x265视频压缩教程**完整版**

欢迎阅读! 若有什么不会的可以加群[691892901](https://jq.qq.com/?_wv=1027&k=5YJFXyf). 本教程仅算盲人摸象的业余分析, 仅具备业余参考价值, 入门先看[x264视频压缩教程综合版](https://www.nazorip.site/archives/63). 手头需要编码压制不妨搭配[急用版](https://nazorip.site/archives/334/)教程

## 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

## Linux Bash报错导出

x265.exe [命令行] 2>&1 | tee home\[用户名]\Desktop\报错.txt

## Mac Terminal报错导出

x265.exe [命令行] 2>&1 | tee User\[用户名]\Desktop\报错.txt

## Windows CMD报错导出

x265.exe [命令行] 2> [桌面路径]\报错.txt

**目录**

分块 1

变换 2

帧间-动态搜索 3

帧间-子像素运动补偿 5

帧间-时域架网搜索 6

溯块向量搜索 7

初始化-Lookahead 8

帧内编码 10

率控制-CRF, ABR与qp 11

自适应量化 14

模式决策 15

率失真优化控制 16

环路滤波-去块滤镜 18

环路滤波-取样迁就偏移 18

熵编码/文本压缩 19

SEI补充与优化消息 29

线程节点控制 30

色彩空间转换, VUI/HDR信息, 黑边跳过 31

IO(输入输出) 32

# D:\Desktop\BZo2ZoY.png分块

hevc中, 帧下结构按面积大小分为帧🡪瓦tile🡪条带分片slice🡪条带段ss🡪ctu树单元🡪cu单元.

图: CTU

cu, cb由ctu经动态搜索ME与运动补偿MC隔离所得. 其U/unit指YCbCr一体; B/block则单指一面. **ipcm-cu**代表跳过MEMC, 直达环路滤波的「帧内编码pcm波形cu」intra pulse code modulation cu, 因为"块"本就是一串像素值的波形, 只是用元数据"换行"到了二维.

## D:\Desktop\x265教程\PU的对称+不对称划分.pngPU - 预测单元

prediction unit是编码完, 用做参考源的块. 支持cu上对称rectangle, 非对称asymmetric划分, 以更好的隔离动静态. 亮度与色度上的分裂法可以不同, 小至4×4像素

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

## D:\Desktop\x265教程\TU.pngTU - 变换单元

transformation unit的划分与cu而非pu同步, 实现变换和量化

## AU - 存取单元

access unit是解码端用于启动播放的块, 一般为IDR-AU

--ctu

<64/32/16, 默认64>编码树单元最大大小. 大则有损压缩效率高, 速度慢. 一般建议保持默认, 除非片有类似jpeg边缘损失的老片设<32>, 分辨率特别小的老片设<16>

--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 --amp

<开关, 默认关, 受limit-modes限制, 开amp需rect>pu的对称与不对称划分, 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

# 变换

## 一维傅里叶变换1D-FT

给出与原信号波形等高, 从最长的频率周期开始不断缩窄(周期增加)并根据源信号调整相位的参考余弦. 在参考余弦波变化的过程中, 记下两条波形吻合度变化的曲线-不同波形周期的振幅, 得频域信号. 反过来将频域所对应的波形加回去就是逆变换. 为将源波形中反相(上下颠倒)的余弦也考虑在内, 所以计算过程含取立方转正. "不断缩窄的参考余弦"可以用表示, 高等数学用表示, 相位可记为及即频域点的亮度和位移(2DFT下是复数旋角). 代表二轴正交坐标系统计一圈360°或2π的旋角, 几何坐标系中同理的超过180°或π会归零而不用, 见[desmos例1](https://www.desmos.com/calculator/qpnz9celzf), [例2](https://www.desmos.com/calculator/ywxqicajbv), [3b1b视频](https://www.youtube.com/watch?v=spUNpyF58BY)及[公开课](https://www.youtube.com/watch?v=tEzgtbnbXgQ)

## 二维傅里叶变换2D-FT

宽高上单拆出线来分别进行1DFT, 通过双求和ΣΣ或双积分∫∫整合. 在频域中相当于每个像素的变换结果相加或干涉. 亮则振幅大, 距中心远则频率高. 强在可编辑性, 可以消除打印喷头, 抖动等均匀噪声

## 二维离散余弦变换2D-DCT

用预制的二维波形模具, 穷举加减列出各波形格子的使用次数, 实现比2DFT更快但精度略差的变换.

--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>大tu使压缩高而慢, 以及瑕疵检测能力越差. 码率换时间+画质. 编码已有边缘损失的老片可以设ctu 32, 搭配max-tu-size 16

# 帧间-动态搜索

逐块于帧间找最小失真朝向direction of minimal distortion/DMD, 组成一张张帧间矢量表. 若找不全, 参考帧与分块的建立就不理想.

图: [十字搜索](https://typeset.io/pdf/displacement-measurement-and-its-application-in-interframe-3me36ivf7b.pdf), [大小菱搜索](https://www.researchgate.net/publication/254040728_Review_of_energy_efficient_block-matching_motion_estimation_algorithms_for_wireless_video_sensor_networks/)x264/5中, 六边形搜索hex将LDSP的上下左右斜8个外点减到6个, SDSP的细化规则不变

--analyze-src-pics

<开关, 默认关>允许动态搜索查找片源帧, 耗时增加压缩

--me

<hex~full, 推荐umh>搜索算法, umh平衡, 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)之后收益递减, sea是优化过的(x264)esa穷举, 相对浪费性能

--merange

<整数, 推荐4的倍数>取决于ME算法和分辨率, 过大会因「找不到更好, 找到也是错」而损失画质和压缩

* 1920x1080下设48左右
* 3840x2160下设52左右
* me hex下设16
* me umh-star设>=32

--no-temporal-mvp

<开关>关闭P-B分片的动态搜索, 除直播外不推荐

--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

<三整数, 需hme-search, 推荐默认16,32,48>对应1/16, ¼和全分辨率三画面



# 帧间-子像素运动补偿

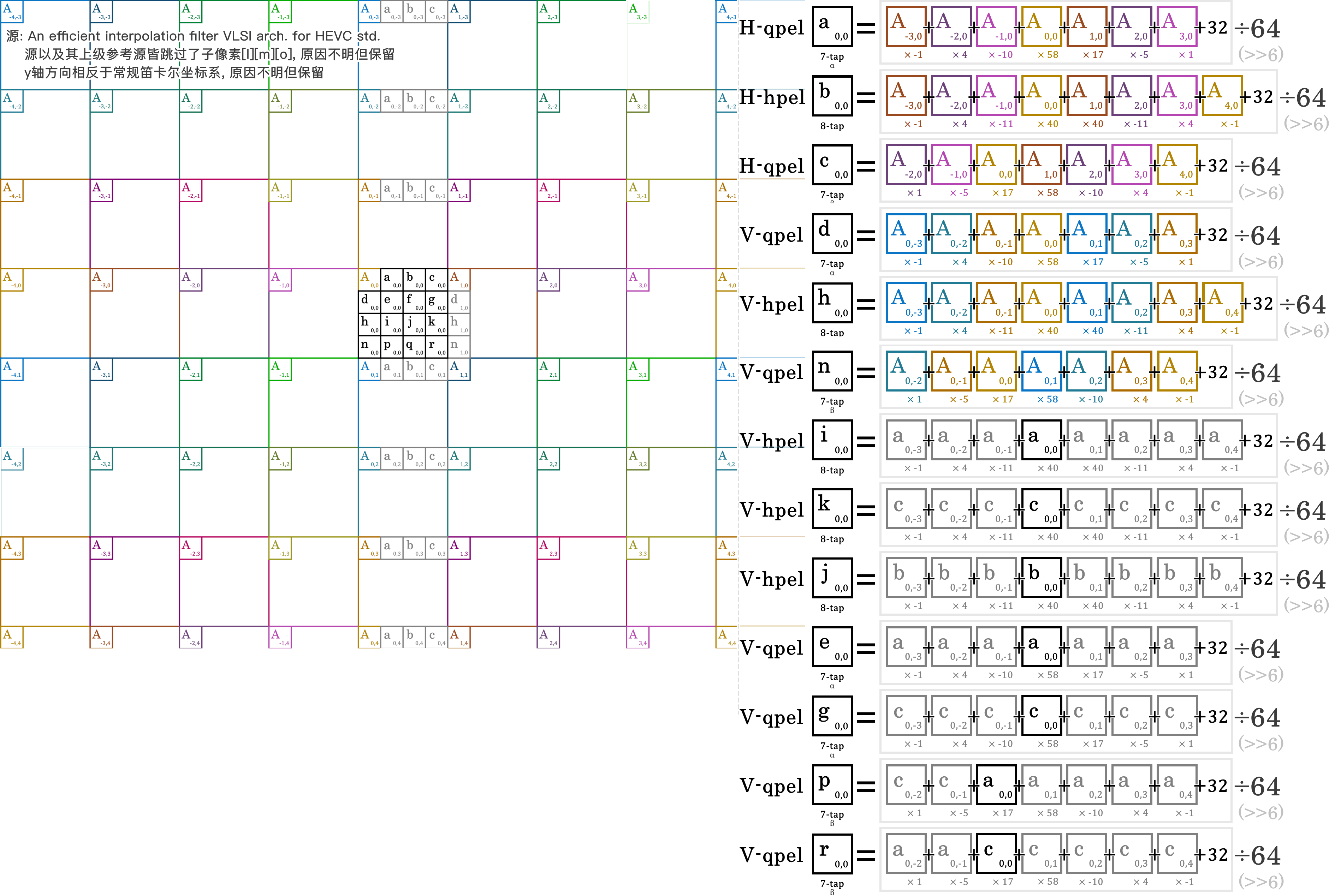
动态预测ME除了与帧内编码的输出帧做差以便推演P/B/i/I帧(见x264教程)外; 还被动态补偿motion compensation「允动画之移, 拦静画所变」的去误差处理; 以及"将帧间矢量表中动态矢量的精度提高到¼像素"的保真处理. 前者消除「动态噪点使静物变化」的误判, 后者修复「动态预测因精度低导致满屏块失真」的"粗加工P帧". 补偿即用「粗加工PU」对比「源视频同位帧上的块」, 使用

### 有限冲激响应滤镜finite impulse response (FIR) filter

放大. 此处指x264-6tap; x265-8tap, 7tap, 4tap滤镜. 放大后, 用SATD差距对准动态矢量, 得「精加工PU」

图: 此"子像素"特指是放大出的half-pixel(hpel)½像素, 以及quarter-pixel(qpel)¼像素. 表: x264/5实现h~qpel插值计算(实现了浮点→整数变量的程序优化)

|  |  |  |  |
| --- | --- | --- | --- |
| 编码器 | 平面/块 | 范围精度 | 插值方法 |
| x264官方 | 亮度**Y** | ½像素/hpel | 6tap FIR |
| x264官方 | 亮度**Y** | ¼像素/qpel | bilinear(lerp) |
| x264官方 | 色度**C** | hpel+qpel | 上下左右加权平均 |
| x265官方 | 亮度**Y** | ½像素/hpel | 8tap FIR |
| x265官方 | 亮度**Y** | ¼像素/qpel | 7tapα/β FIR |
| x265官方 | 色度**C** | hpel+qpel | 4tap FIR |

图: Y平面FIR插值和subme并行, 调用8²或16²块的横/纵向参考源. 若subme所得动态的:

* 向量横分量==0: **[d][n]**分别用7tapαβ采样整像素**[A]**
* 向量横分量!=0: **[f][q]**分别用7tapαβ采样子像素**[b]**
* 向量纵分量==0: **[a][c]**分别用7tapαβ采样整像素**[A]**
* 向量纵分量!=0: **[i][k]**用8tap分别采样子像素**[a][c]**

--subme

<整数范围1~7, 默认2>根据源帧率用下表判断. 由于x264中rdo选项和subme并用, 所以不与x265通用. SATD差异计算见x264教程

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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搭配率失真优化rdo | <3> | 2次 | 4 | 1 | 4 | SATD |
| 48fps搭配率失真优化rdo | <4> | 2次 | 4 | 2 | 4 | SATD |
| 60fps搭配率失真优化rdo | <5> | 1次 | 8 | 1 | 8 | SATD |
| 90fps 搭配率失真优化rdo | <6> | 2次 | 8 | 1 | 8 | SATD |
| 144fps搭配率失真优化rdo | <7> | 2次 | 8 | 2 | 8 | SATD |

## 加权预测weighted prediction

解决淡入淡出过程中, 部分pu因误参考, 亮度变化不同步的块失真问题. 分为P, B条带用的显加权, 编码器直接从原画和编码过的参考帧做差; B条带用的隐加权, 用参考帧的距离做加权平均插值

--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

## 一维小波变换1D-WT

让短波像拉链一样划过一维信号, 时间轴上给出频域信息(音频热度图), 支持更换波形以提取特征. 解决了傅里叶变换只有空-频域, 原生不支持描述信号随时间变化过程的问题. 缺点是分辨率低

