x265视频压缩教程**综合版c(x265-crf可能等于x264-crf值+4的问题暂未解决)**

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Word/浏览器/阅读器的"查找ctrl+f"功能, 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 4

--ctu

--min-cu-size

--rect

--amp

变换·(附录-傅里叶变换) 5

逆变换

--limit-tu

--rdpenalty

--tu-intra-depth

--tu-inter-depth

--max-tu-size

帧间-动态搜索 6

--me

--merange

--analyze-src-pics

--hme-search

--hme-range

帧间-子像素运动补偿 6

--subme

加权预测weighted prediction 7

--weightb

溯块向量搜索 7

--ref

--max-merge

--early-skip

GOP结构建立, 参数集 8

--opt-qp-pps

--opt-ref-list-length-pps

--repeat-headers

--scenecut

--hist-scenecut

--hist-threshold

关键帧 9

参考帧 9

--no-open-gop

--radl

--min-keyint

--keyint

--fades

--pbratio

--bframes

--b-adapt

帧内编码 10

--fast-intra

--b-intra

--no-strong-intra-smoothing

--constrained-intra

量化-质量控制模式 11

CRF上层模式

--crf

--qpmin

CQP上层模式

--qp

--ipratio

ABR上层模式

--bitrate

VBR下层模式

--vbv-bufsize

--vbv-maxrate

--crf-max

--crf-min

--qcomp

2pass-ABR模式

--slow-firstpass

Analysis-2pass-ABR模式

--analysis-save

--analysis-load

--analysis-save-reuse-level; --analysis-load-reuse-level

--dynamic-refine

--refine-inter

--refine-intra

--refine-mv

--scale-factor

--refine-mv-type avc

--refine-ctu-distortion

2pass转场优化 14

--scenecut-aware-qp

--masking-strength

--analysis-reuse-file

Analysis-Npass间调优

--multi-pass-opt-analysis

--multi-pass-opt-distortion

--multi-pass-opt-rps

Analysis-pass2-ABR天梯模式

--abr-ladder

近无损压缩, 真无损压缩上层模式

RDO控制量化; 率控制 15

--rdoq-level

--psy-rdoq

自适应量化 16

--aq-mode

--aq-strength

--aq-motion

--qg-size

--cbqpoffs

--crqpoffs

模式决策 16

--rd

--limit-modes

--limit-refs

--rskip

--rskip-edge-threshold

--tskip-fast

--rc-lookahead

--no-cutree

率失真优化RDO控制 17

率失真优化

拉格朗日值λ

--psy-rd

--rd-refine

--dynamic-rd

--splitrd-skip

峰值信噪比peak signal-to-noise ratio/PSNR 19

环路滤波-去块 19

--deblock

平滑强度 19

环路滤波-取样迁就偏移 19

锐偏移eo 20

带偏移bo 20

--sao-non-deblock

--no-sao-non-deblock

--limit-sao

--selective-sao

熵编码/残差编码-上下文自适应二进制算数编码 20

算数编码 20

SEI维稳优化消息 21

--hrd

--hash

--single-sei

--film-grain

--idr-recovery-sei

--frame-dup

--dup-threshold

线程节点控制 22

--pools

--pmode

--asm

多线程vs多参考

--pme

--frame-threads

--lookahead-threads

色彩空间转换, VUI/HDR信息, 黑边跳过 23

--master-display

--max-cll

--hdr10

IO(input-output, 输入输出) 24

--seek

--frames

--output

--input-csp

--dither

--allow-non-conformance

--force-flush

--field

--input-res

--fps

--chunk-start --chunk-end

Voukoder用法:

CMD操作技巧

预设(该板块有理论错误) 27

# 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(ò\_óˇ)

--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

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](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-高上每条线分别提出, 像素值变化视作波形跑1D-FT. ①源仅在高边上有特征, 宽边就得到零点值-无效. 频域的一对点表示一套波; 亮则振幅大, 远则频率高. ②③, ⑥⑦向外辐射的变换结果与原图形状无关, 而是那条线上有一套用余弦逼近的方形波而已; ⑥⑦灰背景是因为低幅余弦的参与. FT强在可编辑性, 是消除光盘扫图噪声的唯一解(PhotoShop官网教程只会用模糊), 但仅Mathematica之类的算软能用. 分辨率也非核磁共振最高的256², 而是Audition中置声道提取滤镜就能用的32768²; 丢分项是无法表示时间前后. 因此分辨率低但长度短的小波拉链变换随即被研发出来做音频的热度图了

**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<整数1~4, 配合limit-tu>空间域tu分裂次数上限, 默认1代表只能在cu基础上分裂一次. 值越高越慢,越隔离动静态. 建议日常编码设在1~2, 画质和高压编码设3~4

