x264 视频压缩教程完整版

欢迎阅读！本教程面向非专业用途，若有什么不会的可以直接加群 691892901 哦(`・ω・´)ゞ

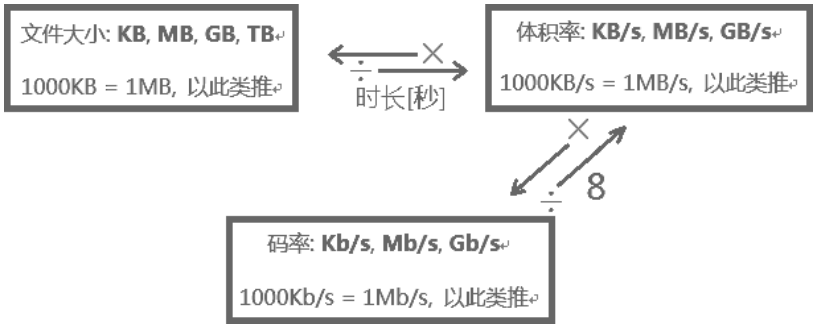
部分 1a: 常识啊常识（｀^´）ノ

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
- 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
- FLV 是针对轻量化封装，更适合网站，电脑，移动端和网站播放，支持切分的格式
- MOV 由苹果公司研发用于兼容自家软硬件，也最适合 Adobe Premiere
- M4A MPEG for Audio 支持多种音频流的封装，实现嵌入封面和详细的艺术家信息
- M3U(8)由苹果公司研发用于在线播放音频的封装(然后被用于视频封装), 8 指 UTF-8 编码

文件，流文件与模拟信号的打包叫 **multiplex/mux** 封装，**demultiplex/demux** 解封装. 常由**编码** encoding，**解码** decoding 操作实现. 编码即压制与转换，解码即播放或解压缩的逆运算. **硬解**代表用低能耗用途少的专用电路嵌入或做成处理器替代**软解**. 视频硬解有 NVDEC, libmxv, OpenCL, MMAL, Direct3D, VDPAU 等方案，**硬件视频编码**有 NVENC, MMF/Venus, QSV, Conexant, Elgato 等缓解录像发热烫手问题的方案. 最常见的编解码方案位于显卡，声卡和网络调制解调器上

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



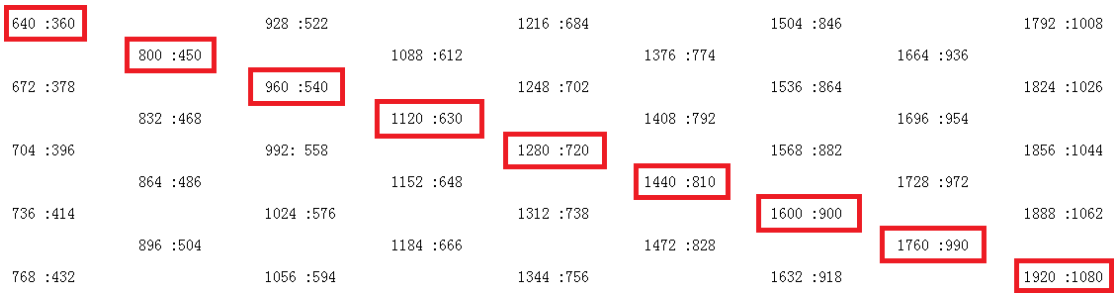
色彩是眼睛捕获特定长度的电磁波，再经脑补而产生的幻觉，实际只是频率不同

滤镜工具 Premiere, After Effects, Audition, ffmpeg, DaVinci Resolve, FL Studio, Reaper, Ableton, Photoshop 等等. AviSynth(+), VapourSynth 严格说是带视频滤镜工具的帧服务器

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（ΦwΦ）



GUIgraphic 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

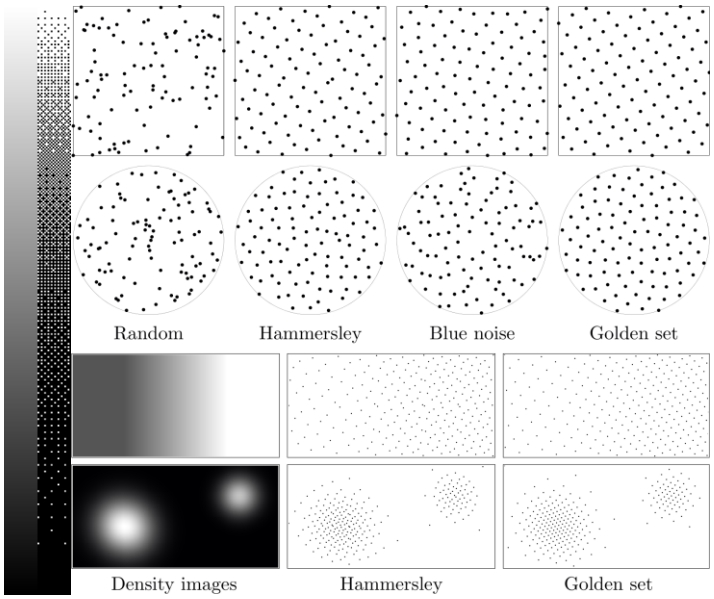
CLIcommand line interface 命令行交互(输命令敲回车)，上手难但越用越方便，缺功能就改用 API

APIapp. 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

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

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

抖动滤镜有随机，蓝噪声，黄金分割，Riemersma 等算法，通过干扰编码器，用分辨率/音频采样率换位深的补偿，因此一般视音频高转低位深时用到，或像奥博拉丁的回归用"抖动即材质"的 ditherpunk 美术. **噪点**是卤化银视频胶片的颗粒 grain，是相机/增益暗场信号/无信号产生的雪花，也是打印喷头-人眼感光细胞分布所致



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
有~无	webp	8 8 8	8 8 8		√		√
有~无	jpg-XR（兼容 jpg）	32 32 32	8 8 8	16 16 16 16	√	√	√
有~无	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	√	√
未压缩	dpx	64 64 64	16 16 16		√	√	

视频		RGB/YUV 色深	编码速度	HDR	透明
有~无	qt, hevc, avc, vvc, vp8/9, DNxHR/HD, prores	12 12 12	慢~快	除 avc, vp8	仅 prores
无损	rawvideo	32 32 32	快		√
无损	cineform	16 16 16	快	√	
有损~无损~未压	flash 动画	里面的图片决定			

音频		音质
有损	mp3, aac, ogg(vorbis/opus)	取决于耳机/音响，混音，声场. 但你要是不知道 HiFi 和监听音频的区别，就无需太关注这个
无损	flac, alac, ape, it	
未压缩	wav	

色彩空间 color space 如下表，或(B, Y 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

RGB → YCbCr → RGB
录制/未压缩 储存/网络发布 播放

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，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 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
4:2:2	亮度不变，色度 960x1080	色素 ← 色素 ← 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)
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 ← ← ← ← 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)
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← 色素 2.667← ↑ 2.848 ↖ ↑ 2.848 ↖	逐行扫描，非方形亮度像素，par 通常是 4: 3 ~~要不然太烧脑子~~

色深 color-depth 代表亮度 Y 的明暗密度．2^{10} 代表 1024，2^8 和子网掩码一样代表 256 个度；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

