

观看本教程意味着你已经阅读了 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，但不一定符合实际要求，因此建议搭配 x264 教程完整版做适当调整。

教程地图，软件下载

地图见 iavoe.github.io。点击下方表格中的[超链接](#)以下载软件。

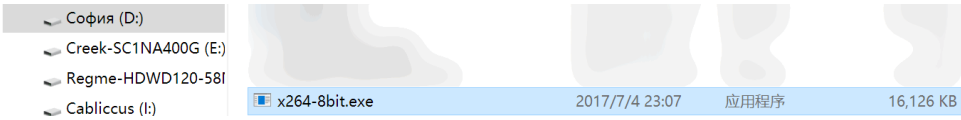
ffmpeg~ffprobe 顶级开源多系统多媒体 CLI 处理~检测工具	
mpv 支持便携的开源多系统现代视频播放器。见 安装与配置教程	
Voukoder 开源 Premiere、Vegas、Aftereffects 压制导出插件，分为 Voukoder 和 V-Connector 两部分	
MediaInfo 开源 GUI 媒体元数据/视音频格式读取器，用于快速查看完整元数据	
x264 by Patman, LigH vlavf 编解码	
x264 tMod by jspdr vlavf 编解码，支持 MCF 线程管理库(比 posix 和 win32 性能更好)	

程序下载与命令行用法

1. 于上表下载 ffmpeg, ffprobe/MediaInfo, x264 并记住路径

选择编码器位深？

有同时含 8-10-12bit 的 x264.exe，以及区分为 x264-8bit.exe, x264-10bit.exe 的版本。一般来说，单个含多位深的程序会方便 ffmpeg/AVS/VS 通过 yuv for mpeg 管道传递位深信息到编码器，使 x264.exe 自动设置位深。



此处置于 Windows 系统 D 盘根目录下，因此路径为 D:\

2. CMD/PowerShell/Bash/Terminal 下分别输入 ffmpeg、x265 的路径并回车，即可确认两点：

路径拼写：直接选择并复制以配置命令行

程序版本：越新越好，编码器的大版本更新有 Bug 修复与性能提升，其它软件的更新有新格式兼容，翻译改进等体验提升

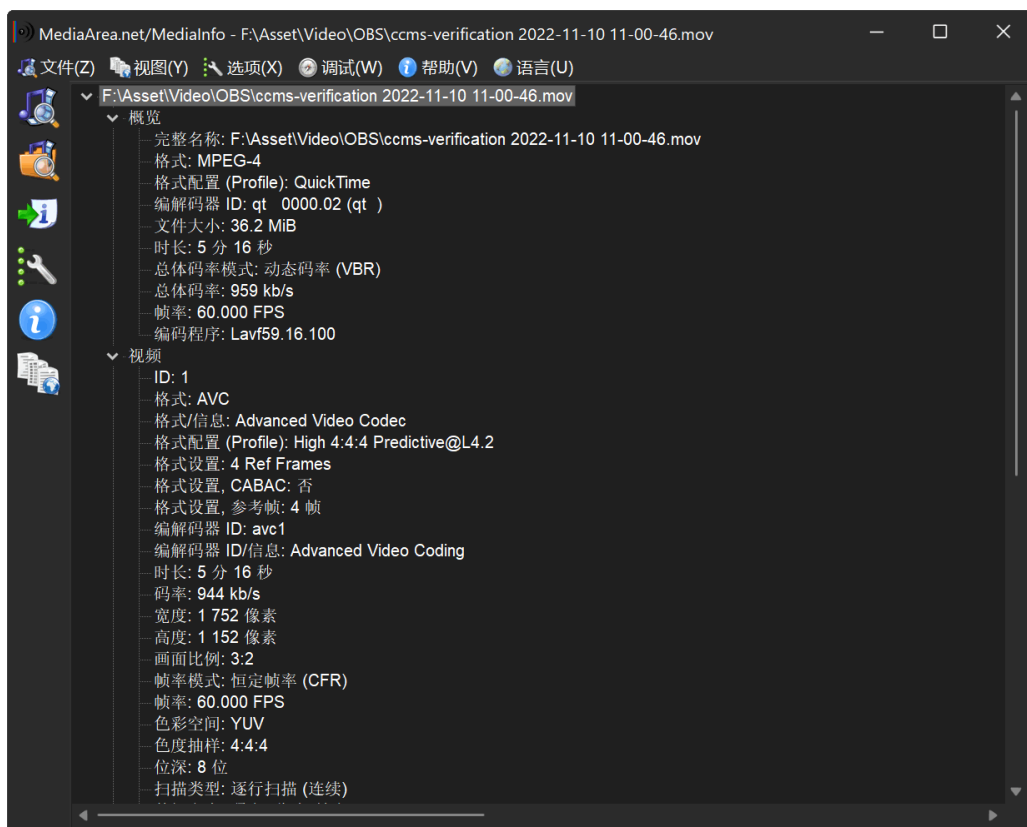
```
管理员: 命令提示符
Microsoft Windows [版本 10.0.17763.2628]
(c) 2018 Microsoft Corporation. 保留所有权利。

C:\Users\JC>D:\x264-8bit.exe -V
x264 0.152.2851+45 9658d1a 7mod [8-bit@all X86_64]
(lsmash 2.14.4)
(libwscale 4.7.101)
(libavformat 57.75.100)
(ffmpegsource 2.23.0.0)
built on Jul 5 2017, gcc: 5.4.0
x264 configuration: --bit-depth=8 --chroma-format=all --enable-openc1
libx264 configuration: --bit-depth=8 --chroma-format=all
x264 license: GPL version 2 or later
libwscale/libavformat/ffmpegsource license: GPL version 3 or later
```

