x265 视频压缩教程综合版 c

欢迎阅读! 若有什么不会的可以加群 691892901. 本教程很难, 入门先看 x264 视频压缩教程综合版. 但现在就要压视频就去拿急用版教程哦(`·ω·´)ゞ 用 ctrl+f, 让电脑帮你找内容((((*. _.)

ffmpeg, VapourSynth, avs2yuv 传递参数

```
ffmpeg -i 〈源〉 -an -f yuv4mpegpipe -strict unofficial - | x265 --y4m - --output
ffmpeg -i 〈源〉 -an -f rawvideo    - | x265.exe --input-res 〈分辨率〉 --fps 〈整/小\/分数〉 - --output
```

-f 格式, -an 关音频, -strict unofficial 关格式限制, --y4m 对应"YUV for MPEG", 两个"-"是 Unix pipe 串流

```
VSpipe.exe       〈脚本〉.vpy --y4m - | x265.exe - --y4m --output
VSpipe/avs2yuv   〈脚本〉.vpy       - | x265.exe --input-res 〈宽 x 高〉 --fps 〈整/小\/分数〉 - --output
avs2yuv.exe      〈脚本〉.avs -raw  - | x265.exe --input-res 〈宽 x 高〉 --fps 〈整/小\/分数〉 - --output
```

ffmpeg 查特定色度采样 ffmpeg -pix_fmts | findstr 〈或 grep 关键字〉

检查/选择色深, 版本, 编译 x265.exe -V, -D 8/10/12 调整色深

多字体+艺术体+上下标.ass 字幕 ffmpeg -filter_complex "ass='F\:/字幕.ass'"滤镜

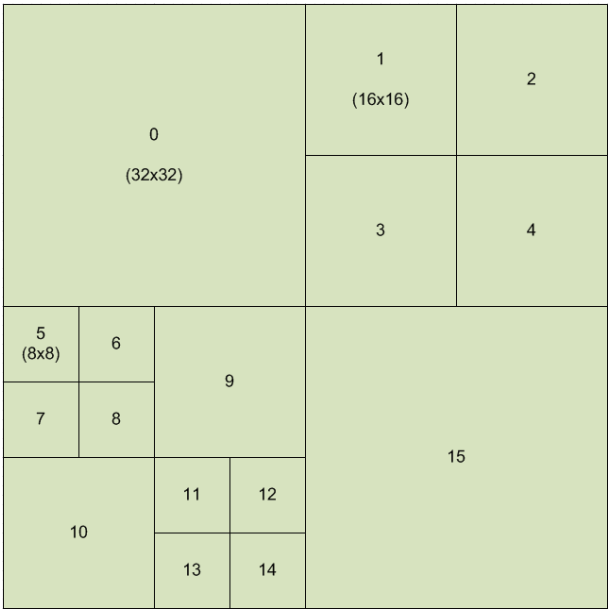
命令行报错直达桌面, 无错则照常运行 [命令行] 2> [桌面]\报错.txt

中途正常停止压制, 封装现有帧为视频 输入 Ctrl+C, x265.exe 自带功能

Bash 报错自动导出+命令窗里显示 x265.exe [参数] 2>&1 | tee C:\x265 报错.txt

分块

hevc 中帧下结构按面积从大到小排列为帧→瓦 tile→条带 slice→条带段 ss→ctu→cu. **CU 和 CB - 编码单元/块**是 ctu 经动静态隔离计算后得出的分块结果, 其大小由用户指定(*・_・)ノ⌒*



PU - 预测单元 prediction unit 是编码完, 可以用作参考的块. 支持 cu 上对称 rectangle, 非对称 asymetric partition 划分, 以更好的隔离动静态. 亮度与色度上的分裂法可以不同, 小至 4×4 像素

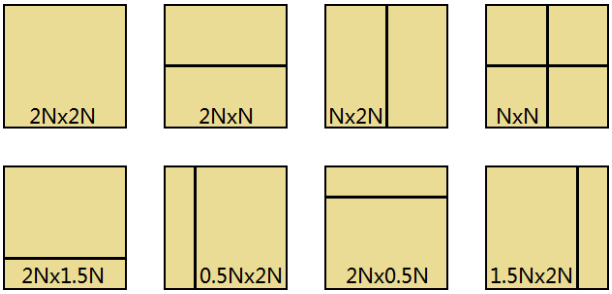


图: pu 的 4 种对称 rectangular 和 4 种不对称 assymetric 划分

TU - 变换单元 transformation unit 划分不与 pu 对齐, 有时还大到跨越多个 pu, 二者关系不大. 在 cu 上完成划分, 变换和量化. 小块 tu 会集中出现在高频信息, 大块会出现在低频(´▽`)ﾉ

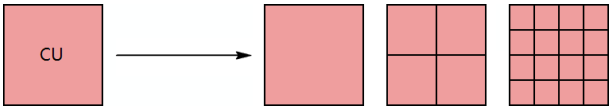
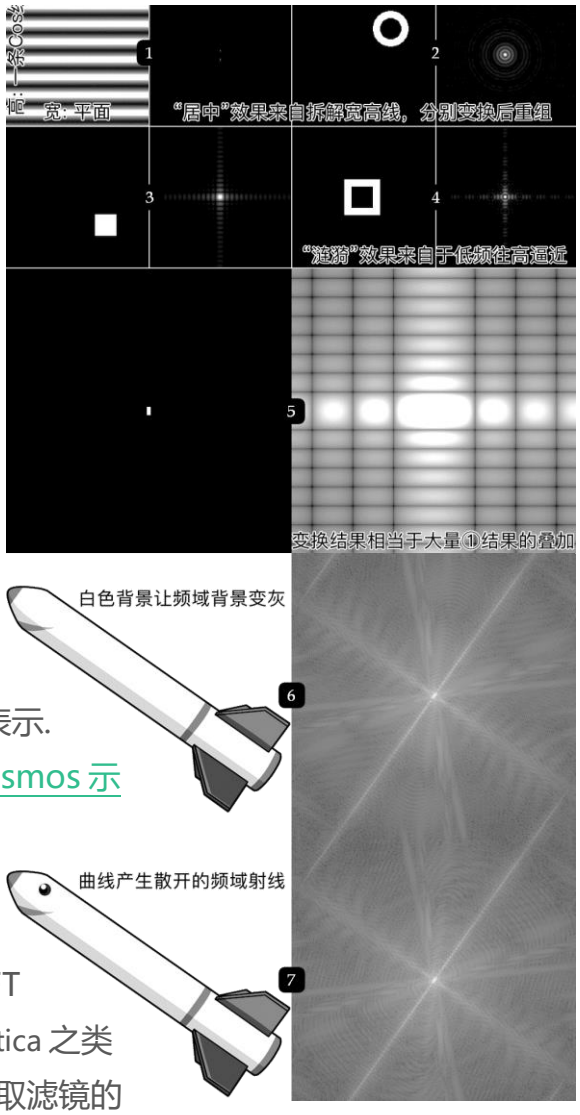


图: tu 的划分

--ctu<64/32/16, 默认 64>指定编码树单元最大大小的参数. CTU 越大, 有损压缩效率越高+平面涂抹越高+速度越慢. 一般建议保持默认, 但考虑到动画的大平面建议辅以低量化. 考虑画质优先时建议设<32>, 当分辨率特别小时建议设<16>且调整下面的参数(^-^*)/

--min-cu-size<32/16，默认 8>限制最小 cu 大小，简化计算步骤，因为使往后步骤 pu，tu 的划分也会更大．用多一点码率换取编码速度的参数．建议日常环境使用 16 或快速编码环境使用 32

--rect<开关，已关>启用 pu 的对称划分方法，用更多编码时间换取码率的参数．只建议有比较充足时间，分辨率大于 1440x810 或通篇颗粒的视频用 **--amp**<开关，已关，须 rect>启用 pu 的不对称划分．用更多更多编码时间换取码率的参数．只建议通篇有大量粒子/噪点，动漫源等分块能带来高收益的视频用



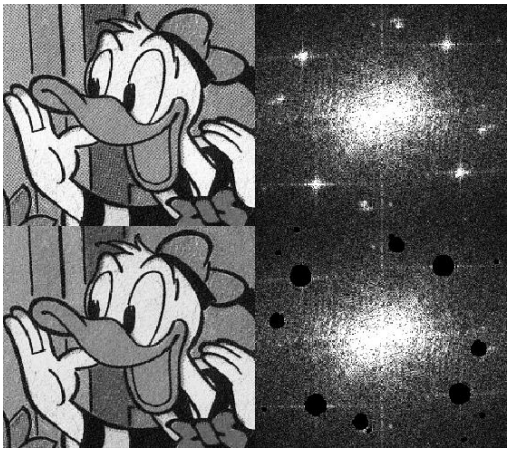
变换·(附录-傅里叶变换)

一维傅里叶变换 1D-FT

1. 给出与原信号波形等高，从类直线开始不断缩窄频率周期的余弦
2. 取立方临时将两波形负值转正，从而将反相的余弦也考虑在内
3. 记下吻合度最高，频率最低的余弦波，然后从原波形减去它；造成源的振幅降低，特征缩水
4. 重复 1~3，直到原信号变成一条直线．将所有记下的余弦按频率低-高排列，就得到了频域，或一维 k 平面的信号
5. 逆变换把频域值还给对应余弦，再加起来就复原了原信号

上面不断缩窄的余弦在低等数学上用 cos(1/Tx·2π) {0≤T≤n}表示．高等数学用欧拉公式卷缠．由螺度代替缩窄．可看巨佬写的 desmos 示 1，示 2，及 3b1b 视频

二维傅里叶变换 2D-FT 宽-高上每条线分别提出，像素值变化视作波形跑 1D-FT 后加到一起．亮则振幅大，远则频率高．FT 强在可编辑性，是消除光盘扫图噪声的唯一解，但仅 Mathematica 之类的算软能用．分辨率有核磁共振最高的 256²，以及中置声道提取滤镜的 32768²，缺点是看不出时间前后．因此遗憾缺席很多时间相关(音视频)处理(后来有人搞出各种小波拉链变换实现了)



二维离散余弦变换 2D-DCT 预制的二维波形模具，通过穷举加减列出每个频段的使用次数，图像就从空间域转换到频域了．优点是快．缺点是只有 8×8 种波形，删高频比 FT 更容易删失真

--limit-tu<整数 0~4，要求 tu-intra/inter-depth>1，默认 0 关>早退 tu 分块以量化/残差编码质量为代价提．tu 大则更容易出现量化涂抹而限码，不利于暂停画质．1 一般，画质编码，取分裂/跳过中花费最小的，2 以同 ctu 内的首个 tu 分裂次数为上限，3 快速编码取帧内帧间附近 tu 分裂平均次数为上限，4 不推荐，将 3 作为未来 tu 的分裂上限，相比 0+20%速度