伽马 gamma/希腊符γ代表亮度 Y 值或信号强度，**伽马曲线**指显示器电压→亮度坐标系中，0~max 电压与亮度 0~100%的曲线关系，以及人眼最为重视低中亮度变化的特性．**伽马矫正**分为消除硬件缺陷，如 CRT 屏要放大电压到$v^{2.2}$才能显示线性灰阶；以及"矫正"RAW 画面的线性亮度于人眼看来是"漂白"两种．sRGB 标准将矫正过程略作"因为显示器默认将画面黑化$\gamma^{1/2.2}$，所以录制时将 RAW 画面白化$\gamma^{2.2}$，显

示器就能线性的显示亮度了". 因此处理 RAW 媒体时, 要开启"按伽马系数混合 RGB 颜色, 伽马值=1 (Adobe 软件默认关)"才能正确显示和滤镜处理画面颜色

逐行扫描 progressive 按左→右及上→下的顺序刷新帧, 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/交错扫描** interlaced 用 4:1:1 采样, 分**上场**: 奇数及**下场**: 偶数像素场(详见教程尾)

帧数是图片数量每秒, 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

视频网站定律指对低流量作者限码, 高流量作者码率优待的国际现象, 所以想要画质得先成名

芯片架构	优	劣
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, 堆核心, 性能提升困难	
Application Specific Integrated Circuit 特定应用集成电路/定制方案(ASIC 架构, TPU)	省电, 快速, 低延迟	功能+1≈设计难度×10
Application Specific Standard Product 特定应用通制方案(SoC, CPU, GPU, NPU)	什么都能往里塞	功耗控制困难, 驱动难写
Field Programmable Gate Array 当场可编程门阵列(FPGA 架构, CPU)	以上什么都能模拟	使用门槛极高+小众

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

编程语言语法	类	优	劣
Fortan, C 语言, Cyclone, Erlang	底层	编译后快	程序员水平≅性能
C#, C++, Rust-Haskell, Java	双层	OOP	
tcl/tk, R 语言, PHP, Ruby, Python, JavaScript, Go, F#, Visual Basic 等	上层	自动指针管理	不适合大型程序
用上层语言调用下层库的语法	折中	以上所有	占硬盘内存
SQL, Bash, CMD, PowerShell, GNU Grub, Cisco IOS, Markup(HTML), CSS, Markdown	论外	能用	离不开环境

数值格式是计算机标准或开发系统自定的变量定义. 种类多是缓解曾经内存极其昂贵问题所用

# 格式说明		位从零数最小最大
Bool	True/False, 由于占二进制一位所以也分别代表 1 和 0, 故用于开关	1 0, 1
Byte	ASCII 字符位, 共 1~256 个符号, 顺便可表示数字大小以省内存	8 0~255
SByte	将二进制下首位改为表示正负符号 Sign, 正整数范围减半	8 -127~127

Integer	整数，简称 int，小数点前 1 位/个位. 不同语境会分别判定长/短整数为 int		
Short	短整数，简称 int 或 int16	16	-32768~32767
Uint	去正负号(Unsigned)的短(或长)整数，一般为短整数(Unsigned short)	16	0~65535
Long	长整数，简称 int 或 int32，调用较慢	32	-2^{31}~2^{31}-1
Long64	长整数，简称 int 或 int64，储存的整数数量最大，调用慢	64	-2^{63}~2^{63}-1
Float	单精度浮点，精确到小数点后 7 位，同时只用了 32 位，调用较慢	32	±3.4×10^{38}
Double	双精度浮点，精确到小数点后 15 位，适用于低误差科学计算，若数据量大则加内存	64	±1.7×10^{308}
Decimal	C#/.Net 中财务计算用，牺牲整数位数保小数位数到 29 位而省算力保精度	96	±7.9228×10^{28}
NaN	非数值 Not a Number 的缩写，判定用户写错了或者写对了		

食用方法，注意事项

教程中的命令行参数说明写法:

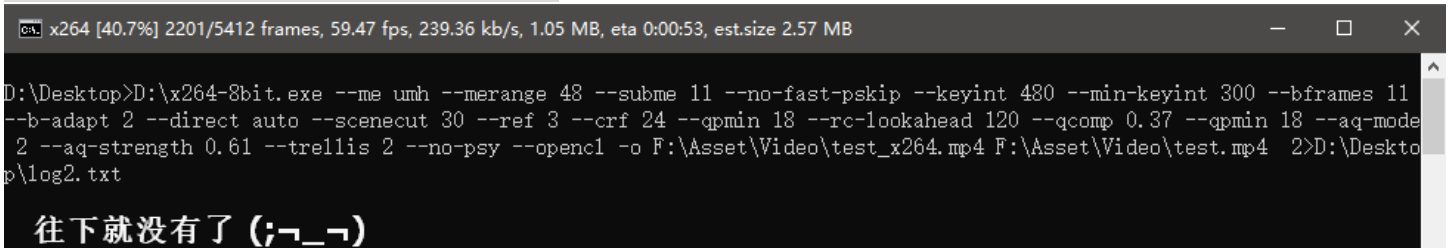
--参数名<开关|整数 A~B|浮点 A~B|其它格式， 默认值， 限制>说明信息

- 细化说明 N，或参数值为 N 时细化说明或适用范围
- 格式为"开关-关"时 在 CLI，API 格式中皆不写，用于保持默认，或写对应否定参数"--no-参"或"no-参=1"
- 格式为"整数-略"时 在 CLI，API 格式中皆不写，用于保持默认，或写对应否定参数"--no-参"或"no-参=1"
- 格式为"开关-开"时 在 CLI 中填"--开关"，API 填"开关=1"
- 格式为"整数-写"时 在 CLI 中填"--参数 值"，API 填"参数=值"
- 串连多个参数时 在 CLI 中填"--参数 值 --开关 --参数 值"，API 中填"参数=值,开关=1,参数=值"

命令行参数 Command prompt 一般用法

```
[引用程序]          C:\文件夹\x264.exe
[CLI 参数]          --me esa --merange 48 --keyint 100 [其它参数]
[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 [参数] --output "导出.mp4" "导入.mp4"
```

Win-CMD/Linux-bash 输出 log 日志

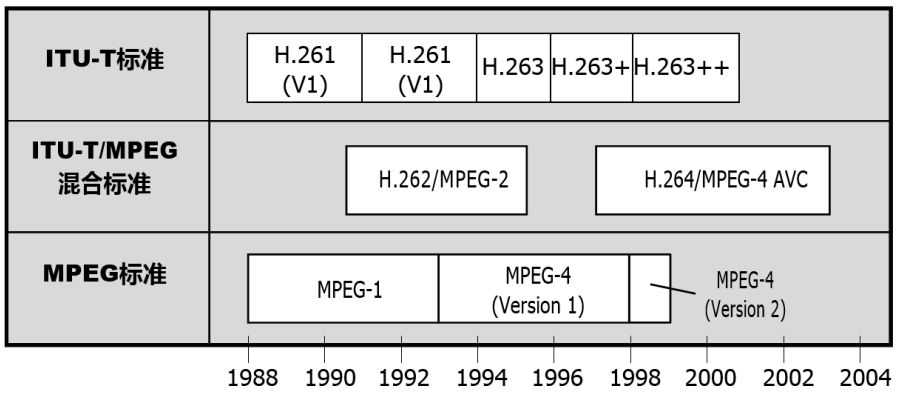


- Windows CMD: x264.exe[参数] 2>C: \文件夹\日志.txt [参数还可以写在右边]
- Linux Bash(或其它): x264 [参数] 2>&1 | tee /home/文件夹/日志.txt