--tu-inter-depth<整数1~4, 配合limit-tu时间域tu分裂次数上限. 建议同上

--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. 建议直接设ctu, 因为也可减少32×32tu

# 帧间-动态搜索

由于噪声光线形变存在, 所以不能做前后块逐像素对比, 而是用SAD, MSE, MAD之类算法把块中像素值加一起比. 于帧间逐块地找最小失真朝向direction of min. distortion, 组成一张张帧间矢量表的计算机视觉技术. 若不到位, 参考帧的建立就欠缺基础. 目标是大范围快速地找准DMD

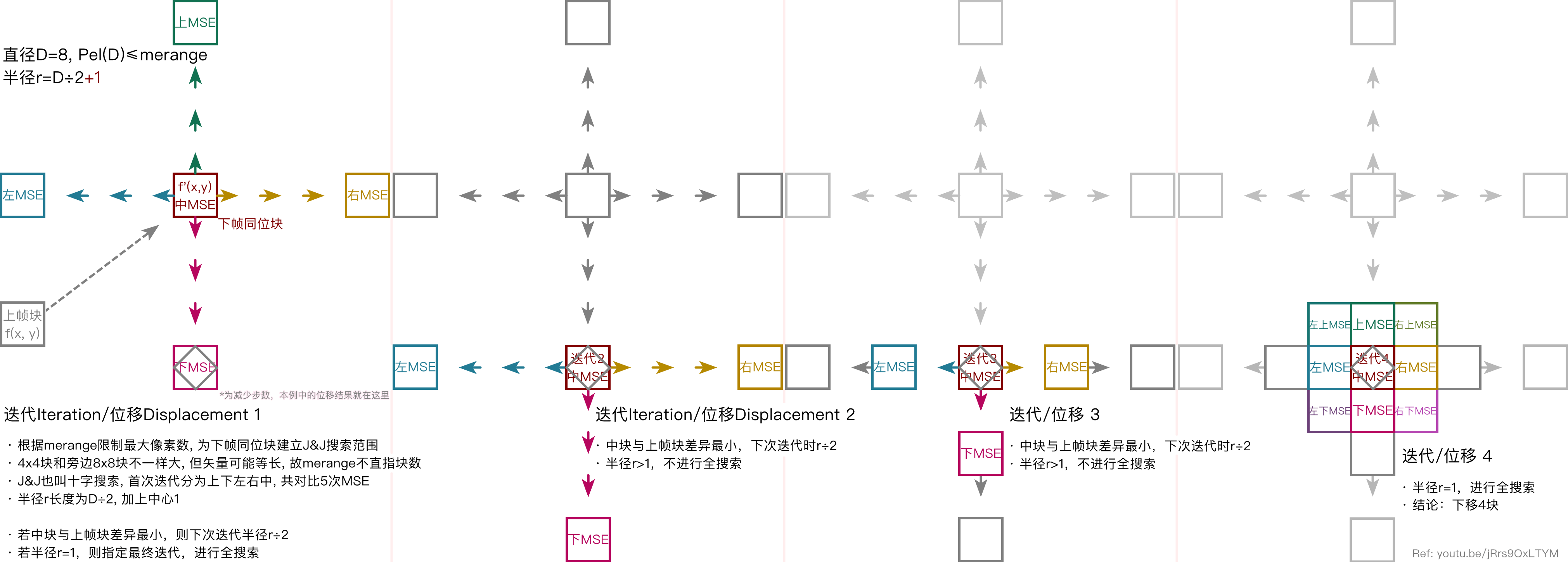


图: [Jain&Jain/十字搜索](https://typeset.io/pdf/displacement-measurement-and-its-application-in-interframe-3me36ivf7b.pdf)算法及[大小菱LS-Dia搜索](https://www.researchgate.net/publication/254040728_Review_of_energy_efficient_block-matching_motion_estimation_algorithms_for_wireless_video_sensor_networks/)步骤. 确立了定位到中心后逐步细化, 找出唯一块的算法, 优化了一开始全搜索的巨大开销. 注意大小菱搜索在位移更偏的情况下, 迭代次数与十字搜索相等, 证明了增加搜索点位才能用更少的迭代找准距离更远, 移动更大的情况. 相比大小菱, 六边形hexagonal搜索将LDSP的外围换成了六个边块, 还是一旦中心块处于差异最小条件下就用SDSP搜索来细化. 理论上在merange 16的范围里是最优解