## 时域动态补偿滤镜

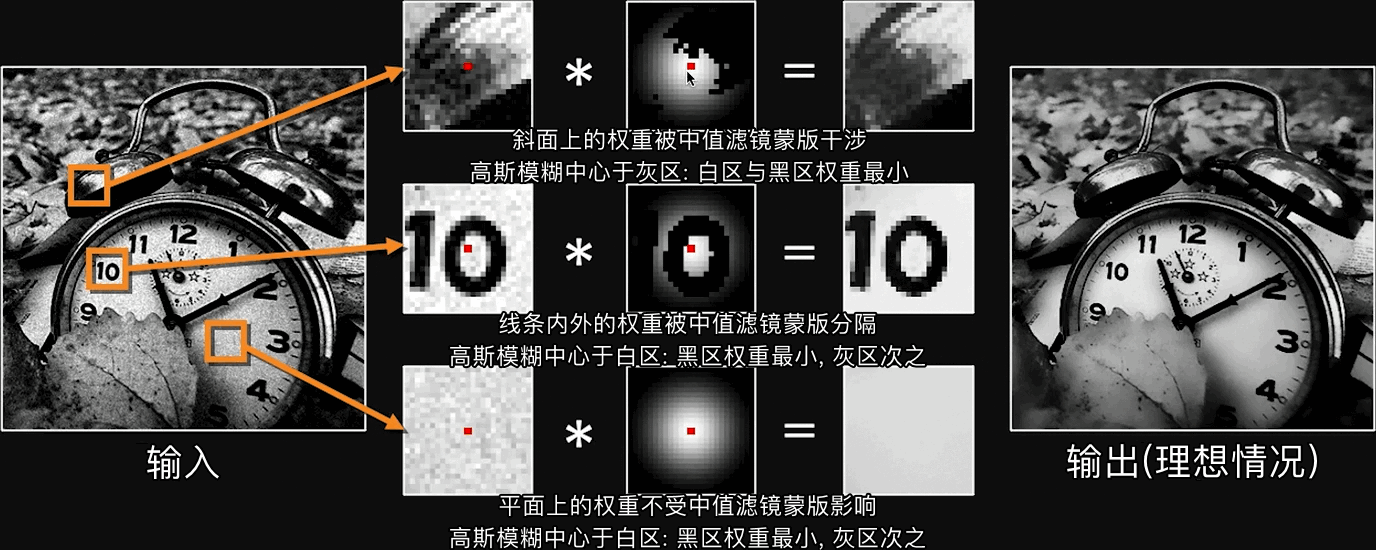
MC-temporal filter/

## 提升式小波变换

lifting-scheme以a-b帧之差来预测b-c区别, 预测对的更新到低频L带, 差错更新到不再参与下轮预测的高频H带. 继续在1层(L₁🡪Lₙ)向右迭代到gop (0-1-2, 2-3-4等帧组成第0层)中所有的L帧完成动态补偿为止. 实现了迭代n次即分离2ⁿ帧动静态, 以及所有的预测与补偿, 故不像传统动态搜索一样受缩放性scalability(分辨率大小)限制. 是SVC的核心算法. 迭代后的高低频用LL₁ LL₂ LH₁ LH₂,来表示低🡪高频的顺序, 字母位数代表迭代次数

--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

**高斯模糊**gaussian blur利用了正态分布函数面积不变的特性, 通过设定偏差程度σ决定正态分布钟的梯度: σ大则钟扁/滤镜中心分到的权重/面积越被更多分到权/面积的旁像素冲淡/模糊掉, 设计行业常用. [**中值滤镜**](https://en.wikipedia.org/wiki/Median_filter#/media/File:Median_filter_example.jpg)median filter和高斯, 卷积滤镜一样用n×n的滤镜格子逐像素扫图, 区别是将滤镜中心替换为旁像素的中值. 如此一来线条边缘会在旁像素两端重复, 使替换中值失效, 噪点/颗粒遭涂抹, 生成仅有平/斜面和完整线条边缘的"模糊"结果. [**双边滤镜**](https://www.youtube.com/watch?v=7FP7ndMEfsc)bilateral filter将后者输出作为高斯的权重蒙版(数学上矩阵点除); 平面/斜面/线条决定高斯模糊的正态分布钟在对应位置被保留/干涉/隔断

# 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. 所以将涉及同物的PB合到一起就能冗余大批PU的动态向量. hevc与avc一样用ref参数于时域上划区, 逐pu创建List0和List1左右两排参考列表. 差别是hevc在avc direct auto的基础上升级了AMVP与merge mode两种方案.

## 高级向量预测

Advanced Motion Vector Prediction用相对繁琐精确的几步为最初的几个块找出动态信息, 写成向量:

* 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
* 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
* 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
* 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
* 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

然后用相对简单含糊的

## 并合搜索

merge mode接手剩余块的向量, 从帧间, 帧内凑五个参选块candidate, 两个备选少服从多地统一动态向量实现合并merge. 计算跳过PU的边缘及当前向量以提速

--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

[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

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. x264中的视频帧数即

sps->vui.i\_num\_units\_in\_tick / sps->vui.i\_time\_scale÷2

所得(÷1则为交错场数)

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

--opt-qp-pps; --opt-ref-list-length-pps

<开关, 默认关. 已知兼容性问题故不推荐>据上个GOP改动当前PPS中默认的qp/ref参数值. [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
* 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/DRA净/脏任意访问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
* 「脏」指一组含i块的P帧, 需要全部解码才能重建出i帧. 压缩更高但相比i帧的解码更容易糊(脏)

**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

--keyint

<整数>一般设9×帧率(9秒一个落点), 影响视频解码计算难度和拖进度条的延迟

* 短视频, 不拖进度条可以设keyint -1稍微降低文件体积
* 剪辑素材, 频繁于多个视频(轨道)上访问画面, 根据画面复杂度设5~2.5秒一个, 即2.5×fps降低解码负载

--min-keyint

<整数, 默认25>判断新发现的转场距上个IDR帧是否小于该值长短. 有两种设定逻辑, 而它们给出的画质都一样:

* 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, x264/5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

--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

* 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

## 参考帧

--ref

<整数1~16, 推荐fps÷100+3.4>向量溯块前后帧数半径, 一图流设1. 要在能溯全所有块的情况下降低参考面积, 所以一般设3就不管了

--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



图: 强力平滑滤镜:

* 从横-纵向两个参考点直接线性插值lerp出所有参考点, 以及所对应的预测像素p
* 缓解了空间域色带问题

图: C-PB缺少B-CB的参考源就用A-CB的最右参考源做副本, EDAB四个CB都缺则拿F-CB的顶部参考源替代, 否则所有参考源填像素中值



图: 强力平滑滤镜的启用条件:

* C-PB的面积小于32×32
* 底-中-顶, 及左-中-右三个纵/横向参考源两两差之和小于视频位深, 8bit下为8
* 非DC, 非平行(夹角10), 非垂直(夹角26)的帧内编码模式

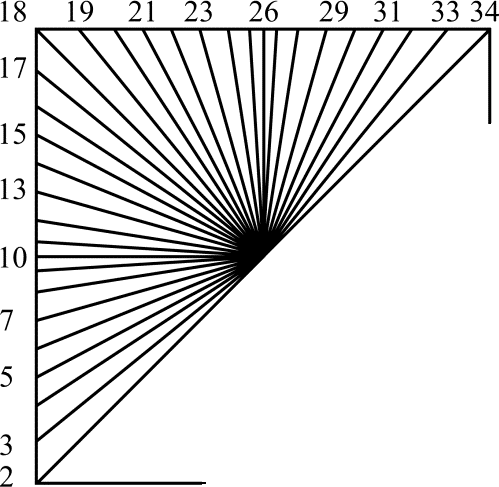
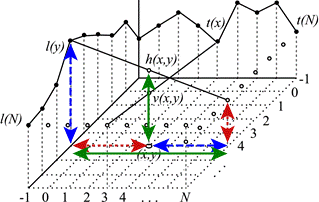


图: 3tap FIR :

用横向t(x)与纵向l(y)各3像素加权平均得预测像素p, 按卷积顺序轮询得到PU



预处理后用**趋平**/**夹角**/**DC**模式初步编码PB到CB. **趋平**模式即双线性插值bi-lerp, 让左-上过渡为右-下平面. 由底·高+底·高=h·底长的关系得过渡线h; t(x)再做一遍得过渡线v;

h, v取平均得预测像素p

**夹角**模式(35种)使渐变/斜面PB无损压缩为参考源+共角θ的拓补结构. 通过穷举所有p(x,y)的角度对齐的t(x)/l(y)参考源实现, 以及差异最小的点/构成直角三角, 用三角函数得一预测像素p的θ. 若角度落于两参考点间, 则对比其加权插值以换精度, **DC**模式则代表"以CB和PB的平均值判断"为平面, 即「直流信号」

组成参考源(I帧)+参考帧的帧间结构后, 数据会集中到I帧/I块上. 故先使用单图无损压缩——补偿参考源, 平滑(3-tap/ss), 和编码PB(趋平/夹角/DC)三步. **补偿**解决PB位于边角, 或等不到其它CB编码完成而缺失的参考源;

## 平滑预处理

即根据情况, 选择3-tap FIR或强力平滑滤镜strong intra smoothing, 卷积插值出「纯预测PU」

--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

--fast-intra, --b-intra

<开关, rd>4时关, 推荐开>fast-intra先查夹角模式2,10,18,26,34, 再加精度到5, 15, 21, 31, 再用最高精度; b-intra代表B分片同样进行帧内编码压缩. 关闭则浪费算力

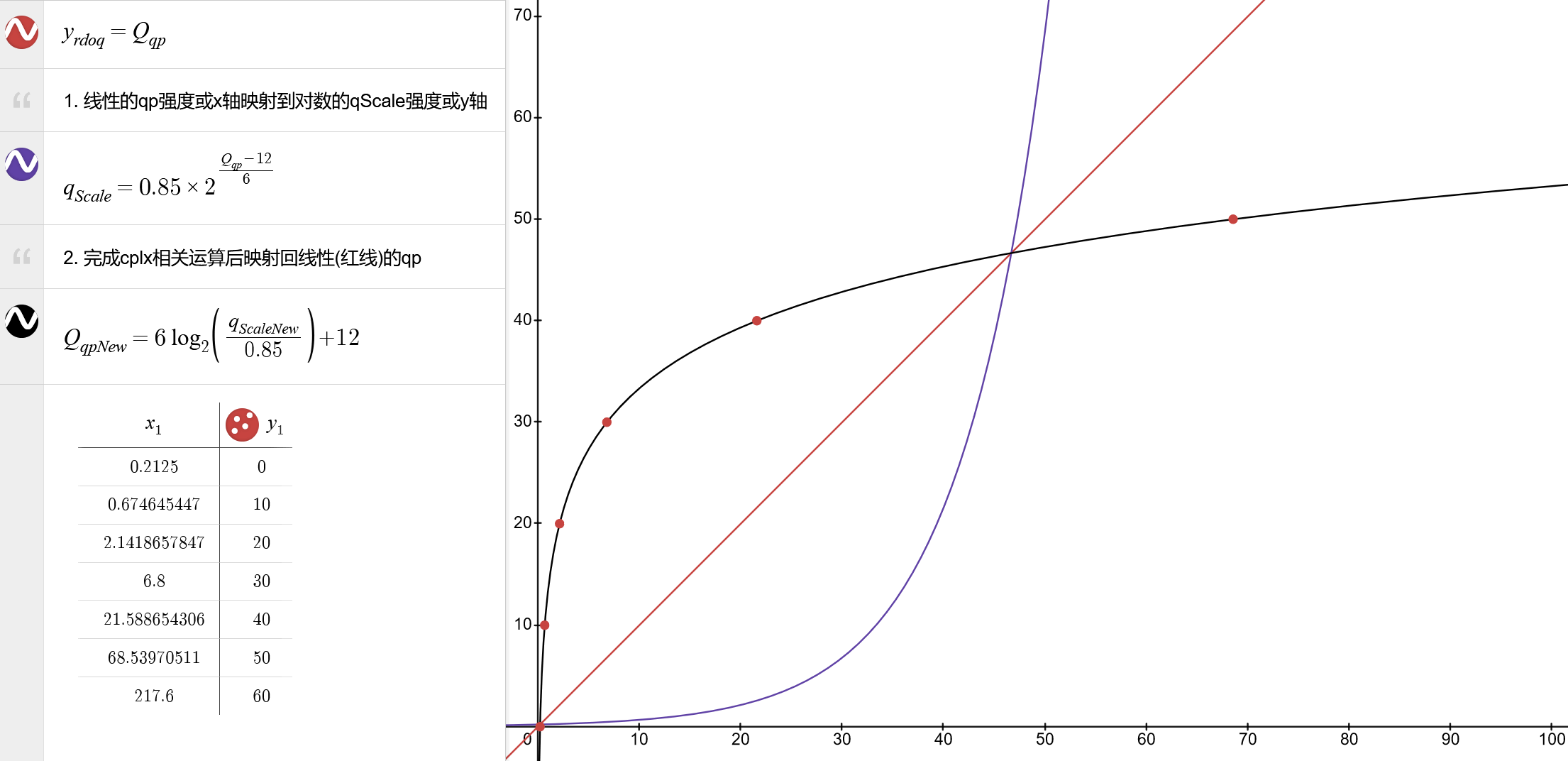
--no-strong-intra-smoothing

<开关, 不建议动>筛选条件苛刻, 平滑参考点而非pu, 去色带

## 量化

见x264教程

# 率控制-CRF, ABR与qp



人对画面细节呈对数㏒状的感知奠定了

图: 量化值qp(x轴)到强度qScale(y轴)的映射, 类似于对显示器(增强暗部细节)的伽马映射曲线.

