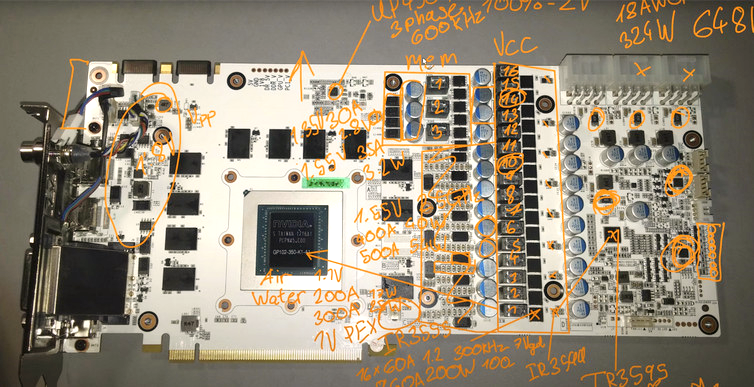
改分辨率是比例换算. 因为长方形的高÷宽能得出如4:3, 16:9这样的比例. 如高720px, 比例是4:3, 那么宽就是720÷3×4=960图: 16:9下的各种宽和高(Ф∀Ф)

视频编码器分为参考和日用两类. 如h.264/h264/avc参考编码器[jm](http://iphome.hhi.de/suehring/tml/)偏重还原算法, x264编码器就是开发出来干活的, 第三方可能会添加新功能和滤镜

**CLI** command line interface命令行交互, 输命令敲回车, 用好了速度最快, 最方便的交互方式; **GUI** graphic user interface图形界面交互, 有开关旋钮滑块的图形化程序, 包括手机app; 而**API** app. programming interface应用编写交互是最广泛的交互法. 比CLI难写, 但多了很多CLI没有的功能, 所以软件里有汇编🡪API🡪CLI🡪GUI的鄙视链(,,◕ ⋏ ◕,,)

mux封装和demux解封装分别缩写于multiplex混流和demultiplex拆流. 压缩包合成多个文件, 频率差距大的波形组成一整波形信号都算混流. 同样硬盘上拷大文件的速度更快. (≚ᄌ≚)

录屏是将显卡输出的画面返回给cpu压制, 或在显卡, 加速卡上做硬件编码再存到硬盘上的过程. 相比压制, 一般情况下的录屏为了不卡就放弃了画质和文件体积. 目前有显卡录屏, 升级处理器, 上计算卡, 或采集卡导入另一台电脑压制几种办法(⁄•˅̥•∖)

超频能让电脑变快. 分为处理器, 显卡和内存超频. 由于是按照最高性能运行, 所以中低端的供电, 电源, 散热方案与芯片体质会成为短板. 硬核超频根据PCB元件分析供电方案, 换掉低端元器件来补短. 需要电路设计与焊接基础. 不如更新bios, 系统, 芯片组驱动; 清灰, 上大散热器实在 图源: Buildzoid