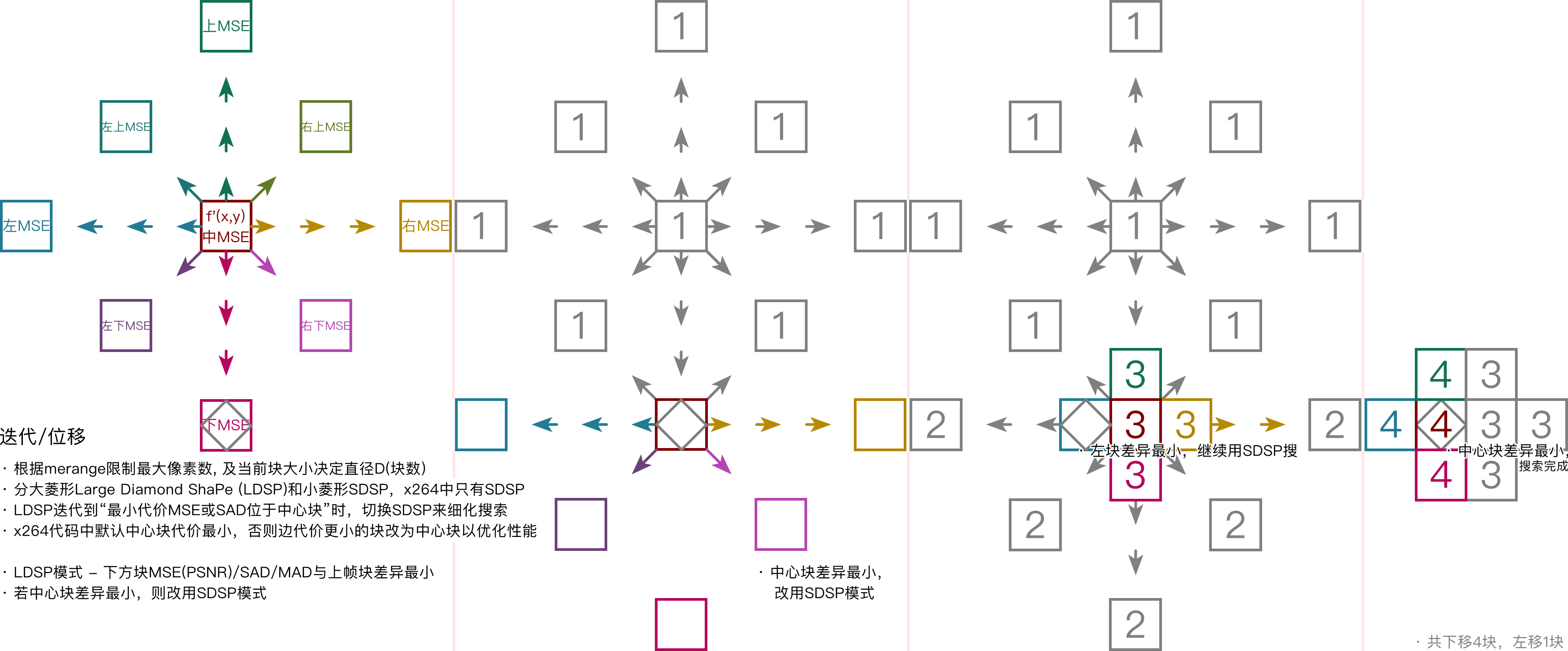
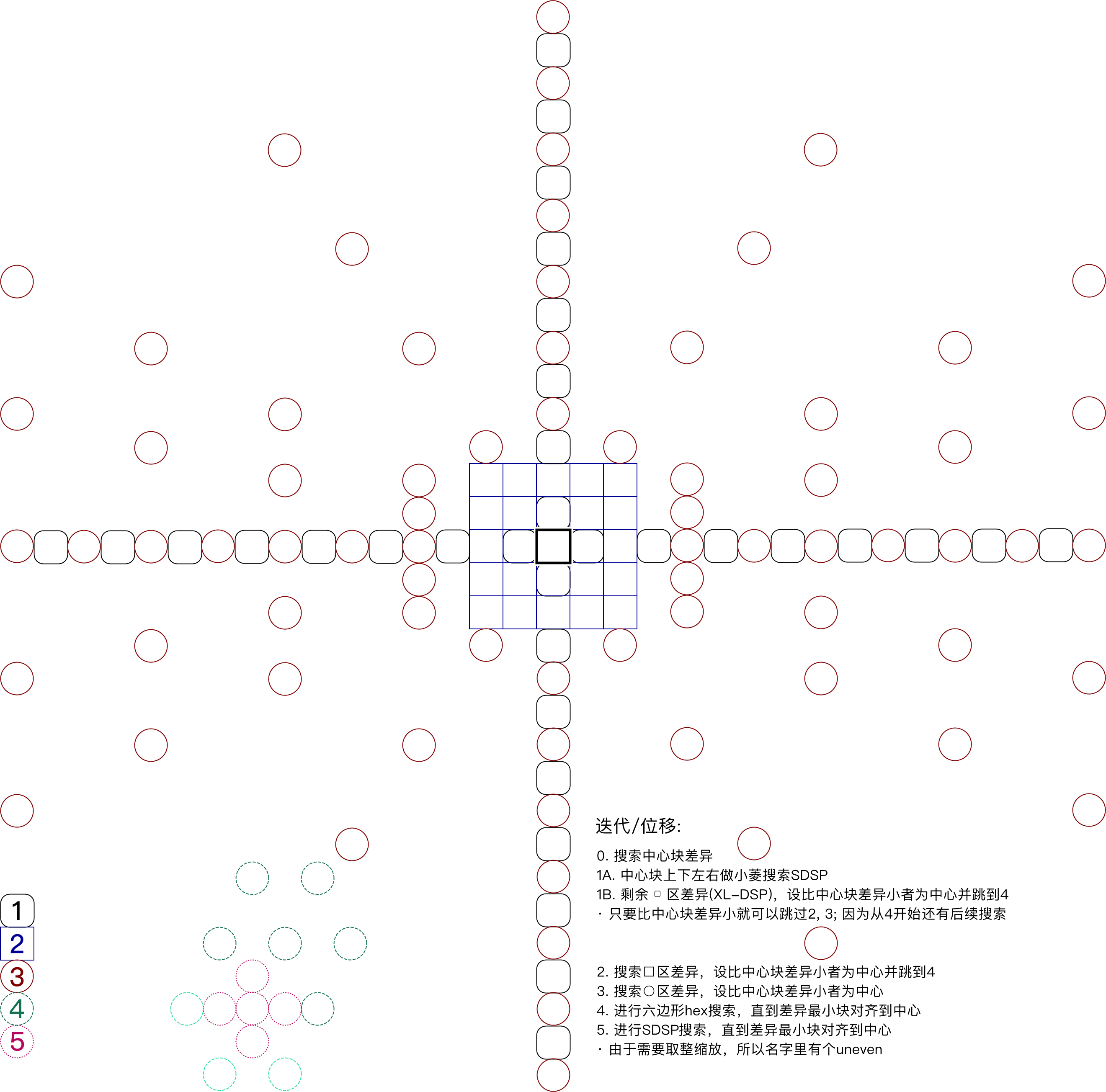


