x265视频压缩教程**综合版**

欢迎阅读! 若有什么不会的可以加群[691892901](https://jq.qq.com/?_wv=1027&k=5YJFXyf). 本教程很难, 入门先看[x264视频压缩教程综合版](https://www.nazorip.site/archives/63). 但现在就要压视频就去拿[急用版教程](https://nazorip.site/archives/334/)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 <宽x高> --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 源.vpy --y4m - | x265.exe - --y4m --output

avs2yuv 源.avs -csp<色> -depth<深> - | x265.exe --input-res <宽x高> --fps <整/小/分数> - --output

avs2pipemod 源.avs -y4mp | x265.exe --y4m - --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字幕渲染** ffmpeg -filter\_complex "ass='F\:/字幕.ass'"滤镜

**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

**[8bit还是10bit色深](https://forum.doom9.org/newreply.php?do=newreply&p=1978001)** x265缩小了精度误差, 8-10差距比x264小, 但10bit仍可减轻色带

**目录**

分块 1

变换·(附录-傅里叶变换) 2

帧间-动态搜索 3

帧间-子像素运动补偿 5

加权预测weighted prediction 6

帧间-时域架网搜索 6

溯块向量搜索 7

GOP结构建立, 参数集 7

关键帧 8

参考帧 8

帧内编码 9

量化-码率质量控制模式 9

2pass转场优化 13

率失真优化-码率质量控制调优 14

自适应量化 15

模式决策 15

率失真优化RDO控制 16

峰值信噪比peak signal-to-noise ratio/PSNR 17

环路滤波-去块滤镜 17

环路滤波-取样迁就偏移 18

带偏移bo 18

熵编码/残差编码/文本压缩-CABAC 18

二进制算数编码 19

SEI维稳优化消息 19

线程节点控制 20

色彩空间转换, VUI/HDR信息, 黑边跳过 21

IO(input-output, 输入输出) 22

# 分块

hevc中, 帧下结构按面积大小列为帧🡪瓦tile🡪条带slice🡪条带段ss🡪ctu🡪cu单元. **cu和cb**是ctu经动静态隔离, 即动态搜索motion estimation与运动补偿motion compensation所得的结果. 其中U即unit, 指YCbCr三位一体, 而B/block则单指其一. **ipcm-cu**代表跳过me-mc, 直达环路滤波的 intra pulse code modulation cu (帧内编码的pcm波形cu), 因为"块"本来就是一串像素值的波形, 只是通过每个"单元"自带的元数据"换行"到了二维而已. intra代表帧内, 当提及帧内编码, 多指GOP建立后, 用于为B, b, P帧提供参考源的IDR和I帧所需要用到的完整单图, 但也包括参考帧内所含的I块



**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asymmetric 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asymmetric划分

**TU - 变换单元**transformation unit的划分与cu而非pu同步, 实现变换和量化 ( ´ ▽ ` )ﾉ



**AU - 存取单元**access unit是解码端用于启动播放的块, 一般为IDR-AU

--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

<开关默认关, 受limit-modes限制>pu对称划分, 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

<开关默认关, 须rect, 受limit-modes限制>pu不对称划分. 用更-多编码时间换取码率的参数. 只建议通篇有大量粒子/噪点, 动漫源等分块高收益的视频用

# 变换·傅里叶变换

**一维傅里叶变换1D-FT:** 给出与原信号波形等高, 从最长的频率周期开始不断缩窄(增加周期)的参考余弦. 在参考余弦波变化的过程中, 记下两条波形吻合度变化的曲线-不同波形周期的振幅, 就得到了频域信号. 反过来将频域信号所对应的波形加起来就实现了逆变换. 为了将源波形中反相的余弦也考虑在内, 所以计算过程中要取立方转正. "不断缩窄的参考余弦"在低等数学上用cos(1/T x·2π) {0≤T≤n}表示. 高等数学用eⁱˣ=cos(x)+i·sin(x). 由复数螺的缩窄以实现核磁共振等看中复数兼容性的用途. 见[desmos示1](https://www.desmos.com/calculator/qpnz9celzf), [示2](https://www.desmos.com/calculator/ywxqicajbv), 及[3b1b视频](https://www.youtube.com/watch?v=spUNpyF58BY)

**二维傅里叶变换2D-FT:** 宽高上单拆出线来分别进行1DFT, 通过双求和ΣΣ或双积分∫∫来缩写. 所以在频域中相当于每个像素的变换结果相加或干涉. 亮则振幅大, 远则频率高. FT强在可编辑性, 是消除光盘扫图等均匀噪声的解

**二维离散余弦变换2D**-DCT**:** 预制的二维波形模具, 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. 比2DFT快但只有8×8种波形, 所以删高频效果比FT差

--limit-tu

<整数0~4默认关, tu-intra/inter-depth大于1>提前退出tu分块, 以量化/残差编码质量为代价提速. [tu大则易出现量化涂抹](https://forum.doom9.org/showthread.php?p=1963250" \l "post1963250), 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0~2, 默认关, 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造成涂抹太强画面糊掉的结果

--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. 若找出的信息不足, 参考帧与动静态隔离分块的建立就欠缺基础. 图: [Jain&Jain/十字搜索](https://typeset.io/pdf/displacement-measurement-and-its-application-in-interframe-3me36ivf7b.pdf), [大小菱LS-Dia搜索](https://www.researchgate.net/publication/254040728_Review_of_energy_efficient_block-matching_motion_estimation_algorithms_for_wireless_video_sensor_networks/)