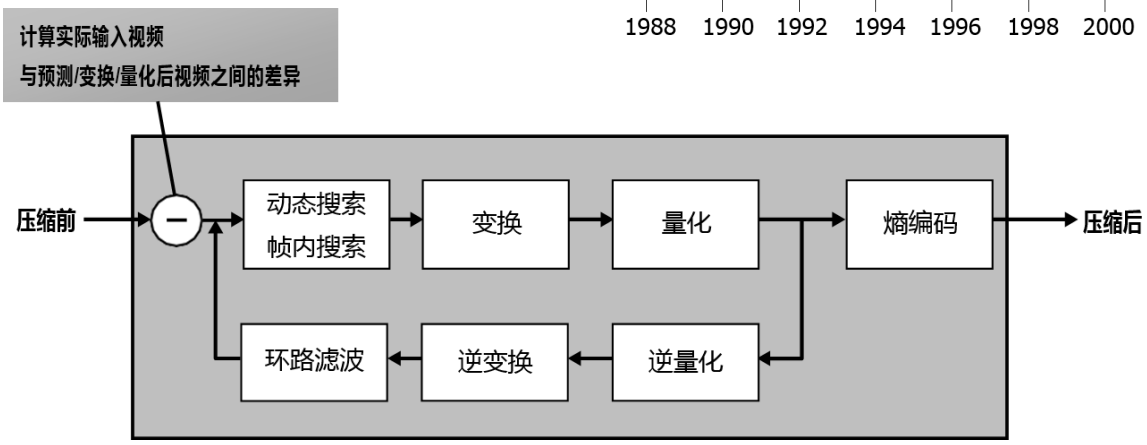
压制软件工作流程概述

正片 - x264 的原理及数据类型丶(; ▽;)ノ

AVC 标准下, h264/x264 编码器给出的文件由大量的图组 Group of Pictures - GOP 堆砌而成, 每帧从宏块开始细化到最小 4x4 的块



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. 在距离 IDR 帧过进又因画面变化触发转场时节省设立 IDR 帧的算力
- IDR 帧创建进度条落点 access point. 使得左右割为独立的子视频, 不再相互参考, 降压缩率将编解码难度
- 将视频分割成独立 GOP 限制了储存或传输介质损坏波及视频的程度, 因为压缩掉的冗余信息会指向参考源, 否则一旦参考源损坏, 或者网络串流丢包就会"烂一大片"
- IDR/I 帧质量为 P 帧的 1.4 倍, 即参数 ipratio 1.4, 约为-3qp 的量化值偏移 offset

参考帧: 有 IDR, I, P, Pyramid-B, 以及 B 帧五种. 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
- P 帧质量为 B 帧的 1.3 倍, 即参数 pbratio 1.3, 约为-2qp 的偏移

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

--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 3 或 1%×fps+3.4>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，所以推荐 3

--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

--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"

初始化与 Lookahead

前瞻与分块等进程最早启动．设定帧类型(关键帧，参考帧)，间接影响了 mbtree, VBV，决定了 ABR，CRF 模式的初始数据．x265 教程中展开 SPS/PPS 的信息．流程如下:

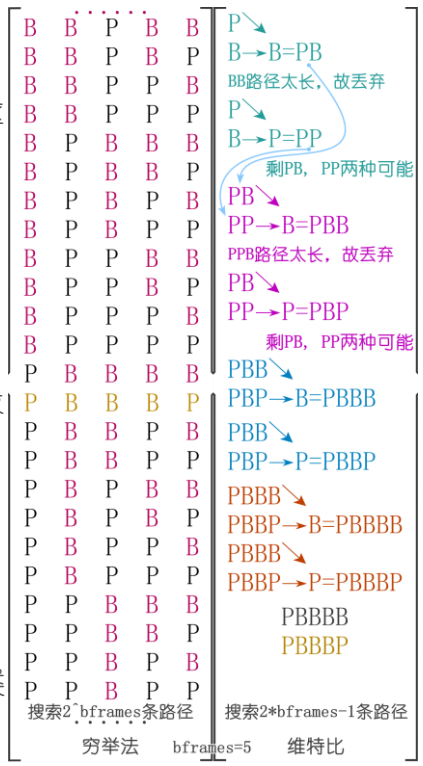
1. 从--seek 参数指定处开始，将视频帧导入 Lookahead 进程，顺序和进度条一样从左往右
2. 后续步骤发现任何 GOP 结尾时，当前的 h->Lookahead->next->list 帧列表就后移相应的帧数
3. **转场判断①**：当输入的第 0 帧是 AUTO/I 帧，且与之前/左侧的非 B 帧差距达到--scenecut 阈值-触发转场时设 IDR
   - 设 IDR(或受--open-gop 限制而设 I 帧)后，其前一帧强行设为 P 帧，min-GOP 遂变 GOP，Lookahead 列表后移
4. **P/B 区分**：据--b-adapt 指定的算法，--bframes 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
5. 单 B 帧/连续 B 帧与其后的 P 帧视为 min-GOP
6. 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
7. **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
8. 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
9. 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

--rc-lookahead<帧数量，范围 1~250，推荐 keyint÷2>指定 mbtree 的检索帧数，常设帧率的 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

图：bframes=5，每层迭代保留 B/P 枝梢，每枝梢反推一次最短根路径，最后 P 帧收尾，得到累计距离最短(即总和帧大小最小)而获最小码率方案

Viterbi-SPF 是多起点-多终点的最短路径 shortest path 算法. 与 GPS 的 AStar, Dijkstra 不同, 推演 B/P 帧是多起点多终点最短的隐马尔科夫模型

--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

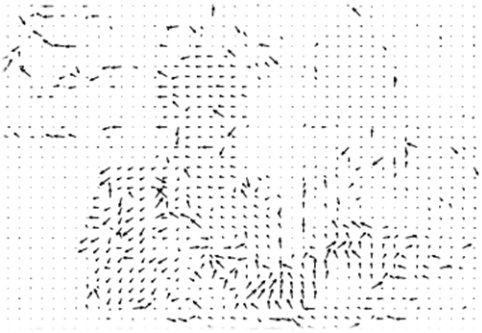
--vbv-bufsize<整数 kbps, 默认关=0, 小于 maxrate>编码器解出原画后, 最多可占的缓存每秒.

bufsize÷maxrate=编码/播放时解出每 GOP 原画的缓冲用时(秒). 相当于要求每个编码 GOP 的平均大小. 编码器用到是因为模式决策需要解码出各个压缩步骤内容与原画作对比

--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

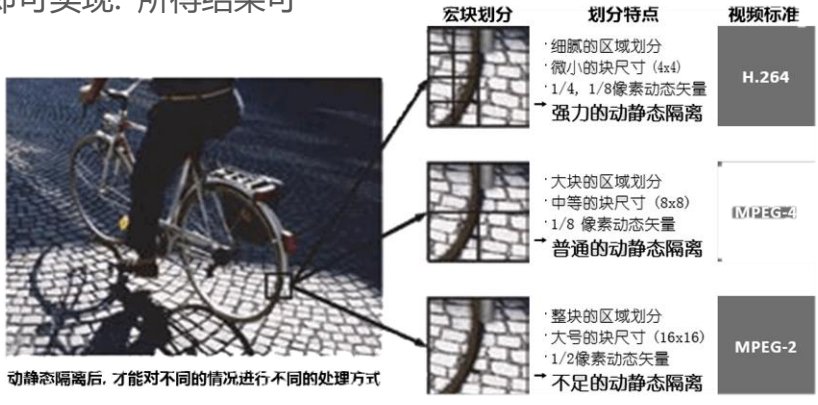
- 大:GOP 大小=bufsize=2×maxrate, 码率超+缓存满后压缩, 避多数码率涨落, 适合限均码率串流
- 小:GOP 大小=bufsize=1×maxrate, 码率超限后压缩, 避部分码率涨落, 适合限峰值码率串流
- 超:GOP 大小<bufsize=1~2×maxrate,超码超限后压缩, 但因视频小/crf 大所以没啥作用
- 欠:GOP 大小>bufsize=1~2×maxrate,码率超限后压缩, 但因视频大/crf 小所以全都糊掉
- 由于 gop 多样, 4 种状态常会出现在同一视频中. buf/max 实际控制了这些状态的出现概率