图: x264引入的不均匀多六边形Uneven MultiHexagon搜索, 用小-中-大-中-小的迭代策略. 慢但解决了远距追踪大动态的需求. 实现了1920x1080下, merange 32及更高的解, 还有点像宗教符号



--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( ﾟ▽ﾟ)/

<umh>x264经典(°∀°)b <star>[SSSP+LDSP+SDSP三步搜索](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>优化过的esa穷举: 超大杯 <full>全搜索: 费时, 水洒了

--merange<整数, 据动搜算法选>搜索范围, 简单说hex选16, umh选48以下, star选52以下; 然后根据视频画面动态最大的地方, 选上下两帧截图测量像素距离. 太大会同时降低画质和压缩率(找不到更好的, 找到了也是错的), 所以一般推荐--me umh --merange 48

--analyze-src-pics<开关>已关, 允许动态搜索直接搜索片源帧, 用更多时间换取码率的参数

--hme-search<hex,umh,star,sea,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算法科普见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

avc首发, 治了少数淡入淡出过程中部分pu误参考, 亮度变化不同步的瑕疵. 分为P, B条带用的显加权explict WP<编码器直接从原画和编码过的参考帧做差>与B条带用的隐加权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哦

--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的动态向量了ヽ(　￣—￣;)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就不管了ヽ(｀◇´)/

--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(¬\_¬)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 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gop爆炸

--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, 所以一般情况下默认相对平衡

反过来若有参考的价值, 也会因为P帧和B帧本身可以包含I宏块的原因而更可能插入P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

上段若VBV开启则不成立. VBV反而会开高量化将码率尖峰干掉, 画质反而会下降

--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(◑‿◐)