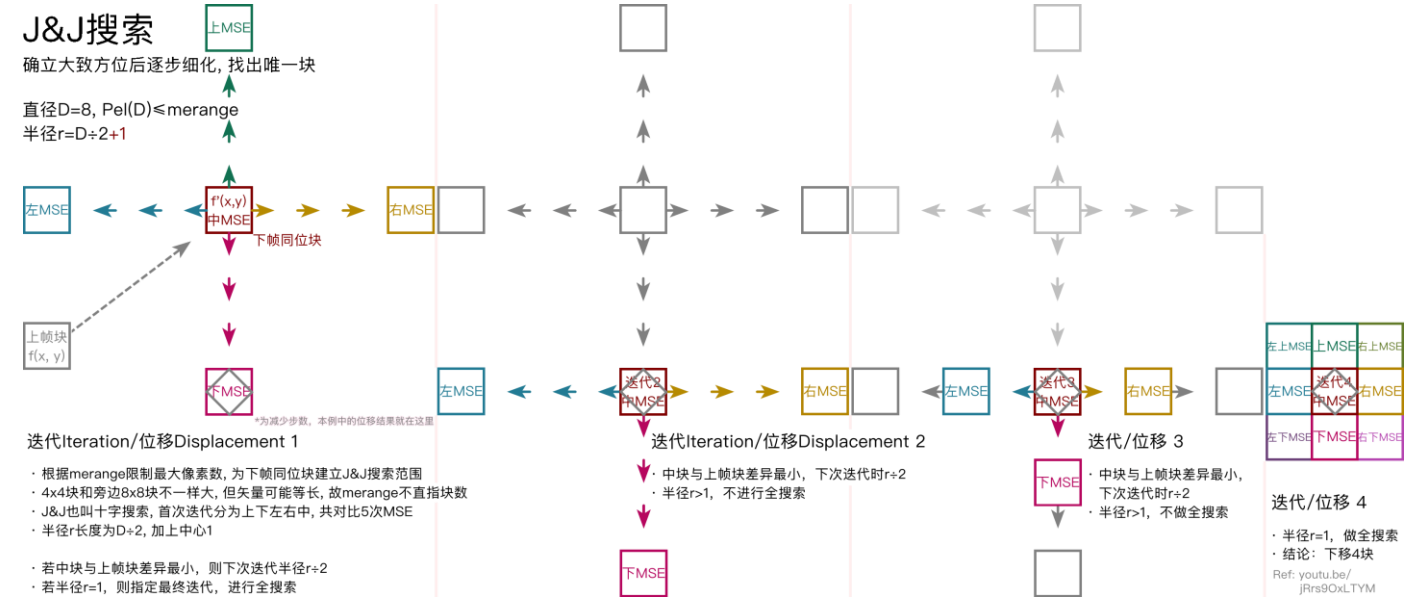
--rdpenalty<整数，默认 0 关，tu-intra-depth 1 时失效；2 则 32×32 帧内 cu 可用；3 才支持 64×64 帧内 cu>强制 tu 分块以提高细节保留降低涂抹．1 提高率失真代价而减少 32×32tu，或设 2 强制 32×32tu 分块．用途与 limit-tu 相反，但可理解为 tu 分块的下限，例如高 limit-tu，高 crf 时用 rdpenalty 2 避免 32×32tu 造成比 x264 16×16mb 涂抹还要烂的结果

--tu-intra-depth，**--tu-inter-depth**<整数 1~4，配合 limit-tu，默认 1>空间域 tu 分裂次数上限，默认只在 cu 基础上分裂一次．决定量化质量所以建议开高．建议日常编码设在 2，提升画质设 3~4

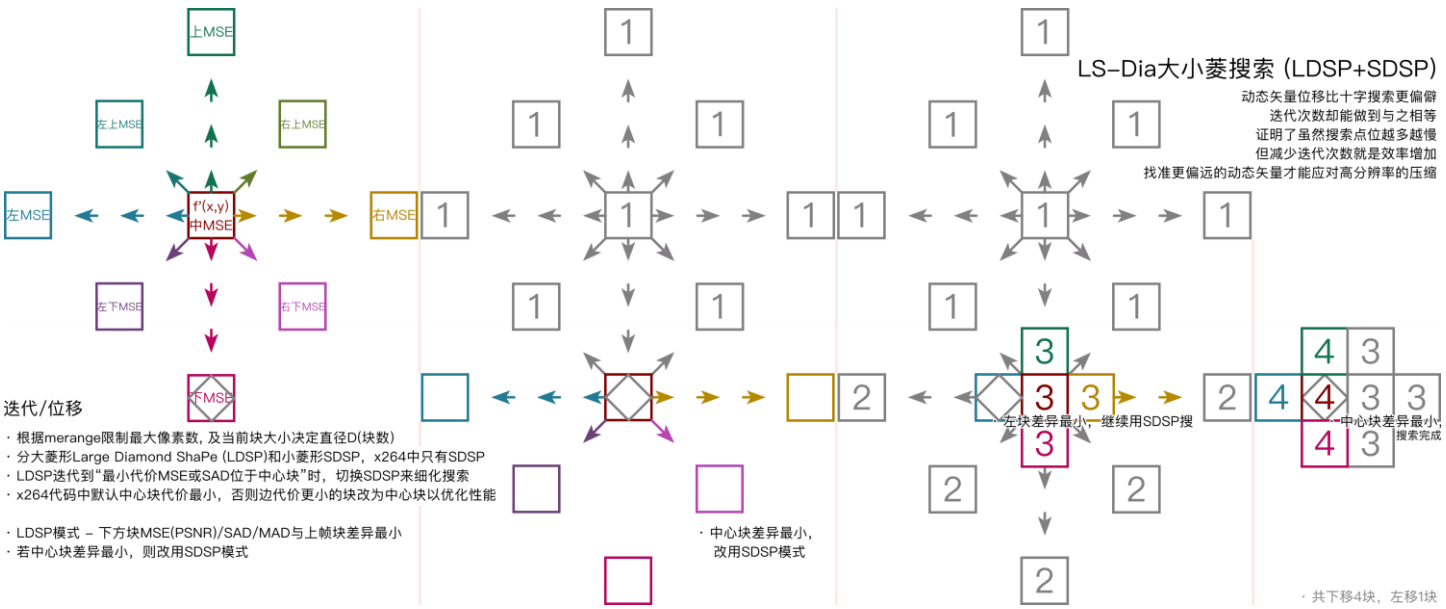
--max-tu-size<32/16/8/4，默认 32>更大的 tu 大小能提高压缩，但也造成了计算量增加和瑕疵检测能力变差．码率换时间+画质．建议不如直接设 ctu，因为也可减少 32×32tu

帧间-动态搜索

于帧间逐块地找最小失真朝向 dir. of min. distortion，组成一张张帧间矢量表的计算．若不到位，参考帧建立就欠缺基础．图：Jain&Jain/十字搜索，大小菱 LS-Dia 搜索；下页图：x264 umh 搜索



六边形 hexagonal 搜索将 LDSP8 外点换成 6 个，SDSP 细规则不变，相比 LS-dia 和 SSSP-LDSP-SDSP（四角星形 star 搜索）在 merange 16 的范围里效率更高

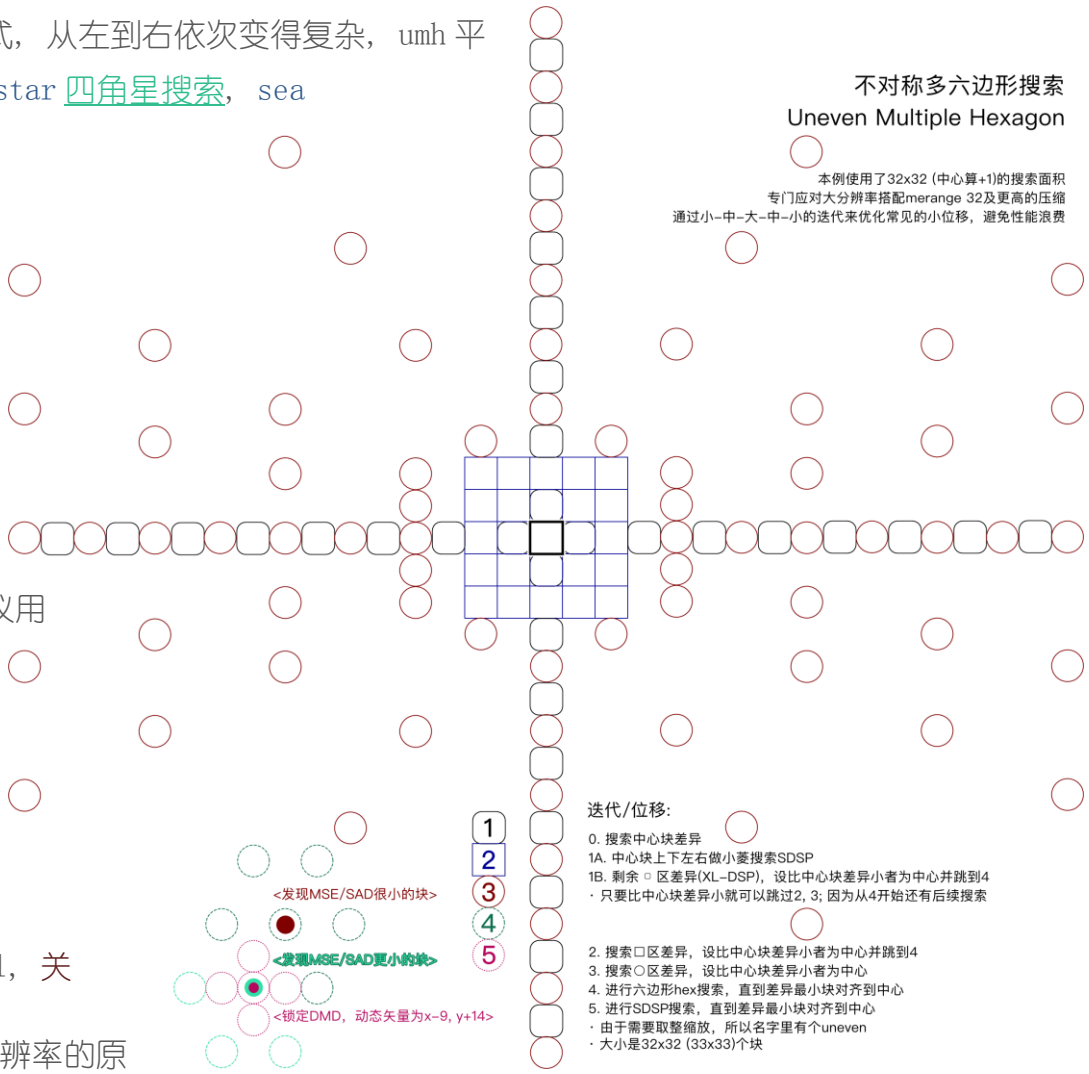


--me<hex~full>搜索方式，从左到右依次变得复杂，umh 平衡，star 之后收益递减. star 四角星搜索，sea 优化过的 esa 穷举

--merange<整数搜索范围，据动搜算法选>简单说 hex 选 16，umh-star 选>=32；一般推荐 me umh merange 48. 精致一些需要 DIY 测量距离. 太大会同时降低画质和压缩率，因为找不到更好的，找到也是错的)，所以 48 左右顶天，同时建议用 8 的倍数

--analyze-src-pics<开关，默认关>允许动态搜索查找片源帧，用更多时间换取码率的参数

--hme-search<hex~full，关me>hieratical motion estimation，制作三种分辨率的原画，分别宏观微观的搜索动态信息，用途待查



--hme-range<三个整数，默认 16, 32, 48>对应 1/16，1/4 和完整分辨率三个画面；建议 16, 24, 40

帧间-子像素运动补偿

motion compensation 将动搜所得信息做块-帧插值，让帧间连贯起来. 防止畸变相对复杂的动态信息让块脱离参考压缩. 冲激响应滤镜 imp. response filter 对超阈值的输入模拟信号出 1，否则出 0 的滤镜. 冲激~响应与音符~波形的关系所同. hevc 标准要求使用 7-tap 精度(1/4 像素补偿), avc 要求 6-tap. 影响模式决策和率失真优化. SAD, SATD 计算见 x264 教程完整版

