

观看本教程意味着你至少已经阅读了 x264 教程完整版，对视频压缩相关的软硬件和技术有了初步了解。本教程的目的是把参数直接贴到软件里用，因此建议搭配 x265 教程完整版做适当调整

## 教程地图，软件下载

见 [iavoe.github.io](https://iavoe.github.io)。软件下载：[x264 x265 AV1 视频压制相关工具下载合集](#) 或 [GitHub 副本](#)

### 命令行用法

于链接下载 ffmpeg, ffprobe/MediaInfo, x265 并记住路径。如果使用 ffmpeg 内置的 libx265 动态链接库，则可以不下载 x265，但要求是须确保 ffmpeg 的版本为最新

### 目标色深

ffmpeg 有能够发送视频帧元数据的 yuv-for-mpeg pipe (管道)，和只发送视频帧的 raw pipe，而管道下游的 x265.exe 根据版本和 mod 不同，不一定能够识别 yuv-for-mpeg 的元数据；同时，x265 的位深设定是仅 CLI (CLI-ONLY) 参数，例如在 ffmpeg 的 libx265 中，位深由 ffmpeg 自身指定。因此，本教程中 ffmpeg pipe 的参数会要求 -D 选项指定视频色深，而 ffmpeg libx265 则没有。

### x265 HDR 设置参数:

**x265**      `--master-display` <手动告知播放器拿什么色彩空间解码>

**DCI-P3:** `G(13250,34500)B(7500,3000)R(34000,16000)WP(15635,16450)L(maxCLL × 10000,1)`

**bt709:**    `G(15000,30000)B(7500,3000)R(32000,16500)WP(15635,16450)L(maxCLL × 10000,1)`

**HDR 标识** **bt2020:** `G(8500,39850)B(6550,2300)R(35400,14600)WP(15635,16450)L(maxCLL × 10000,1)`

- 找到 HDR 元数据中的色彩范围，确认用以下哪个色彩空间后填上参数
- L 的值没有标准，每个 HDR 视频元数据里可能都不一样

**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` <maxCLL,maxFALL>最大,平均光强度, MediaInfo 查不出来就不用填

`--colormatrix` <照源, 例: gbr bt709 fcc bt470bg smpte170m YCgCo bt2020nc bt2020c smpte2084 ictp>

`--transfer` <照源, 例: gbr bt709 fcc bt470bg smpte170m YCgCo bt2020nc bt2020c smpte2084 ictp>

**杜比视界 dolby vision/DV** 有 DV-MEL (BL+RPU)和 DV-FEL (BL+EL+RPU)两种带 RPU 的格式，x265

支持共 3 种样式/profile 的 DV-MEL

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

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

--dolby-vision-rpu<路径>导入 rpu 二进制文件(.bin)用

x265 管道输入参数变更

x265 v4.0 版中引入了 Multiview Encoding (多视角输入编码)，因此 ffmpeg pipe 的格式从自 x264 以来的"- "变更为"--input - "参数

选择规格 Profile，级别 Level

根据视频位深选择规格 Profile，分辨率和帧率选择级别 Level，最后细分到档次 Tier (Main/High)

级别 Level	最大亮度像素流量 (Luma Samples)	最大亮度平面面积 (Luma Size)	8-10bit 最大码率	12bit 最大码率	4:4:4 12bit 最大码率	最高分辨率@帧率
1	552,960	36,864	Main: 128 Kbps High: -	Main: 192 Kbps High: -	Main: 384 Kbps High: -	176 × 144@15 fps
2	3,686,400	122,880	Main: 1500 Kbps High: -	Main: 2250 Kbps High: -	Main: 4500 Kbps High: -	352 × 288@30 fps
2.1	7,372,800	245,760	Main: 3000 Kbps High: -	Main: 4500 Kbps High: -	Main: 9000 Kbps High: -	640 × 360@30 fps
3	16,588,800	552,960	Main: 6000 Kbps High: -	Main: 9000 Kbps High: -	Main: 18 Mbps High: -	960 × 540@30 fps
3.1	33,177,600	983,040	Main: 10 Mbps High: -	Main: 15 Mbps High: -	Main: 30 Mbps High: -	1280 × 720@33.7 fps
4	66,846,720	2,228,224	Main: 12 Mbps High: 30 Mbps	Main: 18 Mbps High: 45 Mbps	Main: 36 Mbps High: 90 Mbps	1280 × 720@68 1920 × 1080@32 fps
4.1	133,693,440	2,228,224	Main: 20 Mbps High: 50 Mbps	Main: 30 Mbps High: 75 Mbps	Main: 60 Mbps High: 150 Mbps	1920 × 1080@64 fps 2048 × 1080@60 fps

5	267,386,880	8,912,896	Main: 25 Mbps High: 100 Mbps	Main: 37.5 Mbps High: 150 Mbps	Main: 75 Mbps High: 300 Mbps	3840 × 2160@32 fps 4096 × 2160@30 fps
5.1	534,773,760	8,912,896	Main: 40 Mbps High: 160 Mbps	Main: 60 Mbps High: 240 Mbps	Main: 120 Mbps High: 480 Mbps	3840 × 2160@64 fps 4096 × 2160@60 fps
5.2	1,069,547,520	8,912,896	Main: 60 Mbps High: 240 Mbps	Main: 90 Mbps High: 360 Mbps	Main: 180 Mbps High: 720 Mbps	3840 × 2160@128 fps 4096 × 2160@120 fps
6	1,069,547,520	35,651,584	Main: 60 Mbps High: 240 Mbps	Main: 90 Mbps High: 360 Mbps	Main: 180 Mbps High: 720 Mbps	7680 × 4320@32 fps 8192 × 4320@30 fps
6.1	2,139,095,040	35,651,584	Main: 120 Mbps High: 480 Mbps	Main: 180 Mbps High: 720 Mbps	Main: 360 Mbps High: 1440 Mbps	7680 × 4320@64 fps 8192 × 4320@60 fps
6.2	4,278,190,080	35,651,584	Main: 240 Mbps High: 800 Mbps	Main: 360 Mbps High: 1200 Mbps	Main: 720 Mbps High: 2400 Mbps	7680 × 4320@128 fps 8192 × 4320@120 fps

