x265命令行参数教程**精简版**

欢迎阅读! 本教程精简了科普, 强烈建议先看懂有科普的[完整版,](https://nazorip.site/archives/63/) 入门先看[x264视频压缩教程综合版](https://www.nazorip.site/archives/63)。有什么不会的可以直接加群[691892901](https://jq.qq.com/?_wv=1027&k=5YJFXyf), 现在要压视频就去拿[急用版](https://nazorip.site/archives/334/)吧(`･ω･´)ゞ

## 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)

首选10bit

# 分块

--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, 需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-量化

--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direction of minimal distortion/DMD, 组成一张张帧间矢量表的计算. 若找出的信息不足, 参考帧与分块的建立就欠缺基础.

--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, 待查>三份异分辨率原画分别宏-微观的搜索动态信息;

--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

--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 | ½像素迭代 | ½像素搜索方向 | ¼迭代 | ¼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

--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

--mcstf

<开关, 默认关, 会关闭多线程>mctf+双阈滤镜(作用于空间域), 基础上增加了自动降噪能力

# 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.

--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

# 初始化 - Lookahead建立GOP结构

[过程见x264教程](https://www.nazorip.site/archives/63). 最先启动, 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)

--scenecut

<整数> Lookahead进程触发转场的阈值, 或

--hist-scenecut

<开关, 默认关, 推荐8bit下开, 12bit源会导致压制中途崩溃>亮度平面边缘+颜色直方图SAD阈值触发转场. v3.5+69后编码彩色视频, 尤其[HDR源](https://forum.doom9.org/showthread.php?p=1978502)中超越传统转场近20%, 降低了正误判(设I帧, closed-gop下帧间冗余效益降低)和负误判(不设I帧, 分为多个带I块的P帧, 帧内编码效益降低), 因此[除黑白视频外](https://forum.doom9.org/showthread.php?p=1978737)推荐. 目前以上推荐停留于理论. 注: hist-threshold于v3.5+69被删

--rc-lookahead

<帧数量, 范围1~250, 推荐keyint÷2>指定cutree的检索帧数, 通常设在帧率的2.5~3倍. 高则占用内存增加延迟, 低则降低压缩率和平均画质. cutree会自动选择--rc-lookahead和max(--keyint, max(--vbv-maxrate, --bitrate)÷--vbv-bufsize×fps)中最小的值作为检索帧数

--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

## P/B帧推演

: 算法见x264教程

### --b-adapt 2

<所有情况, 整数0~2, 建议2>0代表不设B帧

--bframe-bias

<整数-90~100, 推荐默认>设立B帧判定偏移, 增大的同时搭配低pbratio可增加B帧数量, 用负值搭配高pbratio可以减少B帧数量

## 参数集

在网络抽象层单元/Network abstraction layer unit中表现为含解码配置(profile, level)的数据包

--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, 默认关>在流未封装的情况下提供SPS, PPS等信息, 正常播放h.265源码

## VBV - 基于缓冲条件的量化控制

手动指定网络/设备下所允许的缓冲速度kbps以控制CRF/ABR模式. 与CRF一并使用时叫VBR

--vbv-bufsize

<整数kbps, 默认关=0, 小于maxrate>编码器解出原画后, 最多可占的缓存每秒. bufsize÷maxrate = 编码与播放时解出每gop原画帧数的缓冲用时秒数. 值的大小相对于编完GOP平均大小. 编码器用到是因为模式决策要解码出每个压缩步骤中的内容与原画作对比用

--vbv-maxrate

<整数kbps, 默认关=0>峰值红线. 用"出缓帧码率-入缓帧码率必须≤maxrate"的要求, 让编码器在GOP码率超bufsize, 即缓存用完时高压出缓帧的参数. 对画质的影响越小越好. 当入缓帧较小时, 出缓帧就算超maxrate也会因缓存有空而不被压缩. 所以有四种状态, 需经验判断

* 大: GOP大小=bufsize=2×maxrate, 超限后等缓存满再压, 避开多数涨落, 适合限平均率的串流
* 小: GOP大小=bufsize=1×maxrate，超码率限制后直接压，避开部分涨落, 适合限峰值的串流
* 超: GOP大小<bufsize=1~2×maxrate，超码率限制后直接压，但因视频小/crf大所以没啥作用
* 欠: GOP大小>bufsize=1~2×maxrate，超码率限制后直接压，但因视频大/crf小所以全都糊掉
* 由于gop多样, 4种状态常会出现在同一视频中. buf/max实际控制了这些状态的出现概率