有损, 无损, 未压缩, 图像vs视频vs音频

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 📷图像 | | 最大RGB YUV CMYK色深 | | | 动图 | HDR | 透明 |
| 有损 | jpg | 8 8 8 | 8 8 8 | 24 24 24 24 |  |  |  |
| 有~无 | gif（编码无损, 颜色少了） | 24×3中取8×3 |  |  | √ |  | 仅1bit |
| webp | 8 8 8 | 8 8 8 |  | √ |  | √ |
| jpg-XR（兼容jpg） | 32 32 32 | 8 8 8 | 16 16 16 16 | √ | √ | √ |
| avif, heif/heic, filf | 32 32 32 | 16 16 16 |  | √ | √ | √ |
| 无损 | pdf, jpg-LS, png | 16 16 16 |  | 32 32 32 32 | 仅mng |  | √ |
| 有损~无损~未压 | tif | 16 16 16 | 8 8 8 | 128 128 128 128 |  | √ | √ |
| 未压缩 | raw, bmp | 24 24 24 |  |  | 仅raw | √ | √ |
| dpx | 64 64 64 | 16 16 16 |  | √ | √ |  |
| 🎥视频 | | RGB/YUV色深 | | 编码速度 | | HDR | 透明 |
| 有~无 | qt, [hevc](https://x265.readthedocs.io), avc, vvc, vp8/9, DNxHR/HD, prores | 12 12 12 | | 慢~快 | | 除avc, vp8 | 仅prores |
| 无损 | rawvideo | 32 32 32 | | 快 | |  | [√](#_生成透明rawvideo) |
| [cineform](https://github.com/marksfink/cfenc) | 16 16 16 | | 快 | | √ |  |
| 有损~无损~未压 | flash动画 | 里面的图片决定(=^-ω-^=) | | | | | |
| 🎶音频 | | 音质 | | | | | |
| 有损 | mp3, aac, ogg(vorbis/opus) | 取决于耳机/音响, 混音和声场. 但你要是不知道HiFi和监听音频的区别, 就不用太关注这个啦 | | | | | |
| 无损 | flac, alac, ape, it |
| 未压缩 | wav |

色彩空间color space如下表(=ｘェｘ=)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 色彩格式 | 构成色 | 特点 | 存在原因 | 支持范围 |
| RGB | 红绿蓝 | 最常用 | 使显示器/照相机通用 | 几乎所有可视媒体 |
| ARGB | α红绿蓝 | 透明通道 | 不用抠图了 | 图片, 部分视频 |
| CMYK | 湛洋黄黑 | 减法色彩 | 多卖一盒墨 | 打印纸 |
| YCbCr (YUV, 近似YPbPr) | 白蓝-黄红-绿 | 压缩 | 压缩视频图片 | 所有有损压缩 |

色度采样chroma subsampling写作A:B:C; 每个长A的空间, 首行B个色度像素, 第二行C个

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 采样 | 1920x1080下宏观 | 色度微观, 宽A高2 | | | | 特点 | |
| 4:4:4 | 亮度, 色度皆1920x1080 | 色素 | 色素 | 色素 | 色素 | 逐行扫描, 直接读写亮色度而不用算, 所以剪视频最快 |
| 色素 | 色素 | 色素 | 色素 |
| 4:2:2 | 亮度不变, 色度960x1080 | 色素 ← | | 色素 ← | | 逐行扫描progressive, 颜色像素靠插值interpolation还原. 直接读写亮度所以剪视频比RGB快, (反正一般看不出来) | |
| 色素 ← | | 色素 ← | |
| 4:2:0 | 亮度不变, 色度半宽半高960x540 | 色素 ← ↑ ⬉ | | 色素 ← ↑ ⬉ | |
| 4:1:1 | 亮度不变, 色度1/4宽全高480x1080 | 色素 ← ← ← ← | | | | 分行扫描interlaced, 横着扫就不会把另一场的帧参考而浪费算力了. pixel aspect ratio指变宽比(原视频宽: 输出视频宽) | |
| 色素 ← ← ← ← | | | |
| 4:1:1 par4:3 | 亮度3/4宽全高, 1440x1080色度3/16宽全高, 360x1080 | 色素 5.333×← | | | |
| 色素5.333×← | | | |
| 4:2:0 par4:3 | 亮度3/4宽全高, 1440x1080色度3/8宽半高, 720x540 | 色素2.667←↑ 2.848 ⬉ | | 色素2.667←↑ 2.848 ⬉ | | 逐行扫描, 非方形亮度像素, par通常是4: 3~~要不然太烧脑了啊~~ | |

色深color-depth代表亮度Y的明暗密度. 和子网掩码转一样, 210代表1024, 28代表256个度; 度数不够会造成色带banding问题. 虽然一般用8bit就行, 但素材存档收藏什么的就用10bit吧. 浮点色深一般指单精度浮点数. 整数即232个度, 浮点则把科学计数法装进去, 代表(2-2-23)×2127个度哦ヾ(＠† ▽ †＠）ノ

