

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)

欢迎阅读本教程！若有什么不会的可以直接加群 691892901 哦(´·ω·´)ゞ

1. 使用 Word/浏览器/PDF 浏览器的"查找"功能, 让电脑帮你找内容(((((*. _.)
2. x265 很难, 不懂就先看 [x264 视频压缩教程综合版](#). 做好看 5 遍以上, 经常查阅的准备←V←

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 [] - --output
```

-i 导入, -f 格式, -an 关音频编码, -strict unofficial 允许超标格式, --y4m 指定"YUV for MPEG"格式, "程序 1 - | 程序 2 -"是 pipe 格式

```
VSpice.exe [脚本].vpy --y4m - | x265.exe - --y4m --output
```

```
VSpice/avs2yuv [脚本].vpy - | x265.exe --input-res [分辨率] --fps [] - --output
```

```
avs2yuv.exe [脚本].avs -raw - | x265.exe --input-res [分辨率] --fps [] - --output
```

带 lavf 的 x265 直接压制: 原作 Yuuki-mod. 但属于强行装上所以不可靠, 需复查压制结果

检查/选择色深, 版本, 编译: 用-V 来检查色深, 版本, 内嵌滤镜等信息, -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

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, 在 Windows 和各种 Unix 系统上通用

目录	变换.....5
--ctu	动态搜索.....7
--min-cu-size	--me
--limit-tu	--subme
--tu-intra-depth	--merange
--tu-inter-depth	--analyze-src-pics
--max-tu-size	--hme-search

--hme-range		--vbr-maxrate	
帧内搜索(该板块有理论错误).....	7	--crf-max	
--max-merge		--crf-min	
--early-skip		--qcomp	
帧内编码.....	9	--cbqpoffs	
3-TAP 滤镜.....	9	--crqpoffs	
强力平滑滤镜.....	9	率失真量化.....	13
DC 模式.....	9	--rdoq-level	
夹角模式.....	9	--psy-rdoq	
--fast-intra		自适应量化.....	14
--b-intra		--aq-mode	
--no-strong-intra-smoothing		--aq-strength	
--constrained-intra		--aq-motion	
运动补偿.....	10	--qg-size	
帧控制/率控制.....	10	模式决策.....	15
关键帧.....	10	--rd	
参考帧.....	11	--limit-modes	
--no-open-gop		--limit-refs	
--radl		--rect	
--min-keyint		--amp	
--ref		--rskip	
--keyint		--rskip-edge-threshold	
--fades		--tskip-fast	
--pbratio		--rc-lookahead	
--bframes		--no-cutree	
--weightb		率失真优化.....	16
--b-adapt		--rd-refine	
转场.....	12	--dynamic-rd	
--scenecut		--rdpenalty	
--hist-scenecut		--splitrd-skip	
--hist-threshold		环路滤波-去块.....	17
2PASS 转场优化.....	12	--deblock	
--scenecut-aware-qp		--no-deblock	
--masking-strength		环路滤波-取样迁就偏移 SAO.....	17
量化 质量控制.....	12	--sao	
--qp		--sao-non-deblock	
--crf		--no-sao-non-deblock	
--qpmin		--limit-sao	
--vbr-buFSIZE		--selective-sao	
		真无损压缩 近无损压缩.....	18

--lossless	
--tskip	
--cu-lossless	
熵编码-CABAC(新内容待复查, 来源)	19
VPS PPS SPS RPS 解码参数集.....	20
--opt-qp-pps	
--opt-ref-list-length-pps	
--repeat-headers	
SEI 维稳优化消息.....	20
--frame-dup	
--dup-threshold	
--hrd	
--hash	
--single-sei	
--idr-recovery-sei	
线程	21
--pools	
--pme	
--pmode	
--asm	
多线程 vs 多参考	
--frame-threads	
--lookahead-threads	
色彩空间转换, VUI/HDR 信息, 黑边跳过	22
--master-display	
--max-cll	
--hdr10	
2PASS 模式.....	23
--pass	
--stats	
--slow-firstpass	
2PASS 模块.....	24
x265 有两种 2pass 模块:	
ANALYSIS 模块 SAVE/LOAD 部分:	24
--analysis-save	
--analysis-load	
--analysis-save-reuse-level; --analysis-load-reuse-level	
MULTI-PASS-OPT 模块:.....	24
--multi-pass-opt-analysis	

--multi-pass-opt-distortion	
--multi-pass-opt-rps	
ANALYSIS 模块 REUSE 部分:	24
--analysis-reuse-file	
--analysis-reuse-mode	
优化:	25
--dynamic-refine	
--refine-inter	
--refine-intra	
--refine-mv	
--scale-factor	
--refine-mv-type <i>avc</i>	
--refine-ctu-distortion	
ABR 天梯.....	26
--abr-ladder	
IO(INPUT & OUTPUT, 输入输出).....	26
--seek	
--frames	
--output	
--input-csp	
--dither	
--allow-non-conformance	
--force-flush	
--field	
--input-res	
--fps	
--chunk-start --chunk-end	
WINCMD LINUXBASH 输出压制 LOG	27
CMD 操作技巧 <i>color 08</i>	

CU 和 CB - 编码单元/块 coding unit 在 ctu 的基础上, 可进一步分裂/不分裂, 目的同样是隔离动静态

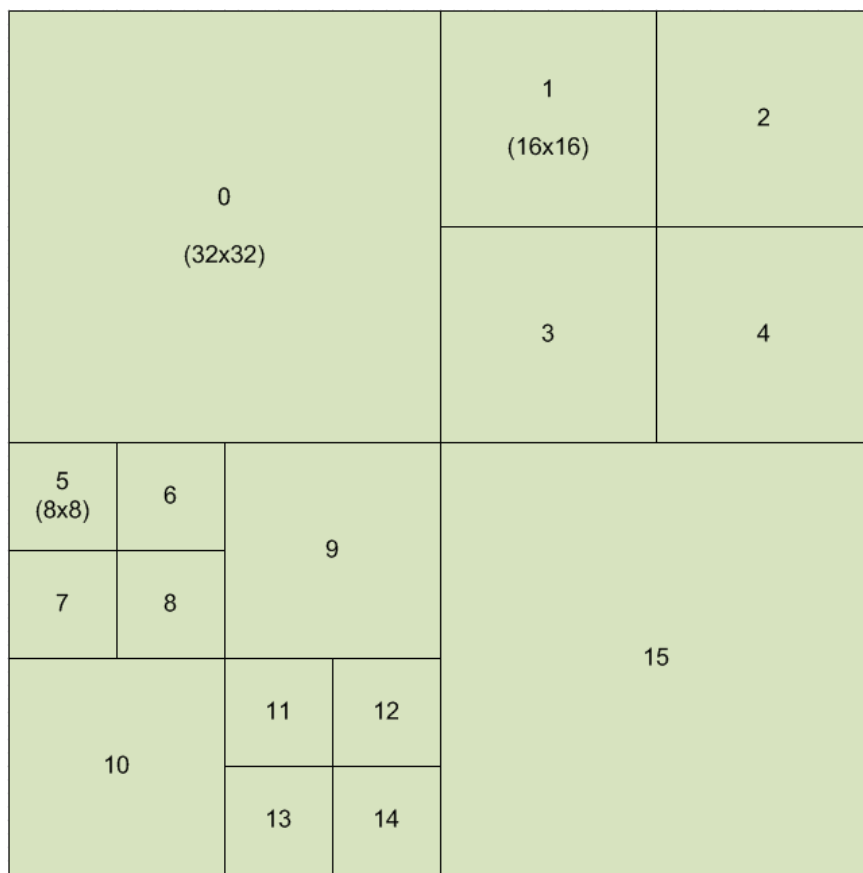


图: CTU 分裂为 CU, CTU 大小上至 64x64, 宏块最大只有 16x16