--subme<整数默认 2，范围 1~7，24fps=4，48fps=5，60fps=6，+=7>根据片源的帧率判断. 分四个范围. 由于动漫片源制于 24~30fps，因此算力可省；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. 由于 x264 中 rdo 选项直接塞进了 subme，所以相比 x265 偏高

推荐范围	值	HPel 迭代	HPel 搜索方向	QPel 迭代	QPel 搜索方向	算法
不推荐	<1>	1 次	4	1	4	SAD
低算力	<2>	1 次	4	1	4	SATD
30fps 搭配 rdo	<3>	2 次	4	1	4	SATD
48fps 搭配 rdo	<4>	2 次	4	2	4	SATD

60fps 搭配 rdo	<5>	1 次	8	1	8	SATD
90fps 搭配 rdo	<6>	2 次	8	1	8	SATD
144fps 搭配 rdo	<7>	2 次	8	2	8	SATD

加权预测 weighted prediction

x264 首发，修复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 implicit WP<用参考帧的距离插值>插值计算在帧内编码板块有说明

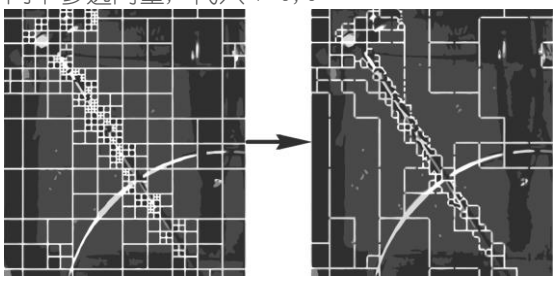
--weightb<开关，默认关>启用 B 条带的显，隐加权预测. 条带所在 SPS 中可见 P，B 加权开关状态，及显加权模式下解码器须知的权重. 光线变化和淡入淡出在低成本/旧动漫中少见

溯块向量搜索

与帧内编码并行，给动态搜索提供溯块向量(cu 帧内/帧间朝向，大小)的步骤. 由于移动的物件会跨越多个 pu. 所以将涉及同物件的块匹配到一起就能冗余一大批 pu 的动态向量了丶(￣—￣;)ノ

hevc 与 avc 一样据 ref 参数划时域之区，逐 pu 创建左右两排参考列表: List0 和 List1. 两种编码差在 hevc 新增了高级向量预测 advanced motion vect. pred; 并合搜索 merge mode 两种方案. 是 x264 direct auto 的升级版. AMVP 的任务是找出向量信息，麻烦但准:

1. 在帧内看当前 pu 左下的邻 pu，优先匹配向量往帧内指的邻 pu
2. 参考那些向量往它帧指的临 pu；并等比缩放，对齐到邻 pu 已按帧间差异对齐好的向量
3. 若以上步骤没找到参选向量，就把同样的步骤于当前 pu 右上角进行一次
4. 若应了如早批 pu 刚开始算，找不到参选向量的情况下就直接从时域搜索：照帧间参考图像变化的内容差异做缩放调整，从右下角的相邻 pu 找参选
5. 若仍不可用，就找当前 pu 中心位置的其它同位 pu. 若最后没凑不齐两个参选向量，代入 v=0,0



merge mode 简单粗暴，从时空域凑五参两被. 漏算 pu 边缘且不顾 pu 当前向量以提速，所以可看做是给 AMVP 打下手的

--ref<整数-0.01×帧数+3.4，范围 1~16>向量溯块前后帧数半径，一图流设 1. 要在能溯全所有块的情况下降低参考面积，所以一般设 3 就不管了丶(` ◇´)/

--max-merge<整数 0~5，默认 2>重设 merge mode 被选数量. 用更多时间换取质量的参数. 建议高压编码设<4>，其它可设<2，3>(+_+)

--early-skip<开关默认关，暂无建议>先查 2nx2n merge 被选，找不到就关 AMVP

GOP 结构建立，参数集

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

1. 按 IDR 帧间隔(keyint)分区，同时 scenecut 分配额外关键帧
2. 按 open/closed-gop 标记 gop 间预设，同时 gop 内的帧拆为条带 slice
3. 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

视频参数集 video parameter set→(分枝-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

--opt-qp-pps<开关，已关>据上个 GOP 改动当前 PPS 中默认的 qp 值(●∀●)

--opt-ref-list-length-pps<开关，已关>据前 GOP 改当前 ref 值，而且是前后帧独立改动

--repeat-headers<开关，已关>在流未封装的情况下提供 SPS，PPS 等信息，正常播放 h.265 源码

--scenecut<整数,推荐默认 40>设 x264/5 立 I 帧的敏感度

--hist-scenecut<开关，已关>亮度平面边缘差异+颜色平面直方图差异检测．如果两帧差之和 sad(见率失真优化板块)达到判定，就触发转场．可能是应对有了字幕，特效的新式视频而开发的

--hist-threshold<0~2.0，默认 0.01>标准绝对差异和 normalized sum of absolute differences 大于判定，就触发转场．每两帧都要计算一次．需要更多测试才能推荐使用

关键帧

idr 刷新解码帧 instant decoder refresh

- gop 间划界分段，令解码器清缓存的完整图片的 I 帧，清缓存是为了防治参考/内存错误(¬_¬)/

cra 净任意访问 clean rand. access

- open-gop 间划界，带帧内参考，gop 内帧间参考可越界的 I 帧，一般直接叫 cra 帧

dra 脏任意访问 dirty rand. access

- 一组含 i 块，全解码才重建出 i 帧的 P 帧．压缩更高但比 i 帧更易出错．需要低 min-keyint

bla 断链访问帧 broken link access

- open-gop 间划界，访问并加载出异分辨率，帧率视频流用的特殊 cra 帧ヽ(ー▽ー)ノ

参考帧

rap/随机访问点 random access point "访问"代表播出画面前读数据的过程; "任意"代表拖进度条，打开直播，使进度条上任意一点都要能解出视频的需求，增加码率提升体验

rasl 任访略前导, radl 任仿解前导 random access skiping/decoding lead

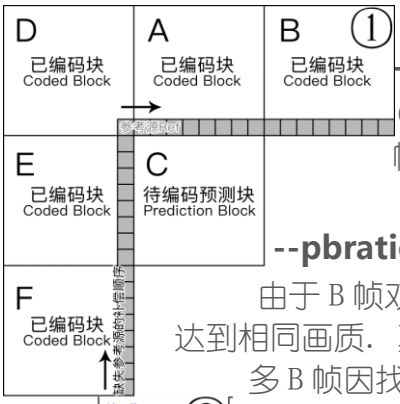
- 正常播过来没它们事，但进度条落在 cra 附近(缺参考)时指定解码/略过的前导帧. 防止拖进度条让 gop 崩坏

--no-open-gop<开关，默认关，建议长 gop 用>不用 cra/bla，增加码率增加兼容

--radl<整数默认 0，小于连续 B 帧，建议 2~3>原理见上

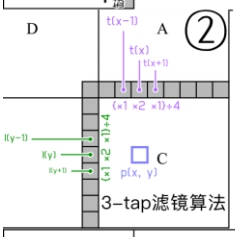
--min-keyint<整数，1 高画质，=keyint 高压缩+最快>指定最小 IDR 帧间隔. 防止编码器在 closed-gop 里将两个 IDR 帧挨太近，导致 P 和 B 帧参考距离受限而设计的. 由于一旦转场就通常意味着前后帧参考的价值降低，而大平面画面可以让脏访问重建的 I 帧干净许多，所以默认值就相对平衡. 反过来若有参考价值，也会因为 P 和 B 帧本身含 I 宏块的原因而更可能不用 I 帧. IDR 自身和之后的 P 帧 B 帧也完全可以相互参考. 同时在激烈画面多设立 IDR 帧还可以减少拖拽进度条的延迟，所以画质相对更高. 若 VBV 开启则不成立. VBV 反而会开高量化将码率尖峰干掉，画质反降

--keyint<整数>指定最大的 IDR 帧间隔，单位为帧. 由于 min-keyint 有设立 IDR 帧的能力，建议照不精确索引下拖动进度条的偏移延迟 vs 码率设置. --keyint -1 即 infinite. 在长度短到不需要拖动进度条，或者用户一定不会拖动进度条的视频可以使用以降低码率(◑‿◐)

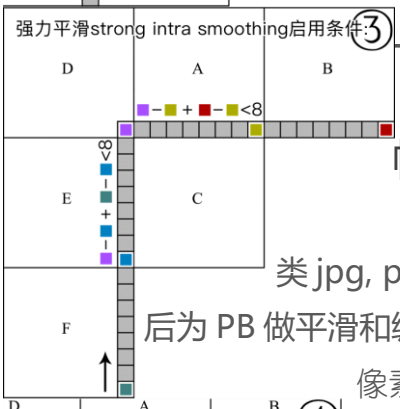


--fades<开关，默认关>找流中的虚实渐变 fade-in，给小到帧间条带(slice，一组横向 ctu)，整个帧间范围改用 I 条带，并根据渐变后最亮的帧重设码率控制历史记录，解决转场致模糊的问题

--pbratio<浮点，默认 1.3>P，B 帧的 qp 值待遇差(如 B 块至少是 P 块 1.3 倍 qp). 由于 B 帧双向参考能从更远的 I，P 帧中定位参考信息，所以 qp 更高也能通过参考来达到相同画质. 真人录像片源中保持默认即可. 动漫片源中 B 帧的出现几率增多，导致很多 B 帧因找不到合适的参考信息损失画质. 所以编码画质的动漫时要通过降低 B 帧的 qp 值分配来保护其画质，通常使用<1.2>或更激进



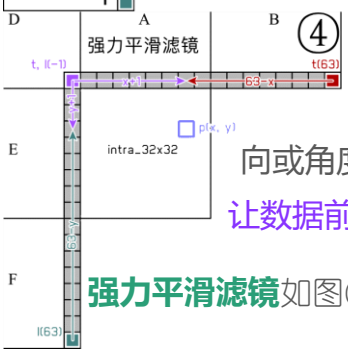
--bframes<整数范围 0~16>最多可连续插入的 B 帧数量. <3~9>快速，<12 左右>正常，若播放设备配置偏高的话就放心的设在<13 左右>吧



--b-adapt<整数 0 关，1 快 2 精确，推荐 2>B 帧适应性算法

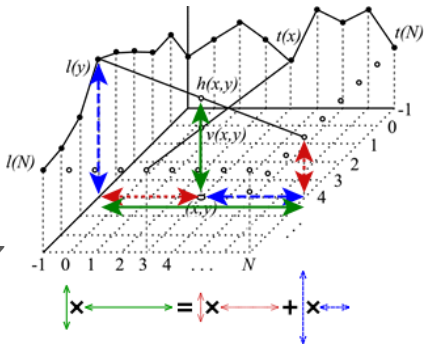
帧内编码

类 jpg, png 的单图内部做参考压缩. 分为补齐因 PB 位置不好而缺的编码块 CB，然后为 PB 做平滑和编码的流程. ①：如缺 EDAB 块就拿 F 块最上面的参考源填；如果全缺就拿像素平均值填，8bit 下是 127；2.**参考源数据平滑预处理**，3.**用趋平/夹角/DC 模式编码**. **3-tap 滤镜**如图②也叫(1 2 1)÷4 滤镜，按每个预测值 p 给横纵轴对应的参考源做 1 2 1 权平均计算. 和强力平滑滤镜一样会自动在 DC，垂直，横向或角度近似垂/横的情况下跳过. 平滑与模糊的区别在于直接砍高频也叫模糊；平滑让数据前后连贯好压，且高频砍得少

