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 AV1 教程完整版做适当调整

# 教程地图，软件下载

SVT-AV1 因缺少第三方编译项目，因此可以选用 ffmpeg 的 libsvtav1，或者手动编译。SVT-AV1 编码器的参数说明，下载与编译方法见 [iavoe.github.io](https://iavoe.github.io/)（附录——下载）。

### 下载并编译 SVT-AV1(-PSY)

#### （1/3）下载并安装编译工具（节选自 AV1 编码教程）

1. **LLVM（clang 支持）**：[github.com/llvm/llvm-project/releases/latest](https://github.com/llvm/llvm-project/releases/latest)，
   * 在打开的网盘路径中根据系统和指令集位宽找最新版程序
     + 例如，Windows 64bit 操作系统选 *LLVM-\*\*\*-win64.exe*
   * 安装时选择 *Add LLVM to the system PATH for all/current users*
2. **Microsoft C++ Build Tools**：[visualstudio.microsoft.com/visual-cpp-build-tools](https://visualstudio.microsoft.com/visual-cpp-build-tools" \t "_blank)
   * 如果已经安装 Visual Studio，且直接下载了 C++ 桌面应用开发组件，则可以跳过这步
   * 下载并运行安装程序，选择 *Desktop development with C++*
   * 打开一个记事本，将实际安装位置路径拷贝进去
   * 安装页面中，可以仅选择安装 MSVC 和最新的 Windows 10/11 SDK
3. **NASM**：[nasm.us/pub/nasm/releasebuilds/?C=M;O=D](https://www.nasm.us/pub/nasm/releasebuilds/?C=M;O=D" \t "_blank)
   * 在打开的网盘路径中根据系统和指令集位宽找最新版程序
     + 例如，Windows 64bit 选择 *最新版/win64/nasm-\*\*\*-installer-x64.exe*
   * 安装时可以去勾选 *Manual*（说明书）和 *VS8 Integration*（Visual Studio 8 集成）
4. **CMake**：[github.com/Kitware/CMake/releases](https://github.com/Kitware/CMake/releases" \t "_blank)
5. **Git**（可选）：[git-scm.com/download/win](https://git-scm.com/download/win" \t "_blank)

#### （2/3）获取 SVT-AV1(-PSY) 项目源代码

#### 浏览器下载项目源代码：

* **SVT-AV1-PSY：**将 Github/gianno-rosato/SVT-AV1-PSY 项目 [打包下载](https://github.com/psy-ex/svt-av1-psy/archive/refs/heads/master.zip) 并解压
* **SVT-AV1：**将 GitLab/AOMediaCodec/SVT-AV1 项目 [打包下载](https://gitlab.com/AOMediaCodec/SVT-AV1/-/archive/master/SVT-AV1-master.zip) 并解压

#### Git 下载项目源代码：

若照上面安装了 Git，则打开 Git Bash，并使用下方命令下载（若在安装时选择了其它编辑器，则命令行格式可能会有所差异，如CMD：cd /D 盘符:\文件夹、Bash：cd /盘符/文件夹）：

# 1. 移动到下载路径（不同的命令行工具路径格式和要求不同）

cd <下载路径>

# 2A. 下载 SVT-AV1-PSY

git clone --depth 50 https://github.com/gianni-rosato/svt-av1-psy.git

cd svt-av1-psy

# 2B. 下载 SVT-AV1

git clone https://gitlab.com/AOMediaCodec/SVT-AV1.git

cd svt-av1

注意：Windows 中，后续命令须在 CMD / Git Bash（后者暂未验证）中运行，**不要用 PowerShell**

#### （3/3）SVT-AV1(-PSY) 编译步骤

0. 在 CMD 中确认以上程序已正确安装并可运行。注意：只有全部正常才能继续：

"C:\Program Files\LLVM\bin\clang.exe" --version

:: clang version \*\*\*

:: Target: x86\_64-pc-windows-msvc

:: Thread model: posix

:: InstalledDir: C:\Program Files\LLVM\bin

where clang-cl

:: C:\Program Files\LLVM\bin\clang-cl.exe

nasm --v

:: NASM version \*\*\* compiled on \*\*\* \*\*\* \*\*\*

cmake --version

:: cmake version \*\*\*

:: CMake suite maintained and supported by Kitware (kitware.com/cmake)