PU - 预测单元 prediction unit 从 cu 上做对称 rectangle, 非对称 asymmetric partition 划分, 以更好的隔离动静态+判断每个区域用帧间/帧内参考与否. 亮度与色度上的分裂度可以不同, 小至 4x4 像素



图: PU 的 4 种对称和 4 种不对称划分

TU 和 TB - 变换单元/块 transformation unit 独自在一些 cu 上用一棵二叉树往复杂度高处划分. 这在对 pu 预测推算出的新图做检查, 找出与源的差距 residual. 接着, 用 DCT 的整数版/给 4x4 亮度 tb 的 DST 来将其编码. tu 的划分不用与 pu 对齐, 二者关系不大(´▽`)/

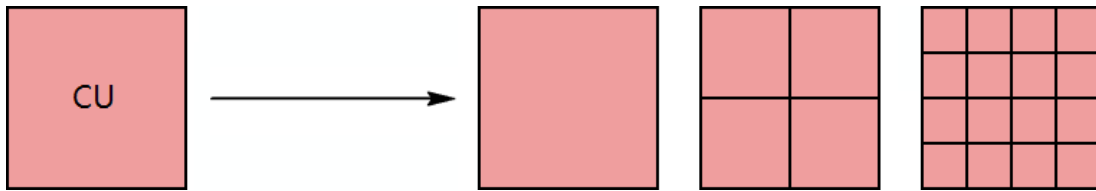


图: TU 的划分, 用 block 称呼以上的块, 例如 CTB, CB, PB 和 TB 是在单指一个块

--ctu<64/32/16>指定编码树单元最大大小的参数, 默认 64. CTU 越大 = 视频的压缩效率越高 + 对画质的破坏(平面涂抹)越高 + 压缩的速度越慢

考虑体积优先时建议保持默认 64, 当考虑画质优先时建议设在 32, 当考虑速度优先时建议设在 16;

ctu_size = (宽 × 高 - 1713600) ÷ 11250, 判断 ≤ 16 还是 ≤ 32, 皆不符就设 64

--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(ˆ-ˆ*)/

--limit-tu<整数 0~4, tu-intra/inter-depth 要大于 1>早退 tu 分块步骤的提速. 以瑕疵修复换取速度. 日常与画质编码可以设 1~3, 快速编码可以设 4. 处理 2k 或以上的分辨率再在基础上减 1. 不建议分辨率小于 1280x720 的视频使用 4. 由于 tu 对高峭度细节的压缩关键, 所以建议为了画质而少退出, 而更大的 tu 可以让 DCT 变换更容易压缩掉细节, 所以为了码率应多退出

<默认 0>关, <1>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(/ˆv`*)/

<3>以时间+空间域附近 tu 分裂次数的平均值为上限

<4>在 3 基础上, 将其作为接下来其他的分裂上限, 相比 0 有最快 20%的速度提升

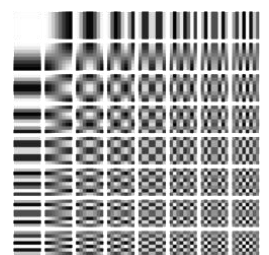
--tu-intra-depth<整数, 1~4>空间域 tu 分裂次数上限. 值越高就使用越多时间换取码率+画质的参数. 默认 1 代表只能在 cu 基础上分裂一次, 建议日常编码设在 1~2, 画质和高压编码设 3~4

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

--max-tu-size<32/16/8/4>更大的 tu 大小能够增加压缩率, 但也造成了计算量的增加和瑕疵压缩能力降低. 使用以限制最大 tu 大小. 用更多码率换取时间和画质的参数. 默认 32, 建议画质编码设在 16

变换

将信号的高低频排队, 从而方便量化, 降噪, 修图的处理. 所有从傅里叶级数 fourier series 发展而来. 将高低频罗列出来的空间叫做频域, 通过逆运算, 俗称逆变换就能组装这些频段, 还原原声原画. 离散代表一次变换一部分, 连续就是一次变换整个信息(* · _ ·) / ˆ *

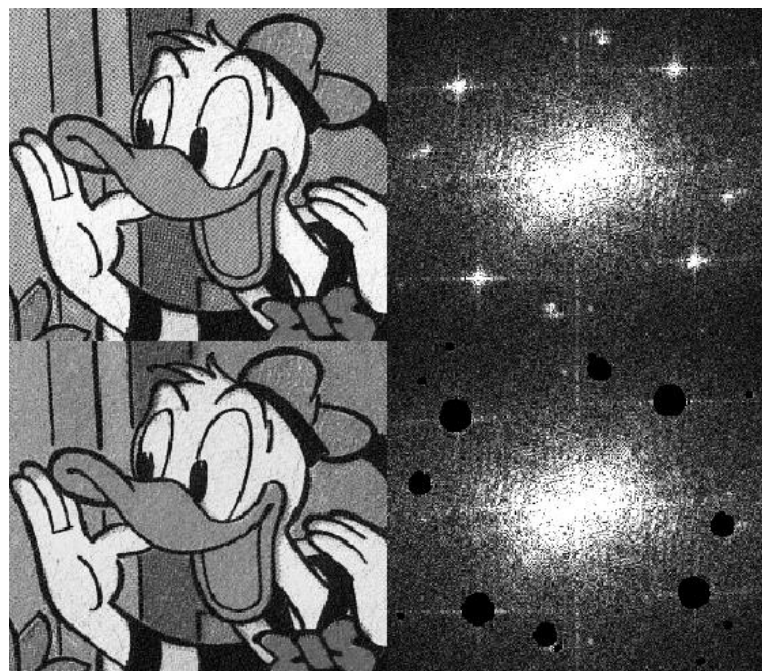
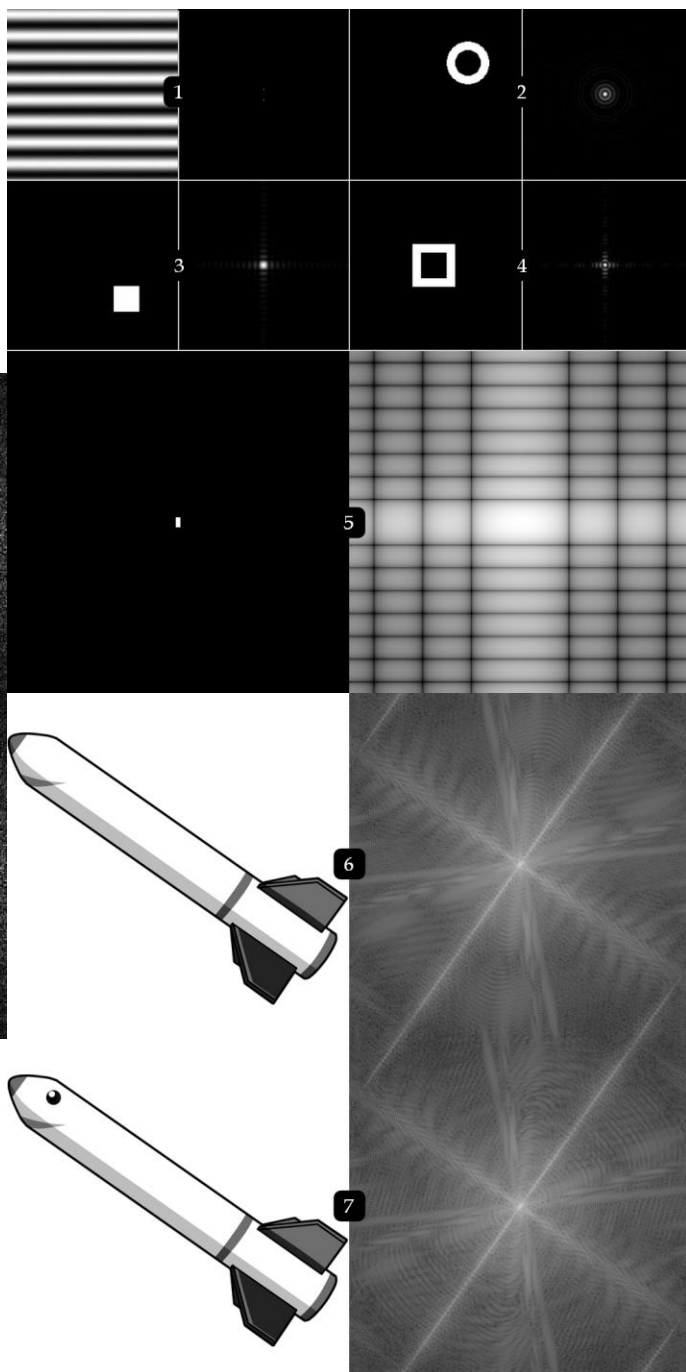


二维离散余弦变换 2D discrete cosine transform 傅里叶级数预制的万能模具, 通过穷举加减列出每个频段的使用次数, 图像就从空间域转换到频域了. 优点是无损还快, 缺点是无法旋转模具\(\#\`^`)/

二维傅里叶变换 2D fourier transform 要自制所有的波形, 所以用到了傅立叶级数. 二维上用一对点表示一套波形(如图①), 点越亮振幅越大, 距离远代表频率高. ②③, ⑥⑦上能看到所有信号被无视原来位置地重排, 从中心向外辐射高频. 这就是傅立叶级数无限逼近, 也叫收敛的性质. 目前关于还原图形原本位置的原理还未知

⑤⑥⑦上频域的点旋转了 90° , 这是因为一对点表示的是顺着它们的波形(见①), 而不是原图 $\equiv (\cdot \cdot) \equiv$

傅里叶变换的意义是音画处理, 如通话降噪, 图↓:
光盘扫图降噪等. 此处用 Mathematica 编辑



二维[不对称]离散正弦变换 2D [Asym] sine transform 缺少资料, 暂时略过. 但据说 x265, av1, vp9 和 vvc 都在用, 所以以后会更新

量化: 减少高频信息, 即将 DCT/DST 的表格项目和对应的量化表相除, 得到了压缩的图像. 而"脏边"也是因为原本遮住它们的高频波形被删掉了. 图 10~11: 左边除数小, 右边除数大

动态搜索

-370	29.7	-2.6	-2.5	-1.1	-3.7	-1.5	0.08
-231	44.9	24.5	-0.3	9.3	3.9	4.3	-1.4
62.8	8.5	-7.6	-2.7	0.3	-0.4	0.5	-0.8
12.5	14.6	-3.5	-3.4	2.4	-1.3	2.7	-0.4
-4.9	-3.9	0.9	3.6	0.1	5.1	1.1	0.5
-0.5	2.3	-1.7	-1.6	1.1	-2.7	1.1	-1.4
4.4	0.1	-1.4	0.2	-1.1	-1.5	-0.1	0.9
10.2	1.8	5.9	-0.9	-0.4	0.3	-0.4	0



1	1	1	1	1	2	2	4
1	1	1	1	1	2	2	4
1	1	1	1	2	2	2	4
1	1	1	1	2	2	4	8
1	1	2	2	2	2	4	8
2	2	2	2	2	4	8	8
2	2	2	4	4	8	8	16
4	4	4	4	8	8	16	16

低量化



1	2	4	8	16	32	64	128
2	4	8	16	32	64	128	256
4	8	16	32	64	128	256	512
8	16	32	64	128	256	512	1024
16	32	64	128	256	512	1024	2048
32	64	128	256	512	1024	2048	4096
64	128	256	512	1024	2048	4096	8192
128	256	512	1024	2048	4096	8192	16384

高量化

搜索块与块在帧间有没有联系，在时间上冗余

--me<hex, umh, star, sea, full>搜索方式，从左到右依次变得复杂，star 之后收益递减(°▽°)/

<umh>可变六边形：适中效果好，中杯(°▽°)b <star>星形三步搜索：慢效果好，大杯(ㄟ_ㄟ)

<sea>优化后的 esa 穷举：极慢效果好，超大杯 <full>全搜索：费时，水洒了

--subme<整数，默认 2，范围 1~7>不要小于 3，静态图~简单平移变形~快速编码等用 3~4 即可，遇到更复杂动态(比如 FPS 游戏录屏，动作片)再设 5，以此类推

--merange<整数>动态搜索范围，由于是在找动态矢量附近有没有更优的矢量，所以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)。hex 用范围 4~ctu 边长减 4，减 subme 的偶数；umh 用 ctu 大小减 4 减 subme 的 4 步模数(例如，52 mod 4 ≡ 0)

考虑 ctu 大小的原因是既然已经做了分块，那么动静态自然会被隔离，而超出范围搜索不但慢(访问新的 ctu)，而且不会有好的结果理论有误，可能只使用 dia-hex，未来会修复(๑•_•)๑