--ratetol<浮点, 百分比, 默认 1>ABR/2pass/VBR 的码率超限容错 tolerance, 参数值代表允许误差码率对比 maxrate 设定的超限%



动态搜索

查找帧间的移动变化信息, 以冗余掉帧间重复块. 剪辑软件中偏移 1~5 帧重叠两层所同视频, 用相减混合模式即可实现. 所得结果可以用减淡-添加混合模式作动态强化层用



--me<dia, hex, umh, esa, 推荐 umh>算法从左到右依次复杂化, umh 之前会漏掉信息, 之后收益递减

--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"，反之是"欠算性"

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

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

$$\text{绝对(变换后)差异和 } \mathrm{SA(T)D} = \sum x_{0\to Tx} \sum y_{0\to Ty} |\mathrm{f(x,y)} - \mathrm{f'(x,y)}|$$

- ∑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

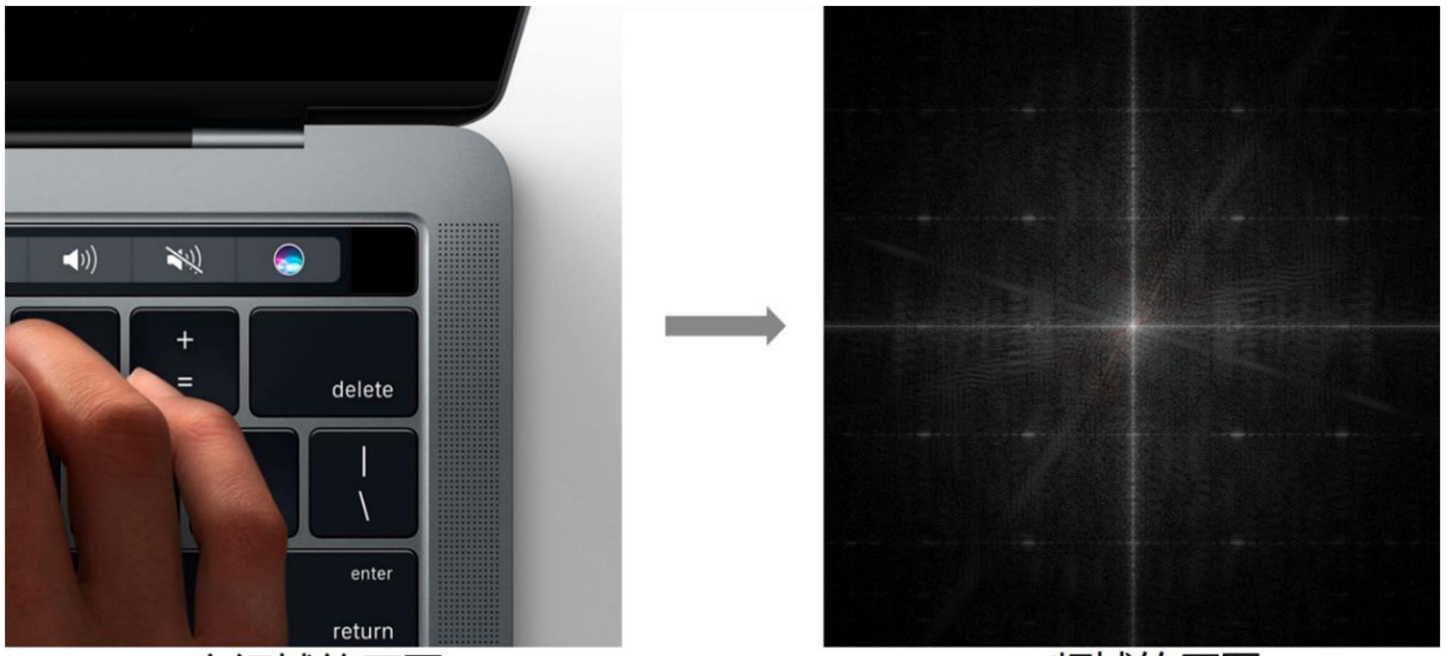
--subme<x264 中影响模式决策和率失真优化．整数范围 0~10，60fps=9，-=8，+=10>根据片源的帧率判断．分四个范围．由于动漫片源制于 24~30fps，因此可节省一些算力；但同是动漫源的 60fps 虚拟主播则异．主流 120Hz 的手机录屏目前最高也不够用．由于性能损耗大，所以不建议一直开满

- <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

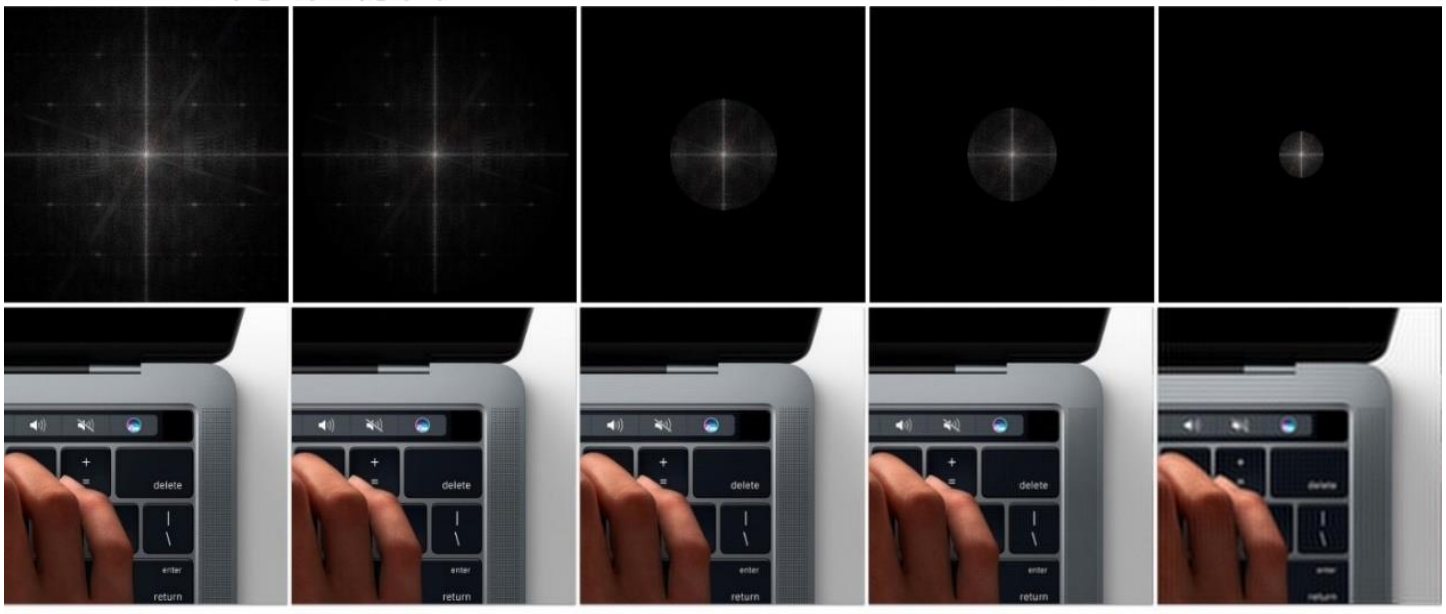
加权预测 weighted prediction / 帧内编码: x265 教程中展开