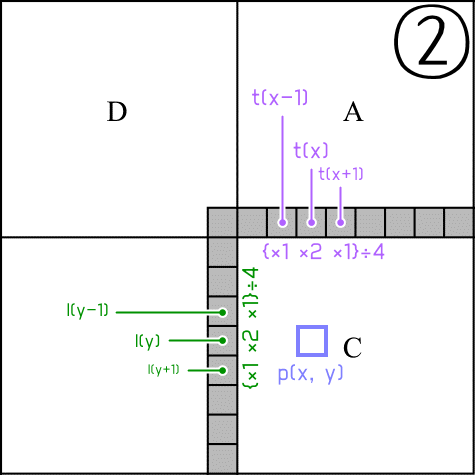
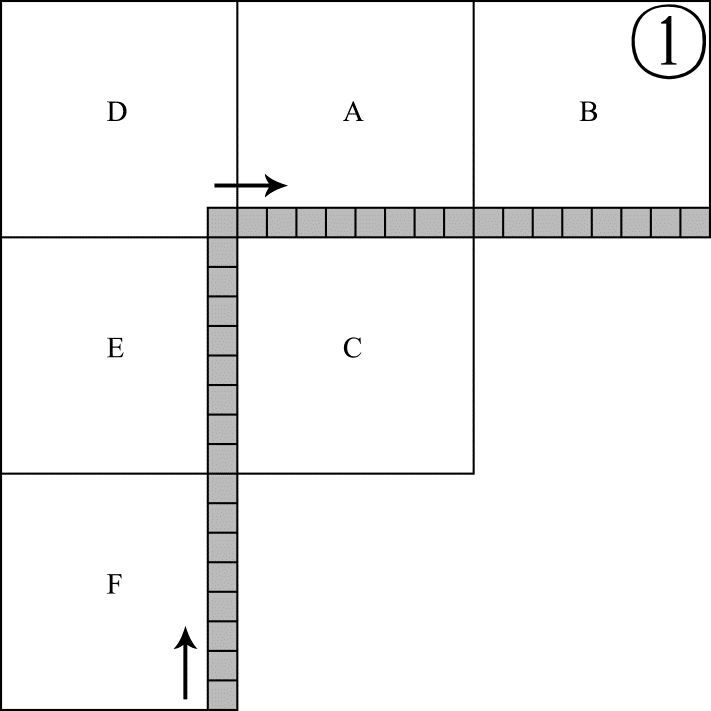
--fades<开关, 已关>找流中的虚实渐变fade-in, 给小到帧间条带(slice, 一组横向ctu), 整个帧间范围改用I条带, 并根据渐变后最亮的帧重设码率控制历史记录, 解决转场致模糊的问题

--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

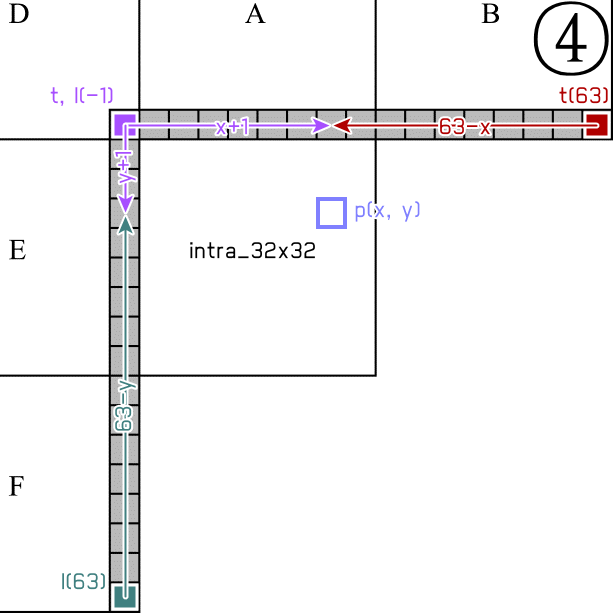
# 帧内编码



类jpg, png的单图编码. hevc标准下可略作补块, 平滑和编码三步. 图①: C块即待编码块PB, ABDEF块即已编码块CB, 灰像素即参考源, 箭头指代填补缺失参考源的顺序. 1.补齐因PB位置不好而缺的块. 图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. 平滑smoothing与模糊blurring区别在于直接砍高频也叫模糊; 而平滑让数据前后连贯好压, 且高频砍得少

**3-tap滤镜**如图②也叫(1 2 1)÷4滤镜, 按每个预测值prediction value, p(x, y)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

**强力平滑滤镜**如图③: ■~■, ■~■两差异的和, 及左列■~■~■差异之和要<8才准用. 原理如图④: 从■~■, ■~■线性插值出每个p(x,y)对应的参考源

**趋平模式**代表从左-上做双线性bilinear插值到右-下边形成DC平面. 具体见右图: 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**以角度φ为准. 分为横轴正角26~34, 横轴负角18~25, 纵轴正角10~17及纵轴负角2~9四大类. 由三角函数tan(φ)=远边÷临边, 再乘以临边得出p(x, y)在某角度下对应到横轴, 纵轴上的哪个参考源; 且大多情况下因角度差异, 角度落在两个参考源■间, 要判断该角度下p(x, y)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; 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**规定全部参考源的平均等于全部预测值平均(大平面)

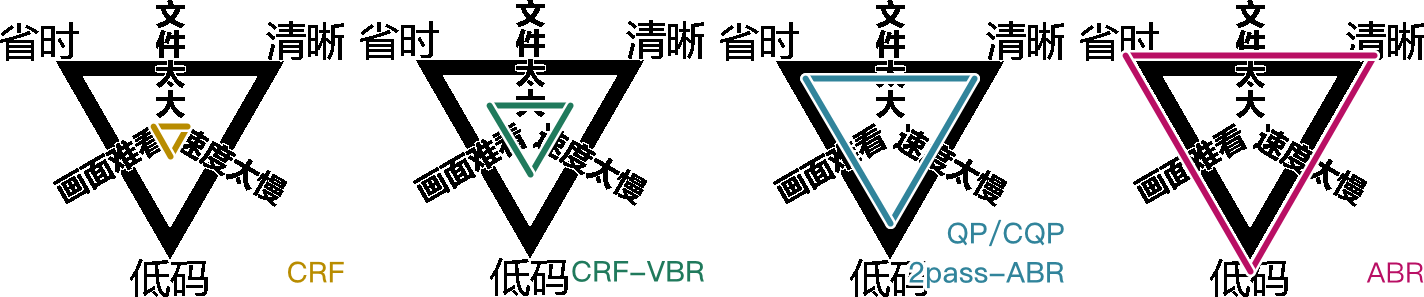
--fast-intra<开关, 默认rd大于4则关>5个夹角地跳着判断夹角模式. 理论上画面复杂才有效