--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>分叉式 hierarchical-me，性能提升未知，原理是三种分辨率下，宏观微观的搜索动态信息，比 me 快，可能在 merange 小于 28 的时候比 me 好

--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

帧内搜索(该板块有理论错误)

当视频段被总结为 IDR/CRA 关键帧，搭配 I 块组成初步的参考网络 GOP/SOP 时，x265 就该查找这些关键帧内部，在空间里冗余了、(͡° ͜ʖ ͡°)ノ

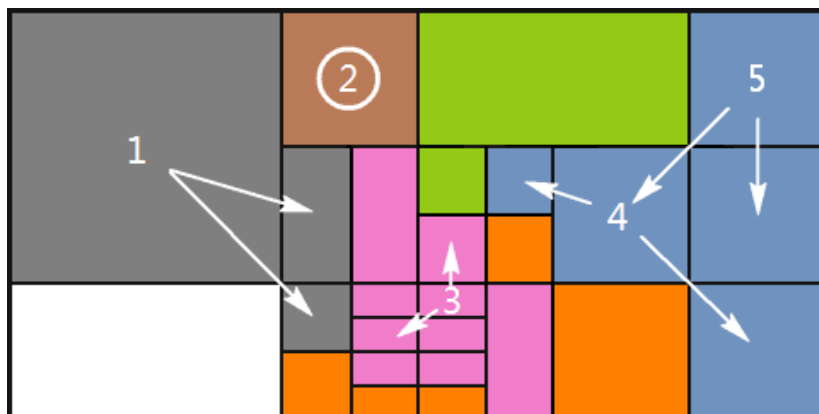


图 10: 完成 PU 划分, 正在进行 PU 搜索的图. 颜色相同表示信息相似

AMVP - 高级动态矢量预测 advanced motion vector prediction 用相邻 PB 的矢量导出若干候选块

先搜索左下和左, 其中最匹配, 最靠近左下角的块就选为被选 1, 如 3 和 5 找到的匹配

再搜索右上, 上和左上, 找到就选为被选 2, 如 3 找到的匹配(ϵ)

若未找齐两个候选就搜索时间域(帧间)右下角或自身中心(标记为圈)的块. 若还找不到就放弃, 如白色块

merge mode - 并合搜索(严格说不只是帧内搜索). 不同于 AMVP 繁琐的步骤, 直接从空时间域凑 5 个参选+两个被选. 不过因为会漏掉其中一边, 从而忽视了潜在的可降低码率

不论画质, 只要能降低码率就能成为被选. 所以增加被选的数量并让画质优先就可保证画质

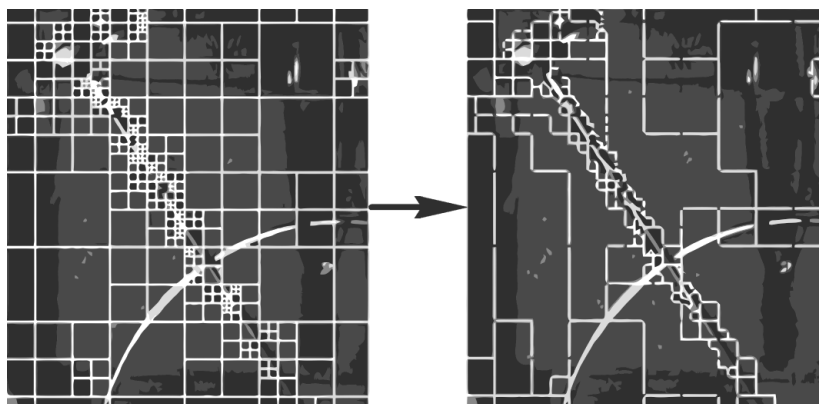


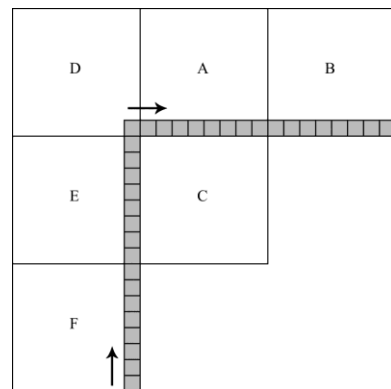
图: 分块 \rightarrow merge 后, 块与 P 帧相互参考而得以在被压缩的情况下重建画面一样的原理是一样的

--max-merge<整数 0~5>重设 merge mode 被选数量的功能. 默认<2>, 即选出被选 1 和 2, 用更多时间换取质量的参数. 建议高压编码设<4>, 其它可以设<2, 3>(+_+)

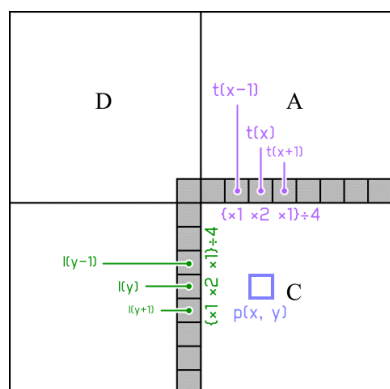
--early-skip<开关, 已关>先找 2nx2n merge 被选块, 找不到就停掉 AMVP, 暂无建议

帧内编码

类似 jpg, png 的单图像编码. hevc 标准的帧内编码可略作补块, 平滑和编码三步(o · · o)/. 图中: C 块即待编码块 PB; ABDEF 块即已编码块 CB; 灰像素点即参考源, 箭头指代填补缺失块的顺序



1. 补齐因 PB 位置不好, 或编码没跟上而缺的块: 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. 参考源预处理: 叫数据平滑 data smoothing. 它和模糊 blur 的区别是直接砍了高频也叫模糊; 而平滑让数据前后更连贯, 也偏向保留连续高频. 右图: 如果预处理不好, 照参考源编码的新 CB 会有蠕虫 worm artifact, 也叫条失真 directional edge 参考错误



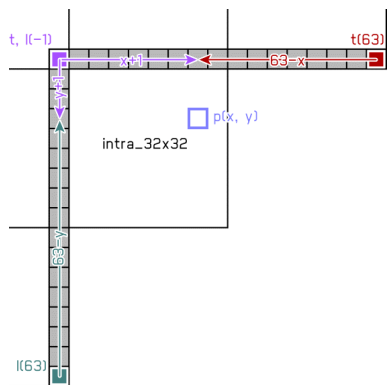
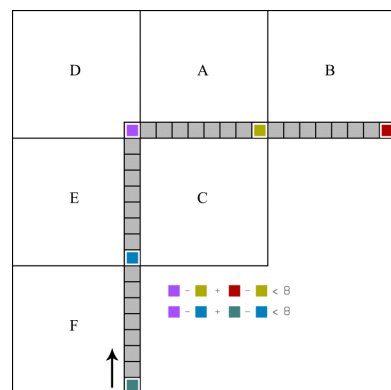
3-tap 滤镜

如左图: 也叫 $(1 \ 2 \ 1) \div 4$ 滤镜. 按每个预测值 prediction value, $p(x, y)$ 给横纵轴对应的参考源做加权平均计算



强力平滑滤镜

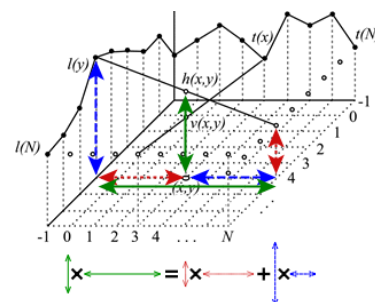
右图中: $\text{■} \sim \text{■}$, $\text{■} \sim \text{■}$ 两差异的和, 还有左侧 $\text{■} \sim \text{■} \sim \text{■}$ 的差异和要小于 8 才准用. 原理如下图中: 从 $\text{■} \sim \text{■}$, $\text{■} \sim \text{■}$ 线性插值出每个 $p(x, y)$ 所对应的参考源. 两种滤镜会在 DC, 垂/平夹角和接近垂/平夹角的情况下自动跳过



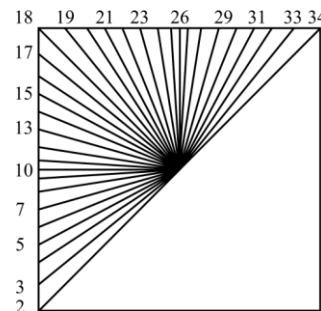
3. 由直流波 direct current, 趋平 planar 和 夹角模式构成的编码:

DC 模式代表"全部参考源的平均=全部预测值的平均", 也就是所谓的大平面. **趋平模式**代表从参考源过渡到 DC; 由双线性 bilinear 插值得来. 具体见图: 底-高相乘, 加底-高相乘就有了"三角形"的面积. 该面积随 $h(x, y)$ 的移动而变化; 再除以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 $\tan(\varphi)$ 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映射到了两个参考源之间, 要判断该角度下 $p(x, y)$ 的投影离左右哪个参考源更近, 以进行插值计算出参考结果. 最终穷举出 33 种参考结果(可缩水); 取最像 PB 的那个为新 CB



--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从而节省时间. 理论上在复杂画面中能提供有效加速

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

--no-strong-intra-smoothing<开关, 推荐保持默认>关闭强力平滑滤镜. 32x32 的 PB 改用 3-tap 滤镜. 由于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. 除非分块步骤跳过太激进, 又遇上了画面复杂的情况. 没 64x64 是因为 PU 最大只有 32x32

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

运动补偿

motion compensation 将动态搜索得到的信息利用起来的步骤, 如压缩和插帧. x264/5 用矢量信息得出宏块/编码单元在时间上的移动, 实现冗余或量化压缩. 其中 h.265 要求 7 遍可数冲应滤镜 finite impulse response filter, 找出小至 1/4 像素程度的镜头抖动以彻底隔绝错误, h.264 要求 6 遍~(ˇˇ)~

帧控制/率控制

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

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, 清缓存是为了治参考错误(¬_¬)/

cra 净任意访问 clean rand. access

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

dra 脏任意访问 dirty rand. access

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

bla 断链访问帧 broken link access

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

参考帧

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