为了方便使用，本教程设定为一直打开 `--high-tier` 选项，详见 [x265 教程完整版](#)

## 有兼容性问题的参数

### ffmpeg

`-hwaccel auto`：自动选择硬件解码，由于部分硬件厂商的解码实现较差，所以可能会花屏

### x265

`--allow-non-conformance`：允许不合规参数以提高压缩率和画质，但可能会遇到播放、剪辑兼容问题

`--analyze-src-pics`：功能误解，实际上使用只提高了多线程利用率，反而会降低画质

## 非必要参数

### ffmpeg

`-hide_banner`：解决命令行窗口被版权协议等信息填满的问题

`-pix_fmt`，`-strict`：见上方：`-pix_fmt` 与 `-strict` 参数

### x265

`--hash`：每帧校验，纠错的效果和没有纠错差不多

`--radl`：支持 I 帧前放置 RADL 帧，会改动 GOP 结构，虽然播放没问题，但分段拼合时兼容性差

`--mcstf`: 仅支持单线程动态搜索，有残影失真

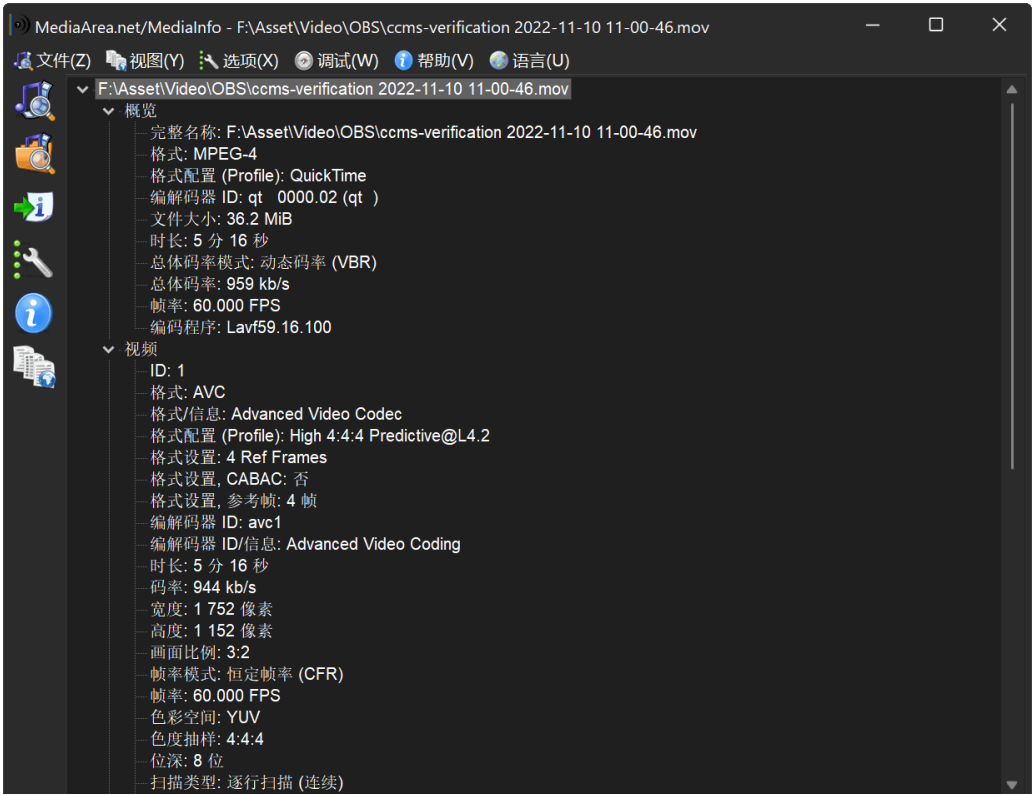
`--rd 5`: 大多情况下 3 就够了，使用该参数会产生高于 Prime95 的计算压力，而很多 CPU 超频并不考虑这种“游戏用不到”的场景，因此可能会死机，或因电压过高导致快速老化。

`--qp-adaptation-range`: 有人在 x265 v4.1 遇到了编码后视频帧播放一小段后冻结的问题，但这可能是硬解错误

# 格式识别

## 格式识别——视音频格式

下载并打开 [MediaInfo](#) 并将视频文件拖放到图形界面中，菜单栏的视图/View 中可以选择树状图（需要精确小数点可以选 JSON），可以选择菜单栏（Language）可选简体中文，即可得到视频信息。



## 操作——识别与处理交错/分行扫描

在 MediaInfo 可以看出视频是否为分行扫描，包括是否使用了 Telecine 等处理。SVT-AV1 并不支持分行扫描。将分行以高画质重新渲染为逐行可以参考 [iavoe.github.io 的这篇教程](#)。

