x264视频压缩教程**完整版**

欢迎阅读！本教程难度一般, 不具备学术价值, 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/Venus, 英特尔QSV, Conexant, Elgato等, 专治手机录像烫手的问题

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, ×8bit=266.7kbps



色彩

是眼睛捕获特定长度的电磁波, 再经过脑补而产生的幻觉

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

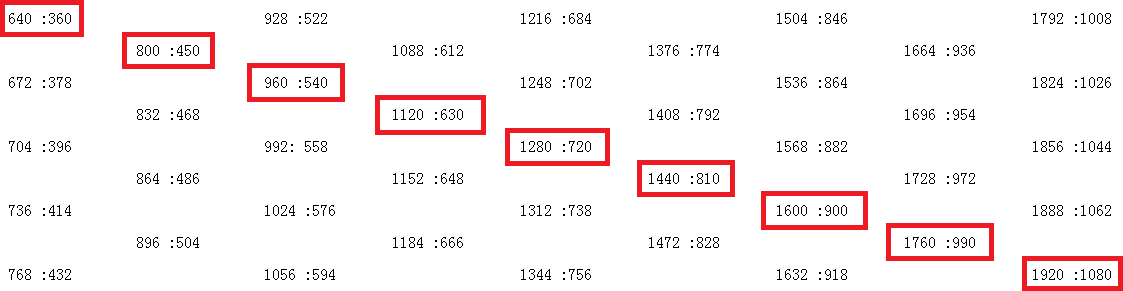
向量

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

### 宽高比/展弦比

aspect ratio是视频的宽高标准. 修改分辨率

需要按比例换算.换算时将宽/高除以其对应的比例, 再乘以临边的比例即可缩放出目标边的比例, 小数结果取最接近的偶数即可. 如高720px, 比例是18:9, 宽就是720÷9×18=1440. 图: 16:9下的各种宽和高(Ф∀Ф)



**GUI**

graphic user interface图形界面交互(开关旋钮滑块), 上手容易, 但有时缺功能就只能改用CLI

**CLI**

command line interface命令行交互(输命令敲回车), 上手难但越用越方便, 缺功能就改用API

**API**

app. programming interface应用编程交互(GUI/CLI套壳到程序或驱动), 排版瞎眼, 因为设计出来就是给程序用的… 一些视频编码器的修改版就是新加了CLI命令来关联到某个API的版本

mux

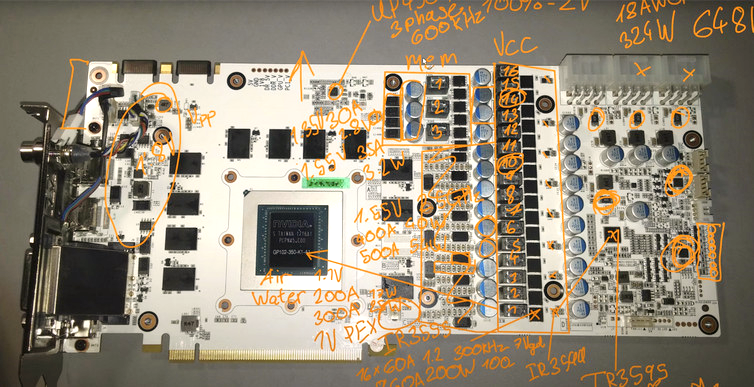
封装,

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

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, 散热方案是短板, 需要换掉低端元器件来补短; 或者购买供电方案, 切断原厂供电, 再桥接到方案板上获取稳定的超频结果. 同时需要BIOS编写, 电路设计与PCB焊接经验. 内存超频对于视频编码等长时间高强度计算有帮助, 但对此之外, 包括游戏体验的提升都难以察觉 图源: Buildzoid

单核性能

由处理器核心的设计与制造决定. 增核心面积=降良率, 缩小晶体管=挑战硅元素, 增核心数量=程序员折寿≈优化烂, 提高频率=挑战热力学. 因此虽然主要看多核性能, 单核性能的提升才是关键