--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



根据对短板容忍度及用途的不同选择适合的方式. 短板来自处理器, 及不同模式的完算性上的区别

### CRF上层模式

--crf<浮点, 范围0~69, 默认23>据情况给每帧分配各自的量化值qp, constant rate factor固定质率因子, 或简称质量呼应码率模式, 统称crf. 素材级画质设在16~18, 收藏~高压画质设在19~20.5, YouTube是23. 由于动画和录像的内容差距, 动画比录像要给低点

虽然相比于 x264的量化一样. 但 crf 越高，x265 要执行的优化计算也越多，速度明显越慢

--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 (\*~▽~)

### 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, 可能是西方人的反留白习惯所致, 目前--qpmin足以[-\_-]╠

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

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

### **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( ^..^)ﾉ

--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 (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-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天梯模式

--abr-ladder<实验性的[苹果TN2224](https://streaminglearningcenter.com/blogs/the-evolving-encoding-ladder-what-you-need-to-know.html)/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

# 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](https://sigport.org/sites/default/files/docs/poster_66.pdf), 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(;\*△\*;)

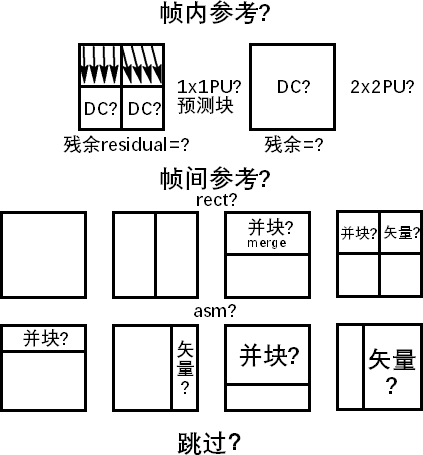
--qg-size<64/32/16/8>实验性参数. 自适应量化能影响到的最小cu边长/最大分裂深度. 默认64可换取更多速度, 减少可略增优化效果. 高画质/平衡都建议设在32~16. 用途不明的<最浅,最深>格式能自定义范围, 如32,8代表只在32, 16和8px的cu上起作用( •ω•́)σ

--cbqpoffs<整数>调整蓝色平面相比亮度平面的△qp值(三角指delta, 变动量), 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

--crqpoffs<整数>此为红色平面, 其余同上. 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没事就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. 码率vs速度vs画质的参数

--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

--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到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>关闭和x264 mbtree一样的功能. 只有近无损, 可能crf小于17才用的到

# 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.

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

(x264) Sum of Squared Err = ∑x=0🡪Tx ∑y=0🡪Ty |f(x,y)-f'(x,y)|²

(x264) Noise SSE (--fgo) = ∑x=0🡪Tx ∑y=0🡪Ty |[N(x,y)- N'(x,y)]·fgo| + |f(x,y)-f'(x,y)|²

* ∑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

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关闭>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, 声音的强度都用分贝表示, 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/当前就是最大像素值÷MSE; 8bit下即256/MSE; RGB色彩空间的视频要算亮度空间, 所以PSNR算式更长(¬‿¬)

# 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(੭•ω•˚)੭̸\*✩

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



* **平滑4：**a与1皆为帧内块, 且边界位于CTU/宏块间, 最强滤镜值
* **平滑3：**a或1皆为帧内块, 但边界不在CTU/宏块间
* **平滑2：**a与1皆非帧内块, 含一参考源/已编码系子
* **平滑1：**a与1皆非帧内块, 皆无参考源/已编码系子, 溯异帧或动态向量相异
* **平滑0：**a与1皆非帧内块, 皆无参考源/已编码系子, 溯同帧或动态向量相同, 滤镜关

--deblock<滤镜力度:搜索精度, 默认1:0>两值于原有强度上增减. 一般推荐0:0, -1:-1, -2:-1

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

# 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へ(゜∇、°)

建议crf 15+用no-sao, 或在线条较锐利的源上开纹理增强+limit-sao. crf 20+可以用limit-sao, crf 24+直接保留sao即可. 不过若在通篇画面动态高且混乱的情况下就算用了也不会被手机端用户察觉, 所以看情况吧ヾ(@°▽°@)ノ