1. 切换文本编码到 UTF-8：

chcp 65001

2. 在先前下载的 VS Build Tools 路径中运行 vcvars64.bat 环境配置脚本（根据实际安装位置调整）：

:: 若已经有下载了 C++ 桌面应用开发组件的 Visual Studio，且 Visual Studio 是 64bit，

:: 则路径可能是 C:\Program Files\Microsoft Visual Studio\2022\Community\VC\Auxiliary\Build\vcvars64.bat

call "C:\Program Files (x86)\Microsoft Visual Studio\2022\BuildTools\VC\Auxiliary\Build\vcvars64.bat"

3. 运行 vcvars64.bat 后会进入一个新的 CMD 界面，在此指定两个变量以定义使用 Clang-CL 编译器：

set "CC=clang-cl"

set "CXX=clang-cl"

4. 用 CMake 生成 Ninja 项目并配置编译选项（根据压制电脑的 CPU 架构进行调整）：

:: 如果当前目录不在 SVT-AV1 或 SVT-AV1-PSY 目录下，则照上例使用 cd 命令移动过去

:: 换行符取决于终端软件，如 CMD 使用 ^，Bash 使用 \。未知则直接删除所有换行

:: 如果要编译 Debug 版而非发布版，则去掉 -DCMAKE\_BUILD\_TYPE=Release 整行

cmake --fresh -B svt\_build -G Ninja ^

-DCMAKE\_BUILD\_TYPE=Release ^

-DBUILD\_SHARED\_LIBS=OFF ^

-DENABLE\_AVX512=OFF ^

-DSVT\_AV1\_LTO=OFF ^

-DCMAKE\_CXX\_FLAGS\_RELEASE="-flto /DNDEBUG /clang:-O2 -march=native" ^

-DCMAKE\_C\_FLAGS\_RELEASE="-flto /DNDEBUG /clang:-O2 -march=native"

5. 最后使用 Ninja 编译，使用 -DCMAKE\_BUILD\_TYPE=Release 则会输出到 Bin\Debug ）：

ninja -C svt\_build

:: \*\*\* warnings generated.

:: D[242/242] Linking C executable \*:\\*\*\*\SVT-AV1-master\Bin\Release\SvtAv1EncApp.exe

### 下载——基本工具

|  |  |
| --- | --- |
| [ffmpeg](http://ffmpeg.org/download.html) | 强大的 CLI 开源视音频处理工具，已内置 libsvtav1 |
| [mpv](https://mpv.io/installation/) | 开源，支持便携的现代视频视频播放器。见[安装与配置](https://nazorip.site/archives/1052/" \t "_blank)教程 |
| [Voukoder](https://www.voukoder.org/) | 开源 Premiere Vegas After Effects 压制导出插件，分为本体和连接器两部分 |
| [OBS](https://obsproject.com/zh-cn/download) | 强大的开源直播框架和软件，设置略比传统录屏软件复杂，但效果也更好 |
| [MediaInfo](https://mediaarea.net/zh-CN/MediaInfo) | 开源的 GUI 媒体元数据/视音频格式读取器，用于配置正确的压制参数 |
| [ffprobe](http://ffmpeg.org/download.html) | CLI 视音频格式读取器，若检测所得信息与 MediaInfo 所异，则优先参考 ffprobe 见[基本使用](https://nazorip.site/archives/169/" \t "_blank)，以及[搭配 Excel 的视频数据可视化](https://nazorip.site/archives/1068/)教程 |