- 进度条落在 cra 附近(缺参考)时指定解码还是略过的分类. 编码顺序上紧挨 cra 帧的被参考位. 避免 gop 崩坏. 如果进度条正常播过来则没它们事

--no-open-gop<开关已关, 建议长 gop 用>不用 cra/bla 之类, 增加码率增加兼容. open-gop 适合短于帧率的短 gop

--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

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

--ref<整数-0.01×帧数+3.4, 范围 1~16>溯块参考前后帧数半径, 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 就不用管了、(`◇`)/

--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, 目前根据 1p 的建议直接使用 24*帧率即可. --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 与 B 帧的 qp 值待遇差(如 B 块 qp 值至少是 P 块 1.3 倍). 由于 B 帧的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(ㄟ_ㄟ)

--weightb<开关>允许 B 帧的加权预测. 由于渐变场景中的 B 帧较多会导致码率不足的画面质量损失, 加权预测可以搜索并预测渐变时的亮度运动变化来提供更好的渐变画面的编码结果

--b-adapt<整数 0 关, 1 快速, 2 精确>B 帧适应性算法, 建议一般情况设 2

转场

--scenecut<整数, x264 教程省略>设定 x264/5 设 I 帧的敏感度. 默认值 40 很合适, 但为了降低平均码率可以在通篇画面平静的视频减少 5, 而对通篇画面激烈的视频拉高~5 以提升 I 帧和 i 帧数量

--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 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. x265 v3.4+9 版有效果优化

--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

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 2pass 的 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, qp-delta-ref, qp-delta-nonref<被 x265 v3.5 踢出房间>

量化-质量控制

--qp<整数, 范围 0~69>恒定量化模式. 每 ± 6 可以将输出的文件大小减倍/翻倍. 直接指定 qp 会导致 x265 不再判断那些内容可以被更高的压缩, 所以除非是以研究为目的, 其他情况皆不建议

--crf<浮点, 范围 0~51, 默认 28>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. 收藏级画质设在 16~18, 收藏级画质设在 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

相比于 x264, 虽然量化本身一样, 但 crf 越高, x265 要执行的额外计算也越多, 速度就越慢

--qpmin<整数, 范围 0~51>设置最小 qp 值, 默认 10. 在高压中画质环境建议设 --qpmin 22. 若视频质量差还可以设的更高, x265 早期?的理想状态下是 crf 值减 1.2(手动约分)

--vbv-buFSIZE<整数 kbps, >maxrate>编码器解出原画后, 最多可占的缓存每秒. $\text{bufsize} \div \text{maxrate}$ = 编码器或播放器解码每 GOP 原画帧数的缓冲秒数, 值的大小相对于编完 GOP 平均大小

--vbv-maxrate<整数 kbps>峰值红线. 用“出缓帧码率-入缓帧码率必须 $\leq \text{maxrate}$ ”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 GOP 大小(“▽”)

- 大: $\text{GOPsize} = \text{bufsize} = 2 \times \text{maxrate}$, 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: $\text{GOPsize} = \text{bufsize} = 1 \times \text{maxrate}$, 超码率限制后直接压, 避开部分涨落, 适合限峰值的串流
- 超: $\text{GOPsize} < \text{bufsize} = 1 \sim 2 \times \text{maxrate}$, 超码率限制后直接压, 但因视频小/crf 大所以没啥作用
- 欠: $\text{GOPsize} > \text{bufsize} = 1 \sim 2 \times \text{maxrate}$, 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 多种多样, 所以 4 种状态常会出现在同一视频中. buf-max 实际控制的是这些状态的所占比

--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, 不如加一层模糊滤镜

--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[-_-] ㄴ

--qcomp<浮点, 范围 0.5~1, 一般建议默认 0.6, x264>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>跟着纹理的增加而增加 qp, 有助于控制码率, 适合线上视频; <接近 1>有延迟的跟着纹理的增加而增加 qp, 有助于防止画质突然下降, 适合收藏画质; <1>停止分配, 类似直接设 --qp, 受到自适应量化 aq 的调整所以不如直接设 --qp

--cbqpoffs<整数>调整蓝色平面相比亮度平面的 Δqp 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($\Delta -3$ 左右)的值

率失优量化

rdoq 和量化平行发生, 把 tu 分到 4x4 并按所在 tu 编组, 每组测试其它量化值并对比当前 qp; 找损失小又消除 tu 系数 coefficients 的 qp 再分配(系数好像是非平面像素+子像素, 总之是趋平面反纹理)

--rdoq-level<整数, 范围 0~2>指定复杂度. 0=关闭(psy-rdoq 失效); 1=不分 tu; 2=分到 4x4, 很慢

--psy-rdoq<浮点 0~50, 默认 0 关>心理视觉优化影响率失真量化的程度, 提高能量以改变 rdoq 的用途, 使其更不愿消除系数, 使模式决策 mode decision 不会遇到差选项

1080p 高码率下设<2.3~2.8>给动漫, <3~4.8>给电影. 分辨率高低, 画面颗粒影响了系数数量和密度, 所以要改参数值 $\rho(\omega^{-1}d^{-1}\omega)$

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 (〰 ~ 〱;)

--aq-mode<范围 0~4, 0 关, 默认 2>建议如下, aq 只在码率不足以还原原画时启动(. . . // ε // . . .)