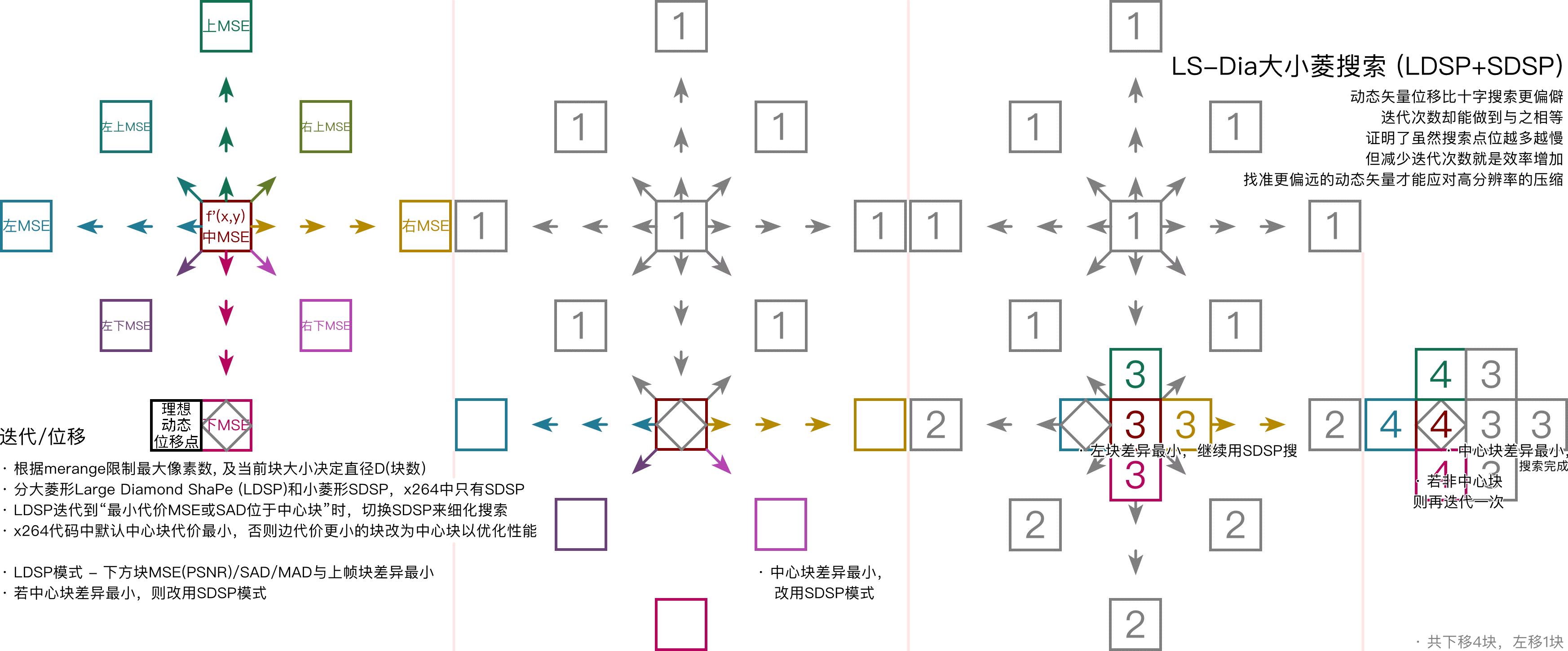
六边形hexagonal搜索将LDSP的上下左右斜, 共8个外点砍成6个, SDSP细化规则不变, 相比LS-dia和SSSP-LDSP-SDSP(四角星me star)在小范围里更高效

--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](https://www3.ntu.edu.sg/home/ekkma/1_Publications_files/A%20NEW%20STAR%20SEARCH%20ALGORITHM%20FOR%20FAST%20BLOCK%20MATCHING,%20International%20Workshop%20on%20Very%20Low%20Bit%20Rate%20Video%20Coding%20(VLBV'98),%20Urbana,%20Illinois,%20USA,%208-9%20October%201998,%20pp.%20173%20-%20176..PDF), sea是优化过的esa穷举, 浪费性能

--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, 耗时增加压缩

# 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🡪频谱的关系所同. hevc标准要求使用7-tap精度(1/4像素), 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. 分四范围. 由于动漫片源制于24~30fps, 因此可调低; 但同是动漫源60fps的虚拟主播则异. 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 |
| 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, 亮度变化失去同步的瑕疵. 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. x265默认开weightp

--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. 略提高压缩和画质

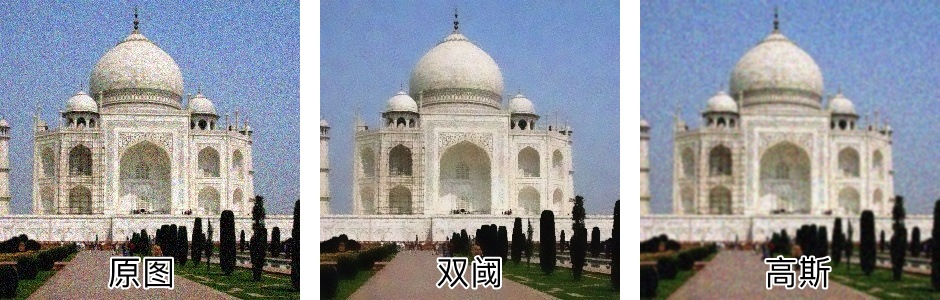
# 帧间-时域架网搜索

motion-compensated temporal filter基于快而简单的提升式小波变换lifting-scheme. 只需前后信息通过1-2之差来预测2-3的区别, 出的残差更新到高频分区, 没预测错的更新到低频分区. 帧间就是照奇偶奇帧推演, 从gop开始到末尾的0-1-2, 2-3-4, 4-5-6帧组成第零层奇偶奇网络, 通过分离的低频L带和高频段残差H带是第一层网络. 一层的L带再重组为0-1-2, 2-3-4等计算奇-偶-奇推演, 得到更远的动静态区间, 即LL带, LH带的第二层网络. 该结构做到了迭代n次即分离2ⁿ帧的动静态, 以及所有的预测与残差, 故不像传统动态搜索一样受分辨率大小的缩放性scalability限制. 是svc的核心算法. 非常依赖转场计算