### [下载——SVT-AV1-Essential 编码器](https://github.com/nekotrix/SVT-AV1-Essential/releases/tag/v3.1.0-Essential)（fork）

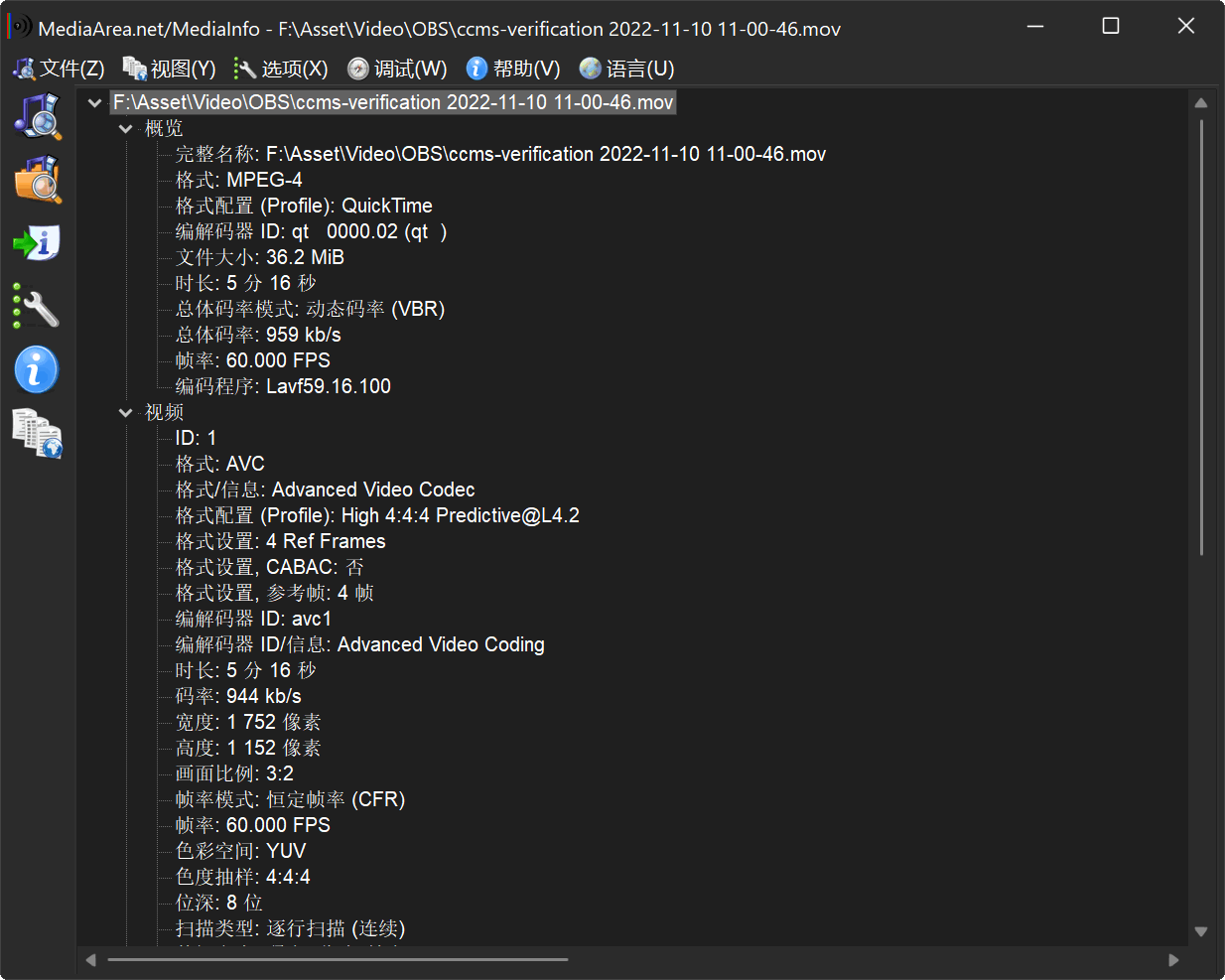
SVT-AV1-Essential同 SVT-AV1-PSY 是第三方修改版的编码器，增加了新参数，新功能，目前（3.1）版也与官方版进度一致。在完整版教程的测试，以及接下来的参数配置部分仍然围绕 SVT-AV1 进行。选择使用第三方修改版需要额外参考针对它们的参数变化，它们往往不会被 ffmpeg 收录为动态链接库。使用时请确保自己至少有一天之闲来调试，测试。

[**下载——SVT-AV1-HDR 编码器**](https://github.com/Patman86/SVT-AV1-Mod-by-Patman)**（fork）**

## 格式识别

### 格式识别——视音频格式

下载并打开 [MediaInfo](https://mediaarea.net/en/MediaInfo/Download) 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](https://iavoe.github.io/deint-ivtc-web-tutorial/HTML/index.html)。

### 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](https://en.wikipedia.org/wiki/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](https://www.nazorip.site/archives/44/) 或 [Github](https://github.com/iAvoe/QAAC-Tutorial-Standalone/blob/master/%E6%95%99%E7%A8%8B.md)。