<1>标准自适应量化(急用, 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

--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

模式决策

mode decision 整合搜到的信息, 在各种选项中给 ctu 定制如何分块, 参考, 跳过的优化. 若没有 psy-rd, psy-rdoq(x265)和 trellis 优化, 模式决策就一定用码率最小, 复杂动态下全糊的方案集. mb 树也是 md 的一部分(" ◡ ◡ ◡)八(" ◡ ◡ ◡ ")

--rd<1/2/3/5, 默认 3>率失真优化参与 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>+矢量/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(¬_¬)

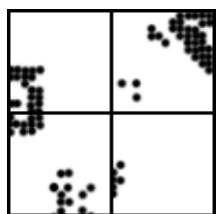
--limit-modes<开关>用附近的 4 个子 CU 以判断用 merge 还是 AMVP, 会大幅减少 rect/amp 块的存在感, 明显提速. 会增大或减少体积, 微降画质但难以察觉

--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; <2>据单个 cb 的差异信息建立 pu; <3=1+2> 建议除快速编码外的环境用 1 ◡ (-_-;)

--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 或通篇有颗粒的视频使用

--amp<开关, 已关>启用 pu 的不对称划分方法, 使用时必须打开--rect. 用更多更多编码时间换取码率的参数. 只建议有充足时间, 分辨率大于 1600x900 或通篇有大量粒子的视频使用

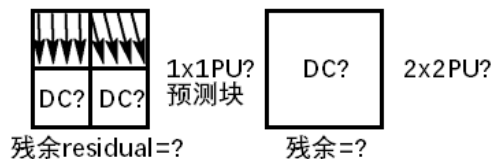


--rskip<0/1/2/3>找不到残余动态矢量/宏观上出现 cu 再分块被跳过时, 判断后面 cu 接着搜索分块还是提前退出的参数. 和 merge/AMVP 的区别是管辖 cu 内部的再分块

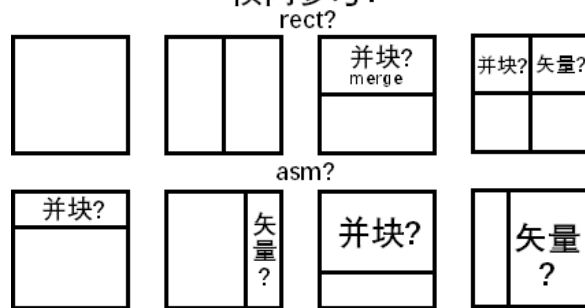
图: edge-density×边缘密度, √纹理密度(@_@)

<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分块, 大平面建议

帧内参考?



帧间参考?



跳过?

--rskip-edge-threshold<百分比 0~100, 默认 5, **rskip** 需大于 1>用迅速的 sobel 算子检测 cu 内纹理密度 edge density, 纹理面积和块面积的百分比. 若密度超过值就分块

--tskip-fast<开关, 已关>跳过 4x4 tu 的变换, 忽略部分系数 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 (3J <) 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, crf 小于 17 时才用的到

率失真优化

rate-distortion optimization 继承 x264 **能量=开销=码率=失真+λ·最低码率**得出优化需求, 拿量化过的块和原画做差能得出残差 residue, 也就是当前 qp 下, 每个块需要的最低码率需求, 底率了. 另外 λ 是 $0.85 \times 2^{(qp-12)/3}$ (o(1-x))

--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. 注意要搭配 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

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(* > ω <)

图: 复杂度对真人录像优化的重要性, 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, 建议在高压编码环境开 rd 5 时使用>使率失真优化参与对完成帧内搜索的 CU 的 crf 计算, 使用更多更多时间来换取码率的参数 (V) (o\o) (V)

--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

--rdpenalty<整数, 默认 0 关闭>惩罚性的, 用更高画质换取码率或反过来的参数



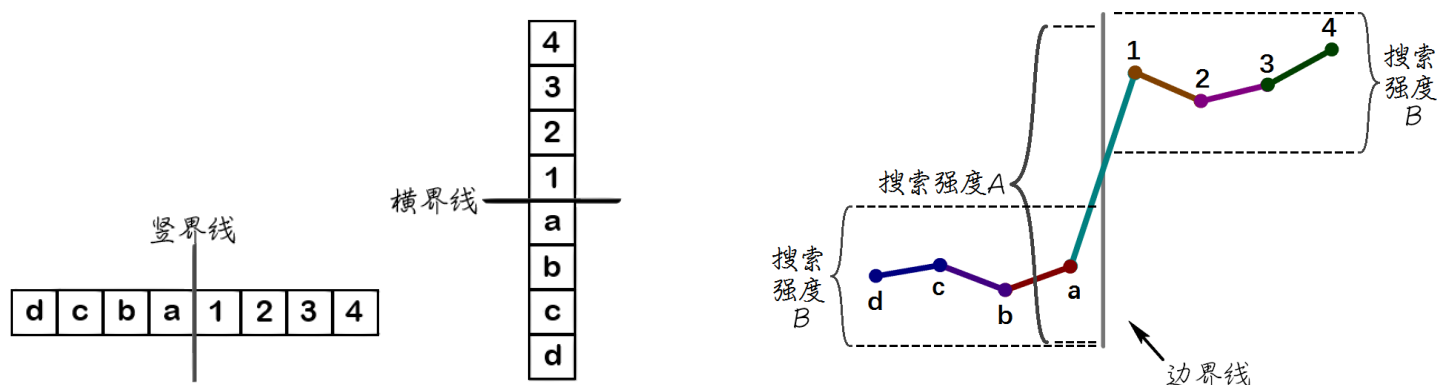
日常和高压推荐<1>强制 32x32 tu 用更差的率

失真优化策略节省码率; <2>强制 32x32 tu 继续向下分块, 有助于质量但不保证(π^π)

--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

环路滤波-去块

用于修复低码或为难以分块画面进行变换编码，运动补偿造成宏块之间生出横纵向纹路的问题，如不及时修复，问题会随着流水线被当成画面的一部分。可以通过外部滤镜修改，但 deblock 可以利用 x264/x265 编码器内的信息，来区分和微调每一个情况，使输出画面更加自然(๑•ω•๑)๑*☆*°



图：为方便理解，取 1234abcd 八个块之间的界线来举例

当 “1-a 的落差小于搜索强度 A”，同时 “2-1，以及 b-a 的落差都小于搜索强度 B”，就说明这是异常界线(= 〇、〇) 因为其他情况很难有精确到块与块之间正好有一个像素宽的线(、、)

--deblock<滤镜力度:搜索精度>默认 1:0, 两个值是在原有的强度上增加或减少, 码率不足/意外决定了一定会出现块失真, 所以除直播时关掉以加速外, 任何时候都应该使用

滤镜力度决定涂改, <大于 2>会糊; <小于-1>和关差不多, 推荐<默认 1>, 或低量化选<0>; 不推荐关, 因为一定会出现块失真, 而去块失败/过度皆影响画质. 可以在高压下为提高压缩率而开高(ノ* °▽ °*)

搜索精度: <大于 2>易误判, <小于-1>会遗漏, 建议保持<默认 0>, 除非针对高量化开高

--no-deblock<开关>快速编码，直播时实在是没法优化了就关掉以降低占用(@‘_’@)

环路滤波-取样迁就偏移 SAO

sample adaptive offset 并不在 h.265 标准范围内, 而是后来引入的新技术. 相比 x264/5 在去块完成后, 加了滤镜. 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() /__o

--sao<默认开启>sao 采用带偏移 band offset, 边缘偏移 edge offset 以及不偏移三种策略修改画面, 使上下帧画面微小细节一致化的处理

其中, 边缘偏移 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, 就迁就多数的锐化/柔化这个区域. 就 x265 版本 3 来说, eo 不能自行判断应该锐化/柔化时就一律柔化, 这样做虽然降低了码率, 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 image segmentation. 然后对偏离程度最大的, 4 个连一起的分带进行迁就处理(ò_ó`)D 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

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<开关>关闭--sao. 早期版本的 x265 中没有 limit-sao 参数, 所以编码画质时可以使用以防止 sao 造成的画面问题(ノ へ ヽ,)

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

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

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