## 格式识别——可变帧率

帧率模式显示 VFR 或 avg\_frame\_rate 异于 r\_frame\_rate。需要确保视频在剪辑前被渲染并重编码为恒定帧率 CFR，以保证剪辑软件/工具链上全部视频滤镜和的兼容性，以及避免剪辑工程音画不同步的问题。ffmpeg 可以通过 `-vsync cfr` 指定渲染换为恒定帧率 Constant Frame Rate。

## 格式识别——音频兼容性

如果要更换封装文件，则需要确认其中的音频流是否兼容到目标格式，如果不兼容则需要转码。格式兼容

列表可见于维基百科：[Comparison of video container formats](#) - Video coding formats

support。兼容性不错的 QAAC 音频编码可以参考 [这篇教程](#) 或 [Github](#)。

## 格式识别——压制用时

时长秒数 = 总帧数 ÷ 压缩速度 fps。通过系统查看封装文件属性，或 MediaInfo、ffprobe 得到视频时

长，即可在视频编码器不预估完成时间（如某些情况下未提供总帧数信息）的情况下手动计算。

## ffmpeg 参数：-pix\_fmt 与 -strict

ffmpeg 能够像 MediaInfo 一样自动检测元数据并设定 `-pix_fmt` 参数，但有时源视频的元数据中会缺

少这些信息（MediaInfo 同样看不到），便要手动设定。需要确认时可以使用 ffprobe 查找，有：

```
yuv420p, yuv422p, yuv444p, yuv420p10le, yuv420p12le, yuv422p10le, yuv422p12le, yuv444p10le,
yuv444p12le, yuv444p10le, yuv444p12le, gray, gray10le, gray12le, nv12, nv16
```

在使用管道/pipe 参数时，超过 8bit 的 YUV for MPEG 流并不合规，因此需要额外提供 `-strict` 参数

解除合规性限制，而在使用 ffmpeg 内置库时则不会用到管道，故同时无需指定 `-pix_fmt` 与 `-strict`。

```
[yuv4mpegpipe @ 0000018dde853540] 'yuv420p10le' is not an official yuv4mpegpipe pixel format. Use '-strict -1' to encode to
this pixel format.
[out#0/yuv4mpegpipe @ 0000018dde8d9e40] Could not write header (incorrect codec parameters ?): Invalid argument
[vf#0:0 @ 0000018dde85e640] Error sending frames to consumers: Invalid argument
[vf#0:0 @ 0000018dde85e640] Task finished with error code: -22 (Invalid argument)
[vf#0:0 @ 0000018dde85e640] Terminating thread with return code -22 (Invalid argument)
[out#0/yuv4mpegpipe @ 0000018dde8d9e40] Nothing was written into output file, because at least one of its streams received
no packets.
```

## 客观画质指标跑分

进行画质跑分的原因有二。一，好的显示器非常昂贵，与购置 CPU 等设备的预算冲突，而差的显示器会隐藏失真；二，实践是检验真理的唯一标准。如果源需要经过滤镜处理，那么操作上的确会多出“导出无损渲染结果”的一步，才能测试。关于更详细的说明见 [AV1 教程](#)。

- 视觉无损: XPSNR  $\geq$  45dB, VMAF  $\geq$  95
- 优秀: XPSNR  $\geq$  42dB, VMAF  $\geq$  90
- 良好: XPSNR  $\geq$  38dB, VMAF  $\geq$  80
- 达标: XPSNR  $\geq$  32dB, VMAF  $\geq$  70

### 客观画质指标跑分——XPSNR

快速低占用，ffmpeg 内置，倾向于计算源与压缩结果之间的差异，注重暂停画质。推荐每完成一次编码后就运行，以进行快速自查，单位 dB。

```
:: 普通版本
ffmpeg -i ".\原画源.mkv" -i ".\压缩源.ivf" -lavfi xpsnr="stats_file=-" -f null -

:: 时间基对齐版本
ffmpeg -i ".\原画源.mkv" -i ".\压缩源.ivf" -lavfi "[0:v]setpts=N*(时间基)[src]; [1:v]setpts=N*(时间基)[enc]; [src][enc]xpsnr=stats_file=-" -f null -
```

### 客观画质指标跑分——VMAF

快速，ffmpeg 内置，倾向于检查视觉模型观感体验，而非压缩前后差异。支持同时使用 VMAF 4K（远距，客厅/影院，高频细节敏感）和 VMAF（显示器，对二次编码、重采样、染色等失真敏感）一并计算。

```
:: 普通版本 (VMAF4k + VMAF1080p)
ffmpeg -i ".\原画源.mkv" -i ".\压缩源.ivf" -lavfi
libvmaf="model=version=vmaf_4k_v0.6.1\\:name=vmaf_4k|version=vmaf_v0.6.1\\:name=vmaf_1080p" -f null -

:: 时间基对齐版本，应对计算错误 (VMAF4k + VMAF1080p)
ffmpeg -i ".\原画源.mkv" -i ".\压缩源.ivf" -lavfi "[0:v]setpts=N*(时间基)[src]; [1:v]setpts=N*(时间基)[enc]; [src][enc]libvmaf=model=version=vmaf_4k_v0.6.1\\:name=vmaf_4k|version=vmaf_v0.6.1\\:name=vmaf_1080p" -f null -
```

### 客观画质指标跑分——时间基 time base 对齐 (DTS 单调增加)

未对齐与对齐得出的画质分数差距巨大（未对齐远低于正常，约 -659%）：

```
XPSNR average, 14315 frames y: 33.8192 u: 41.5568 v: 42.3078 (minimum: 33.8192)
XPSNR average, 14315 frames y: 41.9978 u: 44.6105 v: 45.2615 (minimum: 41.9978)
```