**一维小波变换1D-WT**让短波像拉链一样划过一维信号, 此外和1DFT一样得到频域信号-热度图的变换算法. 解决了傅里叶变换只有空-频域, 把一整个信号变换而不能描述变化过程的问题, 还可以像拉链一样更换波形. 但缺点是分辨率低, 可编辑性差. 加上mctf对预测和残差的追踪后就可以分析视频了

--mcstf

<开关, 要求32GB内存? 会关闭多线程>mctf帧上应用空域双边滤镜, 是迭代到最后的L帧?

~~~~**高斯模糊**是设定范围阈值-模糊波形圆周大小来决定强度的滤镜. 比方块模糊柔和, 在媒体和设计上常用. [**双阈滤镜**](https://www.geeksforgeeks.org/python-bilateral-filtering/)bilateral filter加了最小纹理强度的判定, 只模糊振幅超阈的像素. 所以瞎起名叫双阈

# 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. hevc与avc一样用ref参数于时域上划区, 逐pu创建List0和List1左右两排参考列表. 差别是hevc在avc direct auto的基础上升级了**高级向量预测AMVP**; **并合搜索merge mode**两种方案. 其中adv. motion vector prediction的任务是找出向量信息, 分以下几步:

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, 所以可看做是打下手的

--ref

<整数1~16, 推荐fps÷100+3.4>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

--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), 三来搭建NALU为基础传输ss的网络串流架构. 过程是按IDR间隔keyint分区🡪scenecut分配额外关键帧🡪gop内的帧拆为条带slice🡪条带拆成条带段ss以降低解码错误的影响

* 视频参数集video parameter set
* 序列参数集sequence parameter set(分枝-负责播放时间戳, 显加权与其它特定解码要求)
* 图参数集picture parameter set(分枝-负责解码信息)
* 条带段slice segment(分枝-负责防止ctu中的错误传播到整个条带, ctu以上最小的单位)

--opt-qp-pps

<开关, 默认关>据上个GOP改动当前PPS中默认的qp值.

--opt-ref-list-length-pps

<开关, 默认关>据前GOP改当前ref值, 而且是前后帧独立改动. <暂停推荐: 播放器默认PPS跨GOP不变, 造成解码兼容性错误. [应该用`hev1`而非`hvc1`封装进ISO-BMFF](https://forum.doom9.org/showthread.php?p=1978837)?>

--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>设x264/5设I帧的两帧差距触发转场阈值

--hist-scenecut

<开关, 默认关, 推荐开>亮度平面边缘+颜色直方图SAD阈值触发转场. 在x265v3.5+69后在彩色视频中超越了近20%,尤其是[HDR源](https://forum.doom9.org/showthread.php?p=1978502)比scenecut降低了正误判(设I帧)和负误判(不设I帧), 因此[除黑白视频外](https://forum.doom9.org/showthread.php?p=1978737)推荐. 12bit源下会导致编码中途失败, 目前由于没有配套的压缩逻辑, 所以提高转场精度对于压缩和提速的效果不明. (hist-threshold于x265 v3.5+69被移除)

## 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(¬\_¬)ﾉ

**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╮(─▽─)╭

## 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 skipping/decoding lead**

1. 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>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. 两种选择,给出画质一样

* 设5或更高, 省了设立一些IDR帧拖慢速度. 快速编码/直播环境直接设=keyintヘ(>\_<ヘ)
* 设1来增加IDR帧, 一帧被判做转场本来就意味着前后溯块的价值不高. 而P/B帧内可以放置I宏块, x264会倾向插P/B帧. 好处是进度条落点在激烈的动作场面更密集, 画质编码用

--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1.3>P, B帧的qp值待遇差(如B块至少是P块1.3倍qp). B帧双向参考能从更多帧中找到参考源, 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0~16>最多可连续插入的B帧数量. <3~5>快, <8>电影平衡, <12左右>正常, 若播放设备配置偏高的话可放心的设在<13左右>. bframes>8+keyint>250会大增内存占用

--b-adapt

<整数默认1快, 推荐2精>B帧适应性算法

--bframe-bias

<整数-90~100, 推荐默认>设B帧阈值偏移, 增大适合压缩放B帧收益大的片源, 如桌面录屏和低成本动漫. 调整此值不会带来性能优化

# 帧内编码

组成参考源(I帧)+参考帧的帧间结构后, 数据会集中到I帧上, 其中I块间还可以进行冗余. 分为待编码, 待冗余的预测块PB和时间顺序前已编码的CB. 由于PB的位置不一定理想, 所以下图①处中缺EDAB块就拷F块右上角的像素; 不巧没有CB就拿像素中值替代

首先进行数据平滑预处理: ②处的**3-tap滤镜**也叫(1 2 1)÷4滤镜, 模糊横纵轴的参考源; ③④处的**强力平滑滤镜**从最远两参考源线性插值出每个预测像素值p(x,y)对应的参考源, 强力平滑滤镜会自动在DC, 垂直, 横向或角度近似垂/横的情况跳过, 目的是节省算力. 准备完随即进行帧内编码, 即用**趋平**/**夹角**/**DC**模式中之一编码PB

**趋平**代表双线性插值bilinear interpolation, 即横, 纵向地过渡出平面. 下图中横向底×高+底×高就有了三角形随v(x,y)这条线而变化的面积; 此时下图中再除以PB的长度/三角形底就插值出了横线预测值h(x,y); 位于t(x)上的纵线则所同地插值出v(x,y); 双线预测取平均即新p(x,y). 而算错可能会导致块失真

**夹角**处理斜着对齐的参考冗余, 直角三角形的远边长除以临边长opp÷adj=tanθ就得出了夹角θ, 而在仅知道tanθ的情况下再乘以新临边的长得出新远边的长, 也就得到p(x,y)确切参考位置了. 如果参考点落在两个参考源像素间, 则根据长边hyp投影对比两参考源间的加权插值穷举出最接近的像素为参考结果. 为了省算力, 一般是不会算完33种角度θ的

**DC**即直流, 此处代表以CB和当前PB的平均判断为平面

--fast-intra