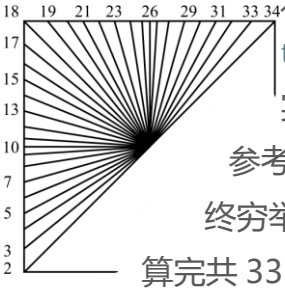


强力平滑滤镜如图③. 原理如图④：从最远两个参考源线性插值出每个 p(x,y)对应的参考源

趋平模式代表从左-上做双线 bilinear 插值到右-下边形成平面. 图: 底×高, 加底×高就有了"三角形"的面积. 该面积随 h(x,y)高的移动而变化; 再除以 pb 边长底就插值出了预测像素值 h(x,y); 同类计算用在横轴 t(x)上就插值出了 v(x,y); 两者取平均即新的 p(x,y)



夹角模式中, 角度θ分为*宽.正角 26~34, 宽.负角 18~25, 高.正角 10~17 及高.负角 2~9 四大类*. 由直角三角函数 tan(θ)=opp÷adj, 再乘以新临边 adj 得出新的远边 opp, 也就得到了 p 在新角度下实际对应到横轴, 纵轴上的参考像素群; 且大多情况下因小数差异, 角度会落在两个参考源■间, 要判断该角度下 p 投影离左右哪个■更近, 进行加权平均插值出参考源. 最终穷举出参考结果; 才能取最像 PB 的那个为新 CB. 当然为了省算力, 不到不得已是不会算完共 33 种结果的. **DC 模式**就是大平面, 通过全部 CB 的平均值和 PB 的平均值判断



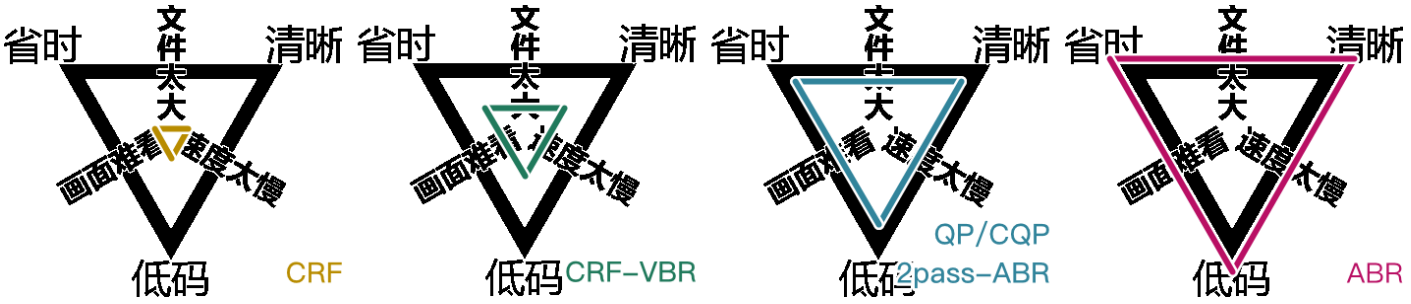
--fast-intra<开关, 默认 rd 大于 4 则关>5 个夹角地跳着判断夹角模式. 理论上画面复杂才有效

--b-intra<开关, 默认关>开启对 B 分片的帧内格式搜索. 建议高压编码开启

--no-strong-intra-smoothing<开关, 推荐默认开>32x32 的 PB 禁用强力平滑滤镜, 改用 3-tap. 因筛选条件苛刻且所平滑的数据是参考源, 所以只会在画面复杂的情况下略降 32x32 块的画质并提速(除非分块步骤太多跳过), 又遇上了画面复杂的情况. 没 64x64 是因为 PU 最大仅 32x32

--constrained-intra<实验性, 默认关>帧内条带不参考帧间像素. 高压/低码下减少误参考的失真

量化-质量控制模式



对数 log(x)是为了方便乘除法交换律, 即路程=速度×时间关系用在指数上, 构成指-根-对数交换律而产生的算法. 和开根号的区别是未知数是指数本身. 例如 $9=3^2$, $3=\sqrt[2]{9}$, $2=\log_3(9)$, 对应 6=2×3, 3=6/2, 2=6/3 改指数关系后的状态. 指数增长即 $y=ab^n$ 关系, 或输出=输入×倍率^次数. 编码器中凡是算 qp 值的地方都用 log, 因为量化强度 qp 本来是对数, 对应着指数变化的量化强度, 在 crf/cqp 不在 18~23 范围里, 码率/画质成倍涨跌的现象中就能体会到

CRF 上层模式

crf/abr 模式下的 qp 计算独用一套率-失真模型. 即 $R=aX_n÷qscale$. 值的关系相当于路程=速度×时间, 套用到"帧内复杂度=率×qscale"上, 为速度=路程÷时间的关系. a 指外界影响. **qscale 是 qp 改用指数表示的量化强度**. qscale 转 qp 即 $qp=6\log_2(85\%qscale)+12$. 图: 紫 qScale 与原本黑 qp 的关系

1 $Q(X)=6\log_2\left(\frac{X}{0.85}\right)+12$

2 Q: qp, X: qscale, 1.18X: qscale÷0.85

3 $a=6\log_2(b)+12$

4 $\log_2(b)=\frac{a-12}{6}$

指数对数交换律: $2=\log_3(9)$ 所以$9=3^2$

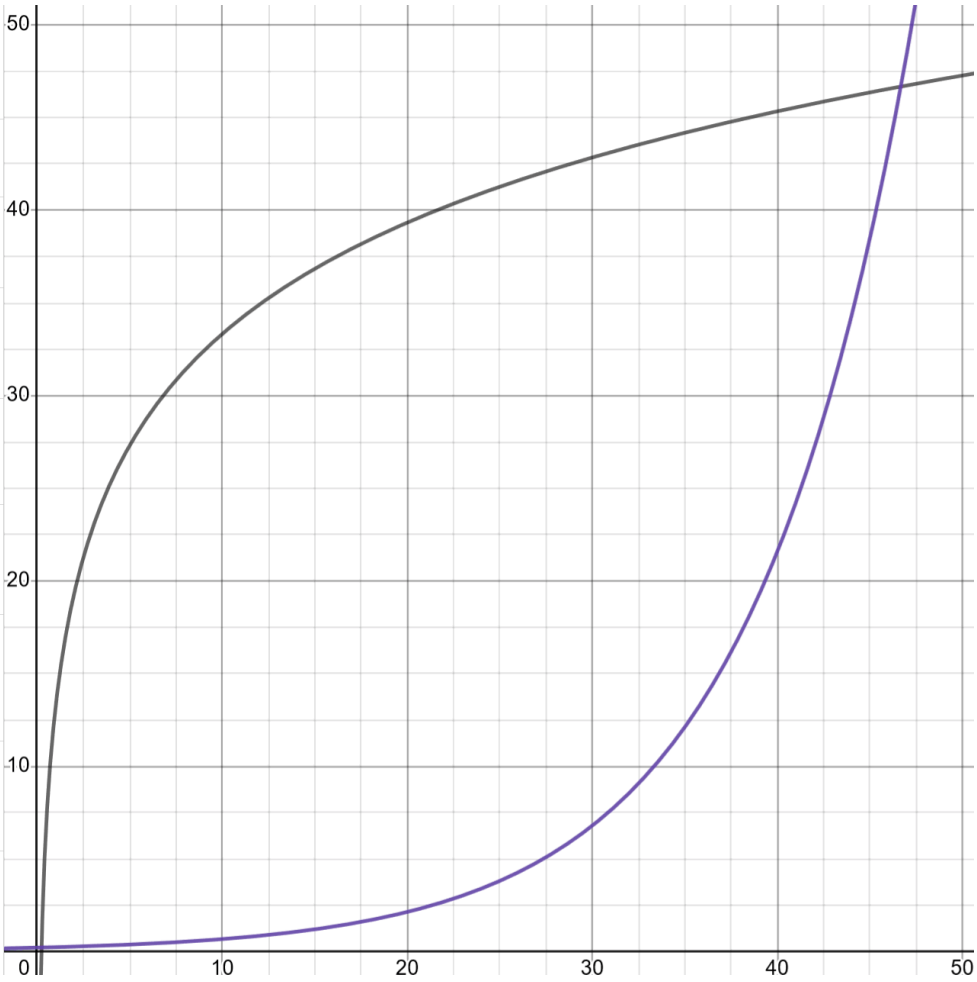
5 $b=2^{\frac{a-12}{6}}$

6 $b=\frac{X}{0.85};\ a=Q(X)$

7 $\frac{X}{0.85}=2^{\frac{Q(X)-12}{6}}$

8 $X=0.85\times 2^{\frac{Q(X)-12}{6}}$

9 $y=0.85\times 2^{\frac{x-12}{6}}$



整那么麻烦将 qp 值改成对数 qscale 是为什么? 因为 qp 转成 qscale 后才能算上 cutree, B 帧偏移, 复杂度计算三步, 逆运算回来才得到最终的 qp 值:

- 当前帧 SATD%复杂度 cplxSum: = cplxSum[上帧] × 0.5 + SATD[上帧]
- 当前帧 100% 复杂度 cplxCount: = cplxCount[上帧]× 0.5 + 1
- 当前帧拟复杂度 cplxBlur(SATD%÷100%): = cplxSum÷cplxCount

- CRF_qScale = cplxBlur^(1-qcomp)÷CRF_RF
- ABR_qScale = qscale × overflow ÷ABR_RF
  - CRF_RF = crf + cutree_qp_offset + bframe_qp_offset
  - ABR_RF = target_rate_window ÷ cplxSum
  - overflow = clip3f(1 + (total_rate - target_rate) ÷ abr_buffer, 0.5, 2)
- qp = $6\log_2(85\%qscale)+12$

最终, 实现了帧内画面简单, qp 值(压缩)高; 反过来越复杂 qp 越低的压缩理念. 然而这种质量判断只有两帧, 不够宏观(abr 模式更严重), 所以有了率失真优化量化的必要

--crf<浮点范围 0~69, 默认 23>据"复杂度 cplxBlur, cutree, B 帧偏移"给每帧分配各自 qp 的固定目

标质量模式，或简称质量呼应码率模式，统称 crf．素材级画质设在 16~18，收藏~高压画质设在 19~20.5，YouTube 是 23．由于动画和录像的内容差距，动画比录像要给低点

虽然相比于 x264 的量化一样．但 crf 越高，率失真优化的需求也越高，速度越慢

--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，其他情况永远不如关 mbtree (*~▽~)

--qcomp<浮点范围 0.5~1，一般建议默认 0.6>crf/abr 模式分配 qp 值的延迟．出自 cplxBlur^(1-qcomp)更改 crf/abr 模式中模糊复杂度 cplxBlur 计算的采用率．cplxBlur 的用处是通过前后两帧推演出复杂度的变化程度，将两帧内突变的画面糊掉

- <1> cplxBlur^1-1　　a^0=1,　　关复杂度判断因素(cplxBlur=1)，和 cqp 模式一样，暂停画质最高
- <0> cplxBlur^1-0　　a^1=a,　　两帧内突变的画面糊掉，突然高码什么的不存在
- <0.5> cplxBlur^1-0.5　　a^0.5=√a,　　√cplxBlur 程度地损失一些画质抵抗突然高码，平衡，适合直播
- <0.7> cplxBlur^1-0.7　　两帧内突变的画面多糊
- <0.3> cplxBlur^1-0.3　　两帧内突变的画面少糊，适合一般~收藏画质