### 格式识别——压制用时

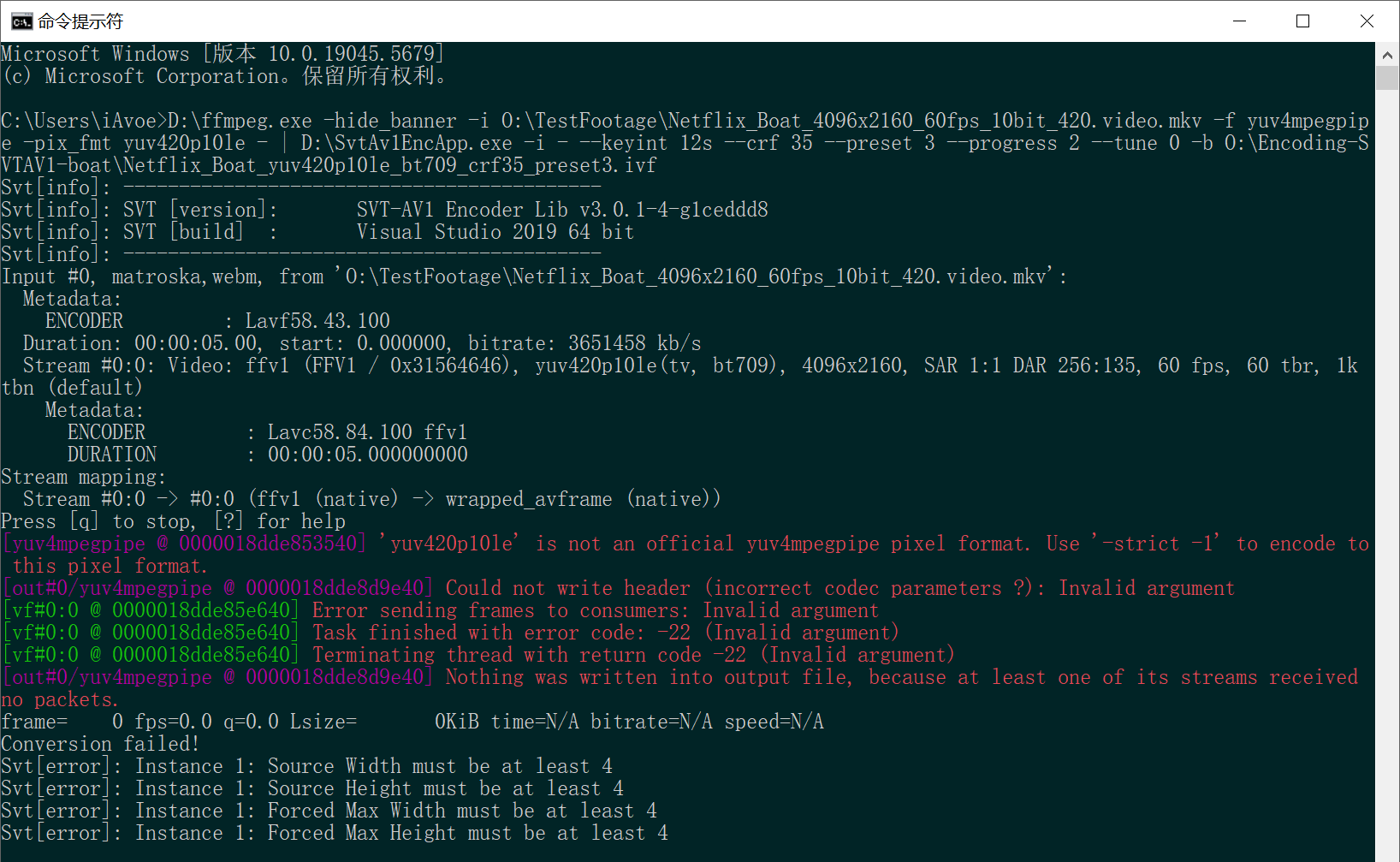
时长秒数 = 总帧数÷压缩速度fps。通过系统查看封装文件属性，或MediaInfo、ffprobe得到视频时长，即可在视频编码器不预估完成时间（如某些情况下未提供总帧数信息）的情况下手动计算。

### 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 。



## 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、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”的前置步骤。详细说明见 [AV1 教程](G://JCSUSBbackup/av1-web-tutorial/HTML/index.html)。

* **视觉无损：** XPSNR ≥ 45dB，VMAF ≥ 95
* **优秀：** XPSNR ≥ 42dB，VMAF ≥ 90
* **良好：** XPSNR ≥ 38dB，VMAF ≥ 80
* **达标：** XPSNR ≥ 32dB，VMAF ≥ 70

### 客观画质指标跑分——XPSNR

快速低占用，ffmpeg 内置，计算“源与压缩结果的差异”，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]; [enc] [src]xpsnr=stats\_file=-" -f null -

### 主观画质指标跑分——VMAF

快速，ffmpeg 内置，倾向于检查视觉观感体验，而非简单地差异（如高压缩下保证大体观感）。可并用 4K（客厅/影院，高频细节敏感）和 1080p（显示器，二次编码、重采样、染色等失真）两种模型。