<开关, rd>4时关>5个夹角跳着判断夹角模式. 纹理复杂时最有效.

--b-intra

<默认关>B分片也查帧内格式, 适合高压编码

--constrained-intra

<默认关, 实验性>帧内条带不参考帧间像素. 高压编码减少误参考

--no-strong-intra-smoothing

<开关, 不推荐>32x32的PB禁用强力平滑滤镜, 改用3-tap. 因筛选条件苛刻, 同时平滑的是参考源而非pu, 所以难以影响画质. 没64x64是因为pu最大仅32x32

# 量化-码率质量控制模式

对数log(x)

是为方便指数上套用乘除法交换关系, 即路程=速度×时间关系, 构成9=3², 3=²√9, 2=log₃(9)对应6=2×3, 3=6÷2, 2=6÷3关系的人造算法. 量化过程中的qp值是一条㏒(x)曲线. 主要体现在qp18~24之间画质vs体积间的变化差距相比其外的值更小, 留出一点压缩空间

图: 紫qp- qScale, 黑qScale-qp的对数映射

由于当前帧未编码, 所以只能用之前的编码帧预测复杂度, 因此叫推演复杂度/模糊复杂度. 而量化是有损压缩, 所以复杂度由当前量化值下, 已编码帧的失真程度推演, crf/abr高低决定复杂度而反直觉. 所以cplxBlur的精确度一般取决于帧率. 虽然基本上不准, 但qp波动变小, 码率变化更稳定

|  |  |
| --- | --- |
| **cplxSum:** 0.5 cplxSum[上帧] + SATD[上帧] | <迭代值, 初始=半数宏块÷CTU, 逻辑是SATD[当前]=SATD[旧]+SATD[当前]>~~查不出为什么要÷2~~ |
| **cplxCount:** 0.5 cplxCount[上帧] + 1 | <迭代值, 初始0, 当前帧数计数, 用于逐帧对cplxBlur加权>÷2是与cplxSum同步, 加权是为了推演越往后参考冗余越好, 压缩越强的规律 |
| **cplxBlur:** cplxSum ÷ cplxCount | <迭代值, 据帧所处位置推演加权的新SATD>接近100%则说明当前帧复杂度应当往高推演, 画面复杂度处上升趋势. 可扭转cplxCount默认的下降趋势 |
| **qScale:** 85% (2^((qp - 12) ÷ 6)) | <迭代值, 已编码旧帧qp转qScale, 方便其它参数修改更新> |
| **ABR\_rate\_factor:**  target\_rate\_window ÷ cplxSum | <迭代+初始值, abr下的qScale转qp> |
| **ABR\_newqScale:**  qscale × overflow ÷ ABR\_rate\_factor | <迭代结果, 据abr控制更新一遍qScale> |
| **cplxBase:**  ctu\_count × (bframes ? 120:80) | <恒定值, crf模式默认的复杂度. 若设bframes则CTU或宏块数量×120, 否则×80. 注: a?b:c是C++中if a:b else:c的缩写> |
| **CRF\_rate\_factor:**  cplxBase^(1-qcomp) ÷ (qScale×(crf+cutree&bframe\_offset)) | <多参数值>^(1-qcomp)是为和CRF\_qScale对齐, 因为运算仅在cplxBase, cplxBlur上而非整个算式中发生. 最终经cutree, b帧偏移再乘进qScale后得到实现质量-码率控制的值 |
| **CRF\_newqScale:**  cplxBlur^(1-qcomp) ÷ CRF\_rate\_factor | <据crf控制更新一遍qScale> |
| **newqp:** 6log₂ (qscale\_new ÷ 85%) + 12 | <目标值>qScale经调整后算出当前帧qp |

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), 所以还有率失真优化量化的必要

### CRF上层模式

--crf

<浮点范围0~69, 默认23>据"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24. 由于动画和录像的内容差距, 动画比录像要给低点. 理论上crf高=量化损失多, 率失真优化也就越慢; 但测试出来是crf+2, 4k4:4:4 12bit会快~0.5fps

--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>cplxBlur迭代值每帧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, mb-cutree, 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, mb-cutree, 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)

--rc-grain

<开关, tune grain时开启>通过cplxBlur抑制qp判断被噪声带偏, 胶片颗粒片源用

--cplxblur

<浮点0~100, 默认20>第-1帧不存在, 无法算出第0帧的cplxBlur所以直接指定

--rc-lookahead

<整数, 范围1~250>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

×rceq

<仅x264, 字串, 默认cplxBlur^(1-qComp)>可以少算一步qcomp, cplxBase还需qcomp, 但不写qcomp就是推荐的0.6, 所以不用写也行

### 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, 以及

--pbratio

<浮点, 默认1.3>B/b帧相比P帧的偏移. 如要求IDR/I帧qp17, P帧qp20, B/b帧p22时填写--qp/crf 17 --ipratio 1.1765 --pbratio 1.1. ratio即比率，如I帧qp10, 那么ipratio 1.4就代表10×1.4=14, 就得到P帧qp14了

### **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[-\_-]╠

### SBRC-CRF上层模式:

分段式率控制segment based rate control是为视频平台实现DASH, M3U8串流用的功能

### --sbrc

<启用分段式率控制, 要求min-keyint=keyint, no-open-gop>由于提高了初始crf值的利用率, 所以建议搭配--cplxblur=crf使用

### SBRC-CRF-VBR双层模式:

SBRC-CRF基础上增加两个VBV参数, 同时在新分段上重设VBV缓存

### SBRC-ABR上层模式:

最适合直播推流, 就是把CRF模式下的参数换成bitrate