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, 或([B](https://www.bilibili.com/video/BV1TA41147RY/), [Y](https://www.youtube.com/watch?v=FTKP0Y9MVus)站)的科普视频, 包括色度采样, 色彩平衡, 调色基础, 调色工具原理, 心理视觉原理, 多媒体色彩控制

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 🎆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. 210代表1024, 28代表256个度, 和子网掩码一样; 度数不够会出现画面渐变色带banding, 以及音频低音糊掉的问题. 可用频闪或抖动来缓解



### 抖动滤镜

有[随机](https://bartwronski.com/2016/10/30/dithering-part-one-simple-quantization/), [蓝噪声, 黄金分割](https://bartwronski.com/2016/10/30/dithering-part-two-golden-ratio-sequence-blue-noise-and-highpass-and-remap/), [Riemersma](https://www.compuphase.com/riemer.htm), Floyd等算法, 通过破坏量化使输出结果更接近原始信号, 用分辨率/音频采样率换位深的补偿方法, 因此一般在视音频高转低位深时用到, 或像奥博拉丁的回归[用不同抖动算法来区分材质](https://forums.tigsource.com/index.php?topic=40832.msg1363742#msg1363742)的ditherpunk美术. 噪点

源于胶片表面粗糙, 相机/电视强化暗场信号/无信号而产生的白噪, 而降噪手段只有模糊, 或者堆算力来尽量避开画面细节的模糊.

偏色

是相机内置拜耳滤镜补偿CMOS空缺区间, 因算法不同导致的画面问题. 催生了直接导出

RAW格式

. 用PhotoLab, Lightrooms软件内置拜耳滤镜统一处理的需求

### 伽马

gamma/希腊符γ代表亮度Y值或信号强度.

### 伽马矫正

分为消除录剪播环节的硬件缺陷, 如CRT屏要放大电压到才能正确显示灰阶; 以及解决未校正/RAW画面在演化出黑化非线性光感()的人眼看来是漂白变灰"问题"的两用. 由于电压-光感的幂互为倒数, 所以sRGB标准中将这个过程简化为"录制时直接白化(, 显示器跳过黑化". 因此用任何工具处理媒体时, 要开启"按照伽马系数混合RGB颜色, 伽马值=1 (Adobe软件默认关)"才能正确(且好看)的混合和模糊颜色

### **逐行扫描**

progressive按左🡪右及上🡪下的顺序播放各帧, 例如宽达4像素, 高1080像素的逐行扫描视频就和1920x1080逐行扫描视频一并叫做1080p视频.

分行/隔行扫描

interlaced用4:1:1采样, 分

上场优先

: 先传输上帧/奇数行及

下场优先

: 先传输下帧/偶数行, 用于降低传输模拟电视信号干扰. 同期不受干扰问题的电影标准用逐行扫描. 由于标准同时还决定分为电影24fps, NTSC电视60fps, 和PAL电视50fps三种帧率, 所以标准间做转换还要将电影源进行NTSC 3:2 pulldown或PAL 2:2 pulldown转为分行; 反过来要做inverse telecine (IVTC)才能转回24fps的逐行视频, 方法见教程尾

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, 而是抗模拟信号干扰用, 网络视频用了会丢帧(( ;°Д°)

视频网站定律

指对低流量UP限码, 高流量UP画音质优待的普遍现象, 所以想提升码率要先火起来

|  |  |  |
| --- | --- | --- |
| 🧠架构 | 优点 | 缺点 |
| Reduced Instruction Set Computer  精简指令集电脑(RISC架构, CPU) | 堆软件设计, 堆核心 | 编程难优化, 同频率更慢 |
| Complex Instruction Set Computer  复杂指令集电脑(CISC架构, CPU) | 堆硬件工艺, 堆制程 | |
| Application Specific Integrated Circuit  特定应用集成电路/定制方案(ASIC架构, TPU) | 交换机/矿机/防火墙省电+快+延迟低+对电源友好 | 芯片界苹果, 功能+1≈∝价格×10 |
| Application Specific Standard Product  特定应用通制方案(SoC, CPU, GPU, NPU) | 什么都能塞 | ASIC, C/RISC缺点, 贵, 总有一代产品会炸功耗 |
| Field Programmable Gate Array  当场可编程门阵列(FPGA架构, CPU) | 以上什么都能模拟, 产品未到驱动先写测试先行 | 使用门槛高, 同性能下性价比远低于ASSP |
| Advanced RISC Machines (ARM公司) | 一司包办现成设计 | 只有少量公司能做, 性能通常不发烧 |
| X86架构 | 主CISC, 副RISC |