**--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(╯︵╰,)

--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(\ )

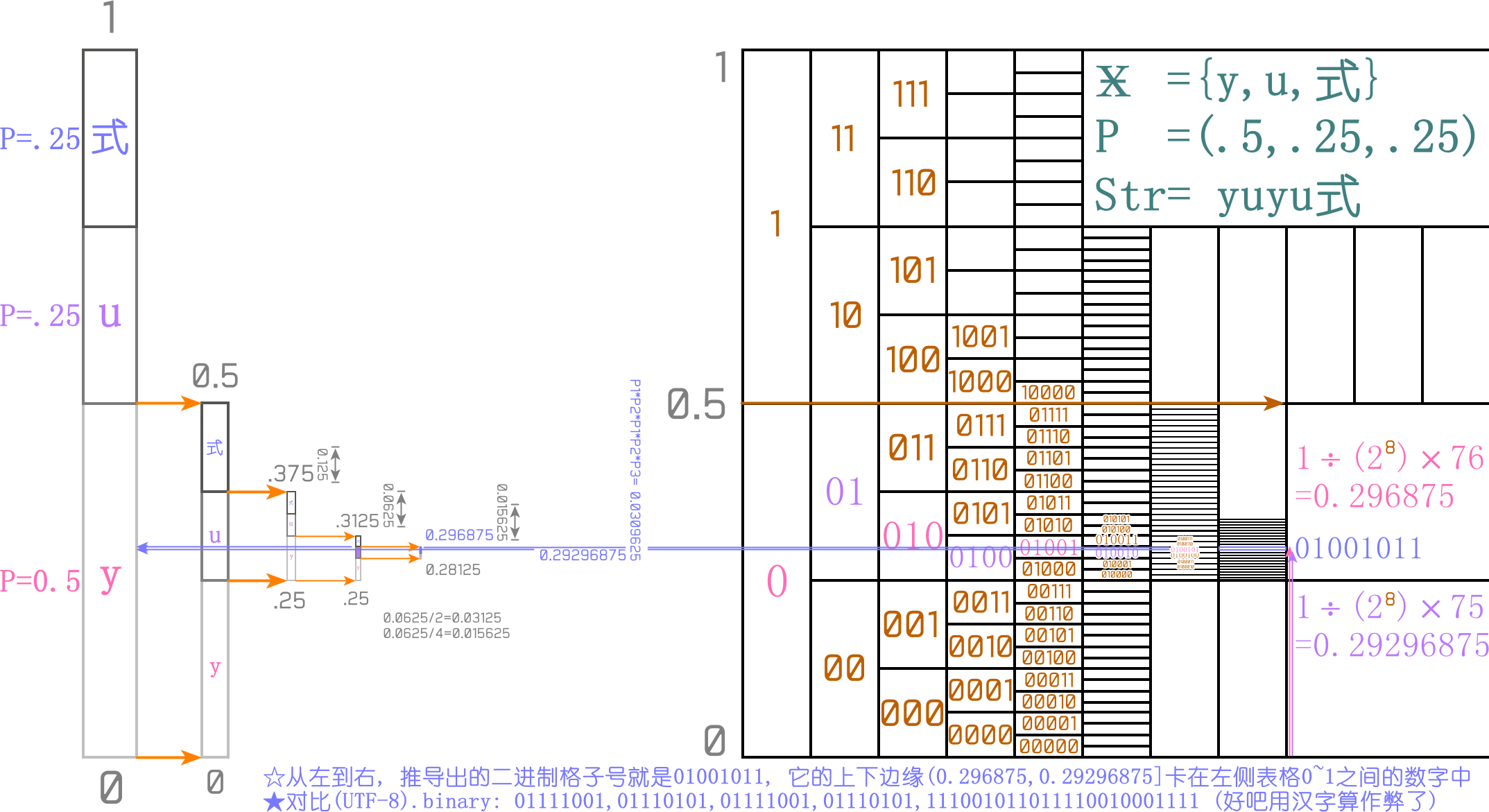
--limit-sao<开关>已关闭. 对一些sao的计算采用提前退出策略, 可以改善sao造成的画面问题

--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-上下文自适应二进制算数编码

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才是现代编码器(压缩包到音视频)的核心算法之一(屮゜Д゜)屮

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૮(⋆❛ہ❛⋆)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](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):*

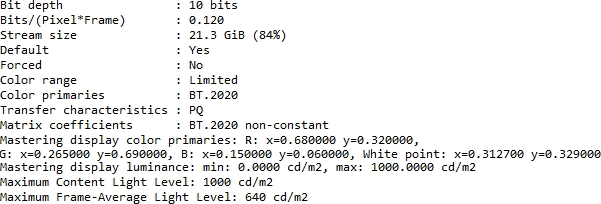
*DCI-P3: G(x0.265, y0.690), B(x0.150, y0.060), R(x0.680, y0.320), WP(x0.3127, y0.329)*