--weightb<开关，默认关，推荐开>启用 B 条带的显隐加权预测，提高全屏明暗变化的编码质量

变换-量化: 将频率从低到高的信息列出来，用低通滤镜去掉高频信息实现压缩. x265 教程中展开

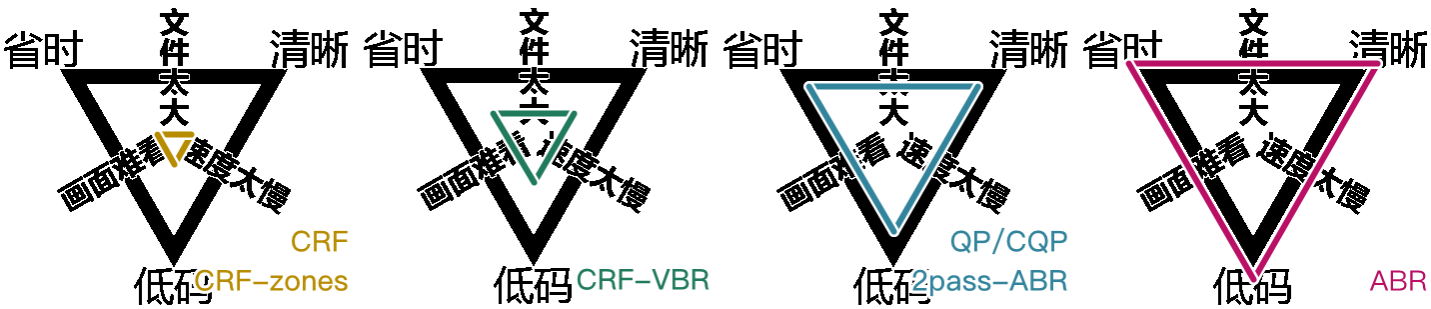


空间域的原图　　频域的原图



100%　69%　30%　11%　2%

码率控制模式



据 CPU 算力，片源与压制的目的而妥协省时，清晰或低码. 简单说一般默认 crf，直播算力够用则 crf 或 vbr，直播节省算力用 abr，码率限制下用 2pass-abr，测试用 cqp

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

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

--crf<浮点 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

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

--qp<整数 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

--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，抠图要物件边缘清晰时设--qpmax 16 以保护录屏时物件的边缘，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