见[desmos例](https://www.desmos.com/calculator/vernjiiphf).

由于当前帧此时还未编码(码率未知), 故寻已编码前帧的量化失真程度(越高则后帧理应越复杂), 做推演复杂度/模糊复杂度. crf越高则除进推演复杂度的分母越大/abr越低则分子越小, 得到的推演复杂度越低, 推演出量化值的qp就越高

|  |  |  |
| --- | --- | --- |
| **cplxSum** | 初始 = 半数宏块÷CTU,  逻辑是SATD[当前] = SATD[旧] + SATD[当前] |  |
| **cplxCount** | 初始0的当前帧数计数, 用于逐帧对cplxBlur加权  除以2与cplxSum同步  加权是制造越往后参考冗余越好, 压缩越强的规律 |  |
| **cplxBlur** | 模糊复杂度. 据帧所处位置推演加权的新SATD  近100%则说明当前帧复杂度往高推演, 画面复杂度呈涨势  可扭转cplxCount默认的跌势, 代表码率为画面增加 |  |
| **qScale** | GOP内累计的直线化qp, 或率失真优化量化的拉格朗日值λ  已编码帧的qp转qScale, 便于其它参数修改更新 |  |
| **ABR\_rate\_factor** | GOP初始值, abr下的qScale(rdoqλ)转qp |  |
| **ABR\_qScaleNew** | 据abr控制更新一遍qScale(rdoqλ) |  |
| **cplxBase** | 恒定值, crf模式默认的复杂度  若用B帧编码则CTU或宏块数量×120, 否则×80 |  |
| **CRF\_rate\_factor** | GOP内累计, 经cutree, B帧偏移乘进qScale后得到  1-qcomp与CRF\_qScale对齐(仅cplxBase, cplxBlur运算) |  |
| **CRF\_qScaleNew** | 据crf rate factor更新此帧的qScale(rdoqλ) |  |
| **qp** | qScale(rdoqλ)经调整后得到当前帧的量化值qp  各qp对应一套DCT变换量化矩阵, 概述见x264教程 |  |

经此实现了帧内画面复杂则qp低, 简则高; 同时据用户设定强度(虽照上式可见影响偏小)的动态变化. 这种质量判断只有两帧而不宏观, 所以引出了各种各样的优化步骤, 如mb/cutree, rdoq等

## 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, 0~51>最小量化值. 由于画质和优质参考帧呈正比, 所以仅高压环境建议设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cu/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

## 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, 就会得到码率比设定的片子; 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, 同时画质差的片子. 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

## 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, 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0~69, 禁用crf/abr+模式决策+率失真优化>直接设定全局量化强度. 影响其后的, 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

--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

--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

--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

--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-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, 速度最慢, 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>以¼tile而非aq的边缘高频信息实现自适应. 据结论[α](https://forum.doom9.org/showthread.php?p=1925373#post1925373), [β](https://forum.doom9.org/showthread.php?p=1925464): hevc-aq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, 默认64, ≥min-cu-size>最小应用自适应量化aq的cu. 越小则aq影响越大, 高画质与平衡都建议设在32/16. <最浅,最深>格式如32,16代表仅这两种cu, 忽略64和8

--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, 宏观地定制分块参考量化等细分方案. 因为选码率最小的压缩方案不平衡, 画质容易崩坏. 注意片源含明显边缘失真时反而要减少决策优化

--rd

<1/2/3/5, 默认3, 大则慢>优化模式决策md的程度. 建议快速用1, 2; 高压用3; 片源数据无损(非视觉无损)时用5

* <1>优化帧内参考, 并块/跳过决策, 含明显边缘失真时用
* <2>+分块决策, 含明显边缘失真时用
* <3>+帧间决策, 高压高量化时可平衡
* <5==6>+向量/帧间方向预测决策. 比3慢一倍, 片源含边缘失真时会强化失真

--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

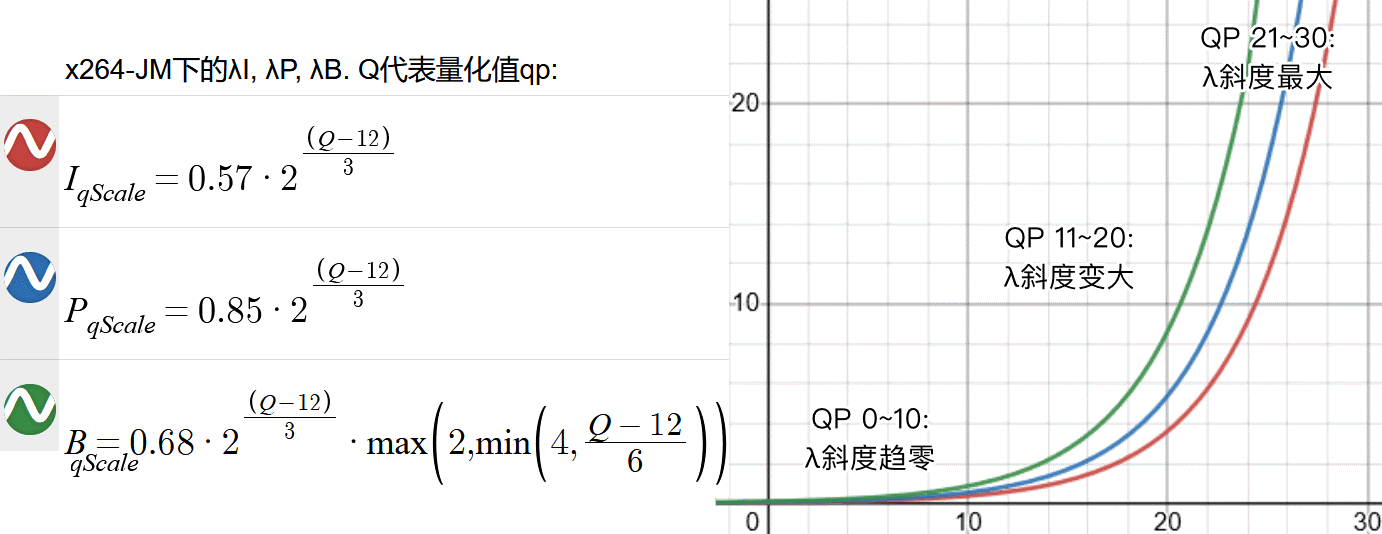
<0~100默认5: 趋向于分块, 需rskip大于1>用Sobel算法获取cu纹理密度, 除以块所占面积的百分比值. 纹理密度>阈值=分块. 量化强度越高越关键, 与x264 deadzone略像. 8×8或16×16块下默认5%即含3或12个系子就分

* <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

率失真优化

rate distortion optimization据多个码率下测得的cu失真(编码前后像素值差D), 挑出R-D曲线下的平衡区间. 由代价函数开销J=失真D + λ⋅码率R实现

模式决策

. 拉格朗日值λ

源于qp, 即crf, abr指定的率失真斜率slope区间. qp越大斜度越小. λ=0是无斜度的「代价=失真(码率变而画面不会变, 应压缩)」. λ趋0则是「开销趋失真(适当压缩不影响画质)」; λ>0则「开销>失真(保画质收益>压缩收益, 应该保画质)」

* 失真用均差异平方MSE判断:
* x264 中则用更快的差异平方和/SSE:
* 或选用画质更好的高频加权总差方NSSE:
* 指块宽度求和范围, f()和f'()分别代表参考块和参考源
* 指块高度求和范围, x, y指块中的像素坐标, | |求绝对值以便Σ正常求和
* MSE相比SSE精度有提升, 但多算了一步平均使其更慢; Noise SSE更强调高频信号(细节纹理或噪声)的权重

## 优化策略-随量化强度增加而增加

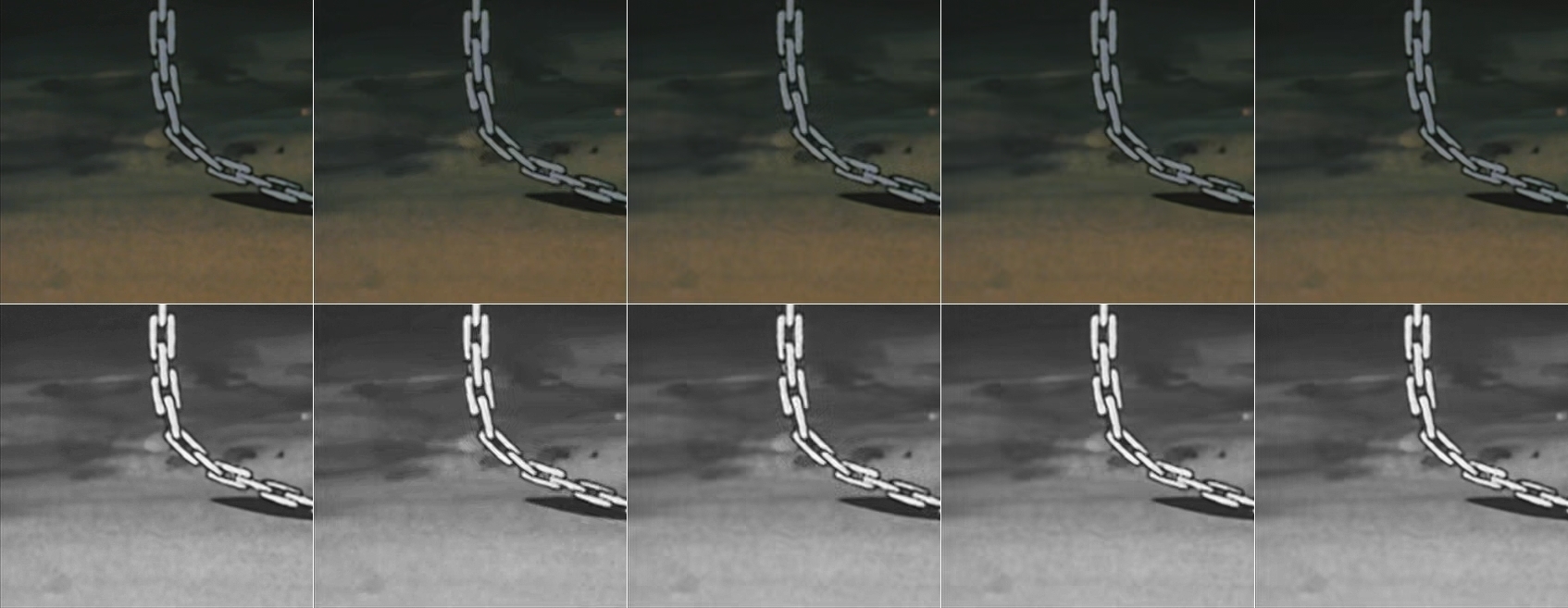
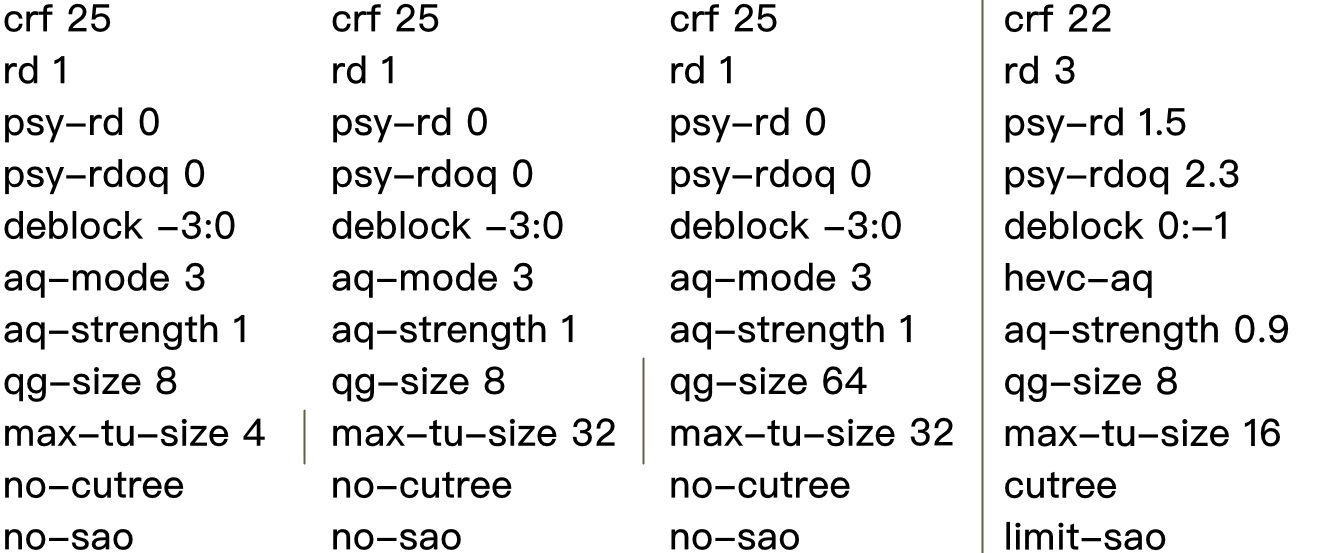