逐行扫描播放时会按左🡪右, 上🡪下的顺序解码一帧. 分行扫描也叫隔行/上下场扫描. 用4:1:1采样, 跳一行像素解一半帧, 再解下一半, 是电视台信号抗干扰的手段之一. 很多旧游戏机为了兼容也加了分行扫描进去(¬‿¬)

ffmpeg fast forward mpeg是功能最全的CLI命令行工具, 可以编解码任何视音频, 渲染字幕, 也有自带的libx264/x265一站式完成视音频压制, 缺点是受版权所限, 导致编码器参数的写法不同

pipe(unix命令) 将内容从多个程序间按预定的顺序传递, 直至完成的功能, 食用方法见教程后半

噪点是录像视频中位于高频信息中的失真, 主要来源于CMOS因为于暗处光线不均出现的噪声. 因此不同设备拍出来的噪声分布不一, 以及算法不同带来的偏色, 就有了直接导出RAW格式用PhotoLab, LightRoom之类软件内置的同一种拜耳滤镜, 而非相机厂商各自的滤镜插值的需求

视频网站多指商业组织, 因此低流量UP不但不会得到画质优待, 反而要被剥削. 不要在画质上费神

# 部分1b: 食用方法, 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＼（＠￣∇￣＠）／

### 压制软件工作流程图解（不严谨）



### 压制软件用法

都是输入视频路径, 指定压缩力度, 然后设置导出路径, 点击压制就出片了

### 命令行参数command prompt用法

[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

有些地方多余的空格与换行会让编码器看不懂你写了什么, 所以带空格的路径会加引号(ΦωΦ)

### 压制报错检查

压制软件中, 如果看到视频一下子就完成, 但输出的视频不能播放, 就顺着log进度窗口往上找, 找到error字样后根据报错分析:

* 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▽(・ｗ・▽)

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

# 部分2: 正片 - x264的大致原理以及常用的参数ヽ(; ▽; )ノ



h.264标准下的视频文件由图组group of pictures组成, 将图拆分成16x16个像素组成的块就叫宏块, 而再根据边缘, 纹理细分到8x8与4x4大小的内容就叫做块. 压缩会因为在块上处理让计算方便了很多



# 动态搜索

通过算法找到多帧之间的移动变化信息, 用来冗余掉几帧之间重复的块 图: 动态搜索结果

--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)/

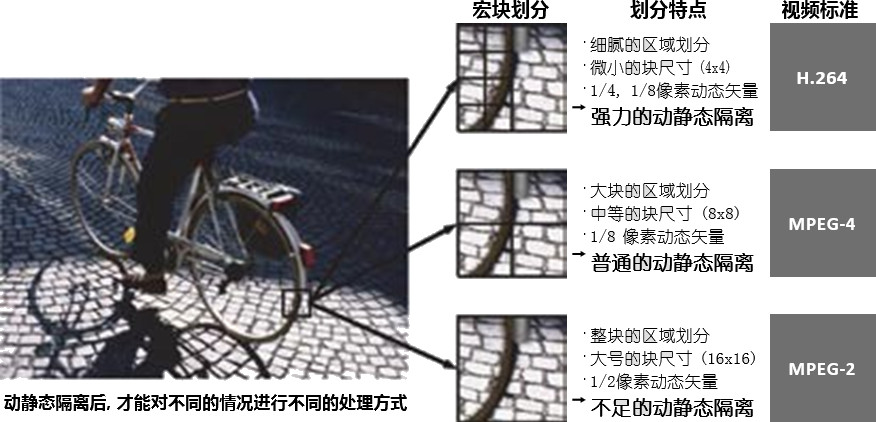
--merange<整数>越大越慢的动态搜索范围, 建议16, 32或48. 由于是找当前动态向量附近有没有更优值, 所以太大会让编码器在动态信息跑不到的远处找或找错, 造成减速并降低画质

--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

motion compensation将动搜所得信息做插值连贯起来, 解决相当于游戏瞄准颗粒度大于显示dpi, 微移鼠标看见画面一跳一跳瑕疵的步骤 (˘▾˘)