:: 普通版本（VMAF4k + VMAF1080p；注意，正确顺序是先压缩，后原画）

ffmpeg -i ".\压缩源.ivf" -i ".\原画源.mkv" -lavfi libvmaf="model=version=vmaf\_4k\_v0.6.1\\:name=vmaf\_4k|version=vmaf\_v0.6.1\\:name=vmaf\_1080p" -f null -

:: 普通版本+标签顺序（VMAF4k + VMAF1080p；在 lavfi 内部调转顺序）

ffmpeg -i ".\原画源.mkv" -i ".\压缩源.ivf" -lavfi [0:v][src]; [1:v][enc]; [enc][src]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]; [enc][src]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](https://github.com/iAvoe/GCDLCMCalculator-VideoQualityMetrics)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

## SVT-AV1 规格与级别

### --profile

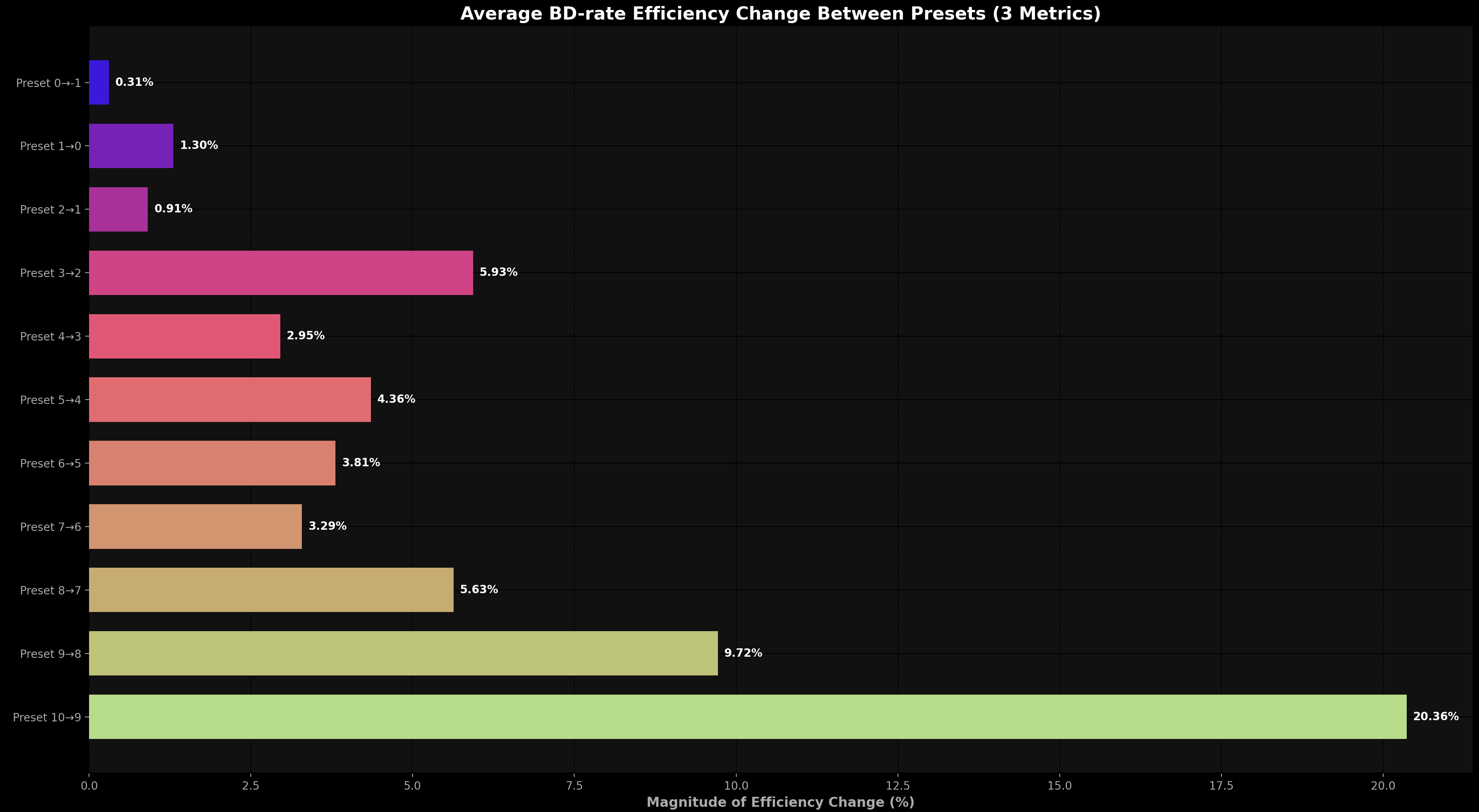
<整数 0~2，默认 0>指定输出视频规格——0：main，1：high，2：professional

|  |  |  |  |
| --- | --- | --- | --- |
| 规格 | Main | High | Professional |
| 采样\位深 | **8/10 bit** | **8/10 bit** | **8/10/12 bit** |
| YUV 4:0:0 | √ | √ | √ |
| YUV 4:2:0 | √ | √ | √ |
| YUV 4:2:2 | × | × | √ |
| YUV 4:4:4 | × | √ | √ |