--rc-lookahead<整数，范围 1~250>用于指定 cutree 的检索帧数，通常设在帧率的 2.5~3 倍，若通篇的画面场景非常混乱则可以设在帧率的 4 到 5 倍通常在 180 之后开始增加计算负担

--no-cutree<开关>关闭少见 CTU 量化增强偏移.只有近无损，可能 crf 小于 17 才用的到

CQP 上层模式

--qp<整数，范围 0~69>恒定量化．每±6 可以将输出的文件大小减倍/翻倍．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，所以除非 yuv4:4:4 情况下有既定目的，都不建议

--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.
例：指定 IDR/I qp17，P qp20，B/b qp22 时填写--qp 17 --ipratio 1.1765 --pbratio 1.1

ABR 上层模式

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

VBR 下层模式

--vbv-bufsize<整数 kbps，小于 maxrate>编码器解出原画的每秒最大码率缓存. bufsize÷maxrate=编码与播放时解出每 gop 原画帧数的缓冲秒数，值的大小关联编完 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，bufsize 的 x 倍>峰值红线. 防止多个>bufsize GOP 连续累积，给出缓帧启用高压的参数. 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 大所以没起作用
- 欠: GOPsize>bufsize=1~2×maxrate，超码率限制后直接压，但因视频大/crf 小所以全都糊掉
- 由于 gop 多样，4 种状态常会出现在同一视频中. buf~max 实际控制了这些状态的出现概率

--crf-max<整数>防止 vbv 把 crf 拉太高，可能适合商用视频但会导致码率失控；**--crf-min**<整数>用途不明，可能是反留白习惯所致，目前--qpmin 足以[-_-]╠

2pass-ABR 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<导出 stats>；**--pass 2**<导入 stats>；**--stats**<文件名>默认在 x265 所在目录下导出/入的 qp 值逐帧分配文件，一般不用设置

--slow-firstpass<开关>pass1 里不用 fast-intra no-rect no-amp early-skip ref 1 max-merge 1 me dia subme 2 rd 2，也可以手动覆盖掉

Analysis-2pass-ABR 模式

在普通 2pass 基础上让 pass1 的帧内帧间分析结果 pass 到 pass2，减少计算量

--analysis-save<"文件名">指定导入 analysis 信息文件的路径，文件名

--analysis-load<"文件名">指定导出 analysis 信息文件的路径，文件名

--analysis-save-reuse-level; --analysis-load-reuse-level<整数 1~10，默认 5>指定 analysis-save 和 load 的信息量，配合 pass1 的动态搜索，帧内搜索，参考帧等参数. 建议 8/9

- <1>储存 lookahead
- <2==4>+同时储存帧内/帧间向量格式+参考
- <5==6>+rect/amp 分块
- <7>+8x8cu 分块优化
- <8==9>+完整 8x8cu 分块信息
- <10>+所有 cu 分析信息(^..^)/

--dynamic-refine<开关，已关闭>自动调整--refine-inter，x265 官方文档中建议搭配 refine-intra 4 使用，相比手动设定提高了压缩率，建议关闭(๑•ᴗ•๑)

--refine-inter<整数默认 0，范围 0~3>限制帧间块的向量格式，取决于 pass1 分析结果是否可信

- <0>完全遵从 pass1 的分块深度和向量格式
- <1>分析所有 pass2 中与 pass1 相同分块的向量格式，除 2pass 中比 1pass 更大的分块
- <2>一旦找出最佳的动态向量格式就应用于全部的块，2Nx2N 块的 rect/amp 分块全部遵从 pass1，仅对 merge 和 2Nx2N 划分的块的动态向量信息进行分析
- <3>保持使用 pass1 的分块程度，但搜索向量格式

--refine-intra<整数默认 0，范围 0~4>限制帧内块的向量格式，取决于 pass1 分析结果是否可信

- <0>完全遵从 pass1 的分块深度和向量格式
- <1>分析所有 pass2 中与 pass1 相同分块的向量格式，除 2pass 中比 1pass 更大的分块
- <2>pass1 找最佳分块程度/向量格式的话 pass2 就跳过
- <3>保持使用 pass1 的分块程度，但优化动态向量；<4>=pass1 丢弃不用

--refine-mv<1~3>优化分辨率变化情况下 pass2 的最优动态向量，1 仅搜索动态向量周围的动态，2 增加搜索 AMVP 的顶级候选块，3 再搜索更多 AMVP 候选（°-°；）ノ゛

--scale-factor<开关，要求 analysis-reuse-level 10>若 1pass 和 2pass 视频的分辨率不一致，就使用这个参数

--refine-mv-type avc 读取 api 调用的动态信息，目前支持 avc 大小，使用 anamyse-reuse 模块就用这个参数+avc(原文解释的太模糊，且未测试)

--refine-ctu-distortion<0/1>0 储存/1 读取 ctu 失真(内容变化)信息，找出 pass2 中可避的失真

2pass 转场优化

--scenecut-aware-qp<整数，默认关，2 仅转后，1 仅转前，推荐 3 前后降低，仅 pass2 用>转场前/后拉低默认 5 qp 以增加画质. 原理是转场本身就缺参考源，所以提高已有参考源的画质

--masking-strength<逗号分隔整数>于 sct-awr-qp 基础上定制 qp 偏移量. 建议根据低~高成本动漫，真人录像三种情况定制参数值. scenecut-aware-qp 的三种方向决定了 masking-strength 的三种方向. 所谓的非参考帧就是参考参考帧的帧，包括 B，b，P 三种帧...大概

- sct-awr-qp=1 时写作<转前毫秒(推 500)>,<参考±qp>,<非参±qp>
- sct-awr-qp=2 时写作<转后毫秒(荐 500)>,<参考±qp>,<非参±qp>
- sct-awr-qp=3 时写作<转前毫秒>,<参考±qp>,<非参±qp>,<转后毫秒>,<参考±qp>,<非参±qp>
- scenecut-window，max-qp-delta，qe-delta-ref，qp-delta-nonref<被 x265 v3.5 移除>

--analysis-reuse-file<文件名 默认 x265_analysis.dat>若使用了 2pass-ABR 调优，则导入 multi-pass-opt-analysis/distortion 信息的路径，文件名

Analysis-Npass 间调优

在 Analysis-pass1~2 之间加一步优化计算. 实现比普通 2pass 更精细的码率控制, 1~N 也行

--multi-pass-opt-analysis<开关, 默认生成 x265_analysis.dat>储存/导入每个 CTU 的参考帧/分块/向量等信息. 将信息优化, 细化并省去多余计算. 需关闭 pme/pmode/analysis-save|load

--multi-pass-opt-distortion<开关, 进一步分析 qp>根据失真(编码前后画面差). 需关闭 pme/pmode/analysis-save|load

--multi-pass-opt-rps<开关, 已关>将 pass1 常用的率参数集保存在序列参数集 SPS 里以加速

Analysis-pass2-ABR 天梯

--abr-ladder<实验性的苹果 TN2224/据说整出一堆 bug, 文件名.txt>编码器内部实现 analysis 模式 2pass abr 多规格压制输出. 方便平台布置多分辨率版本用. 可以把不变参数写进 pass1+2, 变化的写进 txt. 格式为"[压制名:analysis-load-reuse-level:analysis-load] <参数 1+输出文件名>"

```
x265.exe --abr-ladder 1440p8000_2160p11000_2160p16000.txt --fps 59.94 --input-depth 8 --input-csp i420 --min-keyint 60 --keyint 60 --no-open-gop --cutree

1440p8kb_2160p11kb_2160p16kb.txt {

[1440p:8:Anld 存档 1] --input 视频.yuv --input-res 2560x1440 --bitrate 8000 --ssim --psnr --csv 9.csv --csv-log-level 2 --output 1.hevc --scale-factor 2

[2160p1:0:nil] --input 视频.yuv --input-res 3840x2160 --bitrate 11000 --ssim --psnr --csv 10.csv --csv-log-level 2 --output 2.hevc --scale-factor 2

[2160p2:10:Anld 存档 3] --input 视频.yuv --input-res 3840x2160 --bitrate 16000 --ssim --psnr --csv 11.csv --csv-log-level 2 --output 3.hevc --scale-factor 0

} analysis-load 填 nil(不是 nul)代表略过
```

近无损压缩, 真无损压缩上层模式

--lossless<开关>跳过分块, 动/帧/参搜索, 量/自适量化等影响画质的步骤, 保留率失真优化以增强参考性能. 直接输出体积非常大的原画, 相比锁定量化方法, 这样能满足影业/科研用, 而非个人和一般媒体所需, 真无损导出有几率因为参考质量提升, 会比近无损小

--tskip<开关, 已关>不在 tu 上使用 DCT 变换へ(. •◌ -")

--cu-lossless<开关, 已关>将"给 cu 使用无损量化(qp 4)"作为率失真优化的结果选项之一, 只要码率管够(符合λ=R/D)就不量化. 用更多码率换取原画相似度, 无损源能提高参考冗余

RDO 控制量化; 率控制

率失真优化控制码率与量化决策的步骤, 叫 rate control. 因率失真优化不考虑量化对参考帧的损害, 所以 x265 画质在此受到了影响

--rdoq-level<整数, 范围 0~2>率失真优化控制量化的分块深度. 0=关; 1=不分 tu; 2=4x4, 慢

--psy-rdoq<浮点 0~50, 默认 0 关>心理视觉优化偏移率失真优化的程度, 提高能量 J 以改变 rdoq 的用途, 使其更不愿消除系子, 避免模式决策遇到差选项. 类似 crqpoffs

1080p 高码率下设<2.3~2.8>给动漫, <3~4.8>给电影. 分辨率高低, 画面颗粒影响了系子数量和密度, 所以要改参数值 ╭(๑¯д¯๑)╮

- 常用: psy-rdoq 和 psy-rd 功能冲突, 所以保留 rdoq-level 1, 关 psy-rdoq, 开 psy-rd
- 高码: 有颗粒的情况下同时用低强度的 psy-rdoq 和 psy-rd, rdoq-level 2
- 少用: 目前 x265 psy-rd 还没写 cpu 指令集(慢, 待跟进), 所以关 psy-rd, 开 psy-rdoq

自适应量化

根据源图像的复杂度来判断 qp 值分配的计算, 防止 x265 往细节分配太多码率而造成平面的质量亏损. 对防止图像变得模糊有一定作用 (￣~￣;)

--aq-mode<范围 0~4, 0 关, 默认 2>aq 只在码率不足以还原原画时启动, 建议<1>标准自适应量化(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)

目前 1 以上的参数没有达到预期的质量补偿, 出现了色带和过度降噪, 在 x265 v3.3 前建议只用 aq 1, 或测试 aq 4. 在需要胶片颗粒噪点的视频中提高 psy-rd. 根据 doom9 论坛, 当前版本中 hevc-aq 因造成失真, 画质不如 aq 4. 目前官方-第三方间信息较为割裂

--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

--aq-motion<开关, 实验性>根据动态信息微调自适应量化的效果 mode 和强度 strength(;*△*;)