未对齐时间基时，ffmpeg 会在运行开始直接提醒未对齐：

```
[Parsed_xpsnr_0 @ 000001e437db6e80] not matching timebases found between first input: 1/90000 and
second input 1001/24000, results may be incorrect!
```

此时需要记下时间基、按 `Ctrl+C` 停止跑分，用下列脚本计算同步值并修改即可重跑。脚本位于本教程同一下载目录的 GCDLCMCalculator.zip 压缩包中的脚本，或 [GitHub/iAvoe](#) 中找到。

调用方法（以下的脚本变体一致）：

```
# 基本用法：计算两个整数的最小公倍数
lcm <数字1 (num1) > <数字2 (num2) >

# 基本用法：计算两个分数的最小公倍数
fracgcd <分子1 (num1) > <分母1 (denom1) > <分子2 (num2) > <分母2 (denom2) >

# PowerShell 调用示例——24 和 1000 的最小公倍数；1/90000 和 1001/24000 的最小公倍数
gcdlcm.ps1 -Operation lcm 24 1000
gcdlcm.ps1 -Operation fracgcd 1 90000 1001 24000

# Bash 调用示例
gcdlcm.sh lcm 24 1000
gcdlcm.sh fracgcd 1 90000 1001 24000

# Python 调用示例
python3.exe gcdlcm.py lcm 24 1000
python3.exe gcdlcm.py fracgcd 1 90000 1001 24000

# Java (.jar) 调用示例
java.exe -jar gcdLcm.jar lcm 24 1000
java.exe -jar gcdLcm.jar fracgcd 1 90000 1001 24000
```

## 结果示例（中量化强度）

**结果 1：**尽管压缩结果与源的差距（失真损失）极大，但由于视频内容变化剧烈，导致播放时看不出毛病，但仍然应该降低量化强度以提高暂停画质

```
XPSNR average, 6314 frames y: 20.9812 u: 38.0531 v: 35.0405 (minimum: 20.9812)
VMAF 4k: 98.125428, VMAF 1080p: 96.795521
```

**结果 2：**无明显问题，或可略微降低量化，将 VMAF 4k 分数提高到 90，XPSNR Y 提至 42

```
XPSNR average, 36996 frames y: 39.3288 u: 42.4070 v: 42.9840 (minimum: 39.3288)
VMAF 4k: 88.251216, VMAF 1080p: 82.140527
```

**结果 3：**XPSNR 的分数可以，但两个 VMAF 模型之间得分的差距较大，这是因为画面中有“二次编码”，



“上采样”，“块失真”，“色带”等痕迹（此处是含一些低分辨率素材渲染的 3D 动画）。而 XPSNR 在 U、V 得分高的原因单纯是因为源视频的色彩较简单，容易压缩。可以尝试降低量化强度（或同时增加色度平面的量化强度以平衡文件体积），让 VMAF 4k 达到 90 分

```
XPSNR average, 15691 frames y: 32.9889 u: 45.2554 v: 44.5165 (minimum: 32.9889)
VMAF 4K: 83.837285, VMAF 1080p: 77.317822
```

**结果 4:** XPSNR 的分数可以，但两个 VMAF 都给出偏低的分，可以考虑降低量化强度（或同时增加色度平面的量化强度以平衡文件体积），让 VMAF 4k 达到 80 分

```
XPSNR average, 301 frames y: 32.2371 u: 40.8458 v: 42.8932 (minimum: 32.2371)
VMAF 4k: 77.517816, VMAF 1080p: 68.330515
```

**结果 5:** 无明显问题，可以不改

```
XPSNR average, 1199 frames y: 39.8871 u: 42.1991 v: 42.3623 (minimum: 39.8871)
VMAF 4k: 91.383935, VMAF 1080p: 85.874477
```

# 通用·简单

去掉所有自定义项目填 Profile, Level, 方便急用且速度仅比 preset slow 慢几 fps

**兼容性** `--profile<8/10/12bit: main/main10/main12, YUV4:2:2: main422-10/main422-12, YUV4:4:4: main444-8/main444-10/main444-12> --high-tier`

**预设-转场** `--preset slow`

**动态搜索** `--me umh --subme 5 --merange 48 --weightb`

**自适应量化** `--aq-mode 4`

**帧控** `--bframes 5 --ref 3`

**多处理器分配** `--pools ,,,` (举例-,+表示该电脑有两个 CPU 节点, 用第二个. 同时占用多个会造成严重的内存延迟)

**其它** **去黑边加速:** `--display-window <整数"<,↑,>,<,↓"&像素>`, **≥22 核 cpu 优化:** `--pme`, **分场视频:** `--field`, **抖动高质量降色深:** `--dither`, **开始; 结束帧:** `--seek; --frames`, **crf/abr 缓解噪点影响:** `--rc-grain`

**目标色彩空间** `[ffmpeg] -pix_fmt yuv420p / yuv422p / yuv444p / yuv420p10 / yuv422p10 / yuv444p10...`

## α——(ffmpeg pipe) x265 CLI 命令

- `ffmpeg.exe -y -i ".\导入.mp4" -an -f yuv4mpegpipe -pix_fmt ○ -strict -1 - | x265.exe --profile ○ --high-tier --preset slow --me umh --subme 5 --merange 48 --weightb --aq-mode 4 --bframes 5 --ref 3 --y4m --input - --output ".\输出.hevc"`