图中检查 D:\x265-4bit.exe -V 确认程序存在

3. 使用 MediaInfo (图形界面) 或 ffprobe (命令行界面) 获取视音频格式细节：

双击打开 MediaInfo.exe 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



假设 ffprobe 位于 D 盘根目录下，则命令为 D:\ffprobe.exe -i ".\视频.mp4" -select_streams v:0 -v error -show_streams -show_frames -read_intervals "%+#1" -show_entries frame=top_field_first:stream=codec_long_name,width,coded_width,height,coded_height,pix_fmt,color_range,field_order,r_frame_rate,avg_frame_rate,nb_frames -of ini

```

[frames.frame.0]
top_field_first=0          分场-是否上场优先 (1/0)

[streams.stream.0]
codec_long_name=H.264      源视频格式
width=1920                 宽
height=1080                高
coded_width=1920           编码宽 - 若!=宽则代表横向长方形像素源
coded_height=1088          编码高 - 若!=高则代表纵向长方形像素源
pix_fmt=yuv420p            色彩空间
color_range=tv             色彩范围 (pc=full=0~255/tv=limited=16~235)
field_order=progressive    逐行/分场 (progressive/interlaced/unknown)
r_frame_rate=24000/1001    帧率
avg_frame_rate=24000/1001  编码帧率 - 若!=帧率则代表可变帧率vfr
nb_frames=20238            总帧数 - 根据压缩速度fps推测完成时间

```

图中为 ffprobe 输出，得到了编码格式名，视频帧大小和实际大小，色彩空间格式与范围，视频帧率，平均帧率，总帧数。依此可以判断：

交错/分行扫描？

这类视频并非使用帧率，而是“场率”为画面基础。有上场优先、下场优先；搭配原生帧率，有 NTSC 电视标准丢帧，有 PAL 电视标准丢帧，有假丢帧等多种“相信后人智慧”的兼容性需求。需要进一步根据帧率，如果一定要处理成现代的逐行扫描格式，可以参考[这篇教程](#)

可变帧率？

帧率模式显示 VFR 或 avg_frame_rate 异于 r_frame_rate，此时需要确保视频在剪辑前被重编码（渲染为恒定帧率 CFR），以保证剪辑软件/工具链上全部视频滤镜的兼容性，以及避免音画不同步。如 ffmpeg 可以添加 -vsync cfr 转换为恒定帧率 Constant Frame Rate

音频格式兼容？

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](#)

长方形像素?

视频大小和实际编码大小不同，代表了日本电视台缩宽，旧版优酷缩高的古代视频压缩手段。能换源则尽

可能换

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

配置最终参数

在确保了兼容和可行性后，即可使用命令行进行视频压制。在 CMD/PowerShell/Bash/Terminal 中，输入与上述 ffprobe 同类的命令即可。使用 x264 时，导入命令的部分被放在了命令行末尾，与 ffmpeg/ffprobe 不同，x264 的程序要求不使用专门的“-i”命令指定导入文件和路径，而是放在命令行末尾的一段命令

[参数格式] x264.exe --me esa --merange 24 [...] --output “导出.mp4” “导入.mp4”