--qg-size<64/32/16/8>实验性参数. 自适应量化能影响到的最小 cu 边长/最大分裂深度. 默认 64 可换取更多速度, 减少可略增优化效果. 高画质/平衡都建议设在 32~16. 用途不明的<最浅,最深>格式能自定义范围, 如 32,8 代表只在 32, 16 和 8px 的 cu 上起作用(•ω•´)σ

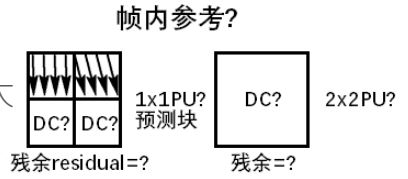
--cbqpoffs, **--crqpoffs** <整数>调整蓝，红色平面相比亮度平面的 qp 值差异，负值降低量化．若当前版本 x265 的算法把色度平面的量化变高，可以用这两个参数补偿回来．由于编码器一直不擅长处理红色，而人眼又对红光敏感可能因为祖先晚上生火所以为了画质建议比 cb 面设更低(△-3 左右)的值

模式决策

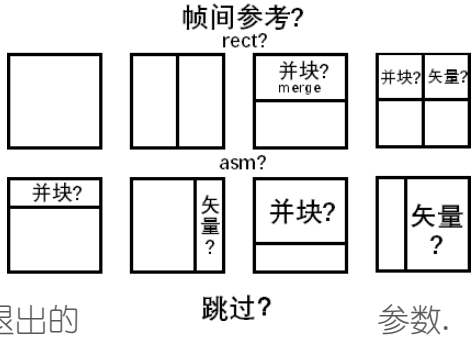
mode decision 整合搜到的信息，给 ctu 分块，参考，跳过，编码，量化全流程定制实现优化．率失真优化是避免模式决策无脑只选不同码率最小，复杂动态下全糊的控件(¬‿¬)

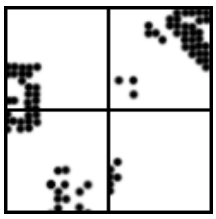
--rd<1/2/3/5，默认 3>率失真优化参与 md 的程度，越大越慢．<1>优化帧内参考，并块/跳过决策 <2+分块决策> <3+帧间决策> <5+向量/帧间方向预测决策>建议快速编码用 1，2；日常/高压使用 3，其他情况(包括高画质高压编码)使用 3，5

--limit-modes<开关>用附近的 4 个子 CU 以判断用 merge 还是 AMVP，会大幅减少 rect/amp 块的存在感，明显提速．会增大或减少体积，微降画质但难以察觉



--limit-refs<0/1/2/3，默认 3>限制分块用信息可参考性．<0 不限>压缩高且慢；<1>用 cu 分裂后的信息+差异信息描述自身(推荐)；<2>据单个 cb 的差异信息建立 pu；<3=1+2>ヾ(-_-;)





--rskip<0/1/2/3>找不到残余向量/宏观上出现 cu 再分块被跳过时，判断后面 cu 接着搜索分块还是提前退出的参数．和 merge-AMVP 的区别是管辖 cu 内部的再分块．<0 不跳>费时费电换一点压缩；<rd=0~4 下 1>看附近 cu 是不是也分不了；<rd=5~6 下 1>看附近 2Nx2N cu 分块难度，推荐；<2>统计 cu 内纹理决定分块，推荐；<3>在 2 基础上直接跳过底部块

--rskip-edge-threshold<0~100，默认 5，rskip 大于 1>sobel 算子检测 cu 纹理密度 edge density，对比块面积的百分比．若密度超过值就分块，越小分块越多

--tskip-fast<开关，已关>跳过 4x4 tu 的变换，忽略部分系子 coefficients 来加速，CbCr-tu 也取决于 Y 块是否被跳过．在全屏小细节的视频中有显著加速效果．建议除高压以外的任何环境使用

率失真优化 RDO 控制

率失真优化 rate distortion optimization 据多个码率下测得的失真程度点，挑出低于率失真曲线的值，再根据码率/crf 做**模式决策**．x264/5 用拉格朗日代价函数 J = D + λ·R．即"开销 = 失真+λ·码率"．x265 中**失真** Distortion 用 MSE 判断．淘汰了 SSE 的宏观处理改为逐块

$$\text{(x265) Mean Squared Error} = 1 \div \text{宽高} \sum_{x=0 \to Tx} \sum_{y=0 \to Ty} |f(x,y)-f'(x,y)|^2$$

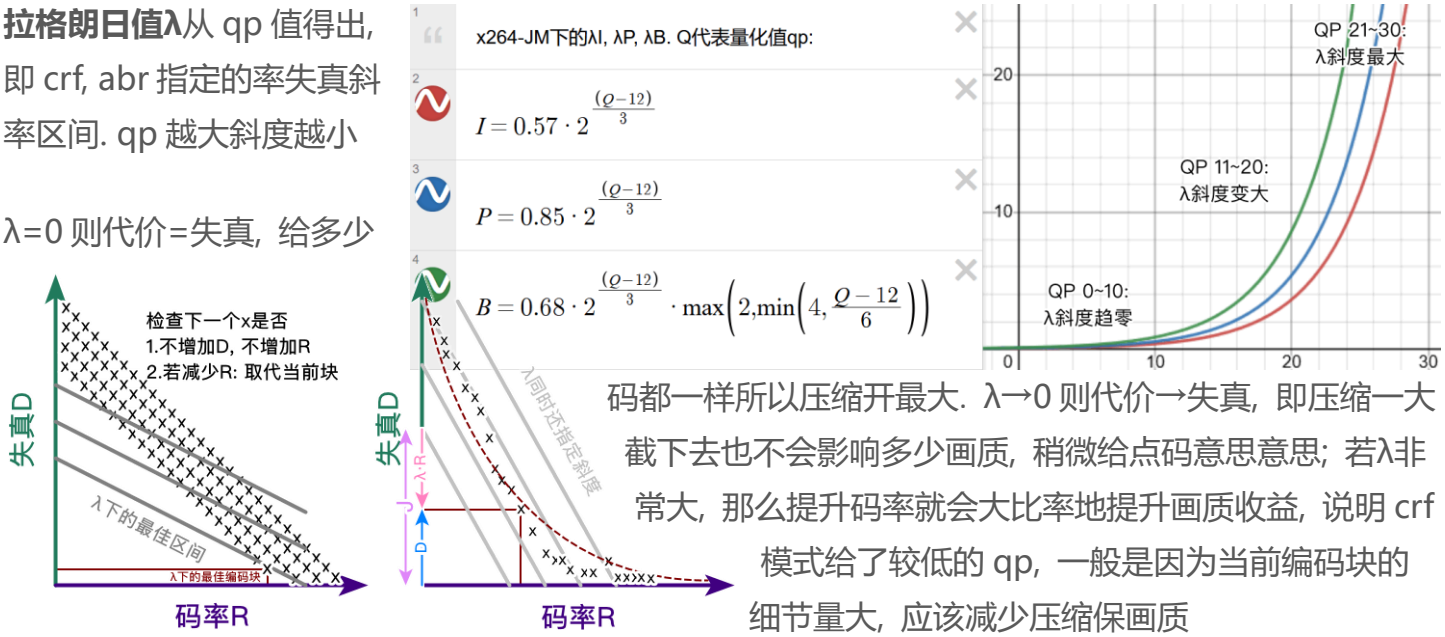
(x264) Sum of Squared Err = $\sum_{x=0\to Tx}\sum_{y=0\to Ty}|f(x,y)-f'(x,y)|^2$

(x264 --fgo) Noise SSE = $\sum_{x=0\to Tx}\sum_{y=0\to Ty}|[N(x,y)-N'(x,y)]\cdot fgo|+|f(x,y)-f'(x,y)|^2$

- $\sum_{x=0\to Tx}$ 代表块宽度求和范围，f()和 f'()分别代表参考块和参考源
- $\sum_{y=0\to Ty}$ 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

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 则代价=失真，给多少码都一样所以压缩开最大. λ→0 则代价→失真，即压缩一大截下去也不会影响多少画质，稍微给点码意思意思；若λ非常大，那么提升码率就会大比率地提升画质收益，说明 crf 模式给了较低的 qp，一般是因为当前编码块的细节量大，应该减少压缩保画质



--psy-rd<浮点 0~50 默认 2，需 rd3，默认 0 关，和 x264 不同>心理视觉优化影响率失真优化的程度，增加量化块的能量，抗拒帧内搜索，使模式决策 mode decision 遇不到差选项. 注意搭配 psy-rdoq 使用. <0.2>高压，动漫据纹理设<0.5~2>. 录像设<1.5~2.5>，星空与 4k+级别的细节量设<2.8>或更高

参数值随分辨率大小变化. 注意噪声和细节都是高频信息，所以开太高会引入画面问题. 图：复杂度对真人录像的重要性，但这些点点毛刺在低成本/大平面动漫里就很难看了

--rd-refine<开关，建议开，需 rd 5>率失真优化分析完成帧内搜索 cu 的最佳量化和分块结果，耗时换压缩率和画质. x264 中直接嵌入 subme 8 中，还多一个最优动态向量分析

--dynamic-rd<整数，范围 0~4>给 VBV 限码的画面调高率失真优化以止损. 1~4 对应 VBV 限码的画面的 rd 搜索面积倍数，越大越慢

--splitrd-skip<开关，已关>启用以在"所有当前 CU 分割致失真程度之总和"大于"任意同帧 CU 分割致失真程度之总和"时，不跟随当前 CU 分割之结果来独立计算 rd 值以加速

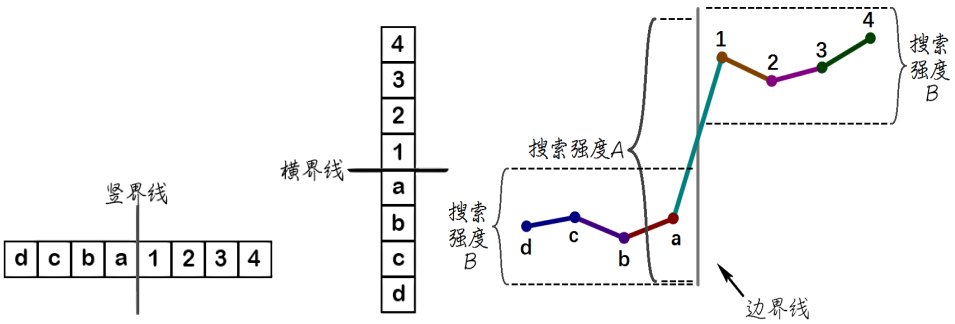
峰值信噪比 peak signal-to-noise ratio/PSNR

信号电压声音强度都用分贝表示，因为分贝通过信号值越大/越小，电平增长越弱的对数线 y=log(x)显示. 功率转 dB 的算式即 $PSNR=10\log_{10}(最大/当前)$. 最大/当前就是最大像素值÷MSE; 8bit 下即 256/MSE

环路滤波-去块

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(ว•ω•˚)ว*☆

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>

环路滤波-取样迁就偏移

sample adaptive offset 不在 hevc 标准内. 这种算法通过采样图像来降低编码后画面细节的偏移量. 从而修复录像设备造成的色带 banding 和振铃 ringing 两种瑕疵. sao 的搜索区域可以是整个画面, 也可以小到一个 CTU. 帧内搜索合并了多个 CTU 后, 还可以将判断结果在合并后的区域里共享