图: 左1源有微量边缘失真以及颗粒. 2~4是高量化导致的worm失真, 蚊噪, 块失真. 5对应失真用psy-rd(oq), (limit)sao, deblock, aq, 降crf抑制

## 优化模式决策

--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0~4, rd小于5>给VBR限码画面调高rd止损. 1~4为rd搜索面积倍数,大则慢

--splitrd-skip

<开关, 默认关>在"所有当前CU分割致失真程度之总和"大于"同帧CU分割致失真程度之总和"时, 不独立计算rd值以加速

--psy-rd

<浮点0~50默认2.0, 需rd3, 需搭配psy-rdoq构成x264的psy-rd>心理视觉优化往高影响量化块的能量J, 使md偏好保留细节的程度, 值随分辨率增加而增加, 随片源边缘失真增加而降低, 并随量化强度增加而增加

* <0~0.1>片源有明显边缘失真, 优化则降低画质
* <0.2>高压或片源暗场有一点点边缘失真
* <0.5~2>根据要保留的边缘峭度以及文件大小, VBR模式的峰值码率决定
* <1.5~2.5>至少2k的录像片源, 根据分辨率和边缘损失程度决定

--qp-adaptation-range

<浮点1~6, 默认1>psy参数改动qp的最大范围, 宏观影响大, 片源含明显边缘失真时会强化失真

## 优化量化

### --rdoq-level

<整数, 范围0关~2>abr/crf分配量化值的宏观调控强度, 大则慢

* <1>逐分块进行率失真优化量化, psy-rdoq开启则倾向低量化
* <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, 大量减少4×4块, 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, 改量化偏好为保留细节. 值随分辨率增加而增加, 随片源边缘失真增加而降低, 并随量化强度增加而增加

* <0~0.1>片源有明显边缘失真, 优化则降低画质
* <2.3~2.8>电脑录屏, 中低成本动漫等几乎没有动态背景变化的情况
* <3~4.8>分辨率从1280x720到3840x2160的高成本电影动漫, 要求片源无明显边缘失真
* <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

### --nr-intra, --nr-inter

<整数0关~2000, 默认0, [1920x1080不建议超250](http://forum.doom9.net/showthread.php?s=81cd7c04a679401fe7c4e689e67eaeb8&p=1947571#post1947571)>基于mc, 给量化前变换完的i帧降噪. 其中nr-intra不如第三方降噪滤镜. 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, 结果类似低配双阈滤镜

## 峰值信噪比Peak signal-noise ratio

PSNR 信号强度用分贝dB表示. 信噪比SNR即信号÷其中噪声, 用于测量软硬件处理, 传输, 变换信号的所剩. dB用对数线y=㏒(x)显示. ㏒同样是为了突出重点范围内的变化, 不管信噪比太好/差的情况. 指数增长应用于信噪比中, 如信号增长1dB=音量/光压×2. 峰值更适合有实际最大值的数字信号. 转dB的算式即. -1代表从0开始数, ㏒₁₀对齐十进制



# 环路滤波-去块滤镜

修复高量化时宏块间出现明显横纵割痕瑕疵的平滑滤镜. 编码器内做去块能用到压缩待遇信息而减少误判. 码率跟不上就一定会出现块失真, 所以除直播关掉以加速外, 任何时候都应该用; 但去块手段目前仍是平滑滤镜, 因此要降低强度才适用于高码视频, 动漫, 素材录屏等锐利画面.

### 边界强度

boundary strength(去块力度判断): 取最小8x8块间的界线举例. (不是4x4)

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

--deblock

<平滑强度:搜索精度, 默认1:0, 推荐0:0, -1:-1, -2:-1>两值于原有强度上增减

* 平滑<≥1>时用以压缩, <0~1>时略微降低锐度, 适合串流
* 平滑<-2~-1>适合锐利视频源, 4k电影, 游戏录屏. 提高码率且会出现块失真
* 平滑<-3~-2>适合高码动画源/桌面录屏. 增块失真, 但观感还是比1好
* 搜索<大于2>易误判, <小于-1>会遗漏, 建议保持<0~-1>, 除非qp>26时设<1>

# 环路滤波-取样迁就偏移

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

* CTB内据分块结果建立一批3x3像素d 采样域
* 由3x3排列规律找出三个像素排成一排的横0, 纵1, 左倾2和右倾3四种可能
* 中心像素值要符合小于/大于旁像素且等于任意一边像素的条件, 否则不是边界
* 确保3x3间的边界连续性, 实现边界验证和性能优化(?)

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

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

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

### --no-sao

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

--sao-non-deblock

<开关>启用后, 未经由deblock分析的内容会被sao分析◑.◑

--no-sao-non-deblock

<默认>sao分析跳过视频右边和下边边界( /)u(\ )

--limit-sao

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

--selective-sao

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

# 熵编码/文本压缩

图: 简化的上下文自适应二进制算数编码[CABAC](https://iphome.hhi.de/marpe/cabac.html)流程, 分转二进制, 出字概率统计和算数编码三步

## 算数编码

CABAC的核心结构. 将输入文本按字数多少分割0~1间的出字概率.

图: 编码即逐字按出字概率缩窄概率条到最后一个字的出字概率, 取上/下区间为编码结果, 解码照概率位置(蓝线)的落点反推实现.

优化是尽可能控制概率分布条上落点(得尽可能短的结果小数), 以及通过降噪变换量化和游程编码以简化输入信号, 实现尽可能高的无损压缩

## 自适应Adaptive与对等Uniform出字概率

预先统计完出字概率(需要全文)限制了算数编码的用途. 解决方法是利用文本编码一次编一字的特性, 以「编一字, 一改出字概率」的推演实现概率自适应. 添切换符⊗代表自适概率0的生字, 以临时用对等概率编码: (仅推演概率, 编码部分以及变化的概率分布结合上图)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **造词典** |  |  |  |  |  |  |  | 切换符 |
| 出字词典 | h | e | l | o |  | w | r | d | ⊗ |
| 自适概率 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 对等概率 | 1/8 | 1/8 | 1/8 | 1/8 | 1/8 | 1/8 | 1/8 | 1/8 | 0 |
| **2** | **编码h** |  |  |  |  |  |  |  |  |
| 自适概率 | 1/2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1/2 |
| 生成字符 | ⊗h |  |  |  |  |  |  |  |  |
| **3** | **编码e** |  |  |  |  |  |  |  |  |
| 自适概率 | 1/3 | 1/3 | 0 | 0 | 0 | 0 | 0 | 0 | 1/3 |
| 生成字符 | ⊗h | ⊗e |  |  |  |  |  |  |  |
| **4** | **编码l** |  |  |  |  |  |  |  |  |
| 自适概率 | 1/4 | 1/4 | 1/4 | 0 | 0 | 0 | 0 | 0 | 1/4 |
| 生成字符 | ⊗h | ⊗e | ⊗l |  |  |  |  |  |  |
| **5** | **编码l** |  |  |  |  |  |  |  |  |
| 自适概率 | 1/5 | 1/5 | 2/5 | 0 | 0 | 0 | 0 | 0 | 1/5 |
| 生成字符 | ⊗h | ⊗e | ⊗ll |  |  |  |  |  |  |
| **6** | **编码o** |  |  |  |  |  |  |  |  |
| 自适概率 | 1/6 | 1/6 | 2/6 | 1/6 | 0 | 0 | 0 | 0 | 1/6 |
| 生成字符 | ⊗h | ⊗e | ⊗ll | ⊗o |  |  |  |  |  |
| **7** | **编码空格** |  |  |  |  |  |  |  |  |
| 自适概率 | 1/7 | 1/7 | 2/7 | 1/7 | 1/7 | 0 | 0 | 0 | 1/7 |
| 生成字符 | ⊗h | ⊗e | ⊗ll | ⊗o | ⊗ |  |  |  |  |
| **8** | **编码w** |  |  |  |  |  |  |  |  |
| 自适概率 | 1/8 | 1/8 | 2/8 | 1/8 | 1/8 | 1/8 | 0 | 0 | 1/8 |
| 生成字符 | ⊗h | ⊗e | ⊗ll | ⊗o | ⊗ | ⊗w |  |  |  |
| **9** | **编码o** |  |  |  |  |  |  |  |  |
| 自适概率 | 1/9 | 1/9 | 2/9 | 2/9 | 1/9 | 1/9 | 0 | 0 | 1/9 |
| 生成字符 | ⊗h | ⊗e | ⊗ll | ⊗o | ⊗ | ⊗wo |  |  |  |
| **10** | **编码r** |  |  |  |  |  |  |  |  |
| 自适概率 | 1/10 | 1/10 | 2/10 | 2/10 | 1/10 | 1/10 | 1/10 | 0 | 1/10 |
| 生成字符 | ⊗h | ⊗e | ⊗ll | ⊗o | ⊗ | ⊗wo | ⊗r |  |  |
| **11** | **编码l** |  |  |  |  |  |  |  |  |
| 自适概率 | 1/11 | 1/11 | 3/11 | 2/11 | 1/11 | 1/11 | 1/11 | 0 | 1/11 |
| 生成字符 | ⊗h | ⊗e | ⊗ll | ⊗o | ⊗ | ⊗wo | ⊗rl |  |  |
| **12** | **编码d** |  |  |  |  |  |  |  |  |
| 自适概率 | 1/12 | 1/12 | 3/12 | 2/12 | 1/12 | 1/12 | 1/12 | 1/12 | 1/12 |
| 生成字符 | ⊗h | ⊗e | ⊗ll | ⊗o | ⊗ | ⊗wo | ⊗rl | ⊗d | EOF |

## 上下文自适应-类LZ的PPM推演法简化版

拓展上例思路作上下文长0——CTX0时的推演, 添加CTX1或更高的范围, 每进一位字就「先查长, 后查短, 生字用对等；匹长概大, 配短概小」的上下文自适应出字概率推演; 如部分匹配预测法Prediction by Partial Matching设最长3字编码DNA链单侧的CTAGGCAATCTAGGTA时:

* CTX-1 无上下文(null)的负1/初始态, 储存对等概率, 编码生字
* CTX0 输入=字符1时, 上下文CTX为""的状态, 一次输一字, 相当于自适应概率本身
* CTX1 输入=字符2时, 上下文CTX为"<字符1>"的状态
* CTXn 输入≥字符n时, 上下文CTX为"<…><字符n-2><字符n-1>"的状态
* ⊗ 切换符, 叠加以显示临时倒退了多少个CTX表, 每个CTX表也因此有⊗的概率

1. 「匹长概大, 配短概小」: 先用最长且小于等于已知上下文长度的CTXn编码输入符号, 该情况下出字概率最大
2. 据已知上下文内容找到各个长度上下文中的概率分组(此处用分色表示)
   1. 新组⊗的概率记为1倍, 匹配到同组则用最新的概率倍数
   2. 旧组⊗的概率为最新的概率倍, 如已存在一项则⊗的概率为1/2
3. **小计**: 统计2, 2A, 2B的编码文本与概率结果
4. **CTX更新**:
   1. 新组建立新的½概率, 添加对应的⊗概率
   2. 旧组更新概率, 如1/2到2/3, 1/3到2/4, 以及更新⊗概率