### **2pass-ABR双层模式**

先用crf模式分析整个视频总结可压缩信息, 后根据abr模式的码率限制统一分配量化值. 有pass 2给特别高的平均码率, 输出最小损失的最小体积近无损模式, 以及pass2给码率硬限的全局整体压缩模式

### --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

--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, --analysis-load

<"文件名">指定导入/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( ^..^)ﾉ

--dynamic-refine

<开关, 已关闭>自动调整refine-inter, x265官方建议搭配refine-intra 4使用, 相比手动设定提高了压缩率

--refine-inter

<整数0~3, 默认0>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0~4,默认0>限制帧内块的向量格式, 取决于pass1分析结果是否可信

* <0~2>同上, <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 (ﾟ-ﾟ；)ノﾞ

--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analyse-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(内容已落后, 待更新)

--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-N**pass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](https://streaminglearningcenter.com/blogs/the-evolving-encoding-ladder-what-you-need-to-know.html)/官方表示bug已修复, 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模块save/load部分:):[analysis-load](#_analysis模块save/load部分:)] <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

# 率失真优化-宏观码率分配调优

### --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设<2.3~2.8>给中低成本动漫, <3~4.8>给电影-高成本动漫, <7~12+psy-rd 3, tskip, tskip-fast, ipratio 1.2, no-sao>留噪. 保留的细节面积越小设得越高

* 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

### --nr-intra

,

### --nr-inter

<整数0~2000, 默认0关>基于运动补偿mc给变换后-量化前降噪变换结果的参数. 其中帧内intra不如VapourSynth里的高端降噪滤镜. 但帧间/时域上降噪的nr-inter和x264的hqdn3d类似, 可以拉近参考源和参考帧的差距/残差, 实现在rc-grain上进一步稳定qp计算, 且在噪点源中相比模糊掉纹理更容易破坏噪点, 结果类似双边滤镜. [1920x1080下最高不建议超过250](http://forum.doom9.net/showthread.php?s=81cd7c04a679401fe7c4e689e67eaeb8&p=1947571#post1947571)

# 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>aq只在码率不足以还原原画时启动, 建议<1>标准(简单平面); <默认2>+启用variance调整aq-strength, 适合录像; <3>+欠码时码率多给暗场些; <4>+让不足以还原原画情况的码率多给边缘些(高锐多线条多暗场少平面)

--hevc-aq

<开关, "aq-mode 5">以1/4-tile而非边缘适应量化. 据测试[1](https://forum.doom9.org/showthread.php?p=1925373#post1925373), [2](https://forum.doom9.org/showthread.php?p=1925464)比aq 4快且适合动漫, 而aq 4更适合录播(?)目前学术方-官方-第三方间信息割裂, 所以暂无适解

--aq-strength

<浮点>自适应量化强度. 据VCB-s建议, 动漫的值太高则浪费码率. 动漫中aq-mode / 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, 需≥min-cu-size>最小支持自适应量化的cu. 默认64可换取更多速度. 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32,16代表仅这两种cu

--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

**x265 jpsdr-mod参数:**

--aq-auto

<对应下表8bit四开关的十进制, 默认0关>.

--aq-mode 5

<开关>启用aq-auto的条件值

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 值 | 逐帧aq | 延迟逐帧aq (hysteresis) | HDR兼容 | aq-mode 5 |
| 1 | √ |  |  |  |
| 2,3 | √ | √ |  |  |
| 4 | √ |  | √ |  |
| 8 | √ |  |  | √ |
| 6 | √ | √ | √ |  |
| 10 | √ | √ |  | √ |
| 12 | √ |  | √ | √ |
| 14 | √ | √ | √ | √ |

--aq-fast-edge

<开关, 需aq-mode4,5>边缘检测跳过高斯模糊过滤, 不适合脏片源

--aq-bias-strength

<浮点, 默认1, 需aq-mode3,5>aq-strength偏给暗场的程度

--aq-strength-edge

<浮点0~3, 默认=aq-strength, 需aq-mode4,5>偏给纹理的aq-strength

--aq-bias-strength-edge

<浮点, 默认=aq-bias-strength, 需aq-mode5>aq-s-edge偏给暗场的程度

# 模式决策

mode decision整合搜来的信息, 宏观上重新定制分块, 参考, 跳过, 编码, 量化方案. 因为选码率最小的压缩方案不平衡, 容易被吐槽画质

--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>建议快速用1, 2; 高压用3, 5比3慢上一倍, 原画画质一般还难以看出提升