锐偏移 eo 会在 ctu 内建立几个采样区并取相互对比, 若出现了代表锐利的凸起或代表平滑的下凹出现, 就少数迁就多数. 就 x265v3.0 说, eo 不能自行判断应该锐化/柔化时就一律柔化. 虽然降了码率, 也造成了 sao 在对付细节复杂, 比如毛茸茸的区域时表现差, 即"糊一片"

带偏移 bo 会将搜索区分为 32 个图像分带 img segment. 然后对偏离程度最大的, 4 个连一起的分带进行迁就处理. 目前, 这样做虽然使编码前后的画面细节变得接近, 但也使得调整后图像的线条出现了

位移的问题，这两种问题使得使用 sao 需要配合其它的辅助或关闭ヘ(˚∇、°)

--no-sao<关闭--sao，默认开 sao>早期版本的 x265 中没有 limit-sao 参数，所以编码画质时可以使用以防止 sao 造成的画面问题(ノ ﹏ ヽ,)

--sao-non-deblock<开关>启用后，未经由 deblock 分析的内容会被 sao 分析◑.◑

--no-sao-non-deblock<默认>sao 分析跳过视频右边和下边边界(/)u(\ )

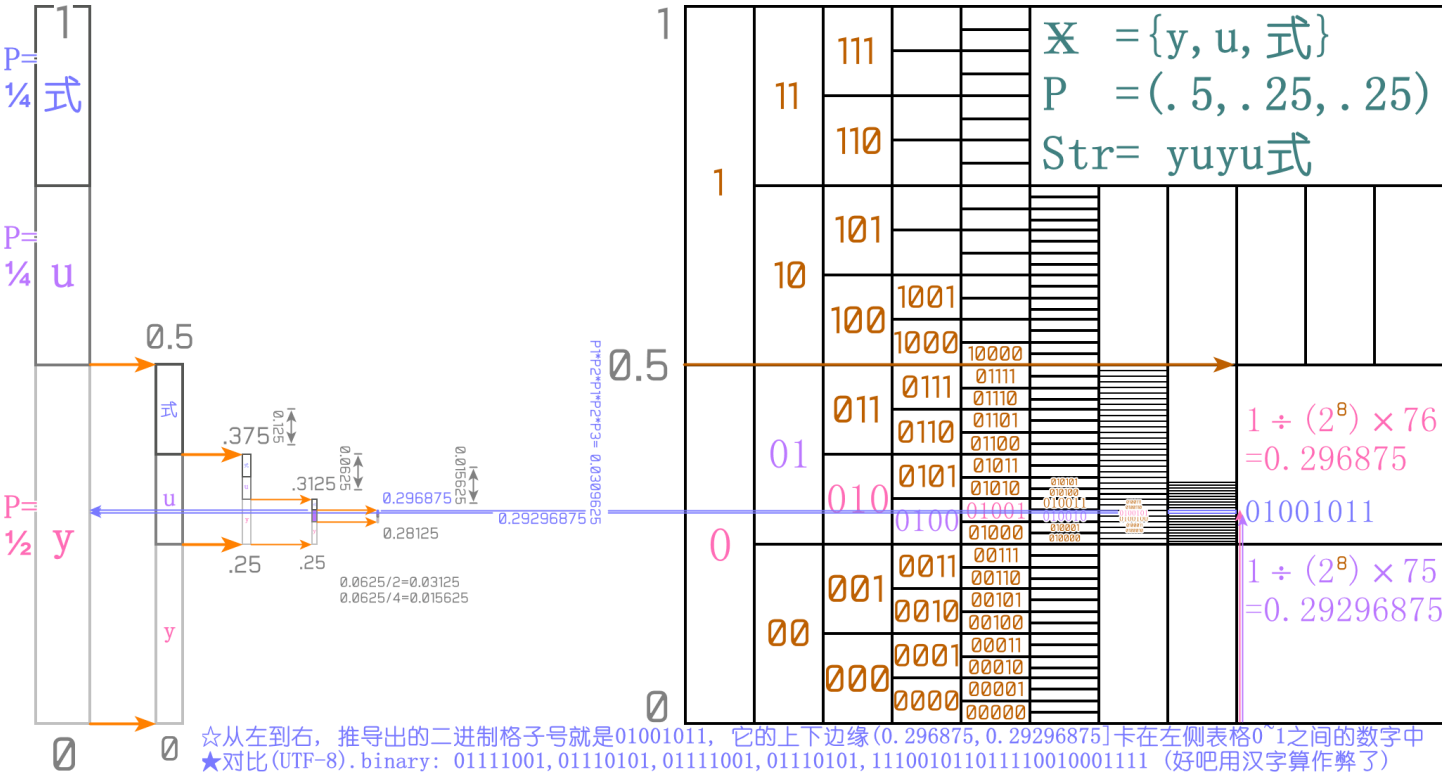
--limit-sao<开关>已关闭. 对一些 sao 的计算采用提前退出策略，可以改善 sao 造成的画面问题

--selective-sao<0~4，默认 0>从条带(横向一组 CTU)角度调整 sao 参数，1 启用 I 条带 sao，2 增加 P 条带，3 增加 B 条带，4 所有条带. 可看作新的 sao 控制方式，或搭配 limit-sao 的新方法

熵编码/残差编码-CABAC

游程编码将降维后的块/条带丢给熵编码，是最后的文本压缩步骤(率失真优化会解码检查每道压缩所以要经历好几次熵编码). x264/5 中使用了 context adapt. binary arithmetic coding. 相比于 cavlc 与霍夫曼编码, cabac 是现代编码器(压缩包到音视频)的核心算法之一(ㄐ ˚Д ˚)ㄐ

算数编码 arithmic coding 在 0~1 间的模具中，用匹配符(阵列)，出字概率(次数)及待压内容压缩. 例如匹配𝔵={y, u, 式}三种符；概率 P=0.5, 0.25, 0.25 那么压缩"yuyu 式"就照下图规律，从左到右细分后给出二进制格子里符合条件的小数：



电脑直接算二进制快，所以 cabac 中加了个二进制转换器，直接算二进制就是 bac；最后直接根据游程编码给的东西，比如𝔛={ng, a, b, b, b, b, be, a, b, qs, q, ..., EOF}，让算法自己根据上下文清点概率，自行调节 0~1 模具，需要的话把阵列里用不到的东西另建一套用，就是所谓的 cabac 了 ℓ(⋆ ❛ ᴗ ❛ ⋆)ა

上图例子若用 0.419, 0.11 这样的出字概率，算数编码仍能精确的编码~~我不想算饶了我吧~~。但霍夫曼编码的精度会被分支一次只能÷2 的限制挡住，造成类似 8bit 视频有时比 10bit 体积大的现象(っ˘ڡ˘ς)

SEI 维稳优化消息

supplemential enhance info 记录每帧的补充信息。主要有正确打开新 gop 用的缓冲 sei，解码卡时间的 pic timing sei，让显示主控切边的 sei, cc 字幕 sei, hdr-sei 等等

缓冲 sei 记录对应 sps 的号；待解码图像缓冲 coded picture buffer/cpb 的延迟安全区等信息；时戳 sei 记录哪些帧上/下场优先的变化；连帧/三连帧的位置等信息

--hrd<开关，已关>开启后将假设对照解码参数 hypothetical ref. decoder param. 在没有丢包和延迟的假想下算好，供解码器临场得知一些瞬间信息，写在每段序列参数集 sps 及辅助优化信息 sei 里，对播放有点好处. 比如 2Mbit 大一帧，30fps 的瞬间带宽就是 2 ÷ (1/30 秒)=60Mbps

--hash<md5, crc, checksum, 默认无>sei 里加效验码，播放时可用以对图像重建纠错来减少失真，三种方式中 md5 播放时所需算力较高，checksum 最快但有忽略概率，crc 平衡

--single-sei<开关>只写一个装全部 sei 信息的大 NALU 而非每 gop 都写，提高压缩率

--film-grain<文件名>将如 libfgm 提取的纹理细节模型 film grain model 写进 SEI，将编码压缩掉的细节另存档，兼容解码器播放时恢复的功能

--idr-recovery-sei<开关>sei 里写进 idr 帧，串流时防止整个 gop 都找不到参考帧而崩坏的机制

--frame-dup<开关默认关，必须开 vbr 和 hrd，有 bug>将 2~3 面近似的连续帧换成同一帧

--dup-threshold<整数 1~99，默认 70>相似度判定值，默认达 70%重复就判为相似

线程节点控制

--pools<整数/加减符,,,, 默认+,+,+,+>x264 中--threads 的升级版. 如--pools +,-,-,-表明 pc 有 4 个节点，仅占用第一个. +代表全部处理器线程. 这样能防止多处理器系统上跑一个 x265 时，所有处

理器访问第一个节点的内存而造成延迟等待. 应该是跑和节点一样多的 x265, 每个节点各自运行. 单 cpu 系统直接作--threads 用, 如--pools 8 指该 pc 有 1 个节点, 占用该节点上处理器的 8 个线程. Enterprise streamer could modify x265 to achieve dual-node parallel analysis-2pass, 4-node-3-res-analysis-ABR-ladder, or 8-node-7-res-analysis-ABR-ladder (e.g., 3840x2160, 2560x1440, 2080x1170, 1920x1080, 1600x900, 1280x720, 960x540), which achieves the fastest multi-res stream deployment possible with better quality than nvenc's parallel 2pass processing

不要企图设置大于实际线程数的 pools/threads 提速. 会因为处理器随机并发的特性从任务数量上冲淡参考帧建立等要之前的步骤算完才能开始的时间窗口. 否则编码器只能跳过参考压缩, 造成处理器占用降低, 码率增加以及压制变慢的副作用

TR1000~2000 系处理器是用多个节点拼出来的, 所以单处理器的内部要按多个节点分开算, 特例是 2990WX, 2970WX, 核心组 1 和 3 没有内存控制器, 0 和 2 有内存控制器, 所以 1, 3 不能用

--pmode<开关>使用平行帧内搜索, 目前出现了难以应付噪点, 会造成画质下降, 码率提高的问题

--asm<avx512>avx512 was a mistake - Intel engineer

多线程 vs 多参考

用多线程一次编码多帧来占满算力, 还是一次只编一帧, 确保所有参考画面可用的决策. 确保所有帧同时吞吐○(・x・). 虽然 x265 有 tile 这种集合多个分片的并行化. 造成多线多参考帧困难的原因有:

1. ctu 比宏块大, 相似性降低了ヾ(▽` ;)ゝ
2. 参考前要等环路滤波和率失真优化, 还有已编码信息的依赖, 使得很多参考, 特别是高 ref 设定下来不及找就跳过
3. 参考帧的波前编码 wavefront parrallel process (压制/播放的多线程改进版)因一行参考 ctu 的存在而卡死, 重启波前编码等没了多余算力

--pme<开关>使用平行动态搜索 parallel ME, 已关. 多开几个动态搜索, 榨干所有剩余的 CPU 算力(如 frame-threads 1 时). 若已占用 100%则别用(@ ￣Д ￣@;)

--frame-threads<整数 0~16~线程数/2, 默认 0 自动>同时压多少帧, 设 1 能让前后整帧可参考, 非 1 就只给 ctu 下方的一行 ctu. 设 1 的代价是 cpu 占用显著降低, 压制减速(-, -)

--lookahead-threads<整数 0~16~线程数/2, 默认 0(关闭)>分出多少线程专门找参考, 而非与帧编码一同占线程, 可能只有开 frame-threads 1 时手动启用以增加 cpu 占用, pme 和 pmode 同理