--subme<整数, 范围0~11>画面插值. 耗费算力但非录屏情况建议直接上最高11(国外mod版的x264版本最高是10). crf模式录屏时可以大幅降低, 3A游戏录屏在3~4左右

## 加权预测weighted prediction

avc首发, 治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

通过预设的间隔及画面差异判断设立. 作为参考帧信息来源冗余掉时间上重复块的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帧是设立后作进度条落点access point的i帧, 左右割为独立视频, 不再相互参考

# 参考帧

* 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
* b帧含I P B宏块, 上下帧几乎一样时设立. 往进度条前-后参考, 叫bi-prediction frame( ´¬`)ノ
* pyramid B帧给左右b帧参考, 实现了更长的连续b帧. 如 IbBbBbBbP形成了尖塔pyramid状的参考树

--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, 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

* SI和SP帧, 代表互动/多分辨率视频得到命令时, 从附近的switching I帧切到另一条流上的SP帧
* 还有利用SI帧独立于视频流, 以及流中SP帧只参考SI帧的特性, 以修复损坏视频的用处

# 率控制; 参数集

给视频分好段以对付接下来的帧内编码, 且形成树叉状的参考结构. 一来冗余, 二来防止参考/解码错误蔓延, 照顾丢包人士, 三来搭建NALU为基础不断传输ss的网络串流架构(ᗒᗣᗕ)՞

1. 按IDR帧间隔(keyint)分区
2. scenecut分配额外关键帧
3. 按开/闭-gop标记gop间的预设
4. 照标准将gop内的帧拆为条带slice
5. 条带一样要拆开来以降低解码错误的影响, 叫做条带段或ss

结构为: (播放时间戳, 显加权与其它特定解码要求)序列参数集sequence parameter set🡪(分枝-解码信息)图参数集picture parameter set🡪(分枝-ctu以上最小单位)条带段slice segment

--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

# D:\Desktop\帧内搜索.png帧内编码

见图

# 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. 图中拿不在视频编码里的傅里叶变换举例



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



avc和jpeg中使用的离散余弦变换相当于把二维傅里叶变换的结果列成表来暴力穷举. 快且信息无损. 傅里叶变换优势在于可编辑性, 当然这里说的是科学计算软件, photoshop之类还是找插件吧…

--crf<浮点, 范围0~69, 默认23>据情况给每帧分配各自的量化值qp, constant rate factor固定质率因子, 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

量化开太高还会降低参考质量. 出现块缝是因为deblock设的太低; 出现亮斑是因为psy, aq太高

--qp<浮点, 范围0~69, 指定所有帧> quantization parameter量化值, 同样每±6可以将输出的文件大小减倍/翻倍. 但直接指定qp会关crf, 影响其后的模式决策, 综合画质下降或码率暴涨. 也叫constant qp. qp=0时启用无损量化, 但因为码率太大所以也不建议

--qpmin<整数, 范围0~51>最小量化值. 由于画质和优质参考帧呈正比, 所以仅在高压环境建议设14~18(〃▽〃)

--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, 其他情况下就是...对标prores了 (\*~▽~)

### **VBR模式**

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

--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种状态常会出现在一个视频中. 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

### **ABR模式**

编码器自行判断量化程度, 尝试压缩到用户定义的平均码率average bitrate上, 速度最快| ⍤⃝

--bitrate<整数kbps>平均码率. 若视频易压缩且码率给高, 就会得到码率更低的片子; 反过来低了会不照顾画质强行提高量化, 使码率达标. 如果给太低则会得到码率不达标, 同时画质\*\*的片子. 平均码率模式, 除2pass分隔, 一般推流用的"码率选项"就是这个参数, 速度快但同时妥协了压缩. 因此算力够的直播建议用crf~vbr模式, 码率>画质, 但画质也抓的压片用1pass-crf+2pass-abr