指令集

是操作处理器与内存间读写复制粘贴计数寻址来满足软件需求的代码, 随处理器发展变迁

|  |  |  |
| --- | --- | --- |
| 📙编程语言 | 优点 | 缺点/妥协 |
| 下层编程语言 Fortan, C语言, Cyclone, Erlang | 编译成机器码后最快 | 手动内存和指针管理 |
| 双层编程语言 C#, C++, Rust-Haskell | +用户输入输出支持 | 学习成本最高 |
| 上层编程语言 tcl/tk, R语言, PHP, Ruby, Python, JavaScript, Go, F#, Visual Basic等 | 自动内存和指针管理 | 比C慢几万倍 |
| 折中编程语言 用Python, Java, .NET调用C/Rust等库 | 以上所有 | 你手机的储存空间 |
| 非编程语言 SQL, Bash-CMD-PowerShell, GNU grub, Cisco IOS, html, CSS, markdown | 反正就是能用 | 离了软件/系统没用 |

# 食用方法, 注意事项

1. 使用急用版搭配本篇/精简版配置压制参数, 忘了就打开文档查ヾ(●ω●)ノ
2. 到后面的下载栏Get压制软件, 然后研究一下怎么用＼（＠￣∇￣＠）／

### 压制软件工作流程图解（不严谨）



### 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

: 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)

# 正片 - x264的原理及数据类型ヽ(; ▽; )ノ

AVC标准下, h264/x264编码器给出的文件由GOP组成, 每帧从宏块开始细化到最小4x4的块

## D:\Desktop\x264历史.png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

## 待定帧:

编码器内部, Lookahead线程内部未处理且没有被手动指定, 记做AUTO的帧

--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

--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, 剪辑素材设5秒一个

--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. 大致流程如下:

1. 从--seek参数指定处开始, 将视频帧导入Lookahead进程, 顺序和进度条一样从左往右
2. 后续步骤发现任何GOP结尾时, 当前的h->Lookahead->next->list帧列表就后移相应的帧数
3. **转场判断①:** 当输入的第0帧是AUTO/I帧, 且与之前/左侧的非B帧差距达到--scenecut阈值-触发转场时设IDR帧

* 设IDR(或受--open-gop限制而设I帧)后, 其前一帧强行设为P帧, min-GOP遂变GOP, Lookahead列表后移

1. **P/B帧区分:** 据--b-adapt指定的算法, --bframes的最大连续B帧长度区分出P/B帧. 得到如PBBBPBP的序列

* 为保证超长连续B帧可行, --b-pyramid参数让偶数的连续B帧变成参考帧

1. 单B帧/连续B帧与其后的P帧视为min-GOP
2. mbtree搜索时-空间范围内的块分布, 找出少见宏块后记下拉高量化值qp的偏移程度
3. **转场判断②:** mbtree跑完后, 计算输入帧与上个关键帧的距离, 超出--keyint阈值参数则设为IDR

* 设IDR(或受--open-gop限制而设I帧)后, 其前一帧强行设为P帧, min-GOP遂变GOP, Lookahead列表后移

1. 按--rc-lookahead参数指定的长度为一组, 通过做差大致给出一组帧的预设量化值qp备用
2. VBV进程检查rc-lookahead范围内给的qp是否能控住码率, 超码则提高这批qp备用

--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

--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, 超出则设立P帧, 一般设14, 手机压片建议设5来降低Lookahead进程的内存占用

## P/B帧推演-维特比算法

Viterbi-SPF是多起点-多终点的最短路径shortest path 1st算法. 类似的有导航上常见的Dijkstra-SPF. 推演B/P帧相当于解决隐马尔科夫模型问题.