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **出字词典** | **C** | **T** | **A** | **G** | **⊗** | **Dict** |
| **等概CTX-1** | **1/5** | **1/5** | **1/5** | **1/5** | **1/5** | **P-uniform** |
| **1:** | **输入C** | **已知null** | **null长0** | **CTAGGCAATC** | **TAGGTA** |  |
| CTX匹配 | **CTX3**匹配⊗ | **CTX2**匹配⊗ | **CTX1**匹配⊗ | **CTX0**匹配⊗ | **CTX-1**配C | **小计** |
| 编码结果 | 已知长0跳过 | 已知长0跳过 | 已知长0跳过 | ⊗ | C | "⊗C" |
| 概率结果 |  |  |  | P⊗ = 1 | P(C)=1/5 | 1×1/5 |
| CTX更新 |  |  |  | P(""C)=1/2 |  |  |
| ⊗概率 |  |  |  | P⊗=1/2 |  |  |
| **2:** | **输入T** | **已知"C"** | **已知长1** | **CTAGGCAATC** | **TAGGTA** |  |
| CTX匹配 | **CTX3**匹配⊗ | **CTX2**匹配⊗ | **CTX1**匹配⊗ | **CTX0**匹配⊗ | **CTX-1**配T | **小计** |
| 编码结果 | 已知长1跳过 | 已知长1跳过 | ⊗ | ⊗ | T | "⊗⊗T" |
| 概率结果 |  |  | P⊗ = 1 | P⊗ = 1/2 | P(T)=1/5 | 1×1/2×1/5 |
| CTX更新 |  |  | P("C"T)=1/2 | P(""C)=1/3 |  |  |
| CTX更新 |  |  |  | P(""T)=1/3 |  |  |
| ⊗概率 |  |  | P⊗=1/2 | P⊗=1/3 |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **3:** | | **输入A** | **已知"CT"** | | **已知长2** | | **CTAGGCAATC** | | **TAGGTA** | |  | |
| CTX匹配 | | **CTX3**匹配⊗ | **CTX2**匹配⊗ | | **CTX1**匹配⊗ | | **CTX0**匹配⊗ | | **CTX-1**配A | | **小计** | |
| 编码结果 | | 已知长2跳过 | ⊗ | | ⊗ | | ⊗ | | A | | "⊗⊗⊗A" | |
| 概率结果 | |  | P⊗ = 1 | | P⊗ = 1 | | P⊗ = 1/3 | | P(A)=1/5 | | 1×1 | |
| CTX更新 | |  | P("CT"A)=1/2 | | P("C"T)=1/2 | | P(""C)=1/4 | |  | | ×1/3×1/5 | |
| CTX更新 | |  |  | | P("T"A)=1/2 | | P(""T)=1/4 | |  | |  | |
| CTX更新 | |  |  | |  | | P(""A)=1/4 | |  | |  | |
| ⊗概率 | |  | P⊗=1/2 | | P⊗=1/2,1/2 | | P⊗=1/4 | |  | |  | |
| **4:** | | **输入G** | **已知"CTA"** | | **已知长3** | | | **CTAGGCAATC** | **TAGGTA** | |  | |
| CTX匹配 | | **CTX3**匹配⊗ | **CTX2**匹配⊗ | | **CTX1**匹配⊗ | | | **CTX0**匹配⊗ | **CTX-1**配G | | **小计** | |
| 编码结果 | | ⊗ | ⊗ | | ⊗ | | | ⊗ | G | | "⊗⊗⊗⊗G" | |
| 概率结果 | | P⊗ = 1 | P⊗ = 1 | | P⊗ = 1 | | | P⊗ = 1/4 | P(G)=1/5 | | 1×1×1 | |
| CTX更新 | | P("CTA"G)=1/2 | P("CT"A)=1/2 | | P("C"T)=1/2 | | | P(""C)=1/5 |  | | ×1/4×1/5 | |
| CTX更新 | |  | P("TA"G)=1/2 | | P("T"A)=1/2 | | | P(""T)=1/5 |  | |  | |
| CTX更新 | |  |  | | P("A"G)=1/2 | | | P(""A)=1/5 |  | |  | |
| CTX更新 | |  |  | |  | | | P(""G)=1/5 |  | |  | |
| ⊗概率 | | P⊗=1/2 | P⊗=1/2,1/2 | | P⊗=1/2,1/2,1/2 | | | P⊗=1/5 |  | |  | |
| **5**: | **输入G** | | | **已知"TAG"** | | **CTAGGCAATC** | | | | **TAGGTA** | |  |
| CTX匹配 | **CTX3**匹配⊗ | | | **CTX2**匹配⊗ | | **CTX1**匹配⊗ | | | | **CTX0**配G | | **小计** |
| 编结果 | ⊗ | | | ⊗ | | ⊗ | | | | G | | "⊗⊗⊗G" |
| 概结果 | P⊗ = 1 | | | P⊗ = 1 | | P⊗ = 1 | | | | P(""G)=1/5 | | 1×1×1 |
| CTX更新 | P("CTA"G)=1/2 | | | P("CT"A)=1/2 | | P("C"T)=1/2 | | | | P(""C)=1/6 | | ×1/5 |
| CTX更新 | P("TAG"G)=1/2 | | | P("TA"A)=1/2 | | P("T"A)=1/2 | | | | P(""T)=1/6 | |  |
| CTX更新 |  | | | P("AG"G)=1/2 | | P("A"G)=1/2 | | | | P(""A)=1/6 | |  |
| CTX更新 |  | | |  | | P("G"G)=1/2 | | | | P(""G)=2/6 | |  |
| ⊗概率 | P⊗=1/2,1/2 | | | P⊗=1/2,1/2,1/2 | | P⊗=1/2,1/2,1/2,1/2 | | | | P⊗=1/6 | |  |
| **6:** | **输入C** | | | **已知"AGG"** | | **CTAGGCAATC** | | | | **TAGGTA** | |  |
| CTX匹配 | **CTX3**匹配⊗ | | | **CTX2**匹配⊗ | | **CTX1**匹配⊗ | | | | **CTX0**配C | | **小计** |
| 编结果 | ⊗ | | | ⊗ | | ⊗ | | | | C | | "⊗⊗⊗C" |
| 概结果 | P⊗ = 1 | | | P⊗ = 1 | | P"G"⊗ = 1/2 | | | | P(""C)=1/6 | | 1×1× |
| CTX更新 | P("CTA"G)=1/2 | | | P("CT"A)=1/2 | | P("C"T)=1/2 | | | | P(""C)=2/7 | | 1/2× |
| CTX更新 | P("TAG"G)=1/2 | | | P("TA"G)=1/2 | | P("T"A)=1/2 | | | | P(""T)=1/7 | | 1/6 |
| CTX更新 | P("AGG"C)=1/2 | | | P("AG"G)=1/2 | | P("A"G)=1/2 | | | | P(""A)=1/7 | |  |
| CTX更新 |  | | | P("GG"C)=1/2 | | P("G"G)=1/3 | | | | P(""G)=2/7 | |  |
| CTX更新 |  | | |  | | P("G"C)=1/3 | | | |  | |  |
| ⊗概率 | P⊗=1/2,1/2,1/2 | | | P⊗=1/2,1/2,1/2,1/2 | | P⊗=1/2,1/2,1/2,1/3 | | | | P⊗=1/7 | |  |
| **7:** | **输入A** | | | **已知"GGC"** | | **CTAGGCAATC** | | | | **TAGGTA** | |  |
| CTX匹配 | **CTX3**匹配⊗ | | | **CTX2**匹配⊗ | | **CTX1**匹配⊗ | | | | **CTX0**配A | | **小计** |
| 编结果 | ⊗ | | | ⊗ | | ⊗ | | | | A | | "⊗⊗⊗A" |
| 概结果 | P⊗ = 1 | | | P⊗ = 1 | | P⊗ = 1/2 | | | | P(""A)=1/7 | | 1/2× |
| CTX更新 | P("CTA"G)=1/2 | | | P("CT"A)=1/2 | | P("C"T)=1/3 | | | | P(""C)=2/8 | | 1/7 |
| CTX更新 | P("TAG"G)=1/2 | | | P("TA"G)=1/2 | | P("T"A)=1/2 | | | | P(""T)=1/8 | |  |
| CTX更新 | P("AGG"C)=1/2 | | | P("AG"G)=1/2 | | P("A"G)=1/2 | | | | P(""A)=2/8 | |  |
| CTX更新 | P("GGC"A)=1/2 | | | P("GG"C)=1/2 | | P("G"G)=1/3 | | | | P(""G)=2/8 | |  |
| CTX更新 |  | | | P("GC"A)=1/2 | | P("G"C)=1/3 | | | |  | |  |
| CTX更新 |  | | |  | | P("C"A)=1/3 | | | |  | |  |
| ⊗概率 | 1/2,1/2,1/2,1/2 | | | 1/2,1/2,1/2,1/2,1/2 | | 1/3,1/2,1/2,1/3 | | | | P⊗=1/8 | |  |
| **8:** | **输入A** | | | **已知"GCA"** | | **CTAGGCAATC** | | | | **TAGGTA** | |  |
| CTX匹配 | **CTX3**匹配⊗ | | | **CTX2**匹配⊗ | | **CTX1**匹配⊗ | | | | **CTX0**配A | | **小计** |
| 编结果 | ⊗ | | | ⊗ | | ⊗ | | | | A | | "⊗⊗⊗A" |
| 概结果 | P⊗ = 1 | | | P⊗ = 1 | | P"A"⊗ = 1/2 | | | | P(""A)=1/8 | | 1/2× |
| CTX更新 | P("CTA"G)=1/2 | | | P("CT"A)=1/2 | | P("C"T)=1/3 | | | | P(""C)=2/9 | | 2/8 |
| CTX更新 | P("TAG"G)=1/2 | | | P("TA"G)=1/2 | | P("T"A)=1/2 | | | | P(""T)=1/9 | |  |
| CTX更新 | P("AGG"C)=1/2 | | | P("AG"G)=1/2 | | P("A"G)=1/3 | | | | P(""A)=3/9 | |  |
| CTX更新 | P("GGC"A)=1/2 | | | P("GG"C)=1/2 | | P("G"G)=1/3 | | | | P(""G)=2/9 | |  |
| CTX更新 | P("GCA"A)=1/2 | | | P("GC"A)=1/2 | | P("G"C)=1/3 | | | |  | |  |
| CTX更新 |  | | | P("CA"A)=1/2 | | P("C"A)=1/3 | | | |  | |  |
| CTX更新 |  | | |  | | P("A"A)=1/3 | | | |  | |  |
| ⊗概率 | 1/2,1/2,1/2,1/2,1/2 | | | 1/2,1/2,1/2,1/2,1/2,1/2 | | 1/3,1/2,1/3,1/3 | | | | P⊗=1/9 | |  |
| **9:** | **输入T** | | | **已知"CAA"** | | **CTAGGCAATC** | | | | **TAGGTA** | |  |
| CTX匹配 | **CTX3**匹配⊗ | | | **CTX2**匹配⊗ | | **CTX1**匹配⊗ | | | | **CTX0**配T | | **小计** |
| 编结果 | ⊗ | | | ⊗ | | ⊗ | | | | T | | "⊗⊗⊗T" |
| 概结果 | P⊗ = 1 | | | P"CA"⊗ = 1/2 | | P"A"⊗ = 1/3 | | | | P(""A)=1/9 | | 1/2×1/3 |
| CTX更新 | P("CTA"G)=1/2 | | | P("CT"A)=1/2 | | P("C"T)=1/3 | | | | P(""C)=2/10 | | ×1/9 |
| CTX更新 | P("TAG"G)=1/2 | | | P("TA"G)=1/2 | | P("T"A)=1/2 | | | | P(""T)=2/10 | |  |
| CTX更新 | P("AGG"C)=1/2 | | | P("AG"G)=1/2 | | P("A"G)=1/4 | | | | P(""A)=3/10 | |  |
| CTX更新 | P("GGC"A)=1/2 | | | P("GG"C)=1/2 | | P("G"G)=1/3 | | | | P(""G)=2/10 | |  |
| CTX更新 | P("GCA"A)=1/2 | | | P("GC"A)=1/2 | | P("G"C)=1/3 | | | |  | |  |
| CTX更新 | P("CAA"T)=1/2 | | | P("CA"A)=1/3 | | P("C"A)=1/3 | | | |  | |  |
| CTX更新 |  | | | P("CA"T)=1/3 | | P("A"A)=1/4 | | | |  | |  |
| CTX更新 |  | | |  | | P("A"T)=1/4 | | | |  | |  |
| ⊗概率 | 1/2,1/2,1/2,1/2,1/2 | | | 1/2,1/2,1/2,1/2,1/2,1/3 | | 1/3,1/2,1/4,1/3 | | | | P⊗=1/10 | |  |
| **10:** | **输入C** | | | **已知"AAT"** | | **CTAGGCAATC** | | | | **TAGGTA** | |  |
| CTX匹配 | **CTX3**匹配⊗ | | | **CTX2**匹配⊗ | | **CTX1**匹配⊗ | | | | **CTX0**配C | | **小计** |
| 编结果 | ⊗ | | | ⊗ | | ⊗ | | | | C | | "⊗⊗⊗C" |
| 概结果 | P⊗ = 1 | | | P⊗ = 1 | | P"T"⊗ = 1/2 | | | | P(""C)=2/10 | | 1/2× |
| CTX更新 | P("CTA"G)=1/2 | | | P("CT"A)=1/2 | | P("C"T)=1/3 | | | | P(""C)=3/11 | | 2/10 |
| CTX更新 | P("TAG"G)=1/2 | | | P("TA"G)=1/2 | | P("T"A)=1/3 | | | | P(""T)=2/11 | |  |
| CTX更新 | P("AGG"C)=1/2 | | | P("AG"G)=1/2 | | P("A"G)=1/4 | | | | P(""A)=3/11 | |  |
| CTX更新 | P("GGC"A)=1/2 | | | P("GG"C)=1/2 | | P("G"G)=1/3 | | | | P(""G)=2/11 | |  |
| CTX更新 | P("GCA"A)=1/2 | | | P("GC"A)=1/2 | | P("G"C)=1/3 | | | |  | |  |
| CTX更新 | P("CAA"T)=1/2 | | | P("CA"A)=1/3 | | P("C"A)=1/3 | | | |  | |  |
| CTX更新 | P("AAT"C)=1/2 | | | P("CA"T)=1/3 | | P("A"A)=1/4 | | | |  | |  |
| CTX更新 |  | | | P("AT"C)=1/2 | | P("A"T)=1/4 | | | |  | |  |
| CTX更新 |  | | |  | | P("T"C)=1/3 | | | |  | |  |
| ⊗概率 | 1/2,1/2,1/2,1/2,1/2,1/2 | | | 1/2,1/2,1/2,1/2,1/2,1/3,1/2 | | 1/3,1/3,1/4,1/3 | | | | P⊗=1/11 | |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **11:** | **输入T** | **已知"ATC"** | **CTAGGCAATC** | **TAGGTA** |  |
| CTX匹配 | **CTX3**匹配⊗ | **CTX2**匹配⊗ | **CTX1**匹配⊗ |  | **小计** |
| 编结果 | ⊗ | ⊗ | T |  | "⊗⊗T" |
| 概结果 | P⊗ = 1 | P⊗ = 1 | P"T"⊗ = 1/3 |  | 1/3 |
| CTX更新 | P("CTA"G)=1/2 | P("CT"A)=1/2 | P("C"T)=2/4 |  |  |
| CTX更新 | P("TAG"G)=1/2 | P("TA"G)=1/2 | P("T"A)=1/3 |  |  |
| CTX更新 | P("AGG"C)=1/2 | P("AG"G)=1/2 | P("A"G)=1/4 |  |  |
| CTX更新 | P("GGC"A)=1/2 | P("GG"C)=1/2 | P("G"G)=1/3 |  |  |
| CTX更新 | P("GCA"A)=1/2 | P("GC"A)=1/2 | P("G"C)=1/3 |  |  |
| CTX更新 | P("CAA"T)=1/2 | P("CA"A)=1/3 | P("C"A)=1/3 |  |  |
| CTX更新 | P("AAT"C)=1/2 | P("CA"T)=1/3 | P("A"A)=1/4 |  |  |
| CTX更新 | P("ATC"T)=1/2 | P("AT"C)=1/2 | P("A"T)=1/4 |  |  |
| CTX更新 |  | P("TC"T)=1/2 | P("T"C)=1/3 |  |  |
| ⊗概率 | 1/2,1/2,1/2,1/2,1/2,1/2,1/2 | 1/2,1/2,1/2,1/2,1/2,1/3,1/2,1/2 | 1/4,1/3,1/4,1/3 |  |  |
| **12:** | **输入A** | **已知"TCT"** | **CTAGGCAATC** | **TAGGTA** |  |
| CTX匹配 | **CTX3**匹配⊗ | **CTX2**匹配A |  |  | **小计** |
| 编结果 | ⊗ | A |  |  | "⊗A" |
| 概结果 | P⊗ = 1 | P⊗ = 1/2 |  |  | 1/2 |
| CTX更新 | P("CTA"G)=1/2 | P("CT"A)=2/3 |  |  |  |
| CTX更新 | P("TAG"G)=1/2 | P("TA"G)=1/2 |  |  |  |
| CTX更新 | P("AGG"C)=1/2 | P("AG"G)=1/2 |  |  |  |
| CTX更新 | P("GGC"A)=1/2 | P("GG"C)=1/2 |  |  |  |
| CTX更新 | P("GCA"A)=1/2 | P("GC"A)=1/2 |  |  |  |
| CTX更新 | P("CAA"T)=1/2 | P("CA"A)=1/3 |  |  |  |
| CTX更新 | P("AAT"C)=1/2 | P("CA"T)=1/3 |  |  |  |
| CTX更新 | P("ATC"T)=1/2 | P("AT"C)=1/2 |  |  |  |
| CTX更新 | P("TCT"A)=1/2 | P("TC"T)=1/2 |  |  |  |
| ⊗概率 | 1/2,1/2,1/2,1/2,1/2,1/2,1/2,1/2 | 1/3,1/2,1/2,1/2,1/2,1/3,1/2,1/2 |  |  |  |
| **13:** | **输入G** | **已知"CTA"** | **CTAGGCAATC** | **TAGGTA** |  |
| CTX匹配 | **CTX3**匹配G |  |  |  | **小计** |
| 编结果 | G |  |  |  | "G" |
| 概结果 | P"CTA"G=1/2 |  |  |  | 1/2 |
| CTX更新 | P("CTA"G)=2/3 |  |  |  |  |
| CTX更新 | P("TAG"G)=1/2 |  |  |  |  |
| CTX更新 | P("AGG"C)=1/2 |  |  |  |  |
| CTX更新 | P("GGC"A)=1/2 |  |  |  |  |
| CTX更新 | P("GCA"A)=1/2 |  |  |  |  |
| CTX更新 | P("CAA"T)=1/2 |  |  |  |  |
| CTX更新 | P("AAT"C)=1/2 |  |  |  |  |
| CTX更新 | P("ATC"T)=1/2 |  |  |  |  |
| CTX更新 | P("TCT"A)=1/2 |  |  |  |  |
| ⊗概率 | 1/3,1/2,1/2,1/2,1/2,1/2,1/2,1/2 |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **14:** | **输入G** | **已知"TAG"** | **CTAGGCAATC** | **TAGGTA** |  |
| CTX匹配 | **CTX3**匹配G |  |  |  | **小计** |
| 编结果 | G |  |  |  | "G" |
| 概结果 | P"CTA"G=1/2 |  |  |  | 1/2 |
| CTX更新 | P("CTA"G)=2/3 |  |  |  |  |
| CTX更新 | P("TAG"G)=2/3 |  |  |  |  |
| CTX更新 | P("AGG"C)=1/2 |  |  |  |  |
| CTX更新 | P("GGC"A)=1/2 |  |  |  |  |
| CTX更新 | P("GCA"A)=1/2 |  |  |  |  |
| CTX更新 | P("CAA"T)=1/2 |  |  |  |  |
| CTX更新 | P("AAT"C)=1/2 |  |  |  |  |
| CTX更新 | P("ATC"T)=1/2 |  |  |  |  |
| CTX更新 | P("TCT"A)=1/2 |  |  |  |  |
| ⊗概率 | 1/3,1/3,1/2,1/2,1/2,1/2,1/2,1/2 |  |  |  |  |
| **15:** | **输入T** | **已知"AGG"** | **CTAGGCAATC** | **TAGGTA** |  |
| CTX匹配 | **CTX3**匹配⊗ | **CTX2**匹配⊗ | **CTX1**匹配⊗ | **CTX0**配T | **小计** |
| 编结果 | ⊗ | ⊗ | ⊗ | T | "⊗⊗⊗T" |
| 概结果 | P"AGG"⊗=1/2 | P"GG"⊗ = 1/2 | P"G"⊗ = 1/3 | P(""C)=2/11 | 1/2× |
| CTX更新 | P("CTA"G)=2/3 | P("CT"A)=2/3 | P("C"T)=2/4 | P(""C)=3/12 | 1/2× |
| CTX更新 | P("TAG"G)=2/3 | P("TA"G)=1/2 | P("T"A)=1/3 | P(""T)=3/12 | 1/3× |
| CTX更新 | P("AGG"C)=1/3 | P("AG"G)=1/2 | P("A"G)=1/4 | P(""A)=3/11 | 2/11 |
| CTX更新 | P("GGC"A)=1/2 | P("GG"C)=1/3 | P("G"G)=1/3 | P(""G)=2/11 |  |
| CTX更新 | P("GCA"A)=1/2 | P("GC"A)=1/2 | P("G"C)=1/3 |  |  |
| CTX更新 | P("CAA"T)=1/2 | P("CA"A)=1/3 | P("C"A)=1/3 |  |  |
| CTX更新 | P("AAT"C)=1/2 | P("CA"T)=1/3 | P("A"A)=1/4 |  |  |
| CTX更新 | P("ATC"T)=1/2 | P("AT"C)=1/2 | P("A"T)=1/4 |  |  |
| CTX更新 | P("TCT"A)=1/2 | P("TC"T)=1/2 | P("T"C)=1/3 |  |  |
| CTX更新 | P("AGG"T)=1/3 | P("GG"T)=1/3 | P("G"T)=1/4 |  |  |
| ⊗概率 | 1/3,1/3,1/3,1/2,1/2,1/2,1/2,1/2 | 1/3,1/2,1/2,1/3,1/2,1/3,1/2,1/2 | 1/4,1/3,1/4,1/4 | P⊗=1/12 |  |
| **16:** | **输入A** | **已知"GGT"** | **CTAGGCAATC** | **TAGGTA** |  |
| CTX匹配 | **CTX3**匹配⊗ | **CTX2**匹配⊗ | **CTX1**匹配A |  | **小计** |
| 编结果 | ⊗ | ⊗ | A |  | "⊗⊗A" |
| 概结果 | P⊗=1/2 | P⊗ = 1 | P"T"A = 1/3 |  | 1/2× |
| CTX更新 | P("CTA"G)=2/3 | P("CT"A)=2/3 | P("C"T)=2/4 |  | 1/3 |
| CTX更新 | P("TAG"G)=2/3 | P("TA"G)=1/2 | P("T"A)=2/4 |  |  |
| CTX更新 | P("AGG"C)=1/3 | P("AG"G)=1/2 | P("A"G)=1/4 |  |  |
| CTX更新 | P("GGC"A)=1/2 | P("GG"C)=1/3 | P("G"G)=1/3 |  |  |
| CTX更新 | P("GCA"A)=1/2 | P("GC"A)=1/2 | P("G"C)=1/3 |  |  |
| CTX更新 | P("CAA"T)=1/2 | P("CA"A)=1/3 | P("C"A)=1/3 |  |  |
| CTX更新 | P("AAT"C)=1/2 | P("CA"T)=1/3 | P("A"A)=1/4 |  |  |
| CTX更新 | P("ATC"T)=1/2 | P("AT"C)=1/2 | P("A"T)=1/4 |  |  |
| CTX更新 | P("TCT"A)=1/2 | P("TC"T)=1/2 | P("T"C)=1/4 |  |  |
| CTX更新 | P("AGG"T)=1/3 | P("GG"T)=1/3 |  |  |  |
| CTX更新 | P("GGT"A)=1/2 | P("GT"A)=1/2 |  |  |  |
| ⊗概率 | 1/3,1/3,1/3,1/2,1/2,1/2,1/2,1/2,1/2 | 1/3,1/2,1/2,1/3,1/2,1/3,1/2,1/2,1/2 | 1/4,1/4,1/4,1/3 |  |  |