### **1pass-CRF + 2pass-ABR模式**

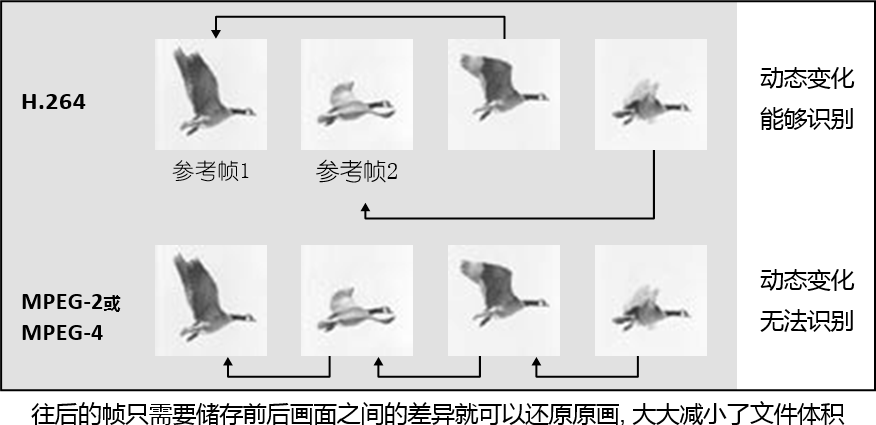
同时利用crf和abr模式的特点, 首遍分析整个视频总结可压缩信息, 第二遍照着给出的码率限制压缩, 是在码率限制下降低最少画质的低画质压制方法, 需命令行手动操作:

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

**--stats**<路径>默认在x264所在目录下导出/入, 一般不用设置

**--output NUL**<开关>视频文件不再输出

**--pass 1**<开关>修改--stats参数作导出 **--pass 2**<开关>修改--stats参数作导入



# 多参考帧

强化对连续-周期性, 有动静态内容, 连续转场画面溯块的能力

--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

--qcomp<浮点, 范围0.5~1, 一般建议默认0.6, ~~x264~~>crf模式分配qp值的延迟. 延迟高代表瞬间画质和码率都能暴涨, 影响码率变化所以也叫rate variability

<0.5>实时跟随画面拉高qp, 助于抗高码, 适合串流; <接近1>延迟跟随画面拉高qp, 有助于防止画质突然下降, 适合一般~收藏画质; <1>不管, [暂停画质最高](https://forum.doom9.org/showthread.php?p=1289361#post1289361), 类似直接设--qp但因qp而画质更差

# 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(. ・//ε//・. )

<1>标准自适应量化(急用, 简单平面); <2>同时启用aq-variance调整aq-strength强度(录像); <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) <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. 根据VCB-s的建议, 动漫的值不用太高(码率浪费). 在动漫中, aq-mode / aq-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

--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

# 环路滤波

修复高量化qp>26时宏块间出现明显横纵割痕瑕疵的平滑滤镜. 编码器内去块相比于外部滤镜能得知压缩待遇信息(两个相邻块的量化, 参考待遇差异是否过大)从而避免误判原画纹路. 码率跟不上就一定会出现块失真, 所以除直播关掉以加速外, 任何时候都应该用; 但去块手段目前仍是平滑滤镜, 因此要降低强度才适用于高码视频, 动漫, 素材录屏等锐利画面(੭•ω•˚)੭̸\*✩

**边界强度Boundary Strength/去块力度判断:** 图: 取1234 | abcd块间的界线举例



* **平滑4：**a与1皆为帧内块, 且边界位于CTU/宏块间, 最强滤镜值
* **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<滤镜力度:搜索精度, 默认1:0>两值于原有强度上增减. 一般推荐0:0, -1:-1, -2:-1

平滑强度勿用以压缩(大于1), 推荐照源视频的锐度设<-2~0>. 设<-3~-2>代表放任块失真以保锐度, 仅适合高码动画源. 设<小于-1>会导致复杂画面码率失控; 搜索精度<大于2>易误判, <小于-1>会遗漏, 建议保持<0~-1>, 除非针对高量化qp>26时设<1>

# 模式决策

mode decision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. 若没有psy-rd, psy-rdoq(x265)和trellis优化, 模式决策就一定用码率最小, 复杂动态下全糊的方案集. mb树也是md的一部分(〃⌒▽⌒)八(〃⌒▽⌒〃)