图: bframes=5, 每层迭代都保留B/P枝梢, 每个枝梢选取一次最短根路径, 最后P帧收尾, 得到累计距离最短, 即码率最小的方案

--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/播放时解出每gop原画帧数的缓冲用时秒数. 相当于要求编完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

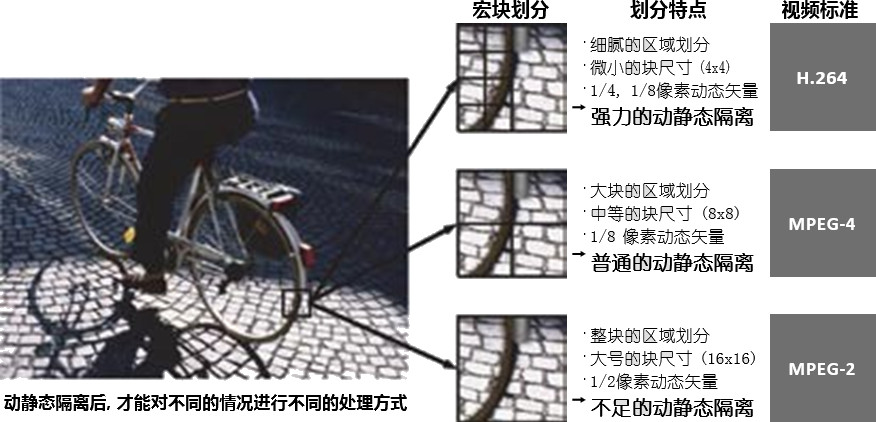
通过算法找到帧间的移动变化信息, 以冗余掉帧间重复块 图: 动态搜索结果

--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, 而非参数设定区域. 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"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

**绝对差异求和Sum of Absolute Difference**: 冗余压缩后, 对比压缩前的原始帧逐像素做差, 取绝对值加到一起, 得差异总数为失真程度;

**绝对变换差求和Sum of Absolute Transformed Difference**: 改用完成变换计算的帧与源做差, 比冗余压缩往后一步, 距离压缩后的帧更近, 更慢但失真算的更准

绝对(变换后)差异和

* ∑x=0🡪Tx代表块宽度求和范围, f()和f'()分别代表参考块和参考源
* ∑y=0🡪Ty代表块高度求和范围, x, y代表块中的像素坐标, | |求绝对值, 否则求和时正像素值差异会减去负

SAD和SATD在多种编码器, 调制解调器以及滤镜中出现. 因为有信号的地方就有失真, 就要判断失真

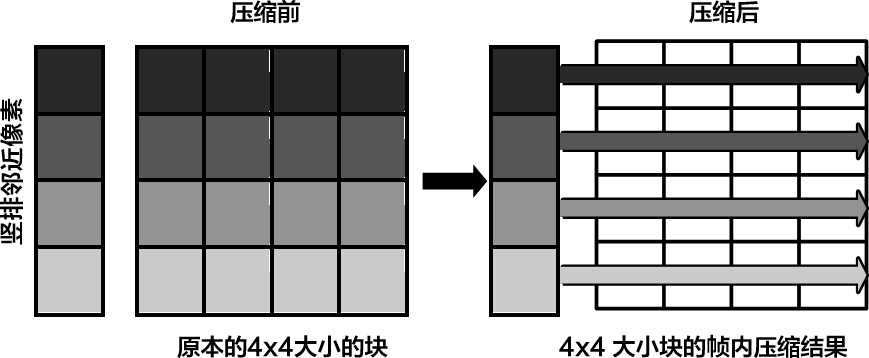
--subme

<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

# 帧内编码

: x265教程中展开

# 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

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"

### 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=

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

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closed-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再根据其信息复杂度来分成高低频两类, 从而将qp细化到块上. AQ与VAQ存在混淆.