--crf-max

<整数0~51>防止vbv把crf拉太高, 可能适合商用视频但会导致码率失控

## 关键帧

**IDR刷新解码帧instant decoder refresh**

* 自身储存完整图片, 但同时还负责GOP间划界分段, 播完令解码器清理前GOP缓存的大写I帧(¬\_¬)ﾉ
* 清缓存是为了防止参考/内存错误传播, 错误可能源自内存/主板/CPU/数据/解码器/网络/操作系统/电子战

**RAP/随机访问点random access point**

* "访问"代表播出画面前获取数据的过程
* "任意"代表拖进度条, 打开直播, 使进度条上任意一点都要正常解码的目的, 增加码率提升体验

**CRA净任意访问clean rand. access**

* open-gop状态下指定包括GOP间划界, GOP内帧间参考, 自身储存完整图片的i帧
* 附近的rasl/radl帧与之相对应

**DRA脏任意访问dirty rand. access**

* 含i块, 需要全部解码才能重建出i帧的一组P帧. 压缩更高但比i帧更易出现解码错误

**BLA断链访问帧broken link access**

* open-gop间划界, 访问不相关/不相连GOP的特殊CRA帧. 用于不暂停播放的分辨率切换

--no-open-gop

<开关, 默认关, 建议开>不用cra/bla, 增加码率增加兼容, 适合长GOP策略

--min-keyint

<整数>指定最小IDR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. 两种选择给出的画质一样

* 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
* 设1来增加IDR帧, 一帧被判做转场本来就意味着前后溯块的价值不高. 而P/B帧内可以放置I宏块, x264会倾向插P/B帧. 好处是进度条落点在激烈的动作场面更密集

--keyint

<整数>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

## 参考帧

**RASL任访略前导, RADL任仿解前导 random access skipping/decoding lead**

* 打开直播, 用户拖动进度条落在CRA附近, 找不到I帧时指定应该解码decode还是略过skip的标签化P帧

--radl

<整数默认0, 小于连续B帧, 建议2~3>原理见上

--ipratio, --pbratio

<浮点, 默认1.4, 1.3>P帧相比IDR/I及B/b帧相比P帧的量化值递增. B帧双向参考能从更多帧中找到参考源, 因此量化强度最高

* 真人录像片源中保持默认
* 动漫片源中连续长B帧出现几率增多, 有时会找不到合适的参考源导致画质损失, 用<1.2>或更小来分配一定码率
* 可据比例换算帧类型的qp, 如I-qp17, P-qp20, B-qp22即--qp/crf 17 --ipratio 1.1765 --pbratio 1.1

--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大于8, 同时keyint大于250会大增内存占用

# 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

--fast-intra

<开关, rd>4时关, 推荐开>先查夹角模式2,10,18,26,34, 再加一倍精度到模式5, 15, 21, 31, 最后解锁最高精度(共搜索10种), 关闭则逐一搜索全部33种夹角造成浪费.

--b-intra

<默认preset slow及更高下启用, 推荐开>B分片同样进行帧内编码压缩

--constrained-intra

<开关, 默认关, 不可备份+长期存档>缺生成参考点的CB时用帧内块或默认值, 不用帧间块生成参考点, 降低参考错误传播距离, 降低压缩率和速度换取数据寿命

--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, 同时平滑的是参考点而非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

## CRF上层模式

--crf

<浮点范围0~51, 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

## **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; 而码率>画质, 但画质也抓的压片用1pass-crf+2pass-abr

## SBRC下层模式 - 可搭配CRF/ABR/CRF-VBR/ABR-VBR

### --sbrc

<启用分段式率控制segment based rate control, 实现DASH, M3U8串流用的功能, 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

## CQP双层模式

--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

## 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双层模式

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, 默认关>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, 默认关>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)/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, 真无损压缩双层模式

### --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, 默认关, 需rd>2>4x4 tu上跳过DCT变换, 可保留深度分块/纹理密集处的放大细节

### --cu-lossless

<开关, 默认关>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

# 自适应量化

CRF/ABR设定每帧量化/qp后, 方差自适应量化variance adaptive quantizer再根据复杂度判断高低频信号, 来实现精确到宏块的qp分配过程. 讨论时注意aq与vaq的混淆

--aq-mode

<整数0~3>据原画和crf/abr设定, 以及码率不足时(crf<18/低码abr)如何分配qp

* <1>标准自适应量化(急用, 简单平面)
* <2>+启用aq-variance, 自动调整aq-strength强度(录像-电影以及crf<17推荐)
* <3>+码率不够用时倾向保暗场(接受更明显的涂抹失真, 慎用)
* <4>+码率不够用时更加倾向保纹理(接受平面上的涂抹失真, 实验性, 慎用)