--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

--no-mbtree<开关>暗场细节的码率占用其实和亮场差不多, 不用mbtree会避免暗场画质损失, 所以收藏画质下关掉(◞⁎˃ᆺ˂)◞₎₎=˳˚॰°ₒ৹๐

# 率失真优化RDO

心理视觉phycological visual就是人眼视觉清晰度的研究. 如图: 复杂度对真人录像的重要性, 但这些点点毛刺在动漫里就难看了. 一开始认为失真越小越好, 后来发现编码整体速度被限死, 码率还超而引入的功能. 大致是给帧内帧间参考帧量化分跑个分. 用开销=失真+λ·模式下码率决定限码还是提画. 失真主要有MSE, SSE, SAD, SATD, RMSE, MAE, RSS等算法, x264中有SSE和Noise SSE, 由于参数依赖, 所以即使超纲也得看算式:

(x265) Mean Squared Error = 1÷宽高 ∑x=0🡪宽∑y=0🡪高 |f(x,y)-f'(x,y)|²

(x264) Sum of Squared Err. = ∑x=0🡪宽∑y=0🡪高 |f(x,y)-f'(x,y)|²

(x264) Noise SSE (--fgo) = ∑x=0🡪宽∑y=0🡪高 |ΔN(x,y)×fgo| + |Δf(x,y)|²

* 1÷T 代表多周期信号中平均一周期长, 此处将整帧的误差平均到1px来增加计算量, 卡自由度
* ∑ 代表多周期信号中划定范围内求和,此处确立了所有要算的像素
* f与f'(x,y)可略做Δf(x,y), 即原画和编码后的像素做差值, f是本体, xy可看做坐标
* | | 意义是防止原画减编码像素减出负值(编码后像素值变高而失真), 需转正
* ² 放大了严重误差(猜测), 编码器内省了算绝对值| |的步骤
* 由于MSE并没有明显带来更好的量化策略, 所以猜测用fgo ~15可以在锐利画面上超越x265

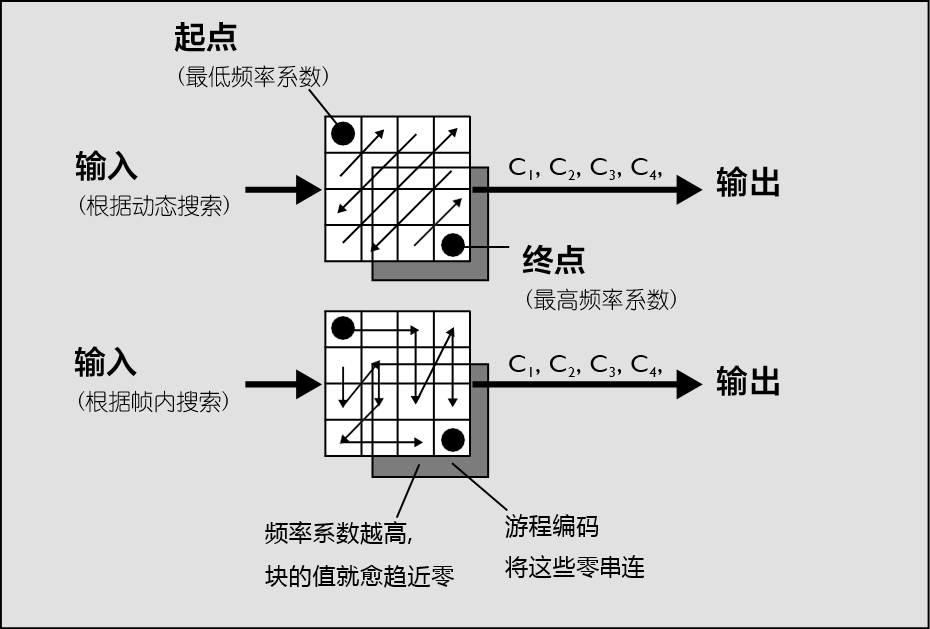
--trellis<整数, 范围0~2, 推荐2>一种率失真优化量化rdoq算法. <1>调整md处理完的块, 快速压制用, <2>+帧内帧间参考和分块￢o(￣-￣ﾒ)