## 方差

variance代表差-方-和-均的计算顺序. 将数据样本逐个与整体平均值做差, 通过累计偏差本身来判断数据的平衡性, 乘方是为方便正负数一起求和计算(不过σ也有平方):

VAQ中, 样本就是宏块里的像素, 要比的平均值就是宏块范围的平均像素值. 求和∑因为差异平方和SSE, 以及差异和SAD算过了所以不写, 样本量N就是宏块范围的像素数量. 通过计算像素值的方差, 就能看出哪些宏块含高频信号多, 就能实现vaq了

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

修复高量化时宏块间出现明显横纵割痕瑕疵的平滑滤镜. 编码器内做去块能用到压缩待遇信息而减少误判. 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.

### 边界强度

boundary strength(去块力度判断): 取最小8x8块间的界线举例. (不是4x4)



* **平滑4：**a与1皆为帧内块, 且边界位于CTU/宏块间, 最高值
* **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

<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

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

# 率失真优化RDO控制



心理视觉phycological visual是对人眼感知清晰度的研究

率失真优化

rate distortion optimization据多码率下测得的失真程度(编码前后像素值差异)点, 挑出低于RD曲线的值.

其中拉格朗日代价函数(开销=失真+λ·码率; J = D + λ⋅R)实现

模式决策

. 失真D用差异平方和SSE和总差异SAD判断. SSE多取一步平方, 使较大的差异呈指数增长, 进而分到更多码率实现补偿. 取平方的负值转正效果是良性副作用

图: 在真人录像上常见, 但不支持动漫的高频细节

x264 差异平方和

拉格朗日值λ

从qp值得出, 即crf, abr指定的率失真斜率区间. qp越大斜度越小. λ=0则无斜度, 即代价=失真, 给多少码画面都一样(允许最大压缩). λ趋0则代价趋失真, 即压缩一截下去不会影响多少画质, 稍微给点码率意思意思; λ远大于0则代价>失真, 提升画质的收益>压缩率降低的收益(保画质)

x264还有直接根据噪声容忍度判断失真的算法. 极为先进, libx264, x265都不支持=\_=

高频加权总差方

--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

--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

从像素上看, 量化后的块属其实就是矩阵数据.用游程编码串联成一维上的字串, 再做文本压缩即可节省最高1/3的体积.

## 游程编码

是文本压缩技术. 目的是建立0, 1代表的二叉树. 传统熵编码是使得于原点最近, 出现最多的字放在树的左边, 同时让枝干尽可能短-即编写~解读步骤越少; 出现少则靠右, 让枝干尽可能长-即编写~解读步骤越多. 这样从树根开始, 通过0🡪左, 1🡪右地走到树梢上, 就实现了霍夫曼编码. x264使用了更好的可变长度编码VLC, 以及目前最好的二进制算术编码CABAC, 统称为熵编码, 考虑到文档受众所以放在了x265教程中. 到此, 视频就被压缩的差不多了

最终还要经过解码, 在宏观层面上对比各种压缩模式, 确保收束到质量最好, 同时做到体积最小的率-失真优化计算, 最终将调优过的方案做熵编码, 然后组装为h264视频流输出, 视频的编码就完成了

# 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: 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

--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

# 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, 最适合新手前两个小时 |
| [**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)) |
| **x264 7mod**[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 |
| [**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 |

# 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 [参数] 2>&1 | tee /home/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

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`""

# 处理图像序列

一组连续图像, 格式不限. 自带lavf的x264/5可根据Win/Linux中的序号表达式, 设定输入文件名+%0Nd序号范围实现文件导入, 0代表编号前有空白补位0; N代表数位. Win/Linux的区别仅为路径:

* x264.exe [参数] --fps 30 --output /home/tmp/导出视频.mp4 /home/tmp/图片%02d.png

ffmpeg还可以用image2解码器搭配起始序号start\_number和表达式类型pattern\_type两个参数编解图像序列; 其中sequence还是%0Nd, glob改用通配符\*

* ffmpeg.exe -r 30 -f image2 -start\_number 0 -pattern\_type sequence -i C:\文件夹\图片%02d.jpg "C:\文件夹\导出视频.mp4"
* ffmpeg.exe -r 30 -f image2 -start\_number 0 -pattern\_type glob -i C:\文件夹\图片\*.bmp "C:\文件夹\导出视频.mp4"