--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的效果, 明显提速. 会增大或减少体积, 微降画质但难以察觉

--limit-refs

<整数0~3, 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

<整数0~2>前cu分块被跳过时, 判断后cu接着搜索分块还是提前退出的参数. 画面越接近录屏, 低成本动漫就用得越多

* <0>继续分析. 适合信噪比差, 噪声高的源. 原画很干净则不如<1>
* <1>rd0~4下据临cu是否细分而定; rd5~6下看附近2Nx2N cu分块难度而定, 高压和一般情况推荐
* <2>直接对比cu纹理密度edge density, 快且不比前者差, 但存在对源的画质要求及客观判断"画质"能力的要求

--rskip-edge-threshold

<0~100默认5: 趋向于分块, 需rskip大于1>用sobel算法获取cu纹理密度, 除以块所占面积的百分比值. 纹理密度>阈值=分块. 量化强度越高越关键, 与x264 deadzone略像. 8×8或16×16块下默认5%即含3或12个系子就分

* <像素风>据像素变大的程度决定. 如画面除以2回到1x1像素大小, 则使默认值乘以2以提速
* <抗涂抹>rskip 2 rskip-edge-threshold 3, 即"有一点不平就分块". 比rskip 0快, 用于已知要保留雪景等全屏大量动态信息的源的情况下, 节省传统分块计算时间用. 可以在不添噪点的情况下达成其抗涂抹的目的

--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据多码率下测得的失真度(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用均差异平方MSE判断:

x264 总差异平方

D·Shikari 高频加权总差方

* 代表块宽度求和范围, f()和f'()分别代表参考块和参考源
* 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
* MSE相比SSE精度有提升, 但多算一步平均使其更慢; Noise SSE更强调高频信号(细节纹理或噪声)的权重

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)

--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. 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, 有瑕疵bug>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. 有斜面上随机出现块失真的bug, 若图中失真可接受则可开:

--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

--qp-adaptation-range

<浮点1~6, 默认1>心理视觉优化调qp的最大范围, 信息不足且没有负值: 高则提高而不是降低qp?

## 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=10log₁₀

(最大÷当前). 最大÷当前即最大像素值

÷MSE

; 8bit下

256÷MSE

; 10bit下则

1024÷MSE

# 环路滤波-去块滤镜

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

### 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

<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>

# 环路滤波-取样迁就偏移

sample adaptive offset滤镜. 逐CTB划分. 界偏移eo缓解纹理边缘因"高频波形的遮盖因强量化或去块丢失"的问题. 可能分以下几步. 算法略像帧内搜索的趋平滤镜, 但是是强制的. 适合修复纹理量化出振铃的损失. 关闭则遮不住平面的蠕虫失真

* CTB内据分块结果建立一批像素大小的3x3采样域
* 由3x3排列规律找出三个像素排成一排的横0, 纵1, 左倾2和右倾3四种可能
* 中心像素的值同时小于旁像素, 同时大于旁像素, 等于任意一边像素的条件, 否则视作非边界
* 确保3x3间的边界连续性, 实现边界验证和性能优化(?)

带偏移bo是一种对比源+补偿编码差异的平面CTB滤镜. 方法是1.划分32条色深带来均分当前色深下的像素值, 2.分为24~31, 0~7的明带, 暗带; 以及8~23的中间带. 限制最大只能补偿4条相连的色深带, 以确保CTB中色深差异不会大到触发eo, 同时涵盖足够大的斜面渐变, 然后率失真优化找出所谓的补偿值: 每条色深带的偏移值. 因此适合补偿平面, 斜面和曲面渐变CU, 适合修复平面量化的损失

* 共有0=无, 1=横E0, 2=纵EO, 3=左倾EO, 4=右倾EO, 5=中间带, 6=明暗带, 共7种补偿方案
* 共有0=无, 1=Y, 2=Y+Cr, 3=Y+Cb, 4=YCbCr, 5=Cr, 6=Cb, 7=CbCr种平面补偿开关

**参数融合merge**将相邻两个CTB的sao信息(补偿方案, 平面补偿开关等)根据参数决定直接用上/左块, 还是对比像素趋势更接近哪个. 和选择bo, eo具体的偏移值一样由率失真优化决定

### --no-sao

<关闭--sao, 默认开sao>由于针对的是强量化环境, 所以高画质源+crf<17的情况下可以关

--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, 默认关>对一些计算采用提前退出策略, 不是改善画质的, 但crf≈18, cutree和bframes 16下可以开, 以保留一定影响

--selective-sao

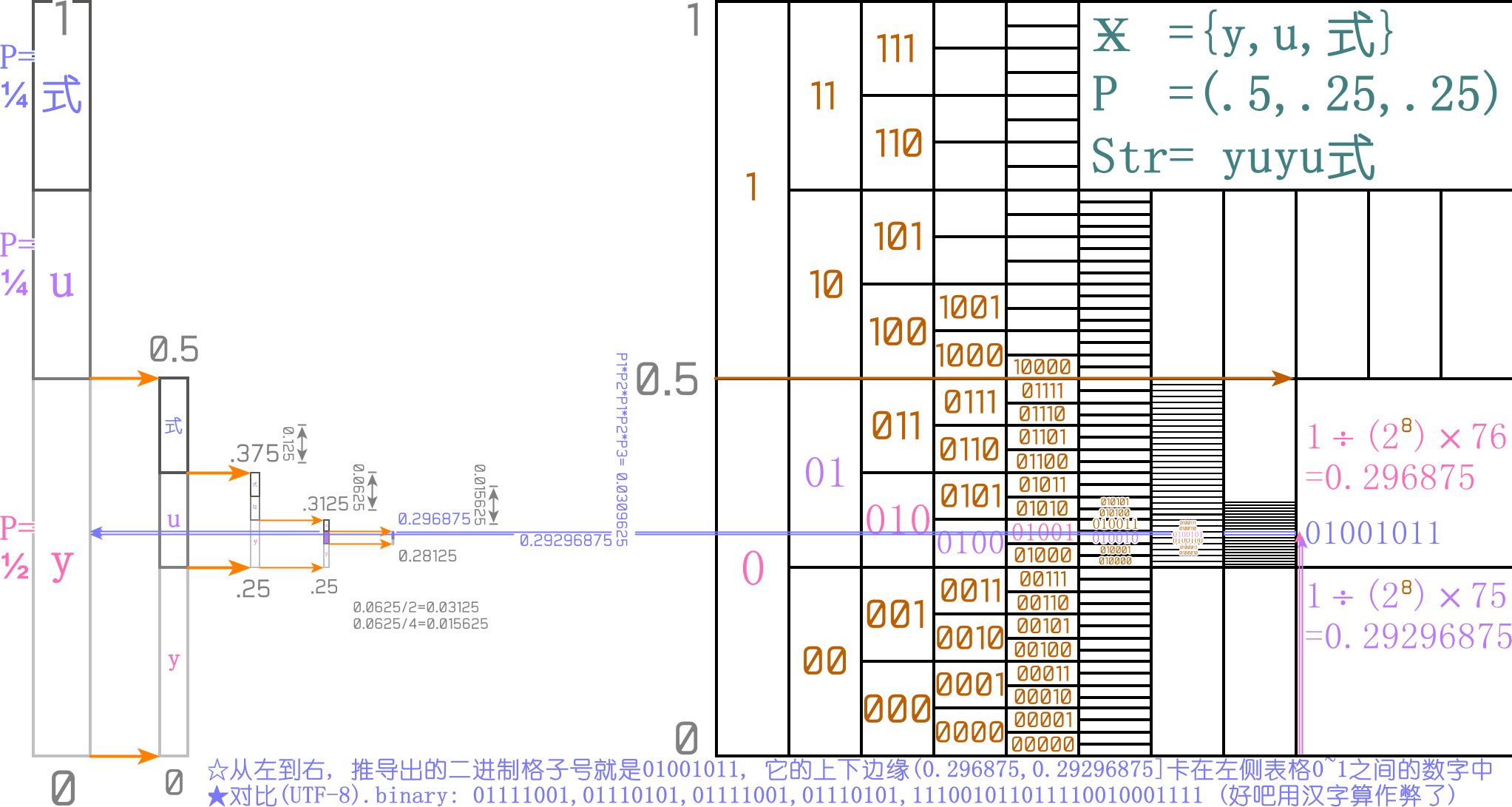
<0~4, 默认0>从条带(横向一组CTU)角度调整sao参数, 1启用I条带sao, 2增加P条带, 3增加B条带, 4所有条带. 可看作新的sao控制方式, 或搭配limit-sao的新方法

# 熵编码/残差编码/文本压缩-CABAC

游程编码将降维后的块/条带丢给熵编码(率失真优化要解码检查每道压缩, 所以要经历多次熵编码). x264/5中用context adapt. binary arithmetic coding. 相比于cavlc和霍夫曼编码, cabac的压缩率更好.

二进制算数编码

bac在0~1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𝔛={y, u, 式}三种符; 出字yuyu式中的概率P=.25, .25, .5故照下图规律, 经8层细分后给出二进制格子里符合条件的小数压缩:



在此基础上增加上下文词典编录能力, 即根据游程编码给的块, 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, 每个阵列适配0~1模具, 就是所谓的cabac了