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

- 多分区用'/'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 总结可压缩信息，再用 abr 的码率限制统一分配量化值. 除非有码率硬限，否则用 crf 模式

```
        --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<文件导入>手动指定特定帧类型；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

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区分高低两种频率，实现细化到块上的量化值 qp. 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做差，以偏差之和窥数据之衡．乘方是为方便正负数一起求和计算(算式左的σ也乘方，优化了正负号在二进制中的额外占用)：$\sigma^2 = \frac{\sum_{i=1}^{N}(x_i-\bar{x})^2}{N}$

VAQ 中，样本即宏块内像素，所比的平均值为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，就实现 vaq 了

$$Variance = \frac{SSE_{pixels\ within\ Marcoblock} - (SAD_{Marcoblock})^2}{256}$$

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+0.1~0.2，画面全是纹理时可再加．注意低成本动漫的平面多过纹理，因此码率不足时反而要妥协纹理

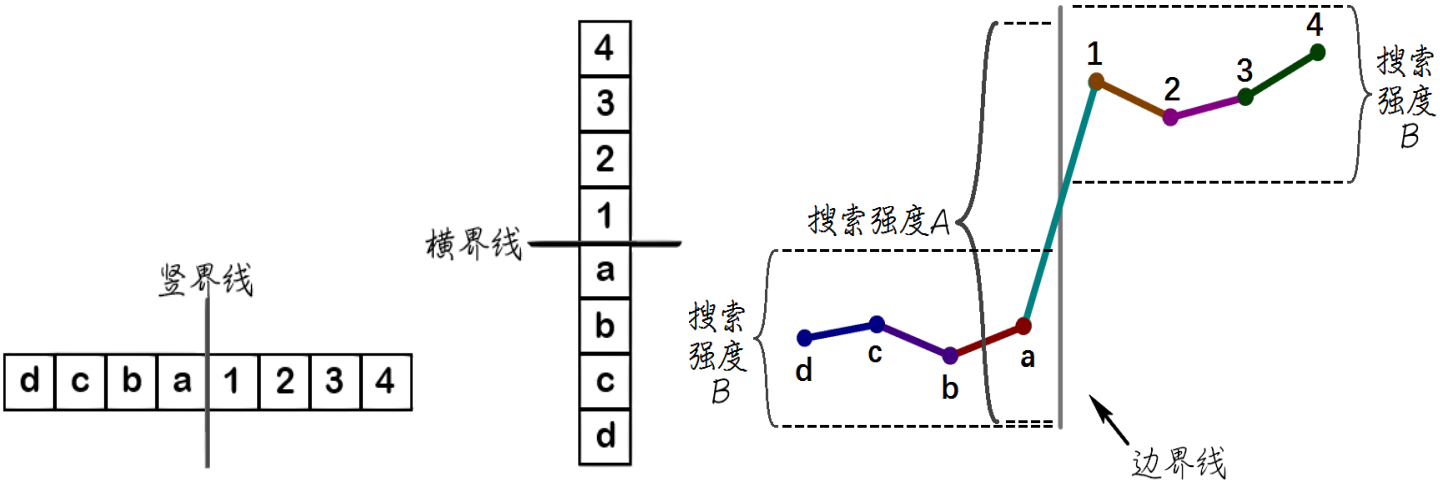
环路滤波

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>

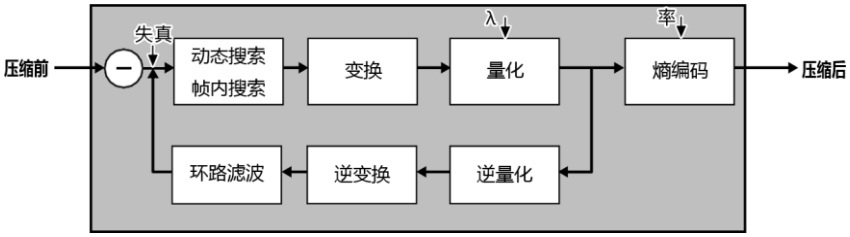
模式决策

mode decision (MD)整合动态搜索所得信息, 宏观定制分块参考量化等细分方案. 因为选码率最小的压缩方案不平衡, 画质容易崩坏. 注意片源含明显边缘失真时反而要减少决策优化

--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

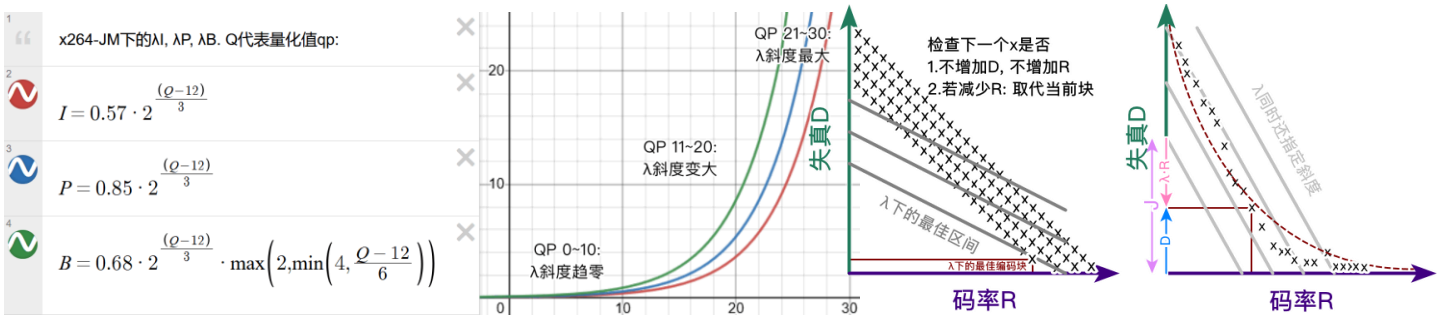
率失真优化 RDO 控制



心理视觉 phycological visual 是对人眼感知清晰度的研究.

率失真优化 rate distortion optimization 穷举测算失真程度(编码前后差异)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; $J = D + \lambda \cdot R$)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

x264 差异平方和 Sum of Squared Error = $\sum x_{0\to Tx} \sum y_{0\to Ty} |f(x,y) - f'(x,y)|^2$



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. 但 libx264, x265 都不支持

高频加权总差方 $NSSE = \sum x_{0\to Tx} \sum y_{0\to Ty} |[N(x,y) - N'(x,y)] \cdot fgo| + |f(x,y) - f'(x,y)|^2$

--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

--psy-rd<a:b 浮点，默认 1:0>心理学优化设置. a 保留纹理，b 在 a 的基础上保留噪点细节. ab 值据画面复杂度拉高. 动漫范围<0.4~.6:0.1~.15>，no-mbtree 时可设 b 为 0; 录像<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

Quantizer curve compression (qcomp)

源自 libavcodec 的量化值曲线缩放：复杂动态场景的帧间参考少而应提高压缩，分码率给参考多而远的简单动态场景，实现同码率下整体画质更高的结果；缺点显而易见：动漫场景等多为背景不动前景动的场景会被误压缩．但说到底这只是处理精度太低而非错误，只要逐宏块地应用这套逻辑就可以了．于是就有了 mbtree. qcomp 则被贬为控制 mbtree 等一系列 qp 分配的缩放，见 desmos 互动例

1. (ABR 模式)根据宏块量与 qcomp 设置初始复杂度百分比：$complexity = (mb_{count} \div 2) \times 700000^{qcomp} \div 100$
2. 获取当前场景的长度及帧间绝对变换差之和 SATD，合成为图像
3. 模糊 SATD 图像以得到模糊复杂度 blurred complexity
4. (关 mbtree)$q = blurred_complexity^{1-qcomp}$转换到质量衡 q，以换算出(ABR 或其它上层模式下的)单帧 qp
5. (开 mbtree)$q = (base_{frame_{duration}} \div clip_{duration} \times i_{duration} \div fps)^{1-qcomp}$据 I 帧在当前片段的权重设单帧 qp
   注：基准单帧用时 ÷ 当前场景用时 × I 帧用时 ÷ 帧率

Macroblock-Tree (mbtree)

强度 stength，帧内参考模式成本 intra_cost，帧间参考模式成本 inter_cost，当前块传播成本 propagate_in，传播成本 propagate_cost 五个参数．若一宏块作帧间参考码率比帧内参考下低 70%，则该块可视做 30%无帧间参考+70%帧间参考两部分.详见 x264 率控制算法及 mbtree_paper

1. 帧内参考成本，帧间参考成本：尝试用于帧间和帧内两种模式编码宏块，累计 SATD(误差)得到
2. 帧间参考成本只能<=帧内参考成本，若帧间参考成本更大则设=帧内参考成本，类似于 P 帧码率不可大于 I 帧
3. **传播成本比率**　$propagate_{fraction} = 1 - (intra_{cost} \div inter_{cost})$
4. **宏块信息成本和**　$intra_{cost} + propagate_{in}$，包括 lookahead 范围内所有溯该块的宏块
5. **传播信息量**　$propagate_{amount} = propagate_{fraction} \times (intra_{cost} + propagate_{in})$，即估计传播到下游溯块宏块的信息量=信息成本和 × 传播成本比率
6. 传播信息量需根据传播到下游不同宏块的像素总数(如块的大小)而删减调整，如因为动态变化导致参考宏块的范围平移到被下游共 4 个宏块共享，此时每个宏块各会参考自身所占的部分

- 对于双向参考的 B 帧，传播信息则一分为二，分别置于连续 B 帧的两端，再传播给 B 帧从而保证压缩率
- 对于加权预测 B 帧，传播信息则不等分，而是权重多的 B 帧的一端参考帧分到的信息量更多

7. **传播成本**　$propagate_{cost}$lookahead 范围从远到近的溯块宏块累计传播信息量$propagate_{amount}$信息越多成本/码率越高，距离越近成本/码率越高
8. **宏块树强度**　$strength = 5.0^{1-qcomp}$qcomp 控制 mbtree 分配 qp 的强度
9. **宏块 qp 偏移**　$\triangle QP = -strength \times log2((intra_{cost} + propagate_{cost}) \div intra_{cost})$ 最后根据(每宏块自身信息成本+传播成本)÷帧内模式成本，缩放到 qp 值范围，qcomp 调整强度，给每个宏块分配宏块树 qp 偏移 offset

- 大部分情况下 mbtree 偏移值为零，因为宏块无溯块信息可用

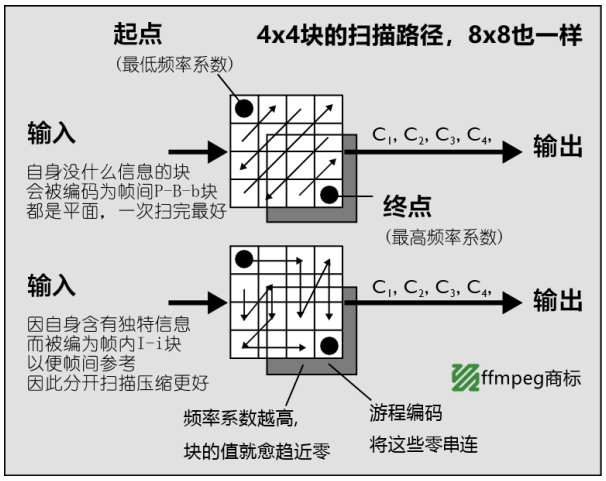
--qcomp<浮点 0~1，默认 0.6>blurred_complexity 迭代值每次能迭代范围的曲线缩放，更改 ABR 模式的初始质量判断，限制 mbtree 偏移 qp 强度的三用，x264 会自动根据 aq-strength 增加 qcomp

- <小于 0.5>中~强 mbtree；CRF，ABR 低延迟逐帧迭代 qp；画面主前景动时用，允许 mbtree 导致零星宏块欠码

- <0.5~0.7>中 mbtree；CRF，ABR 中延迟逐帧迭代 qp，画面含背景动，或混合情况用，平衡优先
- <大于 0.7>中~弱 mbtree；CRF，ABR 中~高延迟逐帧迭代 qp，保留重噪点，或 FPS/STG 游戏录屏场景用
- <关 mbtree，小于 0.5>画面不分前背景，如静态图像，PPT/桌面录屏节约性能用
- <关 mbtree，大于 0.5>动态画面，不分前背景时节约性能用
- <0>(触发 if 判定)启用固定码率模式
- <1>(触发 if 判定)启用固定 qp 模式

--no-mbtree<开关，不推荐>逐宏块分配前景变背景不变画面的码率(如动漫，录播)

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. 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)