## β——ffmpeg libx265 CLI, 拷贝音频并封装为 mp4

- `ffmpeg.exe -y -i ".\导入.mp4" -c:v libx265 -profile:v ○ -x265-params "high-tier=1:preset=slow:me=umh:subme=5:merange=48:weightb=1:bframes=5:ref=3" -fps_mode passthrough -c:a copy ".\输出.mp4"`

# 通用·标准

含大量自定义项目，可以配出高压或高速参数

## 兼容性

--profile<8/10/12bit: [main/main10/main12](#), YUV4:2:2: [main422-10/main422-12](#), YUV4:4:4: [main444-8/main444-10/main444-12](#)> --high-tier

## 分块-变换

--tu-intra-depth 3 --tu-inter-depth 3 --limit-tu 1 --rdpenalty 1 --rect

## 动搜-补偿

--me umh --subme <24fps: [3](#), 48fps: [4](#), 60fps: [5](#), 100fps: [6](#)> --merange <1920:1080: [48](#), 2560:1440: [52](#), 3840:2160: [56](#)> --weightb

## 溯块-帧控

--ref 3 --max-merge <快: [2](#), 中: [3](#), 慢: [5](#)> --early-skip --no-open-gop --min-keyint 5 --fades --bframes 8 --b-adapt 2 <锐利线条: --pbratio [1.2](#)>

## 帧内编码

<快: --fast-intra / 中: 不填 / 慢: --b-intra / 极慢且有兼容性问题: --constrained-intra>

## 量化

--crf <超清: [18~20](#), 高清: [19~22](#)> --crqpoffs -3 --cbqpoffs -1

## 率失优量化

--rdoq-level <快: [1](#), 很慢: [2](#)>

## 自适应量化

<动漫源改--hevc-aq, 关 aq-mode> --aq-mode 4 --aq-strength <多面: [0.8](#), 多线: [1](#)>

## 模式决策

--rd 3 --limit-modes --limit-refs 1 --rskip <快: [2](#), 中: [1](#), 慢: [0](#)> --rc-lookahead <[3](#)×帧率, 大于 bframes> --rect <很慢: --amp>

## 率失真优化

--psy-rd <录像: [1.6](#), 动画: [0.6](#), ctu64: +0.6, ctu16: -0.6> --splitrd-skip

## 去块-取迁

--limit-sao --sao-non-deblock --deblock 0:-1

## 目标色深

-D 8/10/12 <单程序兼容多色深时须手动指定, 默认 8bit, 低勿转高, 高转低开 --dither>

## 多处理器分配

--pools ,,, (举例-,+表示该电脑有两个 CPU 节点, 用第二个. 同时占用多个会造成严重的内存延迟)

## 其它

去黑边加速: --display-window <整数"←,↑,→,↓"像素>, ≥22 核 cpu 优化: --pme, 分场视频: --field, 抖动高质量降色深: --dither, 开始; 结束帧: --seek; --frames, crf/abr 缓解噪点影响: --rc-grain

## α——(ffmpeg pipe) x265 CLI 命令-共 11+2 个自定义

- ffmpeg.exe -y -i ".\导入.mp4" -an -f yuv4mpegpipe -strict -1 - | x265.exe --profile ☐ --high-tier  
--ctu ☐ --min-cu-size 16 --tu-intra-depth 3 --tu-inter-depth 3 --limit-tu 1 --rdpenalty 1 --me  
umh --subme ☐ --merange ☐ --weightb --ref 3 --max-merge ☐ --early-skip --no-open-gop --  
min-keyint 5 --fades --bframes 8 --b-adapt 2 --pbratio 1.2 --fast-intra --b-intra --crf ☐ --crqpoffs  
-3 --cbqpoffs -1 --rdoq-level ☐ --aq-mode 4 --aq-strength ☐ --rd 3 --limit-modes --limit-refs 1  
--rskip ☐ --rc-lookahead ☐ --rect --amp --psy-rd ☐ --splitrd-skip --limit-sao --sao-non-  
deblock --deblock 0:-1 --y4m --input - --output ".\输出.hevc"

## β——ffmpeg libx265 CLI, 拷贝音频并封装为 mp4

- ffmpeg.exe -y -i ".\导入.mp4" -c:v libx265 -profile:v ☐ -x265-params "high-tier=1:ctu=☐:min-cu-  
size=16:tu-intra-depth=3:tu-inter-depth=3:limit-tu=1:rdpenalty=1:me=umh:subme=☐:merange=  
☐:weightb=1:ref=3:max-merge=☐:early-skip=1:open-gop=0:min-keyint=5:fades=1:bframes=8:b-  
adapt=2:pbratio=1.2:fast-intra=1:b-intra=1:crf=☐:crqpoffs=-3:cbqpoffs=-1:rdoq-aq-mode=4:aq-strength=  
☐:rd=3:limit-modes=1:limit-refs=1:rskip=☐:rc-lookahead=☐:rect=1:amp=1:psy-rd=☐:splitrd-skip=1:limit-  
sao=1:sao-non-deblock=1:deblock=0,-1" -fps\_mode passthrough -c:a copy ".\输出.mp4"

# 高压·录像/3D 动画

建议高清源，否则画质不如通用-简单，更慢，但一般压缩率更高

## 兼容性

--profile <8/10/12bit: main/main10/main12, YUV4:2:2: main422-10/main422-12, YUV4:4:4: main444-8/main444-10/main444-12> --high-tier

## 分块-变换

--ctu 64 --tu-intra-depth 4 --tu-inter-depth 4 --limit-tu 1 --rect --tskip --tskip-fast