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压缩~~但是我不想算饶了我吧~~. 但霍夫曼编码的精度会被分支一次只能÷2的限制挡住, 所以尽管自带词典, 但因为压缩率根本比不上所以被淘汰了

# SEI维稳优化消息

supplemental 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, 默认关, 开vbv>开启后将假设对照解码参数hypothetical ref. decoder param. 在无丢包和延迟的假想下算好瞬间码率, 写在每段序列参数集sps及辅助优化信息sei里, 对专门配置了网络串流, NAS播放自动缓冲的播放器有好处? 但应该没啥用

--hash

<整数0~3分别代表默认的关,checksum, crc和md5>sei里加效验码, 播放时可用以对图像重建纠错来减少失真, md5播放所需算力较高, 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, 提高很小一点压缩率

--film-grain

<文件名>将如[libfgm](https://bitbucket.org/multicoreware/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vbv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

参自[pugetsystems.com](https://www.pugetsystems.com/labs/articles/amd-ryzen-5800x3d-vs-5800x-for-content-creation-2331/): Cinebench, 虚幻5上5800X比5800X3D快近10%反映了持续计算下5800X频率更高的优势; Lightrooms等单图处理上5800X3D跑过了5800X, Photoshop上5900X, X3D, X同时打平(优化?); 剪辑视频并非持续高负载, 因此AfterFX，Premiere，DaVinci上X3D和X打成平手. 所以选择时首先看生产力软件的优化, 其次看任务类型. 3D缓存处理器比同型号更慢的原因是目前视频编码对内存带宽需求不大, 更高频率才是优势. 因此压制的视频越长性价比越低. 但中间若有一堆内存读写间歇, 3D缓存处理器就能用低延迟读写赢回速度

--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

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
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 parallel 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, 默认关. [任意年代>=16核处理器](https://forum.doom9.org/showthread.php?p=1963250#post1963250)>使用平行动态搜索parallel ME. 解决了单核性能不足导致me后的编码步骤等待, 占用降低速度变慢的问题. 让旧服务器超多核中低频处理器变得更有编码性价比. 而达到16核的处理器也能提速

--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

纯元数据, 写错或忘写也可以改. hdr应用早期因制定方猜不出元数据主次, 所以制定主master-display, 次maxcll. 但现在...hdr电视只读取"次要"的maxcll和maxfall, 并且忽略"主要"的master-display. 另外光强/光压的单位是candela (1cd=1nit)

由于bt601，bt709，HDR-PQ，HLG标准所重用的亮度范围不一(偏亮或偏暗), 所以在编码，心理学优化算法，编码器参数上其实都要调整适配, 所以出现了适配不当的可能

--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, 搞得这么麻烦是因为hdr作为新标准不敢确定播放硬件需要什么信息, 所以就把master-display写成必须参数了. 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)*

--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, 例子见图: max-cll 1000,640. master-display由 G(13250…开头, L(10000000,1)结尾, 共7个0

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

--hdr10

<自动开关>当master-display, max-cll启用就在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

**--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, smpte428, smpte431, smpte432. 如图→为bt.2020

**--colormatrix**

<字符>播放用矩阵格式/系数: fcc, bt470bg, smpte170m, smpte240m, GBR, YCgCo, bt2020nc, bt2020c, smpte2085, chroma-derived-nc, chroma-derived-c, ICtCp, 不支持图↑的bt2020nc

**--transfer**

<字符>传输特质: bt470m, bt470bg, smpte170m, smpte240m, linear, log100, log316, iec61966-2-4, bt1361e, iec61966-2-1, bt2020-10, bt2020-12, smpte2084, smpte428, arib-std-b67, 上图PQ即st.2084的标准, 所以参数值为smpte2084

ffprobe会将三个信息并一行写: 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ᕙ(⇀‸↼‶)ᕗ

--force-flush

<整数0~2, 默认0>性能不足录屏用. 当编码不过来就跳过几帧的措施(?):

* <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x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)

--temporal-layers

<开关, 默认关>使x265更兼容svc标准, 将非参考b帧(相当于空信息)分离到另一层视频流中, 解码器可以选择跳过而降低性能损耗, 可能会造成兼容性问题

下载 附录与操作技巧

|  |  |
| --- | --- |
| [**LigH**](http://www.mediafire.com/?6lfp2jlygogwa) | .hevc GCC10 [单文件8-10-12bit] 附x86, Windows XP x86版 附libx265.dll |