Windows上对%0Nd的支持略差, 对图片文件名有要求也不便于跳过, 此时可以用ffmpeg concatenation滤镜的编码方法. 首先创建txt列表:

* #Concat格式: 只能用正斜杠以及单引号, 用#注释, Win/Linux区别仅路径
* #可用gif格式的单图, 可混用图片格式, 分辨率一样就行, pdf没试过
* file 'F:/Asset/2013/IMG\_20130411\_182854.jpg'
* file 'F:/Asset/2013/IMG\_20130412\_125214.png'
* file 'F:/Asset/2013/IMG\_20130412\_125307.tif'
* file 'F:/Asset/2013/IMG\_20130508\_155625.bmp'
* file 'F:/Asset/2013/IMG\_20130508\_155819.dpx'
* file 'F:/Asset/2013/IMG\_20130613\_133840.jpg'
* file 'F:/Asset/2013/IMG\_20130727\_191243.ext'
* file 'F:/Asset/2013/IMG\_20130727\_191255.isa'

帧率未知, 以及不确定能不能跑通时, 就做一个长10帧左右的列表副本来生成一段未压缩视频流排查

* ffmpeg.exe -loglevel 16 -hwaccel auto -y -hide\_banner -f concat -safe 0 -r 帧率 -i "C:\导入图片列表.txt" -video\_size 2560x1440 -an -pix\_fmt yuv420p "C:\导出视频帧列表\_帧率x.yuv"

-f concat以选择滤镜, -safe 0以支持绝对路径(从根目录开始的路径), -r决定帧率, -video\_size淘汰了原本的-s决定分辨率(解决图片格式支持奇数分辨率, 但视频不支持的冲突), -an关闭音频编码, -pix\_fmt选择和图片格式匹配的色彩空间. 一般情况下可以直接用视频播放器打开, 且应正确显示帧率

上方命令跑通, 或确认了帧率后, 就能使用ffmpeg-libx264/5, 从急用版教程里找到对应参数来编码了

* ffmpeg.exe -loglevel 16 -hwaccel auto -y -hide\_banner -f concat -safe 0 -r 15 -i "C:\导入图片列表.txt" -video\_size 2560x1440 -an -pix\_fmt yuv420p -x264-params "rc-lookahead=33: me=umh:bframes=12:b-adapt=2: subme=9:merange=48:fast-pskip=0:direct=auto:weightb=1:keyint=230:min-keyint=3:ref=3: crf=19:qpmin=9:chroma-qp-offset=-3:aq-mode=3:aq-strength=1.1:trellis=2:deblock=0,-1:psy-rd=0.5:0.3:nr=4" "C:\导出视频帧列表\_x264\_帧率15.mp4"
* ffmpeg.exe -loglevel 16 -hwaccel auto -y -hide\_banner -f concat -safe 0 -r 15 -i "C:\导入图片列表.txt" -video\_size 2560x1440 -an -pix\_fmt yuv420p -x265-params "ctu=64:tu-intra-depth=4:tu-inter-depth=4:limit-tu=1:me=star:subme=3:merange=48:weightb=1:ref=3:max-merge=4:open-gop=0:min-keyint=3:keyint=230:fades=1:bframes=8:b-adapt=2:radl=3:constrained-intra=1:b-intra=1:crf=22.8:qpmin=8:crqpoffs=-3:ipratio=1.2:pbratio=1.5:rdoq-level=2:aq-mode=4:aq-strength=1:qg-size=8:rd=3:limit-refs=0:rskip=0:rc-lookahead=33:rect=1:amp=1:psy-rd=2:qp-adaptation-range=3:deblock=0,-1:limit-sao=1:sao-non-deblock=1:selective-sao=3:hash=2:allow-non-conformance=1" "C:\导出视频帧列表\_x265\_帧率15.mp4"

如果要导出图片序列, 则使用ffmpeg的image2解码器来导入, 指定图片格式导出即可:

* ffmpeg.exe -loglevel 16 -hwaccel auto -y -hide\_banner -i 导入视频.avi -r 1 -f image2 -start\_number 0 -pattern\_type sequence -fps\_mode cfr /home/tmp/图片%03d.jpg