--aq-strength

<浮点>自适应量化强度. 搭配aq-mode, 如动漫1:0.8, 2:0.9, 3:0.7用. 录像上可加0.1~0.2, 画面混乱/观众难以注意平面时可再增加. 注意低成本动漫的平面居多, 因此码率不足时反而要妥协纹理

--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-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-str., 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, 画质容易崩坏

--rd

<1/2/3/5, 默认3, 大则慢>优化参与md的程度. <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>

--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抗涂抹的目的

--tskip-fast

<开关, 默认关>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

# 率失真优化控制

## 优化量化

### --rdoq-level

<整数, 范围0关~2, 大则慢>解决abr/crf粗暴分配量化值所造成的失衡问题

* <1>每个分块的量化值进行率失真优化, psy-rdoq开启则倾向于低量化
* <2>对比4x4块高频信息/残差是否有利于整个编码组(CTU内分块)画质, 同时对所有系子进行同类分析, 造成大批4x4块被抹除, 降低psy-rdoq效果, 适合一般及高压缩用途

### --psy-rdoq

<浮点0~50, 默认0关>心理视觉往高影响量化块的能量J, 使量化偏好低强度/保留细节. 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
* 取决于分辨率, 画面细节少则取决于表现画面细节所需的最低分辨率

### --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)

## 优化模式决策

--psy-rd

<浮点0~50默认2, 需rd3, 默认0关, 和x264不同, 搭配psy-rdoq>心理视觉优化往高影响量化块的能量J, 使模式决策偏好保留细节. <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, 需rd 5, 有图中斜面上出现的随机块失真>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, 默认关>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>psy参数改qp的最大范围, 宏观影响大, 等待测试数据

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

# 环路滤波-去块滤镜

* **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

### --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, 是最后的文本压缩(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

# SEI补充与优化消息

supplemental enhance info记录每帧的补充信息.

--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

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，PR，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. 但若有一堆内存读写间歇, 3D缓存处理器就能用读写赢回速度

--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, 官方建议rd 3/5占不满算力的情况下开>多线程模式决策, 有难以应付噪点的问题

--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

--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信息, HDR视频, 黑边跳过

纯元数据, 写错或忘写也可以改. HDR电视只读取maxcll和maxfall, 所以可不写master-display. 光强/光压的单位是candela (1cd=1nit). 由于bt601，bt709，HDR-PQ，HLG标准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. 因为HDR相当于音响能发出很大和很小的声音, 所以好屏幕比HDR参数更重要

--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

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



**--colorprim**

<字符>播放用基色, 指定给和播放器默认所不同的源, 查看视频信息可知: bt470m, bt470bg, smpte170m, smpte240m, film, bt2020, smpte428, smpte431, smpte432. 如图为bt.2020

图2: cll 1655,117/L(40000000,50)/colorprim bt2020/colormatrix bt2020nc/transfer smpte2084

**--colormatrix**

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

**--transfer**

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

--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)

**--display-window**

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

# 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/DV**

有两种格式, 单流/DV-MEL和双流/DV-FEL, 两者都带有RPU, 双视频流有base layer视频层和enhance layer强化层, EL可被一般的hevc解码器丢弃而正常播放, 单视频流就只有私有解码器/有特定芯片+固件的设备能播放, 如果开源播放器能播放DV-MEL则是假源; 如果支持DV-FEL的设备丢弃EL, 只播放BL则是假设备. 参考单元reference picture unit是含有动态元数据的特殊NALU, 类似(到现在还没几个播放器能正常解码的)--opt-qp-pps, opt-ref-list-length-pps的功能

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 样式 | 编码 | BL:EL分辨率 | x265支持 | 伽马 | 色彩空间 |
| 4 | 10bit hevc | 1:1/4 |  | SDR | YCbCr |
| 5 | 仅BL (DV-MEL) | √ |  | ICtCp |
| 7 | 4K=1:1/4; 1920x1080=1:1 |  | UHD蓝光 | YCbCr |
| 8.1 | 仅BL (DV-MEL) | √ | HDR10 |
| 8.2 | √ | SDR |
| 8.4 |  | HLG |
| 9 | 8bit avc | 仅BL (DV-MEL) |  | SDR | YCbCr |

--dolby-vision-profile

<选择5/8.1 (HDR10)/8.2 (SDR)>8.1需要写master-display和hdr10-opt

--dolby-vision-rpu

<路径>导入rpu二进制文件(.bin)用

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

**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