SVT-AV1 的 --level 会自动根据 ffmpeg 调用 libsvtav1，或上游程序 YUV for MPEG 管道输入参数给 SVT-AV1 编码器配置。如需手动配置，可见[iavoe.github.io](https://iavoe.github.io/)（SVT-AV1——参数与预设）的 Level 表。

## SVT-AV1速度与保真率

完整的预设参数/主控参数说明，包括牵扯的模式强度变化见[iavoe.github.io](https://iavoe.github.io/)（SVT-AV1——参数与预设）。

图：SVT-AV1 各个 preset 下的保真率差距大小

异于 x264 与 x265，SVT-AV1 的*绝大多数强度*（至少 38 项）咎由 --preset 指定。根据开启的功能和强度来说，最适合高压缩和高画质的模式是 2 ；在用不到、可妥协一定画质、或速度优先的情况下才适合 3 或者更高速度；而 -1 是 0 的上位替代（耗时相似但更聪明一些）。

使用 Clang-LLVM、开启当前 CPU 架构优化、禁 AVX512 编译的SVT-AV1 3.1 版 --preset 2 比x265 4.0 版的 --preset veryslow （x265 教程急用版的录像高压、动漫字幕组参数）快 1/5~1/2。而直接使用 Visual Studio 编译的 SVT-AV1 3.0 版 --preset 2 速度略慢于同样的 x265 参数。

测试发现，SVT-AV1 --preset 2 比 3 慢 1 倍；而 --preset 1 可以比 2 慢 50 倍。

## SVT-AV1色彩格式与 VUI/SEI 标注

### --matrix-coefficients

<部分整数 0~14，默认 2>播放用矩阵格式/系数指标：

* **0：**identity
* **1：**bt709
* **2：**留空（视频播放器默认）
* **4：**fcc
* **5：**bt470bg
* **6：**bt601
* **7：**smpte240m
* **8：**ycgco
* **9：**bt2020-ncl（BT.2020 可变亮度，BT.2100 YCbCr）
* **10：**bt2020-nc（BT.2020 单一亮度）
* **11：**smpte2085
* **12：**chroma-ncl（chroma-derived-nc 色度衍生可变亮度）
* **13：**chroma-cl（chroma-derived-c 色度衍生单一亮度）
* **14：**ictcp (BT.2100 ICtCp)
* **3：**保留值，不可用

### --transfer-characteristics

<部分整数 1~18，默认 2>传输特质：

* **1：**bt709
* **2：**留空（视频播放器默认）
* **4：**bt470m
* **5：**bt470bg
* **6：**bt601
* **7：**smpte240、smpte240m
* **8：**linear
* **9：**log100
* **10：**log100-sqrt10
* **11：**iec61966（-2-4）
* **12：**bt1361
* **13：**srgb、sycc
* **14：**bt2020-10
* **15：**bt2020-12
* **16：**smpte2084
* **17：**smpte428
* **18：**hlg
* **3：**保留值，不可用

### --color-primaries

<部分整数 1~22，默认 2（留空）>三原色及白点指标，详见上述的国际电信联盟（ITU-T）[H.273——Table 2](https://www.itu.int/rec/T-REC-H.273-202407-I)：

* **1：**bt709
* **2：**留空（视频播放器默认）
* **4：**bt470m
* **5：**bt470bg
* **6：**bt601
* **7：**smpte240
* **8：**film
* **9：**bt2020
* **10：**xyz
* **11：**smpte431
* **12：**smpte432
* **22：**ebu3213
* **0、3、13~21、...：**保留值，不可用