色彩空间转换, VUI/HDR 信息, 黑边跳过

光强/光压的单位是 candela. 1 candela=1 nit

--master-display<G(x,y)B(,)R(,)WP(,)L(,)>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)

--max-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

--hdr10<自动开关>当 master-display 和 max-cll 启用就直接在 sei 中指示 hdr10 相关参数，原本参数名叫--hdr(和 hdr-opt 一样)，改名是为了指明它能优化新的 hdr10，而非旧的 hdr

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

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

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

--hdr10-opt<开关，已关>逐块为 10bit bt2020，smpte2084 视频做亮度色度优化，其它视频无效

--display-window<←,↑,→,↓>指定黑边宽度以跳过加速编码，或者用--overscan crop 直接裁掉

--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 等，如→图的 bt.2020

--colormatrix<字符>播放用矩阵格式/系数：GBR bt709 fcc bt470bg smpte170m smpte240m YCgCo bt2020nc bt2020c smpte2085 ictcp，如上图的 bt2020 non-constant

--transfer<字符>传输特质 transfer characteristics：bt709 bt470m smpte170m smpte240m linear

log100 log316 bt2020-10 bt2020-12 smpte2084 smpte428，上图 PQ 是 smpte st.2084 的标准，所以写 smpte2084(⊙.⊙)

ffprobe 会将三个信息并在一行写: Stream #0:0(und): Video: prores (XQ) (ap4x / 0x78347061), yuv444p12le (tv, bt2020nc/bt2020/smpte2084, progressive)

IO(input-output，输入输出)

--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

--output<字符串，两边带双引号>例：--output "输出文件地址+文件名" "输入文件地址+文件名"

--input-csp<i400/i422/i444/nv12/nv16>在输入非默认 i420 视频时需要的参数，rgb 空间需转换

--dither<开关>使用抖动功能以高质量的降低色深(比如 10bit 片源降 8bit)，避免出现斑点和方块

--allow-non-conformance<开关>不写入 profile 和 level，绕过 h.265 标准的规定，只要不是按照 h.265 规定写的命令行参数值就必须使用这个参数≪(¯﹃¯")≫

--force-flush<整数 0~2，默认 0>录像，录屏和损坏源用. 当输入帧速度慢且常迸发很多帧时的措施:

0=等全部帧输入再编码；1=不等全部帧输入完就编码；2=取决于条带种类，调整 slicetype 才能用

--field<开关>输入分行扫描视频时用，自动获取分场视频的帧率+优先场，替代了--interlaced 参数

--input-res<宽×高>在使用 x265 时必须指定源视频的分辨率，例如 1920x1080

--fps<整数/浮点/分数>在使用 x265 时必须指定源视频的帧率，小数帧填小数，勿四舍五入(´∽` *)

--chunk-start --chunk-end<开关，no-open-gop>chunk-start 允许跨 GOP 制作数据包(?)，改由 chunk-end 参数将数据包结尾和剩下的视频帧断开(?). 据描述看，由于数据包接收顺序一定会被打乱，所以只可参考其之前，而不可参考之后的内容，跟 http 的数据包编码协议有关Σ(-_-๑)

下载 附录与操作技巧

LigH	.hevc GCC10 [单文件 8-10-12bit] 附 x86, Windows XP x86 版 附 libx265.dll
Rigaya	.hevc GCC 9.3 [8-10-12bit] 附 x86 版
Patman	.hevc GCC 11+MSVC1925 [8-10-12bit]

ShortKatz

DJATOM-aMod

MeteorRain-yuuki

arm64~64e 加 x86 版 [?] 需 macOS 运行编译命令文件 ？

opt-Intel 架构与 zen1~2 优化 [10bit], opt-znver3 代表 zen3 优化 [10-12bit] GCC 10.2.1+GCC10.3

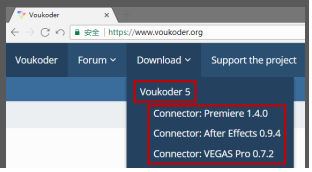
lsmash.mkv/mp4 或.hevc [能封装，但传说 lavf 不如 pipe 可靠] GCC 9.3+ICC 1900+MSVC 1916 [8][10][12bit]+[8-10-12bit]

ffmpeg 多系统兼容，备用地址 ottverse.com/ffmpeg-builds

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

x265GuiEx (Rigaya) 日本語, auto-setup 安装，教程点此

Voukoder; V-Connector 免费 Premiere/Vegas/AE 插件，直接用 ffmpeg 内置编码器，不用帧服务器/导无损再压/找破解. 只要下两个压缩包，放 Plug-Ins\Common 文件夹就行了



gcc 是什么，为什么同版同参的编码器速度不同

把源码编成程序的软件即编译器. x265 有 mingw(gcc 套件)，套件版本新旧影响编出程序的效率, msvc 体积更小，但需要 VCRUNTIME140_1.dll；icc 需要 libmmd.dll；Clang 需要...?

速度不一样还可能源自内建函数. 函数即等待变量输入的算式. 由于 8bit x265 中有大量开发组手动编写的内建函数，所以不同编译者给出的程序速度也不等. 而 10bit x265 完全没有手动编写的内建函数，所以编译者只有优化源码. 同样，速度测试应以 10bit x265 为基准(ﾍ｡ﾍ)

rc 指 release candidate

有的 x265 编译的文件名上有 rc，指已修复所有被提出的问题 且编译者认为 ok 的版本丶(·ω·ゞ)

杜比视界 dolby vision 不深入研究

作者认为, dolby vision 和光线追踪一样，支持的内容少，还需要特定的解码器或游戏 / (v x v) \

有两种 dv 格式，单视频流和双视频流，双视频流有视频层和 db 强化层，强化层可以被一般的 hevc 解码器丢弃，单视频流就只有特定的解码器才能播放

而按照 db 的意思，未来很可能只有硬件解码器用，而这需要高价购买支持这种格式的设备才能看

CMD 操作技巧 color 08

将原本黑景白字改成黑景灰字的单行命令，降低视疲劳

cmd 窗口操作技巧%~dp0

"%~"是填充字的命令(不能直接用于 CMD). d/p/0 分别表示 drive 盘/path 路径/当前的第 n 号文件/

盘符/路径，数字范围是 0~9 所以即使输入“%~dp01.mp4”也会被理解为命令 dp0 和 1.mp4

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

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

cmd for 循环批量压制(确保文件名无重复，预先分离出音频，预先将视频套滤镜渲染好)

给出 bat 文件所在目录下完整 pdf 路径+文件名: `for %%a in ('*.pdf') do echo '%~dp0%%a'`

批量压 mkv: `chcp 65001`

```
@ for %%1 in ('*.mkv') do (x265 [参数] --output 'D:\文件夹\%%~n1.mp4' '%~dp0%%1' & qaac [参数] -o 'D:\文件夹\%%~n1.aac' '%~dp0%%~n1.flac')
```

ffmpeg 批量压 mp4，音频拷到新文件: `chcp 65001`

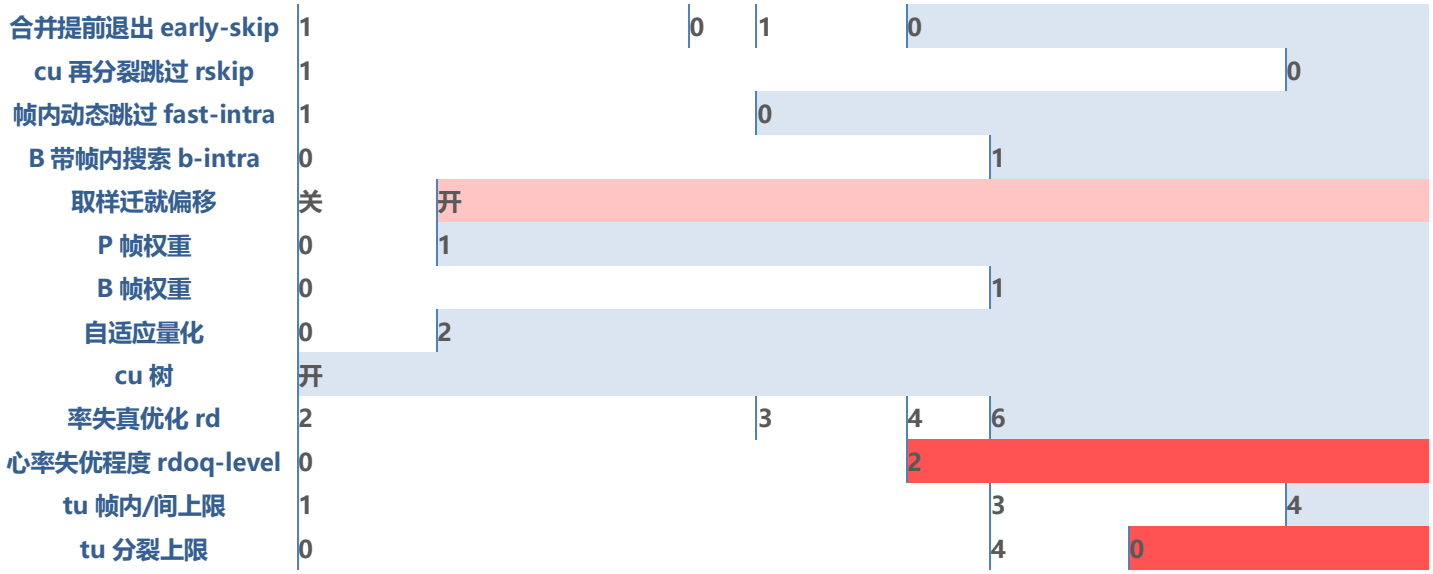
```
@ for %%3 in ('*.mp4') do (ffmpeg -i '%%3' -c:v copy -i '%%~n3.aac' -c:a copy '%%~n3.mp4')
```

chcp 65001 会让 cmd 以 unicode 形式读取，@是不打出输了什么命令进去，%%~n1 是%%1 去掉了文件后缀 o(-_^)

LSMASHWorks 崩溃 0xc0000005 可能是内存问题

预设

--preset	superfast	veryfast	faster	fast	medium	slow	slower	Very slow	placebo
ctu	32	64							
最小 cu	8								
连续 B 帧	3	4					8		
B 帧筛选	0				2				
cu 树向后 rc-lookahead	10	15			20	25	40		60
lookahead-slices	8					4	1		
参考帧	1	2		3		4	5		
参考帧限制 limit-refs	0	3					1	0	
动态搜索	hex					star			
动搜搜索范围	57								92
子像素搜索	1		2			3	4		5
矩形分块	0					1			
非矩分块	0						1		
分块模式快选 limit-modes	0					11		0	
合并模式数量 max-merge	2					3	4	5	



tune zerolatency 去延迟

连续 B 帧	0
B 帧筛选	关
cu 树	关
转场	关
多线程压制帧数	1

tune grain 最高画质

自适应量化	0
cu 树开关	关
I-P 帧压缩比	1.1
P-B 帧压缩比	1
QP 赋值精度 qp-step	1
取样迁就偏移	关
心理率失真优化程度 psy-rd	4
心率失优可用 psy-rdoq	10
cu 再分裂跳过 rskip	0

tune animation 动画片

心理率失真优化程度 psy-rd	0.4
自适应量化强度	0.4
去块	1:1
cu 树	关
B 帧数量	<preset>+2

tune fastdecode 解码加速

B 帧权重	关
P 帧权重	关
去块	关
取样迁就偏移	关

tune psnr 峰值信噪比

自适应量化	关
率失真优化 rd	关
cu 树	关