PPM非常适合推演DNA链的出字概率, 因为人类细胞染色体多达0.5~25亿端, 而上述是在全文未知的情况下把最优出字概率分布总结得差不多了. 这种以查表推演Table lookup是一种计算特性

## 有限状态机Finite-State Machine

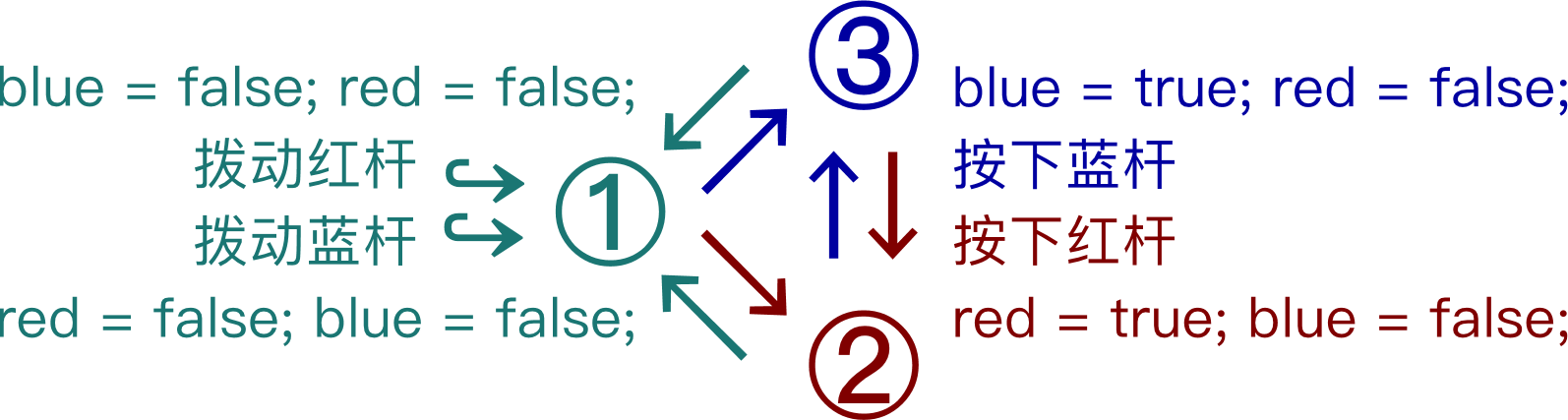
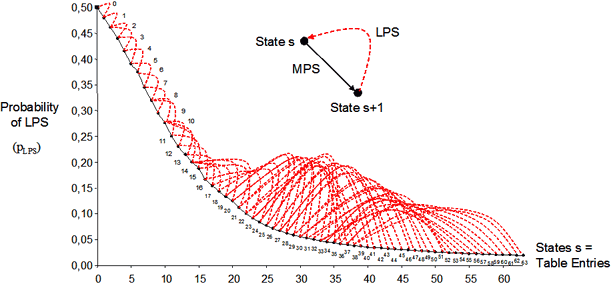
以状态描述程序/机器, 例如1~2间可过渡出∞个数代表机械运动到位的过程; 也可限为仅①↔②两态, 以抽象表示出圆珠笔的伸缩; 输入即按下笔的按钮, 输出即下一状态. 若将笔伸出(需②)而笔已收拢则写作①→⓶; 若需收拢而笔已伸出则②→⓵.