## SVT-AV1 HDR VUI/SEI 标注

### --master-display

<字符串 G(x,y)B(x,y)R(x,y)WP(x,y)L(max,min)>用于高动态范围内容的绿、蓝、红、白点、光强信息

* RGB-WP 原信息见ITU-T H.273 表格，以下为转换出的结果：
* **bt709:** 
  + G(0.30,y0.60)B(x0.150,y0.060)R(x0.640,y0.330)WP(x0.3127,y0.329)L(max,min)
* **bt470m:** 
  + G(0.21,0.71)B(0.14,0.08)R(0.67,0.33)WP(0.31,0.316)L(max,min)
* **bt470bg:** 
  + G(0.29,0.60)B(0.15,0.06)R(0.64,0.33)WP(0.3127,0.329)L(max,min)
* **bt601:** 
  + G(0.31,0.595)B(0.155,0.07)R(0.63,0.34)WP(0.3127,0.329)L(max,min)
* **smpte240:** 
  + G(0.31,0.595)B(0.155,0.07)R(0.63,0.34)WP(0.3127,0.329)L(max,min)
* **film:** 
  + G(0.243,0.692)B(0.145,0.049)R(0.681,0.319)WP(0.31,0.316)L(max,min)
* **bt2020:** 
  + G(0.71,0.797)B(0.131,0.046)R(0.708,0.292)WP(0.3127,0.329)L(max,min)
* **xyz:** 
  + Y(0,1),Z(0,0),X(1,0)WP(0.3333,0.3333)L(max,min)
* **smpte431:** 
  + G(0.265,0.69)B(0.15,0.06)R(0.68,0.32)WP(0.314,0.351)L(max,min)
* **smpte432:** 
  + G(0.265,0.69)B(0.150,0.060)R(0.68,0.32)WP(0.3127,0.329)L(max,min)
* **ebu3213:** 
  + G(0.295,0.605)B(0.155,0.077)R(0.63,0.34)WP(0.3127,0.329)L(max,min)
* **L(max,min):** 见源视频的元数据。SDR 视频的光强是 (1000,1)，单位

### --content-light

<逗号分隔整数 MaxCLL,MaxFall>根据源视频的元数据设置最大内容光强（Max content light level）与最大平均光强（Max Frame-Avg Light level）

## 检查 ffmpeg 内置 libsvtav1 版本

编译好的 ffmpeg 往往会使用静态链接库，导致 ffmpeg -version 等命令检查不到编码器等库的版本，在 SVT-AV1 编码器中，可以通过生成空视频来启动编码器，从而使得其内部信息被打印出来：

ffmpeg.exe -loglevel info -f lavfi -i nullsrc=s=16x16:d=0.1 -c:v libsvtav1 -f null -

从而看到：

Svt[info]: SVT [version]: SVT-AV1 Encoder Lib vX.Y.Z

## SVT-AV1 线程与内存管理

### --lp

<整数 0~6，默认 0：全部>指定当前压制任务的线程数量（Level of Parallelism）

### --pin

<整数 0~n，默认 0：关>指定仅占用前 n 个 CPU 核心，应搭配 --lp 使用

### --ss

<整数 -1~n，默认 -1：自动>指定使用多节点系统中的特定节点编码

使用 --lp 4 --pin 4 将限制编码器运行到核心 0~3，并将资源分配设置为与 --lp 4 关联的线程/内存数量；而使用 --pin 0 --lp 4 会导致相同的线程/内存分配，但编码器在需要时可以在 0~3 之外的核心上运行（根据编码任务自动使用，但 4 线程本身给的也比较少）。 --pin 0 下的线程控制相当于只限制内存占用，这样相比限制核心反而能允许更多编码被同时运行，CPU 利用率更高，批量压制速度更快。见：[Gitlab/AOMediaCodec](https://gitlab.com/AOMediaCodec/SVT-AV1/-/blob/master/Docs/Parameters.md#1-thread-management-parameters)。

## 可能不支持的参数

### --enable-dlf 2

<整数 0~2，默认 1：普通>指定去块滤镜模式——0：关，1：普通，2：精确。需确保 SVT-AV1 版本最新才能支持 2。

# 通用·画质+

在画质高于 x265 急用版教程 CRF18片源优化参数的前提下画质更高、速度更快

主控预设

GOP与前瞻

量化

自适应量化

环路滤镜

--preset <低中: **4**, 中: **3**, 高/如 veryslow（推荐）: **2**, 极高: **1**, 特高: **0**, 特高+: **-1**, 极限: **-3**>

--keyint 12s --scd 1 --enable-tf 2 --tf-strength 2

--crf <中画质: **40**, 中高画质（字幕组）: **35**, 高画质（x265 crf17~18）: **30**, 剪辑素材: **25~15**> --enable-qm 1

--enable-variance-boost 1 --variance-boost-curve 2 --variance-boost-strength 2 --variance-octile 2