[参数用例] D:\x264-8bit.exe --me umh --subme 11 --merange 32 -I 270 -i 1 -b 11 --b-adapt 2 -r 3 --direct auto --crf 19 --qpmin 13 --rc-lookahead 90 --aq-mode 3 --aq-strength 1 --trellis 2 --deblock 0:0 --psy-rd 0.7:0.2 --fullrange --vf hqdn3d:1.1,1.1,1.1,1.1 --output “F:\导出.mp4” “D:\导入.mp4”

空格，标点与拼写错误

在命令行程序中，参数和参数值、命令和命令之间通常使用空格作为分隔符，因此路径（参数值）中如果有空格，则应该用英文直引号 “” 括起来。同样的，程序往往使用英文逗号 ， 或英文冒号 ： 来分隔多值。

如果命令拼写错误，则命令行程序找不到命令，也会报错。空格与引起的报错往往以空格之后的“命令”不明的形式呈现、分隔符引起的报错往往以不明参数值的形式呈现、拼写引起的报错往往以命令不明形式呈现

在 API 命令行程序中的符号不同。拼写起来会麻烦一些，但规则与报错规律同样适用。

一般命令行参数格式，ffmpeg 常用操作，命令行操作技巧

虽然编码器不同，但格式一致，ffmpeg 部分也通用。因此可见 [x265 教程完整版](#)

ffmpeg 非必要参数

-hide_banner (减少命令窗口文本，更容易找到报错信息)

-loglevel 16 (减少命令窗口文本，更容易找到报错信息)

x264 HDR 设置参数:

x264

--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(?,1)

bt709: G(15000,30000)B(7500,3000)R(32000,16500)WP(15635,16450)L(?,1)

HDR 标识

bt2020: G(8500,39850)B(6550,2300)R(35400,14600)WP(15635,16450)L(?,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)>

-- cll <和 master-display 的 L 最大值一样>

--colormatrix <照源，例: gbr bt709 fcc bt470bg smpte170m YCgCo bt2020nc bt2020c smpte2085 ictcp>

--transfer <照源，例: gbr bt709 fcc bt470bg smpte170m YCgCo bt2020nc bt2020c smpte2085 ictcp>

色域标识

通用·简单

去掉了全部自定义项目，方便急用但降低了特定画面的压缩率

前瞻进程 `--rc-lookahead 90 --bframes 12 --b-adapt 2`

动态-帧内搜索 `--me umh --subme 9 --merange 48 --no-fast-pskip --direct auto --weightb`

帧控-参考 `--keyint 360 --min-keyint 5 --ref 3`

量化 `--crf 20 --qpmin 9 --chroma-qp-offset -2`

自适量 `--aq-mode 3 --aq-strength 0.7 --trellis 2`

环滤/RDO `--deblock 0:0 --psy-rd 0.77:0.22 --fgo 10`

降噪 `--nr 8`

色彩范围 `--fullrange<非 7mod x264 用, 检查源视频是否使用完整色彩范围>`

动-帧快速搜索 `--me hex --subme 8 --merange 32 --direct auto --weightb`

参考冗余优先 `--sliced-threads <降低 CPU 占用, 减速但时域复杂画面的压缩率可能提高, 参考错误降低>`

放/裁/边/降噪 `--vf crop:左,上,右,下/resize:缩放后宽,缩放后高,,,,bicubic/pad:左,上,右,下,直接宽,直接高`

滤镜 `/hqdn3d:1,1,1,1.5`

划区压制 `--zones 0,<片头 OP 结束帧>,crf=30 --zones<片尾 ED 开始帧>,<片尾 ED 结束帧>,crf=30`

压制范围 `--seek 从第<>帧开始压 --frame 压制<>帧后停止 --fps 元数据没写多少时手动指定帧数`

α——x264 CLI 命令

`x264.exe --rc-lookahead 90 --bframes 12 --b-adapt 2 --me umh --subme 9 --merange 48 --no-fast-pskip --direct auto --weightb --keyint 360 --min-keyint 5 --ref 3 --crf 20 --qpmin 9 --chroma-qp-`

```
offset -2 --aq-mode 3 --aq-strength 0.7 --trellis 2 --deblock 0:0 --psy-rd 0.77:0.22 --fgo 10 --nr 4 --  
output ".\输出.mp4" ".\导入.mp4"
```