## 动搜-补偿

--me star --subme <24fps: 3, 48fps: 4, 60fps: 5, 100fps: 6> --merange <1920:1080: 48, 2560:1440: 52, 3840:2160: 56> --weightb

## 溯块-帧控

--ref 4 --max-merge 5 --no-open-gop --min-keyint 3 --keyint <9 × 帧率> --fades --bframes 8 --b-adapt 2

## 帧内编码

--b-intra <极慢且可能会造成块失真，增加压缩率: --constrained-intra>

## 量化

--crf 21.8 --crqpoffs -3 --ipratio 1.2 --pbratio 1.5

## 率失优量化

--rdoq-level 2

## 自适应量化

--aq-mode 4 --aq-strength <多面: 1~多线: 1.3> --qg-size 8

## 模式决策

--rd 5 --limit-refs 0 --rskip 0 --rc-lookahead <1.8 × 帧率, 大于 bframes>

## 率失真优化

--psy-rd <录像: 1.6, 动画: 0.6, ctu64: +0.6, ctu16: -0.6>

## 去块

--deblock 0:-1

## 取样迁就偏移

--limit-sao --sao-non-deblock --selective-sao 3

## 多处理器分配

--pools ,,, (举例-,+表示该电脑有两个 CPU 节点, 用第二个. 同时占用多个会造成严重的内存延迟)

## 其它

去黑边加速: --display-window <整数"←,↑,→,↓"像素>, ≥22 核 cpu 优化: --pme, 分场视频: --field,

抖动高质量降色深: --dither, 开始; 结束帧: --seek; --frames, crf/abr 缓解噪点影响: --rc-grain

## α——(ffmpeg pipe) x265 CLI 命令

- ffmpeg.exe -y -i ".\导入.mp4" -an -f yuv4mpegpipe -strict -1 - | x265.exe --profile ☐ --high-tier  
--ctu 64 --tu-intra-depth 4 --tu-inter-depth 4 --limit-tu 1 --rect --tskip --tskip-fast --me star --  
subme ☐ --merange ☐ --weightb --ref 4 --max-merge 5 --no-open-gop --min-keyint 3 --keyint  
☐ --fades --bframes 8 --b-adapt 2 --b-intra --crf 21.8 --crqpoffs -3 --ipratio 1.2 --pbratio 1.5 --  
rdoq-level 2 --aq-mode 4 --aq-strength ☐ --qg-size 8 --rd 5 --limit-refs 0 --rskip 0 --rc-lookahead  
☐ --psy-rd ☐ --deblock 0:-1 --limit-sao --sao-non-deblock --selective-sao 3 --y4m --input - --  
output ".\输出.hevc"

## β——ffmpeg libx265 CLI, 拷贝音频并封装为 mp4

- ffmpeg.exe -y -i ".\导入.mp4" -c:v libx265 -profile:v ☐ -x265-params "high-tier=1:ctu=64:tu-intra-  
depth=4:tu-inter-depth=4:limit-tu=1:rect=1:tskip=1:tskip-fast=1:me=star:subme=☐ :merange=  
☐ :weightb=1:ref=4:max-merge=5:open-gop=0:min-keyint=3:keyint=☐ :fades=1:bframes=8:b-adapt=2:b-  
intra=1:crf=21.8:crqpoffs=-3:ipratio=1.2:pbratio=1.5:rdoq-level=2:aq-mode=4:aq-strength=☐ :qg-  
size=8:rd=5:limit-refs=0:rskip=0:rc-lookahead=☐ :psy-rd=☐ :deblock=0,-1:limit-sao=1:sao-non-  
deblock=1:selective-sao=3" -fps\_mode passthrough -c:a copy ".\输出.mp4"

# 剪辑素材存档

通过减少 P 帧，B 帧数量来降低解码压力，从而降低剪辑软件负载；兼容 ≥ 画质+压缩

**兼容性**      `--profile<8/10/12bit: main/main10/main12, YUV4:2:2: main422-10/main422-12, YUV4:4:4: main444-8/main444-10/main444-12> --high-tier`

**分块-变换**      `--ctu 32 --tskip`

**动态搜索**      `--me star --subme <24fps: 3, 48fps: 4, 60fps: 5, 100fps: 6> --merange <1920:1080: 48, 2560:1440: 52, 3840:2160: 56>`

**帧内搜索**      `--max-merge 5 --early-skip --b-intra`

**帧控制**      `--no-open-gop --min-keyint 1 --keyint <5×帧率> --ref 3 --fades --bframes 4 --b-adapt 2`

**量化**      `--crf 17 --crqpoffs -3 --cbqpoffs -2`

**模式决策**      `--rd 3 --limit-modes --limit-refs 1 --rskip 1 --rc-lookahead <4×帧率, 大于 bframes>`

**率失真优化**      `--splitrd-skip`

**环路滤波去块**      `--deblock -1:-1`

**主控**      `--tune grain`

**其它**      去黑边加速: `--display-window <整数"←,↑,→,↓"像素>`, ≥22 核 cpu 优化: `--pme`, 分场视频: `--field`,

α——(ffmpeg pipe) x265 CLI 命令

- `ffmpeg.exe -y -i ".\导入.mp4" -an -f yuv4mpegpipe -strict -1 - | x265.exe --profile ☐ --high-tier --ctu 32 --tskip --me star --subme ☐ --merange ☐ --max-merge 5 --early-skip --b-intra --no-open-gop --min-keyint 1 --keyint ☐ --ref 3 --fades --bframes 7 --b-adapt 2 --crf 17 --crqpoffs -3 --cbqpoffs -2 --rd 3 --limit-modes --limit-refs 1 --rskip 1 --rc-lookahead ☐ --splitrd-skip --deblock -1:-1 --tune grain --y4m --input - --output ".\输出.hevc"`