--selective-sao<0~4, 默认 0>从条带(横向一组 CTU)角度调整 sao 参数, 0 关闭, 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

真无损压缩 近无损压缩

--lossless<开关>跳过分块, 动/帧/参搜索, 量/自适应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

x264 中 qp 0 是无损量化, 而 x265 中 qp 4 是无损量化, 0~3 关闭量化, 但不确定 4 以上的设定相比 x264 有没有变化, 由于能力不足, 我未能了解这是否影响到了参数 crf, 而其中一个证据即 x265 的默认 crf 值偏高(¬_¬)

真无损的导出有几率会比近无损还小, 因为参考块的质量提升了, 而参考是视频压缩的重点

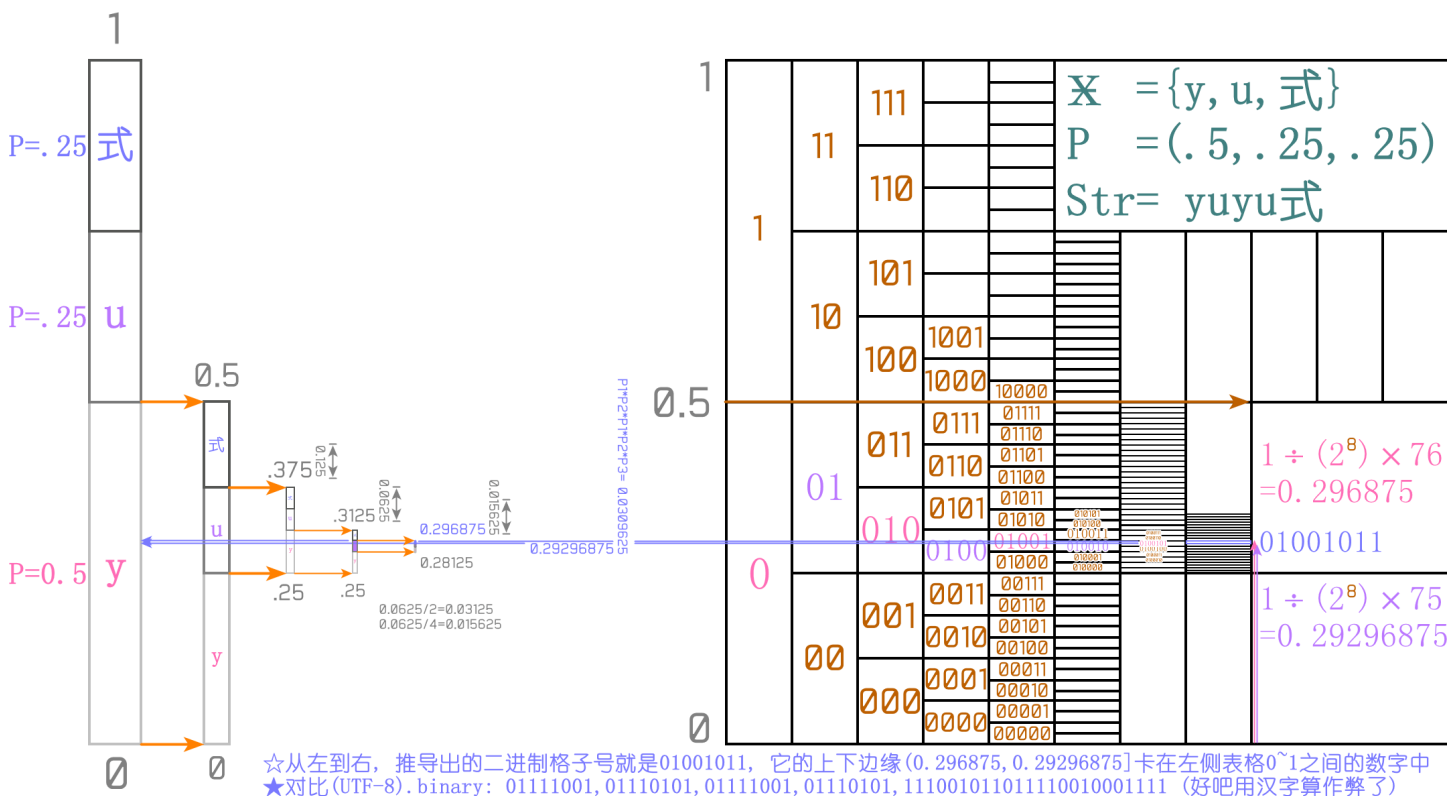
--tskip<开关, 已关>不在 tu 上使用 DCT 变换 (. •̂ - ")

--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, 只要码率允许就不做量化. 用更多码率换取原画相似度, 无损源能提高参考冗余

熵编码-CABAC(新内容待复查, [来源](#))

游程编码将降维后的块/条带丢给熵编码, 也就是最后文本压缩的步骤. x264/5 中使用了 context adapt. binary arithmetic coding 上下文自适应可变长度二进制编码. 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(中 °Δ °)中

其中, 算数编码 arithmetic coding 的原理是在 0~1 之间的压缩模具中按给出匹配符(阵列), 出字概率(其实就是次数)以及待压缩内容从左到右细分. 例如匹配 $x=\{y, u, \text{式}\}$ 三种符, 概率 P 分别为 .5, .25 和 .25, 那么压缩 "yuyu 式" 就按照下图的规律, 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 $x=\{ng, a, b, b, b, b, be, a, b, qs, q, \dots, EOF\}$, 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 $\angle(\star \circ \circ \star)$

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 $\div 2$ 的限制挡住, 造成类似 8bit 视频有时比 10bit 体积大的现象(っ`ヾ`c)

VPS PPS SPS RPS 解码参数集

图参数集 picture parameter set, 序列参数集 sequence PS, 率参数集 rate PS 是在 h.264 中引入的, 避免因丢包造成画面瑕疵的问题. (包括 gop 头 header 掉了, 导致满 gop 画面崩坏的情况). 参数集代表给一段画面解码用的指导, 不是一组命令行参数的集合哦(>O<)

其实就是弄个备用头信息 header 出来, 这样看网络视频就能任解码器以最快的速度, 不按顺序解码了

--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让解码端知道 SPS, PPS 等信息, 以正常播放

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

卡时 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

--frame-dup<开关, 已关, 开 vbr 开 hrd, 有失灵 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

--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 $2 \div (1/30 \text{ 秒}) = 60\text{Mbps}$

--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>串流时只发一个装着全部 sei 的大 NALU 而非每 gop 提供一个, 网速稳定可以试试

--idr-recovery-sei<开关>sei 里写进 idr 帧的位置, 串流防止整个 gop 都找不到参考帧崩坏的机制

线程

--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, 占用其中的第一个节点, +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

参考帧步骤要等其之前的步骤算完才开始, 若 cpu 线程小于 threads, 就会因为处理器随机算的特性, 从任务数量上冲淡参考的优先级. 编码器只能等一会儿再设参考帧, 降低处理器占用. 优先级低, 参考帧计算时间窗口就更小, 码率增加画质降低速度变慢

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

--pme<开关>使用平行动态搜索 parallel motion estimation, 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). 若已经占用 100%则不需要(@ -Д -@;)

--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 - anonymous 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. 由于参考帧 ref 需要关联前后至少 1~2 帧, 所以 x264/5 都限制了参考范围到帧间其下一行的宏块/ctu, 确保了所有帧同时吞吐 $\mathcal{O}(\cdot \times \cdot)$ 造成 x265 多线程多帧困难的原因有三

1. ctu 比宏块大, 相似性降低了 $\simeq(\nabla \cdot)$
2. 参考之前要先让环路滤波处理, 环路滤波还要等之前的编码算完, 使得很多参考, 特别是高 ref 设定下来不及找, 只能跳过

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

--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)>分出多少个线程专门找参考, 而非与帧编码一同占用线程, 可能只有在开启 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 视频一定要看视频信息设, 找不到不要用, 例子见下

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

```
Bit depth           : 10 bits
Bits/(Pixel*Frame)  : 0.120
Stream size         : 21.3 GiB (84%)
Default             : Yes
Forced              : No
Color range         : Limited
Color primaries     : BT.2020
Transfer characteristics : PQ
Matrix coefficients : BT.2020 non-constant
Mastering display color primaries: R: x=0.680000 y=0.320000,
G: x=0.265000 y=0.690000, B: x=0.150000 y=0.060000, White point: x=0.312700 y=0.329
Mastering display luminance: min: 0.0000 cd/m2, max: 1000.0000 cd/m2
Maximum Content Light Level: 1000 cd/m2