β——libx264 私有 CLI, 兼容 libav, 不支持 fgo

```
ffmpeg.exe -y -i ".\导入.mp4" -c:v libx264 -x264-params "rc-lookahead=90:bframes=12:b-  
adapt=2:me=umh:subme=9:merange=48:fast-pskip=0:direct=auto:weightb=1:keyint=360:min-  
keyint=5:ref=3:crf=20:qpmin=9:chroma-qp-offset=-2:aq-mode=3:aq-strength=0.7:trellis=2:deblock=0,0:psy-  
rd=0.77,0.22:nr=4" -fps_mode passthrough -c:a copy ".\输出.mp4"
```

γ——libx264 私有 CLI, 不支持 fgo

上面的 -x264-params 改成 -x264opts, 但功能完全相同

δ——libx264 ffmpeg CLI 命令

```
ffmpeg.exe -y -i ".\导入.mp4" -c:v libx264 -bf 12 -b_strategy 2 -me_method umh -subq 9 -me_range 48 -  
flags2 -fastpskip -directpred 3 -flags2 +wpred -g 360 -keyint_min 5 -refs 3 -crf 20 -qmin 9 -chromaoffset -2 -  
aq-mode 3 -aq-strength 0.7 -trellis 2 -deblockalpha 0 -deblockbeta 0 -psy-rd 0.77:0.22 -nr 4 -flags2  
+bpyramid -fps_mode passthrough -c:a copy ".\输出.mp4"
```

通用·标准

配置起来慢些但自定义范围广。由于 x264 参数不多，所以这套参数足以涵盖高画质高压-录像-动画情形

前瞻进程 `--rc-lookahead <3 × 帧率> --bframes 12 --b-adapt 2`

动态-帧内搜索 `--me umh --subme <电影 10~11, 动漫 9 (11 仅 7mod)> --merange <快 20, 高压 48, 4 的倍数> --no-fast-pskip --direct auto --weightb`

帧控-参考 `--keyint <8~10 × 帧率> --min-keyint <1 增加 IDR, 5 常用> --ref 3`

量化 `--crf 19 --qpmin 9 --chroma-qp-offset <常用-1~-2, 动漫-3~-5>`

自适应量化 `--aq-mode 3 --aq-strength <一般 0.7, 原画 1.1> --trellis 2`

环滤/RDO `--deblock <一般 0:0, 原画-1:-1> --psy-rd<动漫 0.4~.6:0.1~.15, 录像 0.7~1.3:0.12~.2> --fgo 12`

CRF-VBR 压缩 `--nal-hrd --vbv-buFSIZE <最大 kbps 每秒> --vbv-maxrate <buFSIZE 倍数的 kbps>`

色彩范围 `--fullrange<非 7mod x264 用, 检查源视频是否使用完整色彩范围>`

参考冗余优先 `--sliced-threads <降 CPU 占用, 减速但时域复杂画面的压缩率可能提高, 参考错误降低>`

放/裁/边/降噪 `--vf crop:左,上,右,下/resize:缩放后宽,缩放后高,,,bicubic/pad:左,上,右,下,直接宽,直接高`

参数划区压制 `/hqdn3d:1.1,1.1,1.1,1.1`

压制范围 `--zones 0,<片头 OP 结束帧>,crf=30 --zones<片尾 ED 开始帧>,<片尾 ED 结束帧>,crf=30`

α——x264 CLI 命令

x264.exe --rc-lookahead ○ --me umh --bframes 12 --b-adapt 2 --subme ○ --merange ○ --no-fast-pskip --direct auto --weightb --keyint ○ --min-keyint ○ --ref 3 --crf 19 --qpmin 9 --chroma-qp-

offset ☐ --aq-mode 3 --aq-strength ☐ --trellis 2 --deblock ☐ --psy-rd ☐ --fgo 12 --output ".\输出.mp4" ".\导入.mp4"

β——libx264 私有 CLI, 兼容 libav, 不支持 fgo

```
ffmpeg.exe -y -i ".\导入.mp4" -c:v libx264 -x264-params "rc-lookahead=☐:me=umh:bframes=12:b-adapt=2:subme=☐:merange=☐:fast-pskip=0:direct=auto:weightb=1:keyint=☐:min-keyint=☐:ref=3:crf=19:qpmin=9:chroma-qp-offset=☐:aq-mode=3:aq-strength=☐:trellis=2:deblock=0,-1:psy-rd=☐,☐:nr=4"-fps_mode passthrough -c:a copy ".\输出.mp4"
```