```
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.
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
                  White point: x=0.312700 y=0.329000
```

位深: 10 位
数据密度【码率/(像素×帧率)】: 0.251
流大小: 41.0 GiB (90%)
编码函数库: ATEME Titan File 3.8.3 (4.8.3.0)
Default: 是
Forced: 否
色彩范围: Limited
基色: BT.2020
传输特质: PQ
矩阵系数: BT.2020 non-constant
控制显示基色: Display P3
控制显示亮度: min: 0.0050 cd/m2, max: 4000 cd/m2
最大内容亮度等级: 1655 cd/m2
最大帧平均亮度等级: 117 cd/m2

--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

```
[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
 Timebase    : 1000/15360000
 Duration    : 0:01:30
ffms [info]: 1920x1080p 0:1 @ 60/1 fps (vfr)
ffms [info]: color matrix: bt709
```
视频流信息

```
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
.....
[99.4%] 5378/5412 frames, 44.60 fps, 271.22 kb/s, 2.90 MB, eta 0:00:00, est.size 2.92 MB
```
压制信息

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

--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

--sliced-threads<开关，默认关，仅建议录屏画面卡顿时用>x264 r607 版本从默认"每核心分别处理一条带"变成"一帧"，r1364 后条带线程模式作为低延迟模式的可选开关重新引入．开启后导致

1. 熵编码中，CABAC(见 x265 教程)的模具因为内容不连续所以要经常重设，降低压缩率
2. 动态信息的动态矢量长度被限制到条带分片内部，干扰动态补偿，降低压缩率
3. 帧内冗余降级为条带内冗余，降低帧内编码压缩率
4. 增加了相当一部分(多余条带与 NAL 封包的)文件头
5. 大量条带的并行提高了编码单帧的速度，降低延迟

--slices<整数，开 sliced-threads，默认自动>并行条带/线程数量

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:←,↑,→,↓/pad:←,↑,→,↓/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，用于快速预览结果，如:

1. 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
2. 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: --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(((￣へ￣井)

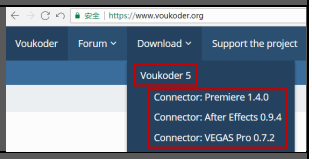
mpv 播放器 比 Potplayer 好在没有音频滤镜, 不用手动关; 没有颜色偏差, 文件体积小

ffmpeg(全系统): 备用地址 ottverse.com/ffmpeg-builds

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, 需自行封装音频. 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，最适合新手前两个小时
Voukoder; V-Connector 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 
OBS 直播与录屏 支持 AVS 滤镜，设置复杂但强大，去 x264 教程急用版照着设置就行了
x264 by Patman, LigH√lavf 编解码，合并 8~10bit
x264 tMod by jspdr√lavf 编解码，支持 MCF 线程管理库(比 posix 和 win32 性能更好)
x264 7mod 谷歌盘/百度云 √lavf 编解码，√hqdn3d 降噪

去可变帧率

片源去 VFR: 防止录像抽干手机电池过快用，但编辑前需转换. 高端机建议录制恒定帧率

Pipe:传递一般数据的 pipe 只要|， 如 md5sum→awk→sort: `for txt in *.txt; do md5sum $txt; done | awk '{print $2, $1}' | sort -bn`; 传递文件则使用- | -，首个-代表输出，第二个-代表输入. 而- | - | - | -即同文件串联多个程序，但因调度复杂所以一般不用，如下:

```
(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 -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， y4m 是 yuv for mpeg

其它

Potplayer 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~~，推荐换 mpv 播放器

CMD 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，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"

.bat 脚本技巧 命令最后换行写上 cmd /k 可以保持 CMD 输入窗口，或 pause

.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 的 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，因此还是得靠 CMD 实现，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

一组格式不限的连续图像. 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 指编号前有空白补位 0; N 即位数. Win/Linux 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 还可用 image2 解码器搭配起始序号 start_number 和表达式类型 pattern_type 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 F:\图%02d.jpg "F:\视.mp4"`
- `ffmpeg.exe -r 30 -f image2 -start_number 0 -pattern_type glob -i F:\片*.bmp "F:\频.mp4"`

Win 系统支持%0Nd 略差，批量改文件名也有风险，此时可用 ffmpeg concat 滤镜. 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:/2013/IMG_20130411_182854.jpg'
- file 'F:/2013/IMG_20130412_125214.png'
- file 'F:/2013/IMG_20130412_125307.tif'
- file 'F:/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 选择滤镜，-safe 0 支持绝对路径(从根目录开始)，-r 帧率，-video_size 决定分辨率(大多图片格式支持奇数分辨率，而视频不支持)，-an 关音频编码，-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"

若导出图片序列，则用 ffmpeg-image2 解码器导入，指定图片格式导出:

- 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

反交错转逐行与 IVTC

播放/检查分行交错扫描源的属性时，所谓的帧数 fps 值其实是 2×fps 的场数 fields per second. 逐行→交错视频的变换统称叫 Interlacing. 反则 Deinterlacing. 进一步修复有去梳 Decomb 和降噪来后

处理剩余的梳状横纹 Combing 失真. 一般用自动滤镜组解决, 或播放器逐帧查看手动辨别(如 mpv 逗句号键), 找出原生交错/pulldown 电影/VFR/PIP/适应蓝光标准的假 pulldown 等情况. 若源经由传媒组织多级剪辑, 则常存在编程意义上的"屎山", 即以上所有情况复数出现的视频

Pulldown 直译为向后拉伸, 因为胶片放映机从上往下曝光胶卷, 即「下等于后」. 反则 pullup

A:B Pulldown 以电影 24fps 标准逐行→分行→上或下场重复出一定的「一帧三场」以拉伸场率到 NTSC 60fps/PAL 50fps 场率, 理论上有无限种实现. 又名 telecine

Inverse Telecine (IVTC) 由于当今显示设备皆逐行, 所以老影片恢复到「一帧二场」. 注意原生录制的交错源转逐行不全算 IVTC, 因为帧率应保持原样而不恢复到电影标准 24fps

2:2:2:4

A01	A02	1
B03	B04	2
C05	C06	3
D07	D08	4
D07	D08	5

2:2:2:4/2:2:4:2/2:4:2:2/4:2:2:2(24d) 「一帧四场」卡顿而不常见

3:2/3:2:3:2 pulldown(24t) 第一帧拷上场, 第三帧拷下场. IVTC 即每 10 场删除场 3, 8(从零数 2, 7)

2:3/2:3:2:3 pulldown 相比 3:2 pulldown 延后到 5, 10 场(从零数设 4, 9)