图: 双/多色按动圆珠笔下, 若红色伸出, 则红色压杆会锁定; 需按动蓝色压杆来复位/收拢红色笔芯. 因此操作有按动压杆和拨动压杆两种, 状态有皆收拢①, 红伸出②, 蓝伸出③

## 上下文自适应-有限状态机推演法

BAC下只有最大概符most-probable symbol MPS和最小概符least-probable symbol LPS两个变量, 若大概为1, 则LPS=0; 已知出0概p0, 出1概率即1-p0, 而只推演LPS上下文即得到两个符号在0~1概率区间的分布. 具体分布由人工筛选, 所以具体逻辑要看源码, 大体为输入LPS为0和为1时的状态切换, 若大于0.5则变成MPS, LPS最小为0.01875; MPS最大为0.98125

## (低精度)二进制算数编码Binary Arithmetic Coding (BAC)

通过MPS, LPS变量简化出字概率的统计. 再降精度到1/2的幂(如8则在0~1区间精确到1/256个位深过渡)来略增误差优化性能, 再加一步将像素值转二进制的编码, 设以便统计过程中交换值, 而算数编码的部分不变, 就实现了二进制的算数编码

## 几何分布/等差等比分布类Geometric distribution

条件过程不变, 各个尝试独立, 只有阴阳两结果的伯努利试验中, 统计n遍成功的概率分布所形成的概率变化曲线. 为. 其中概率(1-p)盖错, (p)盖对([Desmos例](https://www.desmos.com/calculator/zhzjpzsqwu)), 如:

1. 尝试3次得阳记作"错, 错, 对": .
2. 对概率为90%则记做. 且根据上式可见成功概率与尝试次数独立
3. 所谓"尝试3次得对"是可能脱离实际的期望值eXpectation, 记做大写X, 为

* 小概率对时, 一遍对的概率被其它次数分走, 而反过来大概率对时, 一遍对的概率会夺走其它次数的概率
* 因为尝试次数间的差距皆为1(不存在0.3次尝试)所以等差
* 因为尝试次数每次都乘进同个(1-p)比率, 所以等比
* 综上所述(计算过程含多次(1-p)相乘)二维坐标系中可见概率的指数递减下降变化, 得名几何分布

## 小数取整法/进位法以及取模

取整分round的舍入, 进位ceiling ⌈ ⌉的, 去尾floor ⌊ ⌋三方而按需取整. 取模modulus则是将余数乘以原来的除数, 如观察0%3, 1%3, 2%3, 3%3, 4%3, 5%3就会发现结果循环于0, 1, 2中的规律, 就省去狂写if(x==0), if(x==1), if(x==2), if(x==3), if(x==4)…换switch (x%n)了

## 转二进制-伪哥伦布编码

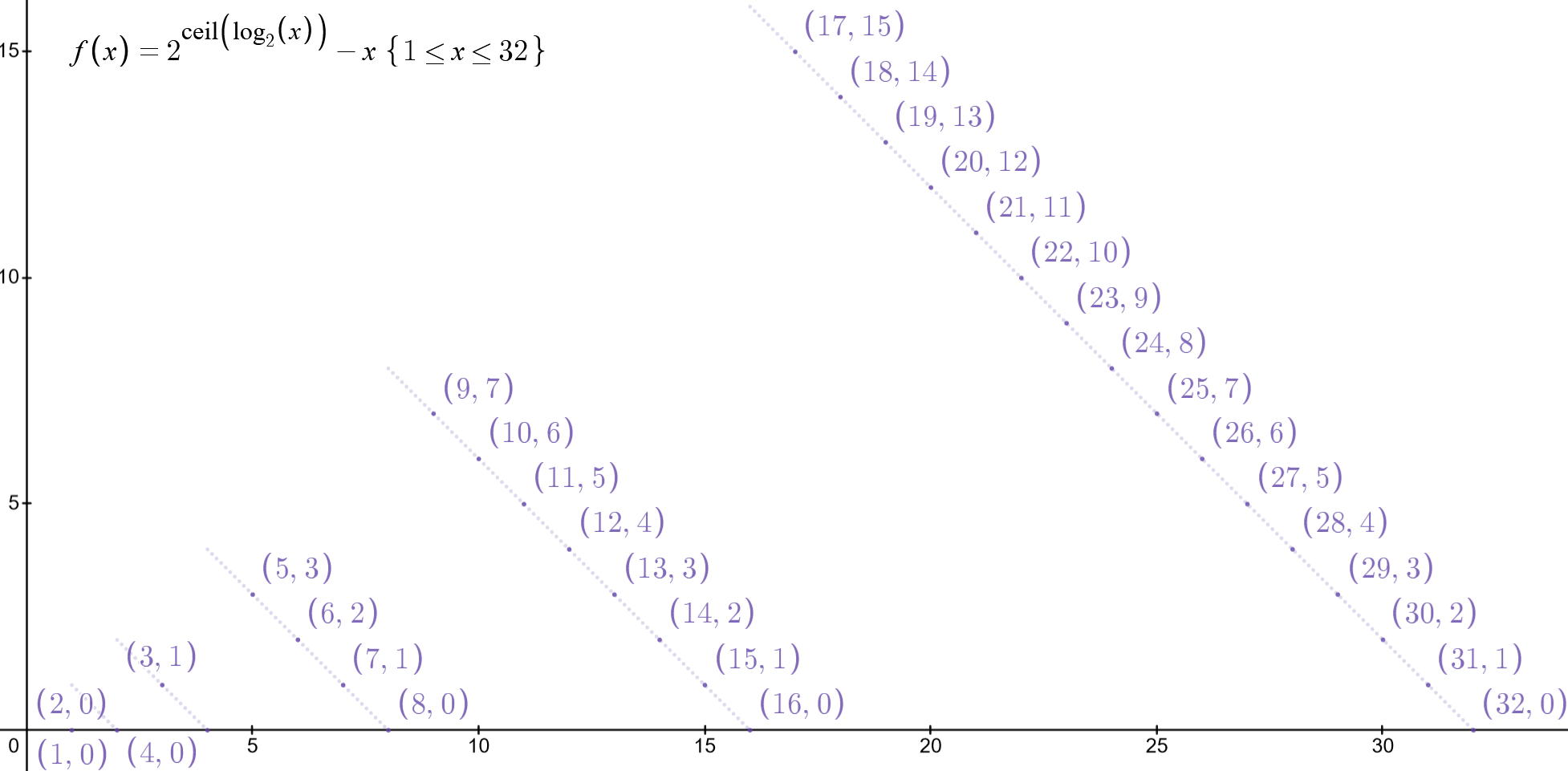
8bit下, 十进制3会转为浪费码率的00000011; 因为前后都挨着别的8bit数而不能删; 所以添加清点有效位数的标记, 共有11两位数而设标记00, 再转后缀为前缀, 就得到相对可靠的(a=00,b=11); 解码器识别出变量a, b, 为两位数的11而还原

## 转二进制-一元码Unary, 截短二进制码Truncated Binary

一元码如2, 3即110, 1110或001, 0001, 末尾分隔后续数, 差在吃带宽. 截短二进制如8bit的十进制5是二进制00000101. 所以能截短到101, 差在难以解码; 被哥伦布编码统一

## 转二进制-哥伦布(-莱斯)编码Golomb(-Rice) coding

将正整数转二进制并压缩. 以输入N, 十进制缩放值M与长度Κ, 一元码, 截短二进制实现. (-莱斯)码在此基础上限制M仅为2的幂, 对计算机更友好. 此处例M=4, 输入N=9:

1. 求反指数缩放的长度K 如, ceil约
2. 求商去尾, 转一元码 如, floor约, 换一元码
3. 求余数模, 判断区间 如, (, )
4. 余数模是否小于刻度 如, 结果否定, 因为
   1. M=1~32时, 边界如图([Desmos例](https://www.desmos.com/calculator/2gonwxfc1m)): x轴为M, y轴为
   2. 简单说, 该边界判断输入N是否位于「二进制前半」, 如2bit下, 00, 01, 10, 11的前半00, 01可省首位0
   3. 小于刻度则用K-1个bit的长度编码
   4. 大于等于则用K个bit的长度编码
   5. (上面简化为if-else结构, 这里略标准些)
5. 中截短truncate方法见上 如, 二进制为00000001, 用2bit编为
6. 将前端Q和后端R`拼合, 即 如

## 转二进制-零阶指数哥伦布编码Exponential Golomb k0

已知十进制数添一位, 量增十倍, 如1, 10, 100按×10倍增; 二进制数添一位, 量增二倍, 如001, 010, 100照×2倍增. 可见「刻度本是指数」, 二进制刻度K的间距设为指数增加的0, 1, 3, 7, 15…或0, 1, 11, 111, 1111…, 每度从底延伸至顶. Kλίμακα设为对齐小的边界:

1. 设定要编码的N并获取其二进制值+1为刻度后端 如输入N=6, 则(二进制)
2. 求二进制刻度K' 如输入N=6, 则, floor约
3. 求刻度内偏移值 如输入N=6, 则, 代表在刻度6处偏移3下
   1. 输入N=0~31时, 十进制K'=[0,1,1,2,2,2,2,3,3,3,3,3,3,3,3,4,4,4,4,4,4,4,4,4,4, 4, 4, 4, 4, 4, 4]
   2. 输入N=0~31时, 十进制O =[0,0,1,0,1,2,3,0,1,2,3,4,5,6,7,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
   3. 注: O就是单纯随K'指数增加而累计偏移量
4. K'值转换为0的数量, 设为前端1 如输入N=6, 则K'=2, 则写作00
5. R值转换二进制为前端2 前端1+后端是常见的零阶指数哥伦布码
6. 前端2+后端少见, 下表可见 转二进制后用游程编码压缩率更低, 但能校验纠错

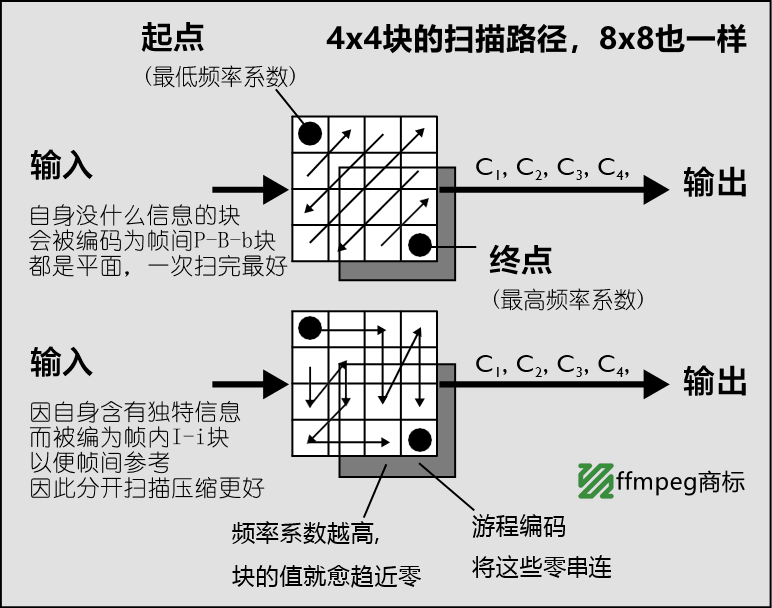
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N | K' | R | 前端1 | 后端 | 前端2 |
| 0 | 0 | 0 |  | 1 | 0 |
| 1 | 1 | 0 | 0 | 10 | 0 |
| 2 | 1 | 1 | 0 | 11 | 1 |
| 3 | 2 | 0 | 00 | 100 | 00 |
| 4 | 2 | 1 | 00 | 101 | 01 |
| 5 | 2 | 2 | 00 | 110 | 10 |
| 6 | 2 | 3 | 00 | 111 | 11 |
| 7 | 3 | 0 | 000 | 1000 | 000 |

注: 此处因篇幅, 难度和不相关而省略一阶及更高阶的哥伦布码

## 汉明距离与欧几里得距离

前者在不考虑大小的情况下测量数字信号的差距, 如100与010的汉明距离为2, 可累计各位数的异或门⊻得到, 构成误差和SAD; 后者测量两个点的差异平方和SSE(见x264教程), 如GPS的地理误差, 电压误差等