--fgo<整数默认关, 推荐15左右>改用NSSE降低压缩, 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-熵编码

量化后的块属于数学/编程上的矩阵. 用游程编码扫描后降维到字符串, 再用文本压缩算法即可耗费一点算力缩减高到1/3的体积. 游程编码(右图)两种蛇形的扫描路径是为了串联更多的0出来, 方便文本压缩算法建立0和1代表的二叉树, 将出现最多的字放在距起点最短, 出现最少则放在距起点最长的位置; 分别采用最少和最多的0和1排序(⌒\_⌒;) 排序完成后, 所有的字符都会被放在树梢上, 而于原点最近, 出现最多的字一般在树的左边. 从树根的分叉0左1右地走到树梢上即可不重复地算出字符了. 主要有可变长度编码VLC, 二进制算术编码BAC和霍夫曼编码, 统称为熵编码. 到此, 视频压缩就完成啦

# 色彩信息

光强/光压的单位是candela. 1 candela=1 nit

--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/真HDR: 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)

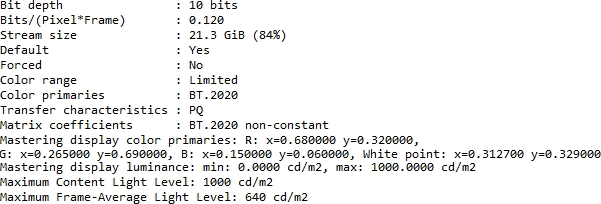
*RGB原信息(对照小数格式的视频信息, 然后选择上面对应的参数):*

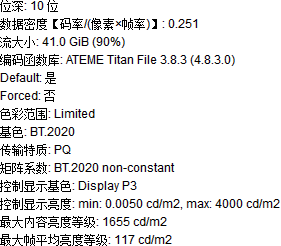
*DCI-P3: G(x0.265, y0.690), B(x0.150, y0.060), R(x0.680, y0.320), WP(x0.3127, y0.329)*

*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←应设max-cll 1000,640. master-display由 G(13250…开头, L(10000000,1)结尾

图↓应设max-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/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, 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\( 'з')/





# 其他命令行参数

--seek<整数>从第<>帧开始压 --frames<整数>一共压<>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

## 节点编码

--stitchable<"缝合"开关>重编视频指定区域, 目前搭配trimx264opt能修复源的损坏处而不二压; 未来可能能搭配ABR天梯(x265教程)一次输出多套参数的压制结果, 取理想片段重组的超级2pass

## 不同参数划区压制

--zones<高级命令>在视频中划区, 每个区域使用不同参数, 高压视频用

格式是帧A,帧B,参数A=x,参数B=x… 可用的参数有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~∞%的码率, 一般用不到
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. 多个划区可以隔开: --zones 0,449,crf=32,me=dia,bframes=10/450,779,b=0.6,crf=8,trellis=1

限制: zones内的--me和--merange强度/大小不能超过zones外

用法: 给除了第一集的OPED高压缩, 反正没人看; 给片尾credit高压缩, 反正没人看(咳咳)

## 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

|  |  |  |
| --- | --- | --- |
| **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 |

/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, 已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, 空域降噪): --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

# 部分3: 开工 - 软件下载与使用方法(((￣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 |
| [**Patman x264**](https://www.mediafire.com/folder/arv5xmdqyiczc) √lavf编解码, 支持subme11, 合并8~10bit |
| **[官方32位x264](I:\\JCSUSBbackup\\x264压制教程\\download.videolan.org\\x264\\binaries\\win32)** **[官方64位x264](I:\\JCSUSBbackup\\x264压制教程\\download.videolan.org\\x264\\binaries\\win64)** ×lavf编解码, 不能封装mp4 |
| **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, √subme11, √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

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"梨%02d.png"就代表从"梨00.png到"梨99.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:** 换颜色看更舒服

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