--enable-dlf 2 --sharpness 6

α——(ffmpeg pipe) SvtAv1EncApp CLI命令

* ffmpeg.exe -y -i ".\导入.mp4" -an -f yuv4mpegpipe -strict -1 - | SvtAv1EncApp.exe -i - --profile main --preset 2 --keyint 12s --scd 1 --enable-tf 2 --tf-strength 2 --crf 30 --enable-qm 1 --enable-variance-boost 1 --variance-boost-curve 2 --variance-boost-strength 2 --variance-octile 2 --enable-dlf 2 --sharpness 6 -b ".\输出.ivf"

β——ffmpeg libsvtav1 CLI，并带音频拷贝封装为mp4

* ffmpeg.exe -y -i ".\导入.mp4" -c:v libsvtav1 -profile:v main -preset 2 -crf 30 -svtav1-params "keyint=12s:scd=1:enable-tf=2:tf-strength=2:enable-qm=1:enable-variance-boost=1:variance-boost-curve=2:variance-boost-strength=2:variance-octile=2:enable-dlf=2:sharpness=6" -c:a copy ".\输出.mp4"

通用·压缩+

关闭自适应量化

主控预设

GOP与前瞻

量化

环路滤镜

--preset <低中: **4**, 中: **3**, 高/如 veryslow（推荐）: **2**, 极高: **1**, 特高: **0**, 特高+: **-1**, 极限: **-3**>

--keyint 12s --scd 1 --enable-tf 2 --tf-strength 2

--crf <中画质: **40**, 中高画质（字幕组）: **35**, 高画质（x265 crf17~18）: **30**, 剪辑素材: **25~15**>

--enable-dlf 2 --sharpness 4

α——(ffmpeg pipe) SvtAv1EncApp CLI命令

* fmpeg.exe -y -i ".\导入.mp4" -an -f yuv4mpegpipe -strict -1 - | SvtAv1EncApp.exe -i - --profile main --preset 2 --keyint 12s --scd 1 --enable-tf 2 --tf-strength 2 --crf 30 --enable-dlf 2 --sharpness 4 -b ".\输出.ivf"

β——ffmpeg libsvtav1 CLI，并带音频拷贝封装为mp4

* ffmpeg.exe -y -i ".\导入.mp4" -c:v libsvtav1 -profile:v main -preset 2 -crf 30 -svtav1-params "keyint=12s:scd=1:enable-tf=2:tf-strength=2:enable-variance-boost=1:enable-dlf=2:sharpness=4" -c:a copy ".\输出.mp4"

#### 此外，可以选择使用弱自适应量化（体积增加 1/5~1/3）：

α --enable-variance-boost 1 --variance-boost-curve 1 --variance-boost-strength 1

β :variance-boost-curve=1:variance-boost-strength=2:variance-octile=2

通用·速度+

主控预设

GOP与前瞻

量化

自适应量化

环路滤镜

--preset <低中: **4**, 中（推荐）: **3**, 高/如 veryslow: **2**, 极高: **1**, 特高: **0**, 特高+: **-1**, 极限: **-3**>

--keyint 10s --scd 1 --scm 0 --enable-tf 2 --tf-strength 2

--crf <中画质: **40**, 中高画质（字幕组）: **35**, 高画质（x265 crf17~18）: **30**,剪辑素材: **25~15**> --tune 0

--enable-variance-boost 1 --variance-boost-curve 2 --variance-boost-strength 2 --variance-octile 2

--sharpness 4

α——(ffmpeg pipe) SvtAv1EncApp CLI命令

* fmpeg.exe -y -i ".\导入.mp4" -an -f yuv4mpegpipe -strict -1 - | SvtAv1EncApp.exe -i - --profile **main** --preset **2** --keyint 12s --scd 1 --scm 0 --enable-tf 2 --tf-strength 2 --crf **30** --tune 0 --enable-variance-boost 1 --variance-boost-curve 2 --variance-boost-strength 2 --variance-octile 2 --sharpness 4 -b ".\输出.ivf"

β——ffmpeg libsvtav1 CLI，并带音频拷贝封装为mp4

* ffmpeg.exe -y -i ".\导入.mp4" -c:v libsvtav1 -profile:v **main** -preset **2** -crf **30** -tune 0 -svtav1-params "keyint=12s:scd=1:scm=0:enable-tf=2:tf-strength=2:enable-variance-boost=1:variance-boost-curve=2:variance-boost-strength=2:variance-octile=2:sharpness=4" -c:a copy ".\输出.mp4"