Maximum Frame-Average Light Level: 640 cd/m2
```

←图应设 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 ictp, 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 会将三个信息一并写在一行, 如 Stream #0:0(und): Video: prores (XQ) (ap4x /
0x78347061), yuv444p12le (tv, bt2020nc/bt2020/smpte2084, progressive)

2pass 模式

--pass<整数, 范围 1~2>指定当前 pass 位置的值, 比如在进行 pass1 就指定 --pass 1, 影响到导出/导
入类以及 2pass 模块参数的功能(・`Д´) /

--stats<文件路径+文件名>指定输出/导入分析数据的位置和命名, pass=1 时导出, pass=2 时读取. 默
认在 x265 所在处导入/导出 x265_2pass.log 所以若不需要指刻意指定就不用设(—)

--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

位深: 10 位

数据密度【码率/(像素×帧率)】: 0.251

流大小: 41.0 GiB (90%)

编码函数库: ATEME Titan File 3.8.3 (4.8.3.0)

Default: 是

Forced: 否

色彩范围: Limited

基色: BT.2020

传输特质: PQ

矩阵系数: BT.2020 non-constant

控制显示基色: Display P3

控制显示亮度: min: 0.0050 cd/m2, max: 4000 cd/m2

最大内容亮度等级: 1655 cd/m2

最大帧平均亮度等级: 117 cd/m2

2pass 模块

x265 有两种 2pass 模块: **multi-pass-opt** 在 pass1 与 2 中完整地储存, 对比动态搜索, 帧内搜索, 分块等信息, 达到最高画质和最有效的码率控制. **analysis** 模块更灵活, 直接沿用 pass1 的信息来, 可直接处理如 ABR 天梯之类多 pass 内分辨率不同的情况, 还能同时输入 multi-pass-opt 和 analysis 给出的两个文件, 更适合 Npass 模式, 当然, 两种模式肯定不能在一块用(づ. ◡. ◡.)づ

analysis 模块 save/load 部分:

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

--analysis-load<"文件名" 无默认>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/帧间矢量格式+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(^..^)/

multi-pass-opt 模块:

--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/矢量等信息. 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>根据失真(编码前后画面差)在 pass1/2 中重分配 qp. 使用更多时间(2pass)换取总体质量的参数. 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 模块 reuse 部分:

需要跑一遍 multi-pass 模式, 所以猜测应该是捡漏用的 \ (° - °) ...

--analysis-reuse-file<文件名 默认 x265_analysis.dat>导入 multi-pass-opt-analysis/distortion 信息的路径, 文件名

--analysis-reuse-mode<save/load 或 1/2>pass=1 时导出, 所以设在"1"或"save", pass=2 时读取, 所以设在"2"或"load". 已关闭

优化:

--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 ̎ ๑)

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

0= pass1 可信, 默认, 完全遵从 pass1 的分块深度和矢量格式(￣^￣)ゞ

1=分析所有 pass2 中与 pass1 相同分块的矢量格式, 除了 2pass 中比 1pass 更大的分块

2=一旦找出最佳的运动矢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 划分的块的动态矢量信息进行分析

3=保持使用 pass1 的分块程度, 但搜索矢量格式

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

0=pass1 可信, 默认, 全遵从 pass1 的分块深度和矢量格式

1=允许分析 pass2 中与 pass1 相同的分块与矢量格式, 除了 2pass 中比 1pass 更大的分块

2=一旦 pass1 找到了最佳分块程度/矢量格式, 就让 pass2 跳过

3=保持使用 pass1 的分块程度, 但优化动态矢量(๑.๑) 4=pass1 不可信, 丢弃不用

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

--scale-factor<开关, 仅配合--analysis-reuse-level 10 使用>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 anamuse-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 中可避免的失真

ABR 天梯

在编码器内部实现 analysis 模式 2pass abr 模式多视频输出的功能，来自苹果公司的 TN2224. 目的是方便平台布置多分辨率视频版本. 对做种，高压，画质党有方便从所有输出中挑出最佳的用途(>_<)

--abr-ladder<实验性参数，文件名.txt>见示例

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

```
[1440p:8:save1] --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:save2] --input 视频.yuv --input-res 3840x2160 --bitrate 16000 --ssim --psnr --csv 11.csv --csv-log-level 2 --output 3.hevc --scale-factor 0
```

[压制代号:analysis-load-reuse-level:analysis-load] <参数组 1+输入输出>

analysis-load-reuse-level 和 **analysis-load** 参数见 2pass→analysis 模块, load 填 nil 非 nul 可略过. 最好把不变的参数写在 x265 命令行内, 变化的写在文本内以减少阅读量

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☞(¬_¬")☞

--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=取决于分片种类, 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

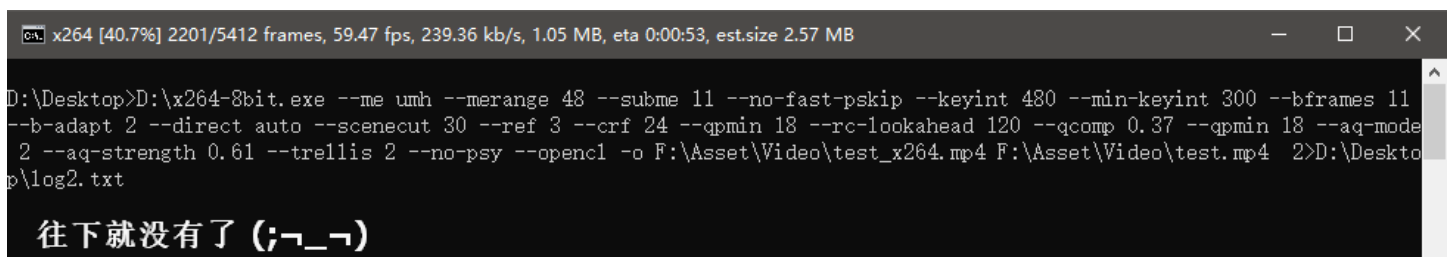
--fps<整数/浮点/分数>在使用 x265 时必须指定源视频的帧率, 小数帧勿做四舍五入('¬`*)

--chunk-start --chunk-end<开关, 需 no-open-gop>chunk-start 允许跨 IDR 帧制作数据包(?), chunk-end 将数据包结尾和剩下的视频帧断开(?)