| [**jpsdr**](https://github.com/jpsdr/x265/releases) | .hevc GCC12.2+MSVC\_llvm 1928 [8-10-12bit] 附Broadwell版 支持aq-mode 5 |
| [**Rigaya**](https://drive.google.com/drive/u/0/folders/0BzA4dIFteM2dWEpvWGZXV3ZhdTA) | .hevc GCC 9.3 [8-10-12bit] 附x86版 |
| [**Patman**](http://www.mediafire.com/folder/arv5xmdqyiczc) | .hevc GCC 11+MSVC1925 [8-10-12bit] |
| [**ShortKatz**](https://forum.doom9.org/showthread.php?p=1937773#post1937773) | arm64~64e加x86版 [?] 需macOS运行编译命令文件 ? |
| **[DJATOM-aMod](https://github.com/DJATOM/x265-aMod/releases/)** | 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 |
| **[MeteorRain-yuuki](https://down.7086.in/)** | 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**](http://ffmpeg.org/download.html) 多系统兼容, 备用地址 ottverse.com/ffmpeg-builds | |
| [**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 | |
| [**x265GuiEx**](https://drive.google.com/drive/folders/0BzA4dIFteM2dRkRzWXZMT0lkM2M) **(Rigaya)** 日本語, auto-setup安装, [教程点此](https://aviutl.info/x265guiex/#toc4) | |
| [**Voukoder**; **V-Connector**](https://www.voukoder.org/)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(⇀‸↼‶)

**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的版本ヽ(･ω･ゞ)

**杜比视界dolby vision不深入研究**

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, 强化层可被一般的hevc解码器丢弃, 单视频流就只有私有解码器能播放. 作者认为dolby vision还不如光线追踪. 内容和设备生产方要各交各钱合作, 所以用户只能交大钱才能达到标准体验. 而多数用户因剧情而非试听购买设备, 所以更方便的平板电脑, 笔记本电脑还是追剧主力. 另外, 在本教程写出的几年时间里该技术如上述所料, 已经变得没人讨论了

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" (ﾉ･ω･)ﾉﾞ

**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

**Worm effect瑕疵** 原因未知, x265低码+no-sao可复现的噪点横向拉伸效果

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **--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** | |
| **合并提前退出early-skip** | **1** | | | **0** | **1** | **0** | | | |
| **cu再分裂跳过rskip** | **1** | | | | | | | | **0** |
| **帧内动态跳过fast-intra** | **1** | | | | **0** | | | | |
| **B带帧内搜索b-intra** | **0** | | | | | | **1** | | |
| **取样迁就偏移** | **关** | **开** | | | | | | | |
| **P帧权重** | **0** | **1** | | | | | | | |
| **B帧权重** | **0** | | | | | | **1** | | |
| **自适应量化** | **0** | **2** | | | | | | | |
| **cu树** | **开** | | | | | | | | |
| **率失真优化rd** | **2** | | | | **3** | **4** | **6** | | |
| **心率失优程度rdoq-level** | **0** | | | | | **2** | | | |
| **tu帧内/间上限** | **1** | | | | | | **3** | | **4** |
| **tu分裂上限** | **0** | | | | | | **4** | **0** | |

|  |  |  |  |
| --- | --- | --- | --- |
| **tune zerolatency去延迟** | | **tune animation动画片** | |
| **连续B帧** | **0** | **心理率失真优化程度psy-rd** | **0.4** |
| **B帧筛选** | **关** | **自适应量化强度** | **0.4** |
| **cu树** | **关** | **去块** | **1:1** |
| **转场** | **关** | **cu树** | **关** |
| **多线程压制帧数** | **1** | **B帧数量** | **<preset>+2** |
| **tune grain最高画质** | | **tune fastdecode解码加速** | |
| **自适应量化** | **0** | **B帧权重** | **关** |
| **cu树开关** | **关** | **P帧权重** | **关** |
| **I-P帧压缩比** | **1.1** | **去块** | **关** |
| **P-B帧压缩比** | **1** | **取样迁就偏移** | **关** |
| **QP赋值精度qp-step** | **1** | **tune psnr峰值信噪比** | |
| **取样迁就偏移** | **关** |
| **心理率失真优化程度psy-rd** | **4** | **自适应量化** | **关** |
| **心率失优可用psy-rdoq** | **10** | **率失真优化rd** | **关** |
| **cu再分裂跳过rskip** | **0** | **cu树** | **关** |