TF | BF Φ

3:2:3:2

A01	A02	1
A01	B04	2
B03	C06	3
C05	C06	4
D07	D08	5

3:3:2:2

A01	A02	1
A01	B04	2
B03	B04	3
C05	C06	4
D07	D08	5

2:2:3:3

A01	A02	1
B03	B04	2
C05	C06	3
C05	D08	4
D07	D08	5

3:2:2:3

A01	A02	1
A01	B04	2
B03	C06	3
B03	C06	4
D07	D08	5

2:3:2:3

A01	A02	1
B03	B04	2
B03	C06	3
C05	D08	4
D07	D08	5

2:3:3:2

A01	A02	1
B03	B04	2
B03	C06	3
C05	C06	4
D07	D08	5

5:5/5:5:5:5 pulldown 「一帧五场」注意音频流被减速二压以匹配视频的可能

0.5:0.5 (not) pulldown(60i, 30i) 「一帧半场/一场占一帧」原生录制的交错源

A01		1
	B04	2
C05		3
	D08	4

TF | BF Φ

A01	A02	1
B03	B04	2
C05	C06	3
D07	D08	4
E09	E10	5
F11	F12	6
G13	G14	7
H15	H16	8
I17	I18	9
J19	J20	10
K21	K22	11
L23	L24	12
L23	M26	13
M25	N28	14
N27	O30	15
O29	P32	16
P31	Q34	17
Q33	R35	18
R35	S38	19
S37	T40	20
T39	U42	21
U41	V44	22
V43	W46	23
W45	X48	24
X47	X48	25

2:2:2:2:2:2:2:2:2:2:3 (Euro) 电影每 12 帧拷一上场, 每 24 帧拷一下场, 加快 25/24x 得 50 场/秒的 PAL 视频. IVTC 即每 50 场删除场 23, 48(从零数设 22, 47)

2:2/2:2:2:2 (not) pulldown 据 PAL 制式直接播放的视频会略快于音频. 所以要注意音频流被加速二压以匹配视频的可能. 若是游戏机原生渲染, 则找到并关闭逐行转分行滤镜就是完美的"转逐行", 滤镜可能位于游戏机软硬件, 也可能要反编译游戏以从内部关掉

PIP pulldowns 画中画 picture in picture 的特效/前景背景 telecine 所异的情况

VFR pulldowns 多种分行视频源被拼接到一起所产生的"可变帧率"危险品

去交错的算法与滤镜

- Half-sizing 上下场各作半高的逐行视频播放，保原生交错 0.5:0.5(not)pulldown 源的帧率但纵向分辨率减半
- Weaving 上下场简单拼成一帧，处理原生交错 0.5:0.5(not)pulldown 源会导致时域信息丢失及梳状横纹 combing
- Blending 上下帧位置上对应的场在播放时临时混成一帧，由于混的帧与另半场不匹配所以导致鬼影 ghosting
- Bobbing 在 Half-sizing 的基础上把图拉回原高，搭配好的插值/补帧滤镜能产出非常好的结果
- EDI：边缘定向插值 Edge-directed interpolation，断线内两端定向连起而插值，差在仅处理空间域而略时域

滤镜	简介	√
fieldmatch	IVTC 的关键滤镜，基本是 AviSynth 场匹配脚本的 TIVTC/TFM. 少另半或处理后仍 combed 的场则糊上并留标记给下列插值滤镜修补	支持 ffmpeg, AviSynth(+) VapourSynth
decimate	删除重复帧的简单滤镜，严格说唯独 fieldmatch+decimate=IVTC，原生交错的源不存在 pulldown 拉出的重复场所以单纯反交错时不加	
yadif	同时参考时域空域临近像素的经典插值滤镜，还有 yadif_cuda	
bwdif	基于 yadif、w3fdif，强在帧内+帧间参考以及_cuda，_vulkan 的 GPU 加速版	
nnedi3	神经网络插值的 bobbing-edi 滤镜，但因 EDI 特性而只参考帧内，动态复杂时可能在帧间连贯上劣于 bwdif	
QTGMC	调用 nnedi3 等多种反交错滤镜+mvtools 动态补偿反交错脚本，SuperFast 及更慢下用 nnedi3，Slower 及更慢下做色度动态搜索，设置 EZKeepGrain 以(据个人喜好)留噪	AviSynth(+) VapourSynth
ivtc_txt60mc 例	用于上述的 PIP 情况，如 24t/24d, 60i/30p 两两混合等需手动测量并标记的段落	

ffmpeg IVTC 用例

fieldmatch 整理并标记场，yadif 不用-deint=all 而用-deint=interlaced 以跟踪 fieldmatch 标记

- -vf "yadif=deint=all"
- -vf "fieldmatch=mode=pc_n_ub:combmatch=full,yadif=deint=interlaced,decimate" -vsync 0 -r 24
- p/c/n 代表尝试匹配当前场另一半的前一 previous/同一 current/后一 next 半场，否则尝试 u/b：反从另半场尝试匹配 uəxʇ 和 bɹəʌıons，建议看文档

原生交错的场率=帧率故不用-r，bwdif 不用-deint=all 而用-deint=interlaced 以跟踪 fieldmatch 标记，用 send_field 设定场率=帧率：

- -vf "fieldmatch=combmatch=full,bwdif=mode=send_field:parity=auto:deint=interlaced" -vsync 0

pulldown 下，用 send_frameIVTC，帧率从 NTSC 恢复到电影标准而用-r 24，decimate 删多余场：

- -vf "fieldmatch=mode=pc_n_ub:combmatch=full,
  bwdif=mode=send_frame:parity=auto:deint=interlaced,decimate" -vsync 0 -r 24
- -vf "fieldmatch=mode=pc_n_ub:combmatch=full:blockx=16:blocky=24:combpel=128,nnedi=weights=C:\点击下载训练数据\nnedi3_weights.bin:field=af:nns=n32:qual=slow:etype=mse,decimate" -vsync 0 -r 24

AviSynth 的手工 VFR IVTC（困难，仅展示）

SeparateFields 换帧为场，隔四场删一场做 3:2 或 2:3 pullup, waving 拼合即 IVTC. 因片源经剪辑而用 Trim 分段补偿 pullup 顺序，顺便展示 124546 帧至片尾后展示 Euro pullup 示例，以此类推:

- `Trim(0, 579).SeparateFields().SelectEvery(10, 1,2,3,4, 6,7,8,9).Weave() +\`
- `Trim(580, 51891).SeparateFields().SelectEvery(10,0,1,2,3, 5,6,7,8 ).Weave() +\`
- `Trim(51892, 70278).SeparateFields().SelectEvery(10,0,1,2, 4,5,6,7, 9).Weave() +\`
- `Trim(70279, 112304).SeparateFields().SelectEvery(10,0,1, 3,4,5,6, 8,9).Weave() +\`
- `Trim(112305,124545).SeparateFields().SelectEvery(10,0, 2,3,4,5, 7,8,9).Weave() +\`
- `Trim(124546, 0).SeparateFields().SelectEvery(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).Weave()`

VapourSynth IVTC 脚本（演示 QTGMC 调用）：

- 关于 VapoutSynth 基础用法，见 VCB 基础[5]教程

```
import vapoursynth
from vapoursynth import core
import havsfunc as haf
import mvsfunc as mvf

# 使用 L-SMASH-Works 导入源
src = core.lsmas.LWLibavSource("<源>")

# 指定输入源，上场优先(1)，p/c+n+u/b 模式匹配(3)
match = core.vivtc.VFM(src, 1, mode=3)

# QTGMC 反交错，指定输入已匹配场的源，上场优先，帧率=0.5 场率，低速预设，边缘预填充防抖
deint = haf.QTGMC(match, TFF=True, FPSDivisor=2, Preset="Very Slow", Border=True)

# 对 VFM 选出含_Combed 标记的帧，替换为 QTGMC 反交错过的帧.
# FilterCombed 清除所有_Combed 标记，而后处理源(post-processed_clip)记做非交错源.
pp_clip = mvf.FilterCombed(match, deint)

# Decimate 删重复帧，res.set_output()，pp_clip.set_output()可预览
res = core.vivtc.VDecimate(pp_clip)
```