根据描述来看, 由于数据包接收顺序一定会被打乱, 所以只可参考其之前, 而不可参考其之后的内容

由于 http 协议里的数据包编码 chunked encoding 会把视频切成一个个几秒长的数据包. 怎么按顺序发送数据就成了串流的重点, 比如服务器增强硬盘预取和顺序读写, 优化并行传输等Σ(¬_¬)

WinCMD LinuxBash 输出压制 log



```
C:\> x264 [40.7%] 2201/5412 frames, 59.47 fps, 239.36 kb/s, 1.05 MB, eta 0:00:53, est.size 2.57 MB
D:\Desktop>D:\x264-8bit.exe --me umh --merange 48 --subme 11 --no-fast-pskip --keyint 480 --min-keyint 300 --bframes 11
--b-adapt 2 --direct auto --scenecut 30 --ref 3 --crf 24 --qpmin 18 --rc-lookahead 120 --qcomp 0.37 --qpmin 18 --aq-mode
2 --aq-strength 0.61 --trellis 2 --no-psy --opencl -o F:\Asset\Video\test_x264.mp4 F:\Asset\Video\test.mp4 2>D:\Desktop\log2.txt
往下就没有了 (;¬_¬)
```

Win 系统使用后只能在 CMD 窗口标题看进度, 因为 log 被输进 txt 里了, 而 Unix 系统能用 tee 命令同时输给窗口和 txt. Win 系统压制软件能保存 log, 所以只需用在手动操作上(图片是 x264 但 x265 可用)

Windows CMD: x264(x265).exe[参数] 2>C:\文件夹\日志.txt . ^·I· ^.

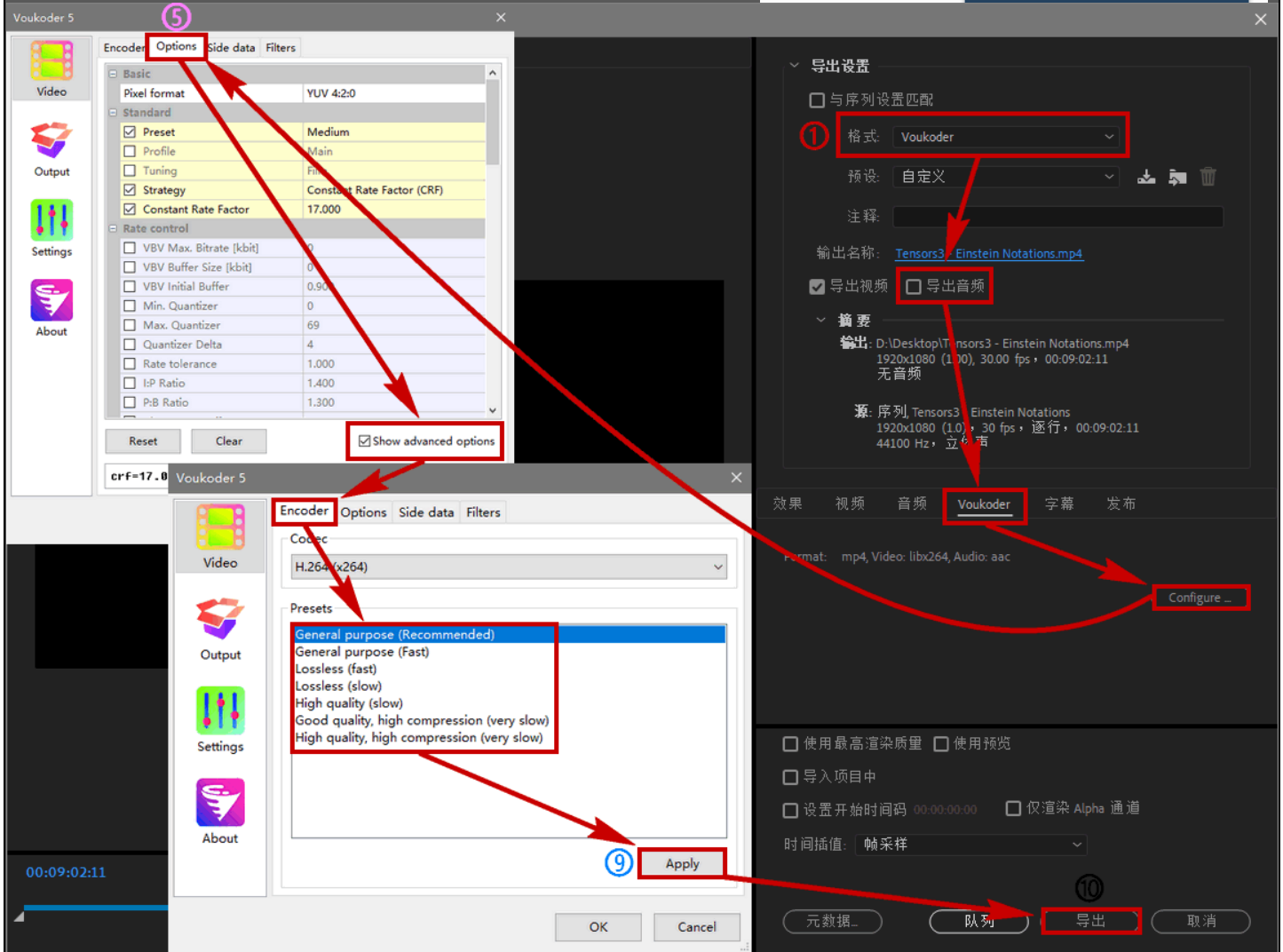
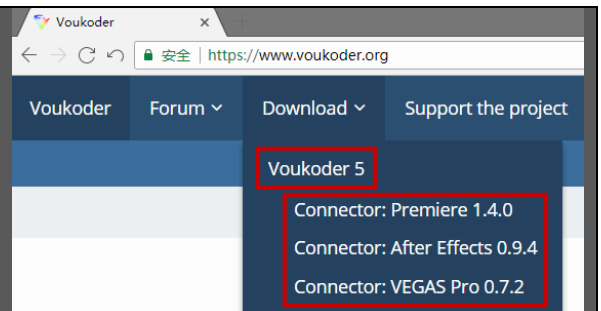
Linux Bash(或其它): x264(x265).exe[参数] 2>&1 | tee C:\文件夹\日志.txt

下载 附录与操作技巧

LigH	.hevc GCC10 [8-10-12bit] 附 x86, Windows XP x86 版 附 libx265.dll
MeteorRain-yuuki	ismash.mkv/mp4 或 .hevc [实验性, lavf 不如 pipe 可靠] GCC 9.3+ICC 1900+MSVC 1916 [8][10][12bit]+[8-10-12bit]
Rigaya	.hevc GCC 9.3 [8-10-12bit] 附 x86 版
Patman	.hevc GCC 11+MSVC1925 [8-10-12bit]
DJATOM-aMod	opt 代表 Intel 架构与 zen1~2 优化 [10bit], opt-znver3 代表 zen3 优化 [10-12bit] GCC 10.2.1+GCC10.3
ShortKatz	arm64~64e 加 x86 版 [?bit] 需 macOS 运行编译命令文件 ?
ffmpeg(全系统) : 备用地址 ottverse.com/ffmpeg-builds	
mpv 播放器 比 Potplayer 好在没有音频滤镜, 不用手动关; 没有颜色偏差, 文件体积小	
x265GuiEx 日语, 需要日语语言包, auo-setup 安装, 安装教程 aviutl.info/x265guiex/#toc4	

Voukoder; V-Connector 免费

Premiere/Vegas/AE 插件, 直接用 ffmpeg 内置编码器全色域导出, 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, 所以不同编译器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, 所以编译器只有优化源码. 同样, 速度测试应以 10bit x265 为基准(¬_¬)

查看 x265.exe 的版本, 编译, 色深 x265.exe -V

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、(·ω·ㄟ)

杜比视界 dolby vision 不深入研究

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

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

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

CMD 操作技巧 color 08

将原本黑景白字改成黑景灰字的单行命令, 有助于降低眼睛疲劳

cmd 窗口操作技巧%~dp0

"%~"是填充字的命令(不能直接用于 CMD). d/p/0 分别表示 drive 盘/path 路径/当前的第 n 号文件/盘符/路径, 数字范围是 0~9 所以即使输入 "%~dp01.mp4" 也会被理解为命令 dp0 和 1.mp4

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

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

.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

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

预设

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

配置项	默认值	推荐值
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

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

tune grain 最高画质

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

tune animation 动画片

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

tune fastdecode 解码加速

B 帧权重	关
P 帧权重	关
去块	关
取样迁就偏移	关

tune psnr 峰值信噪比

自适应量化	关
率失真优化 rd	关
cu 树	关