## β——ffmpeg libx265 CLI, 拷贝音频并封装为 mp4

- `ffmpeg.exe -y -i ".\导入.mp4" -c:v libx265 -profile:v ☐ -x265-params "high-tier=1:ctu=32:tskip=1:me=star:subme=☐:merange=☐:max-merge=5:early-skip=1:open-gop=0:min-keyint=1:keyint=☐:ref=3:fades=1:bframes=7:b-adapt=2:b-intra=1:crf=17:crqpoffs=-3:cbqpoffs=-2:rd=3:limit-modes=1:limit-refs=1:rskip=1:rc-lookahead=☐:splitrd-skip=1:deblock=-1,-1:tune=grain" -fps_mode passthrough -c:a copy ".\输出.mp4"`



# 高压·动漫·字幕组

建议 YUV4:2:0; 8~10bit

兼容性	<code>--profile&lt;8/10/12bit: main/main10/main12, YUV4:2:2: main422-10/main422-12, YUV4:4:4: main444-8/main444-10/main444-12&gt; --high-tier</code>
分块-变换	<code>--tu-intra-depth 4 --tu-inter-depth 4 --max-tu-size 16 --tskip --tskip-fast</code>
动搜-补偿	<code>--me umh --subme &lt;24fps: 3, 48fps: 4, 60fps: 5, 100fps: 6&gt; --merange &lt;1920:1080: 48, 2560:1440: 52, 3840:2160: 56&gt; --weightb --max-merge 5 --early-skip</code>
溯块-帧控	<code>--ref 3 --no-open-gop --min-keyint 5 --keyint &lt;12×帧率&gt; --fades --bframes 16 --b-adapt 2 --bframe-bias 20</code>
帧内编码	<code>--b-intra &lt;极慢且可能会造成画面问题: + --constrained-intra&gt;</code>
量化	<code>--crf 22 --crqpoffs -4 --cbqpoffs -2 --ipratio 1.6 --pbratio 1.3 --cu-lossless --psy-rdoq 2.3 --rdoq-level 2</code>
率失优量化	<code>--hevc-aq --aq-strength 0.9 --qg-size 8</code>
自适应量化	<code>--rd 3 --limit-modes --limit-refs 1 --rskip 1 --rc-lookahead &lt;2.5×帧率, 大于 bframes&gt; --</code>
模式决策	<code>rect --amp</code>
率失真优化	<code>--psy-rd 1.5 --splitrd-skip --rdpenalty 2</code>
去块	<code>--deblock 0:-1</code>
取样迁就偏移	<code>--limit-sao --sao-non-deblock</code>
其它	去黑边加速: <code>--display-window &lt;整数"←, ↑, →, ↓"像素&gt;</code> , $\geq 22$ 核 cpu 优化: <code>--pme</code> , 分场视频: <code>--field</code> , 抖动高质量降色深: <code>--dither</code> , 开始; 结束帧: <code>--seek; --frames</code> , crf/abr 缓解噪点影响: <code>--rc-grain</code> , 外/内网 NAS 串流: <code>--single-sei --idr-recovery-sei</code>

## α——(ffmpeg pipe) x265 CLI 命令

- ffmpeg.exe -y -i ".\导入.mp4" -an -f yuv4mpegpipe -strict -1 - | x265.exe --profile ☐ --high-tier --tu-intra-depth 4 --tu-inter-depth 4 --max-tu-size 16 --tskip --tskip-fast --me umh --subme ☐ --merange ☐ --weightb --max-merge 5 --early-skip --ref 3 --no-open-gop --min-keyint 5 --keyint ☐ --fades --bframes 16 --b-adapt 2 --bframe-bias 20 --constrained-intra --b-intra --crf 22 --crqpoffs -4 --cbqpoffs -2 --ipratio 1.6 --pbratio 1.3 --cu-lossless --psy-rdoq 2.3 --rdoq-level 2 --hevc-aq --aq-strength 0.9 --qg-size 8 --rd 3 --limit-modes --limit-refs 1 --rskip 1 --rc-lookahead ☐ --rect --amp --psy-rd 1.5 --splitrd-skip --rdpenalty 2 --deblock -1:0 --limit-sao --sao-non-deblock --y4m --input - --output ".\输出.hevc"

## β——ffmpeg libx265 CLI, 拷贝音频并封装为 mp4

- ffmpeg.exe -y -i ".\导入.mp4" -c:v libx265 -profile:v ☐ -x265-params "high-tier=1:tu-intra-depth=4:tu-inter-depth=4:max-tu-size=16:tskip=1:tskip-fast=1:me=umh:subme=☐:merange=☐:weightb=1:max-merge=5:early-skip=1:ref=3:open-gop=0:min-keyint=5:keyint=☐:fades=1:bframes=16:b-adapt=2:bframe-bias=20:b-intra=1:crf=22:crqpoffs=-4:cbqpoffs=-2:ipratio=1.6:pbratio=1.3:cu-lossless=1:psy-rdoq=2.3:rdoq-level=2:hevc-aq=1:aq-strength=0.9:qg-size=8:rd=3:limit-modes=1:limit-refs=1:rskip=1:rc-lookahead=☐:rect=1:amp=1:psy-rd=1.5:splitrd-skip=1:rdpenalty=2:deblock=-1,0:limit-sao=1:sao-non-deblock=1" -fps\_mode passthrough -c:a copy ".\输出.mp4"