# IVTC, 分行转逐行

处理分行扫描源时, 注意帧数fps的参数实际上代表2×fps的场数. 将逐行fps转换到分行的变换处理一般叫pulldown/up. 往回转就是Inverse Telecine, IVTC. IVTC的通用解是找出并删掉重复场, 重组回完整帧. 进一步修复包括去梳De-comb和降噪滤镜来缓解合成出的横线失真. ffmpeg用例如下:

* -vf "fieldmatch=mode=pc\_n\_ub:combmatch=full:blockx=16:blocky=24:combpel=128,yadif" -r 24

## A:B Pulldown / Telecine

胶片放映机按照从上往下的顺序曝光胶卷得到视频帧, 所以"下"就是视频的"后", pulldown就是后延. 逐行帧变成NTSC/PAL电视的分行后即上-下场top-bottom field, 因此每帧占据2场(A=2或B=2). 此时将A或B多延后一场(A=3或B=3), 把后面的TF或BF挤兑一个渲染窗口, 从而让24fps的逐行电影拉伸到PAL 25fps/50场或30fps/60场上播放

### 2:2/2:2:2:2 (Not) pulldown

和游戏机一样没有不存在"一帧占三场"的情况, 根据PAL制式要加速匹配电视, 所以视频略快于音频. 注意音频流被加速(重采样)以匹配视频速度的情况. 如果是游戏机原生渲染的帧, 则找到并关闭游戏的逐行转分行滤镜(顺便节约算力)就是画质最高的"转逐行", 不能重录的话用yadif滤镜, 或画质略好的nnedi转逐行

* -vf "nnedi=weights=C:\[下载\nnedi3\_weights.bin](https://github.com/dubhater/vapoursynth-nnedi3/blob/master/src/nnedi3_weights.bin):field=af:nsize=s48x6:nns=n128:qual=slow:etype=mse:pscrn=new3"
* -vf "yadif=deint=1"

### 3:2/3:2:3:2 pulldown

第1帧拷一上场, 第3帧拷一下场. IVTC过程就是每5帧(10场)删除场5和场10, 然后重组回逐行帧. IVTC可以用上面的fieldmatch滤镜, 也能用x264/AviSynth的Select\_every滤镜手动删除, 然后yadif/nnedi转回逐行, 再标注帧率24即可

### 2:3/2:3:2:3 pulldown

相比3:2 pulldown延后一个场. 见右图第2项

### 3:3:2:2/2:3:3:2/3:2:2:3/2:2:3:3

相当于2:3和3:2的四种排列组合, 比较过分

### 2:2:2:4/2:2:4:2/2:4:2:2/4:2:2:2

这样播放会让延迟更明显, 所以不怎么常见

### 5:5/5:5:5:5 pulldown

从上面的例子显而易见, 就是每帧五场, 但这样播放会大幅提高延迟而很少见, 注意音频流被减速(重采样)以匹配视频速度的情况

### 2:2:2:2:2:2:2:2:2:2:2:3 (Euro) pulldown

24fps电影格式每12帧拷一上场, 每24帧拷一下场, 加快25/24倍得到25fps或50场/秒的PAL视频. IVTC过程就是每25帧(50场)删除场25和场50, 然后重排上下场:

* [ffmpeg 3232 pulldown IVTC] -vf "select='eq(mod(n\,5))'"
* [x264/AviSynth 3232 pulldown IVTC] --vf Select\_every(10,1,2,3,4,6,7,8,9)
* [ffmpeg Euro pulldown IVTC] -vf "select='eq(mod(n\,25))'"
* [x264/AviSynth Euro pulldown IVTC]
* --vf Select\_every(50,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23, 24,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49)

### 源视频用的哪个pulldown?:

用播放器的逐帧查看功能识别. 如果片源里出现了多种pulldown, 可能是NTSC/PAL源在没有恢复到电影24fps逐行前就被二次pulldown了. ~~这是片源方的恶性挑衅~~最好的处理是换源