## 软判决与硬判决解码Soft/Hard Decision decoding

定义二进制0为0.33V, 1为0.67V; 再定义编解码格式的有效字为[001, 010, 100, 101, 111], 传输001, 电压为0.33, 0.33, 0.67V, 解码器得到偏移的0.35V, 0.55V和0.75V时的两种解码策略. 硬判决如设置大于0.4V则1; 小于0.5V则0, 得0, 1, 1. 比对有效字, 汉明距离为[1, 1, 3, 2, 1], 共3个误差最小的可能; 软判决则比对电压间的差异平方和, 如同样的偏移值, 对比[001]算, 对比[010, 100, …]分别得欧几里得距离为[0.1912, 0.3272, 0.1572, …]等误差, 而0.0468的误差最小, 就在条件不变的情况下更准确的解码了信息

## 游程/游标编码run-length/run-level coding

前者将二维DCT频域系子以适宜压缩的走线从起点zig-zag串联到终点为一维文本, 以便文本压缩/熵编码, 因量化左上低右下高, 故使DCT系子越往左上出字概率越低, 而不是出字概率最大的强度/level 0; 后者作连续的0和非0值为一个bin分区, 如84 | 2 | 0,0,1 | 0,0,0

## trellis

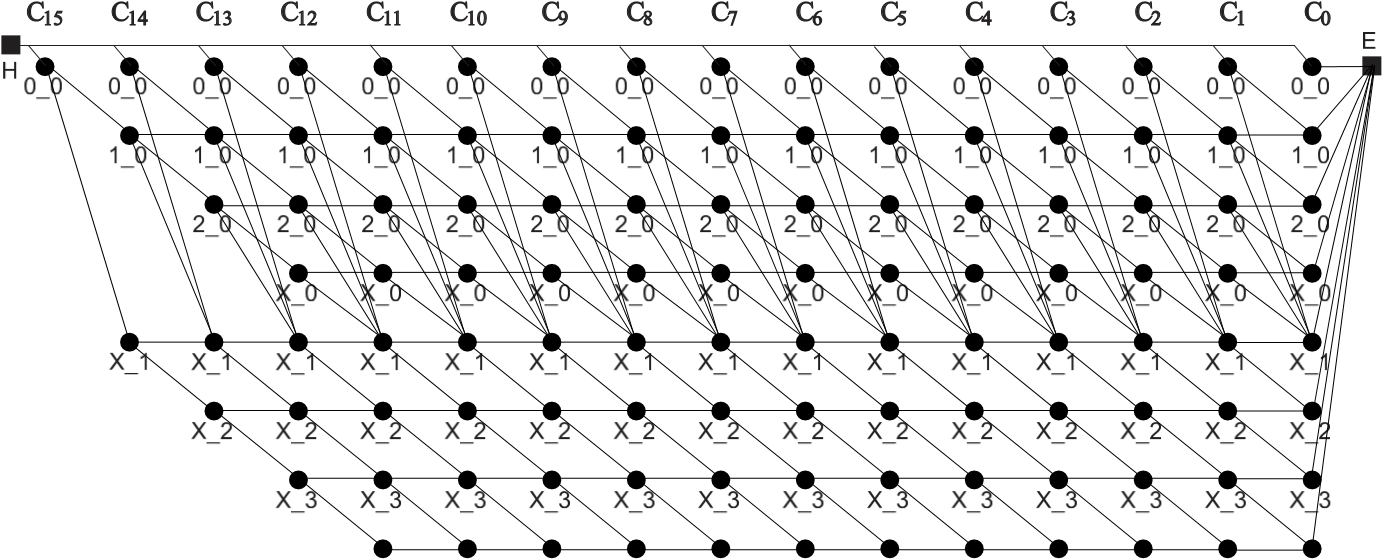
原指北欧建筑的白墙木条棋盘格子装修, 在动态规划编程dynamic programming中代表过程的可视化——各状态为节点, 且都遵循一种从左推演到右的规则而相互连成的网格, 视频编码中包括:

* Psy-trellis的心理学优化, 于x264的--psy-rd
* Viterbi-trellis, 解决最短路径, 隐马尔科夫问题的导航寻路算法, 用于推演B帧位置
* Trellis-coded quantization (TCQ, h.263+, trellis卷积内核的有限状态机)中涉及的量化优化, 停留在理论
* Soft-Decision Quantization trellis, 软判决率失真优化CABAC的再量化, 于x264的--trellis参数

## 软判决率失真优化CABAC的再量化-SDQ trellis

算数编码行游标编码之倒序——先编码最大概率的0, 后编码低概的绝对值强度absolute level (因为含负强度/DCT系子波形的反相). 得到CABAC各路线整合的trellis.

图: CABAC编码4×4块的所有上下文可能性以及变更. 由于此处文本编码是动态程序(一次编一字), 所以路线一次只能改一条且会保持出字概率, 得斜线



4x4块的16个倒序zig-zag系子中, 预测和总结系子强度(等于1\_大于1, 或NumEq1\_NumLg1)的数量归纳为8种CABAC上下文长度的节点(0\_0~any\_any), 如2\_0对应NumLg1×2, NumEq1×1, 即2\_0线. 若NumLg1>0, 则从当前系子节点切到下个系子的×\_1线并不再回到NumEq1; 若NumLg1≥4则构成any\_any或x\_x, 若(NumEq1≥4 && NumLg1==0)时则构成any\_0或x\_0, 为两种上下文长4(最大)的CABAC路线, 一种只编码0和1, 另一种编码0, 1, <其它>. 各线路上的率即强度level, 失真就是其系子节点差异平方的和/SSE, 而率失真最优的线路就是失真SSE + λ⋅码率R最小的线路

叫软判决量化是因为此处SSE测了欧几里得距离, 相对的硬判决量化为qp值所对应的DCT除法表

# SEI补充与优化消息

supplemental enhance info记录每帧的补充信息. 主要有正确打开新gop用的缓冲sei, 解码卡时间的pic timing sei, 让显示主控切边的sei, cc字幕sei, HDR-sei等等. 缓冲sei记录对应sps的号; 待解码图像缓冲coded picture buffer/cpb的延迟安全区等信息; 时戳sei记录哪些帧上/下场优先的变化; 连帧/三连帧的位置等信息

--hrd

<开关, 默认关, 开vbv>将假设对照解码参数hypothetical ref. decoder param. 估计不丢包无延迟的瞬间码率, 写在每段sps及sei里, 对专门配置了网络串流, NAS播放自动缓冲的播放器有好处? 但应该没啥用

--hash

<整数0~3分别代表默认的关,checksum, crc和md5>sei里加效验码, 播放时可用以对图像重建纠错来减少失真, md5播放所需算力较高, checksum最快但有忽略概率, crc平衡

--single-sei

<开关>只写一个装全部sei信息的大NALU而非每gop都写, 提高很小一点压缩率

--film-grain

<文件名>将如[libfgm](https://bitbucket.org/multicoreware/libfgm)提取的纹理细节模型film grain model写进SEI, 将编码压缩掉的细节另存档, 兼容解码器播放时恢复的功能

--idr-recovery-sei

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

--frame-dup

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

--dup-threshold

<整数1~99, 默认70>相似度判定值, 默认达70%重复就判为相似

# 线程节点控制

参考自[pugetsystems](https://www.pugetsystems.com/labs/articles/amd-ryzen-5800x3d-vs-5800x-for-content-creation-2331/): Cinebench, 虚幻5上5800X比X3D快近10%反映了持续计算下5800X频率高的优势; Lightrooms等单图处理上5800X3D跑过了5800X, Photoshop 2022上5900X, X3D, X打平; 剪辑视频并非持续高负载, 因此Ae，Pr，DaVinci上X3D和X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更需要算力; 若程序有大量内存读写间歇, X3D就能用读写赢回速度, 但优化烂又是另一回事了…

--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

不要设置大于实际线程数的pools/threads. 会导致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

### 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. 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:

* ctu比宏块大, 相似性降低了
* 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)来不及找而被跳过
* 参考帧的波前编码wavefront parallel process (压制/播放的多线程改进版)因一行参考ctu的存在而卡死, 重启波前编码等没了多余算力

--pme

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

--frame-threads

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

--lookahead-threads

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

# 色彩空间转换, VUI/HDR信息, 黑边跳过

纯元数据, 写错或忘写也可以改. HDR电视只读取maxcll和maxfall, 所以master-display可略. 光强/光压单位Candela等于尼特, 即1cd=1nit. 因bt601, 709, HDR-PQ, HLG标准重视的亮度范围, 曲线所异(偏亮或偏暗), 故需编码, 心理学优化, 模式决策的重适配; 否则即错适配源的诞生

### HDR

与"不手动干预, 噪声小的情况下可发出极大, 极小声的音响"所同, 即「重硬件轻软件」. 色彩格式的具体转换变量与算法标准见[T-Rec建议](https://www.itu.int/rec/dologin.asp?lang=s&id=T-REC-H.265-201504-S!!MSW-E)的Table E.3所在处, 2015版为p347~353的5~6页

--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>播放用三原色(以及白点)指标, 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/系数指标: 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, 路径+文件名, 有空格则两边带双引号>

--input-csp

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

--dither

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

--allow-non-conformance

<开关>不写入profile和level, 绕过h.265标准的规定, 只要不是按照h.265规定写的命令行参数值就必须使用这个参数ᕙ(⇀‸↼‶)ᕗ

--force-flush

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

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

--field

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

--input-res

<宽x高>在使用x265时必须指定源视频的分辨率, 例如1920x1080

--fps

<整数/浮点/分数>在使用x265时必须指定源视频的帧率, 小数帧填小数, 勿四舍五入

--chunk-start

,

--chunk-end

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

--temporal-layers

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

## 编解码图像序列:

见x264教程完整版

下载 附录与操作

|  |  |
| --- | --- |
| [**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/)开源免费强大便携的现代软件, [安装配置教程见网页](https://nazorip.site/archives/1052/), 无色彩错误, 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/达芬奇插件, 用ffmpeg内置编码器, 不用导无损再压/破解. 只要解两个压缩包, 放Plug-Ins\Common即可 | |

**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

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 | 1:¼ |  | SDR | YCbCr |
| 5 | 仅BL (DV-MEL) | √ |  | ICtCp |
| 7 | 4K=1:¼; 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批量压制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(-\_^)

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

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| --preset | **superfast** | **veryfast** | **faster** | **fast** | **medium** | **slow** | **slower** | **Very slow** | **placebo** |
| **ctu** | **32** | **64** | | | | | | | |
| **最小cu** | **8** | | | | | | | | |
| **连续B帧数bframes** | **3** | **4** | | | | | **8** | | |
| **B帧筛选b-adapt** | **0** | | | | **2** | | | | |
| **cu树向后rc-lookahead** | **10** | **15** | | | **20** | **25** | **40** | | **60** |
| **lookahead-slices** | **8** | | | | | **4** | **1** | | |
| **参考帧ref** | **1** | **2** | | **3** | | **4** | **5** | | |
| **参考帧限制limit-refs** | **0** | **3** | | | | | **1** | **0** | |
| **动态搜索算法me** | **hex** | | | | | **star** | | | |
| **动搜搜索范围merange** | **57** | | | | | | | | **92** |
| **子像素搜索/MC subme** | **1** | | **2** | | | **3** | **4** | | **5** |
| **长矩形分块rect** | **0** | | | | | **1** | | | |
| **非矩分块amp** | **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** |
| **帧内夹角编码优化** | **1** | | | | **0** | | | | |
| **B带帧内搜索b-intra** | **0** | | | | | | **1** | | |
| **取样迁就偏移sao** | **关** | **开** | | | | | | | |
| **P帧权重weightp** | **0** | **1** | | | | | | | |
| **B帧权重weightb** | **0** | | | | | | **1** | | |
| **自适应量化aq-mode** | **0** | **2** | | | | | | | |
| **cutree** | **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tu-intra-depth** | **1** | | | | | | **3** | | **4** |
| **tu分裂上限tu-inter-depth** | **0** | | | | | | **4** | **0** | |

|  |  |  |  |
| --- | --- | --- | --- |
| **tune zerolatency去延迟** | | **tune animation动画片** | |
| **连续B帧数 bframes** | **0** | **优化模式决策psy-rd** | **0.4** |
| **B帧筛选 b-adapt** | **关** | **自适应量化强度aq-strength** | **0.4** |
| **cu树 cutree** | **关** | **去块deblock** | **1:1** |
| **转场 scenecut** | **关** | **cutree** | **关** |
| **多线程压制帧数 frame-threads** | **1** | **连续B帧数bframes** | **<preset>+2** |
| **tune grain最高画质** | | **tune fastdecode解码加速** | |
| **自适应量化aq** | **0** | **B帧权重weightb** | **关** |
| **cutree** | **关** | **P帧权重weightp** | **关** |
| **I-P帧压缩比ipratio** | **1.1** | **去块deblock** | **关** |
| **P-B帧压缩比pbratio** | **1** | **取样迁就偏移sao** | **关** |
| **QP赋值精度qp-step** | **1** | **tune psnr峰值信噪比** | |
| **取样迁就偏移sao** | **关** |
| **优化模式决策psy-rd** | **4** | **自适应量化aq** | **关** |
| **优化量化psy-rdoq** | **10** | **率失真优化rd** | **关** |
| **cu再分裂跳过rskip** | **0** | **cutree** | **关** |