# 穷举法·HEDT 压力高，画质高，压缩率不高，不适合大部分情况

兼容性	<code>--profile&lt;8/10/12bit: main/main10/main12, YUV4:2:2: main422-10/main422-12, YUV4:4:4: main444-8/main444-10/main444-12&gt; --high-tier</code>
分块-变换	<code>--tu-intra-depth 4 --tu-inter-depth 4 --max-tu-size 4 --limit-tu 1 --rect --amp --tskip</code>
动搜-补偿	<code>--me star --subme &lt;24fps: 3, 48fps: 4, 60fps: 5, 100fps: 6&gt; --merange &lt;1920:1080: 52, 2560:1440: 56, 3840:2160: 64&gt; --weightb --max-merge 5</code>
溯块-帧控	<code>--ref 3 --no-open-gop --min-keyint 1 --keyint &lt;12×帧率&gt; --fades --bframes 16 --b-adapt 2</code>
帧内编码	<code>--b-intra</code>
量化	<code>--crf 18.1 --crqpoffs -5 --cbqpoffs -2 --ipratio 1.67 --pbratio 1.33 --cu-lossless</code>
率失优量化	<code>--psy-rdoq 2.5 --rdoq-level 2</code>
自适应量化	<code>&lt;普通: --hevc-aq --aq-strength 1.4; Jpsdr Mod: --aq-auto 10 --aq-bias-strength 1.3 --aq-strength-edge 1.4 --aq-bias-strength 1.1&gt; --qg-size 8</code>
模式决策	<code>--rd 5 --limit-refs 0 --rskip 2 --rskip-edge-threshold 3 --rc-lookahead &lt;2.5×帧率, 大于bframes&gt; --no-cutree</code>
率失真优化	<code>--psy-rd 1.5 --rdpenalty 2 &lt;实验性: --qp-adaptation-range 5&gt;</code>
去块	<code>--deblock -2:-2</code>
取样迁就偏移	<code>--limit-sao --sao-non-deblock --selective-sao 1</code>

α——(ffmpeg pipe) 普通 x265 CLI 命令

- ffmpeg.exe -y -i ".\导入.mp4" -an -f yuv4mpegpipe -strict -1 - | x265.exe --profile ☐ --high-tier --tu-intra-depth 4 --tu-inter-depth 4 --max-tu-size 4 --limit-tu 1 --rect --amp --tskip --me star --subme ☐ --merange ☐ --weightb --max-merge 5 --ref 3 --no-open-gop --min-keyint 1 --keyint ☐ --fades --bframes 16 --b-adapt 2 --b-intra --crf 18.1 --crqpoffs -5 --cbqpoffs -2 --ipratio 1.67 --pbratio 1.33 --cu-lossless --psy-rdoq 2.5 --rdoq-level 2 --hevc-aq --aq-strength 1.4 --qg-size 8 --rd 5 --limit-refs 0 --rskip 2 --rskip-edge-threshold 3 --rc-lookahead ☐ --no-cutree --psy-rd 1.5 --rdpenalty 2 --deblock -2:-2 --limit-sao --sao-non-deblock --selective-sao 1--y4m --input - --output ".\输出.hevc"

## β——(ffmpeg pipe) x265 jpsdr-Mod CLI 命令

- ffmpeg.exe -y -i ".\导入.mp4" -an -f yuv4mpegpipe -strict -1 - | x265.exe --profile ☐ --high-tier --tu-intra-depth 4 --tu-inter-depth 4 --max-tu-size 4 --limit-tu 1 --rect --amp --tskip --me star --subme ☐ --merange ☐ --weightb --max-merge 5 --ref 5 --no-open-gop --min-keyint 1 --keyint ☐ --fades --bframes 16 --b-adapt 2 --b-intra --crf 18.1 --crqpoffs -5 --cbqpoffs -2 --ipratio 1.67 --pbratio 1.33 --cu-lossless --psy-rdoq 2.5 --rdoq-level 2 --aq-auto 10 --aq-bias-strength 1.3 --aq-strength-edge 1.4 --aq-bias-strength 1.1 --qg-size 8 --rd 3 --limit-refs 0 --rskip 2 --rskip-edge-threshold 3 --rc-lookahead ☐ --no-cutree --psy-rd 1.5 --rdpenalty 2 --deblock -2:-2 --limit-sao --sao-non-deblock --selective-sao 1--y4m --input - --output ".\输出.hevc"

## γ——普通 ffmpeg libx265 CLI，拷贝音频并封装为 mp4

- ffmpeg.exe -y -i ".\导入.mp4" -c:v libx265 -profile:v ☐ -x265-params "high-tier=1:tu-intra-depth=4:tu-inter-depth=4:max-tu-size=4:limit-tu=1:rect=1:amp=1:tskip=1:me=star:subme=☐:merange=☐:weightb=1:max-merge=5:ref=3:open-gop=0:min-keyint=1:keyint=☐:fades=1:bframes=16:b-adapt=2:b-

intra=1:crf=18.1:crqpoffs=-5:cbqpoffs=-2:ipratio=1.6:pbratio=1.33:cu-lossless=1:psy-rdoq=2.5:rdoq-  
level=2:hevc-aq=1:aq-strength=1.4:qg-size=8:rd=5:limit-refs=0:rskip=2:rskip-edge-threshold=3:rc-  
lookahead=○:cutree=0:psy-rd=1.5:rdpenalty=2:deblock=-2:-2:limit-sao=1:sao-non-deblock=1:selective-  
sao=1" -fps\_mode passthrough -c:a copy ".\输出.mp4"