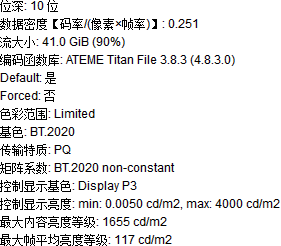
*bt709: G(x0.30, y0.60), B(x0.150, y0.060), R(x0.640, y0.330), WP(x0.3127,y0.329)*

*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

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

****↓图应设max-cll 1655,117/L(40000000,50)/colorprim bt2020/colormatrix bt2020nc/transfer smpte2084

**--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, 从第0帧开始数(. -`ω´-)

--frames<整数, 默认全部>一共压缩x帧, frames 12即处理12帧, 从0号到11号

--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色彩空间需转换

--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ᕙ(⇀‸↼‶)ᕗ

--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, 小数帧勿做四舍五入(´〜｀\*)

--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**](http://www.mediafire.com/?6lfp2jlygogwa) | .hevc GCC10 [单文件8-10-12bit] 附x86, Windows XP x86版 附libx265.dll |
| [**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**

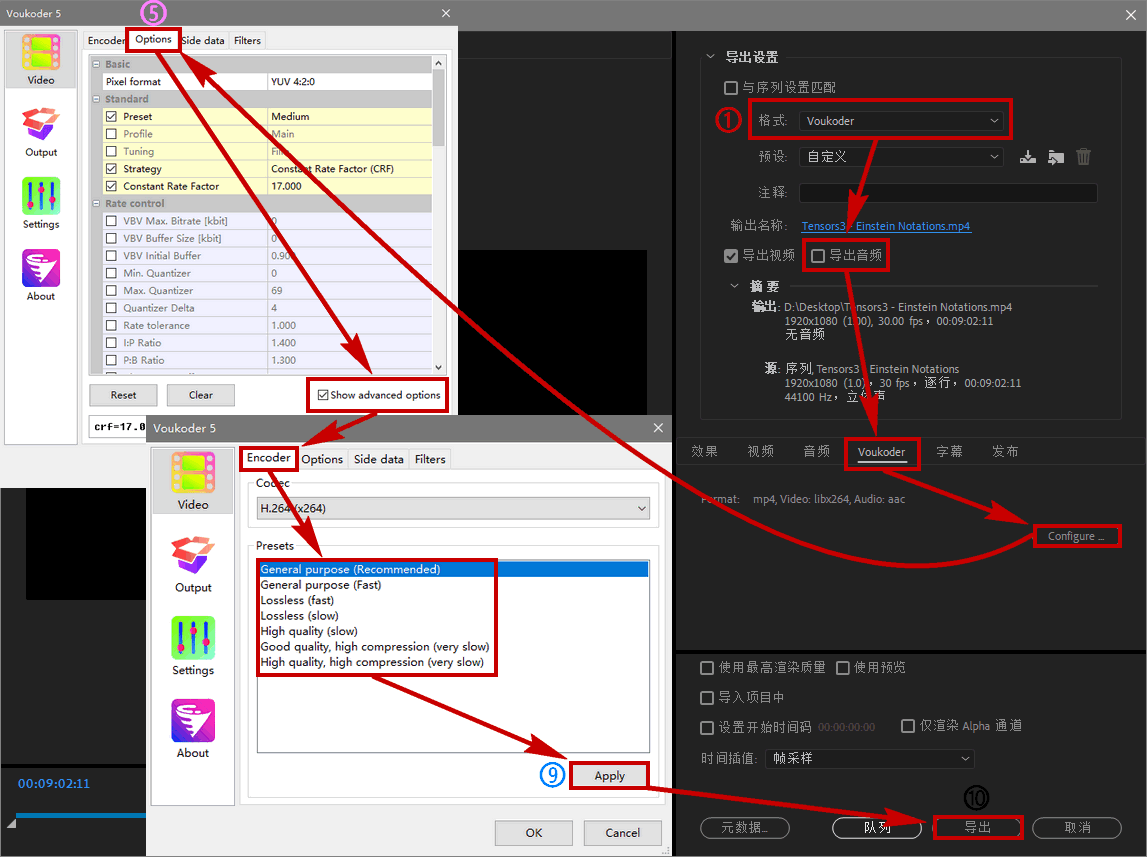
有的x265编译的文件名上有rc, 指已修复所有被提出的问题 且编译者认为ok的版本ヽ(･ω･ゞ)

**杜比视界dolby vision不深入研究**

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

Voukoder用法:

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

**.bat文件操作技巧**

.bat中, 命令之后加回车写上pause可以不直接关闭cmd, 可以看到原本一闪而过的报错(⌐■\_■)

**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**

原因未知, 可能是内存问题, 但是目前所有硬件都正常, 开虚拟机没有事U・x・U

# 预设(该板块有理论错误)

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