γ——libx264 私有 CLI, 不支持 fgo

上面的 -x264-params 改成 -x264opts, 但功能完全相同

δ——libx264 ffmpeg CLI 命令

注: ffmpeg 的 x264 受版权限制而重写了部分参数名; 有的 ffmpeg 版本可能又没有重写, 较为不便

```
ffmpeg.exe -y -i ".\导入.mp4" -c:v libx264 -me_method umh -subq ☐ -me_range ☐ -flags2 -fastpskip -directpred 3 -flags2 +wpred -g ☐ -keyint_min ☐ -bf 12 -b_strategy 2 -refs 3 -crf 19 -qpmin 9 -chromaoffset ☐ -aq-mode 3 -aq-strength ☐ -trellis 2 -deblockalpha 0 -deblockbeta -1 -psy-rd ☐:☐ -nr 4 -flags2 +bpyramid -fps_mode passthrough -c:a copy ".\输出.mp4"
```

剪辑素材存档

加强无损压缩，降低有损压缩，增加 IDR 帧数量. 建议 YUV4:2:2 或 4:4:4 8~10bit

前瞻进程 --bframes 12 --b-adapt 2

动态-帧内搜索 --me esa --subme <电影 10~11, 动漫 9 (11 仅 7mod)> --merange <快速 40, 高压 48> --no-fast-pskip --direct auto --weightb

帧控-参考 --keyint <5~8 × 帧率> --min-keyint 1 --ref 3 --sliced-threads

量化-主控 --crf 17 --tune grain

自适-RDO --trellis 2 --fgo 15

划区压制 --zones 0,<片头/OP 结束帧>,crf=30 --zones<片尾/ED 开始帧>,<片尾/ED 结束帧>,crf=30

压制范围 --seek 从第<>帧开始压 --frame 压制<>帧后停止 --fps 元数据没写多少时手动指定帧数

α——x264 CLI 命令

x264.exe --bframes 12 --b-adapt 2 --me esa --subme ○ --merange ○ --no-fast-pskip --direct auto --weightb --keyint ○ --min-keyint 1 --ref 3 --crf 17 --tune grain --trellis 2 --fgo 15 --output ".\输出.mp4" ".\导入.mp4"

β——libx264 私有 CLI, 兼容 libav, 不支持 fgo

ffmpeg.exe -y -i ".\导入.mp4" -c:v libx264 -x264-params "me=esa:subme=○:merange=○:fast-pskip=0:direct=auto:weightb=1:keyint=○:min-keyint=1:bframes=12:b-adapt=2:ref=3:crf=17:trellis=2" -fps_mode passthrough -c:a copy ".\输出.mp4"

γ——libx264 私有 CLI, 不支持 fgo

上面的 -x264-params 改成 -x264opts, 但功能完全相同

δ——libx264 ffmpeg CLI 命令

```
ffmpeg.exe -y -i ".\导入.mp4" -c:v libx264 -bf 12 -b_strategy 2 -me_method esa -subq ○ -me_range ○ -  
flags2 -fastskip -directpred 3 -flags2 +wpred -g ○ -keyint_min 1 -refs 3 -crf 17 -tune grain -trellis 2 -flags2  
+bpyramid -fps_mode passthrough -c:a copy ".\输出.mp4"
```

OBS 录屏

本版块的内容已经有些过时，但除了少数占用大量 CPU 的游戏以外，使用 CPU 录屏的画质更好、成本更低，不需要额外购置录制设备就可以录屏和直播。因此仍然是不错的选择，由文件菜单打开设置。

高级

颜色格式-色彩空间-色彩范围-文件名格式，一般除文件名外和图中保持一致就行。但由于文件名格式也在这里，因此每次录制都要来这里修改标题。



OBS 使用了 FOURCC 代码来表示色彩空间，与 ffmpeg 的写法不同，根据下表转译。兼容性最高的是 4:2:0 8bit 格式，如果需要保存精细的色度信息，且已验证兼容，则可以尝试 10bit、4:4:4 等高位深/高精度格式。

ffmpeg 空间	OBS-FOURCC	位深	平面布置 (数量)
yuv420p	I420 / YV12	8	planar (3)

nv12	NV12	8	semi-planar (2)
yuv420p10le	I010	10	planar (3)
p010le	P010	10	semi-planar (2)
yuv422p	Y42B	8	planar (3)
yuv444p	YV24	8	planar (3)

热键

开始/停止录制可以使用快捷键实现，看个人习惯，**但应注意测试按键是否冲突**，例如图中的 Ctrl+Shift+X 尽管非常少见，但在浏览器的网址框中其实是切换左~右对齐的按键，而其它专业软件，甚至输入法中可能也存在冲突。



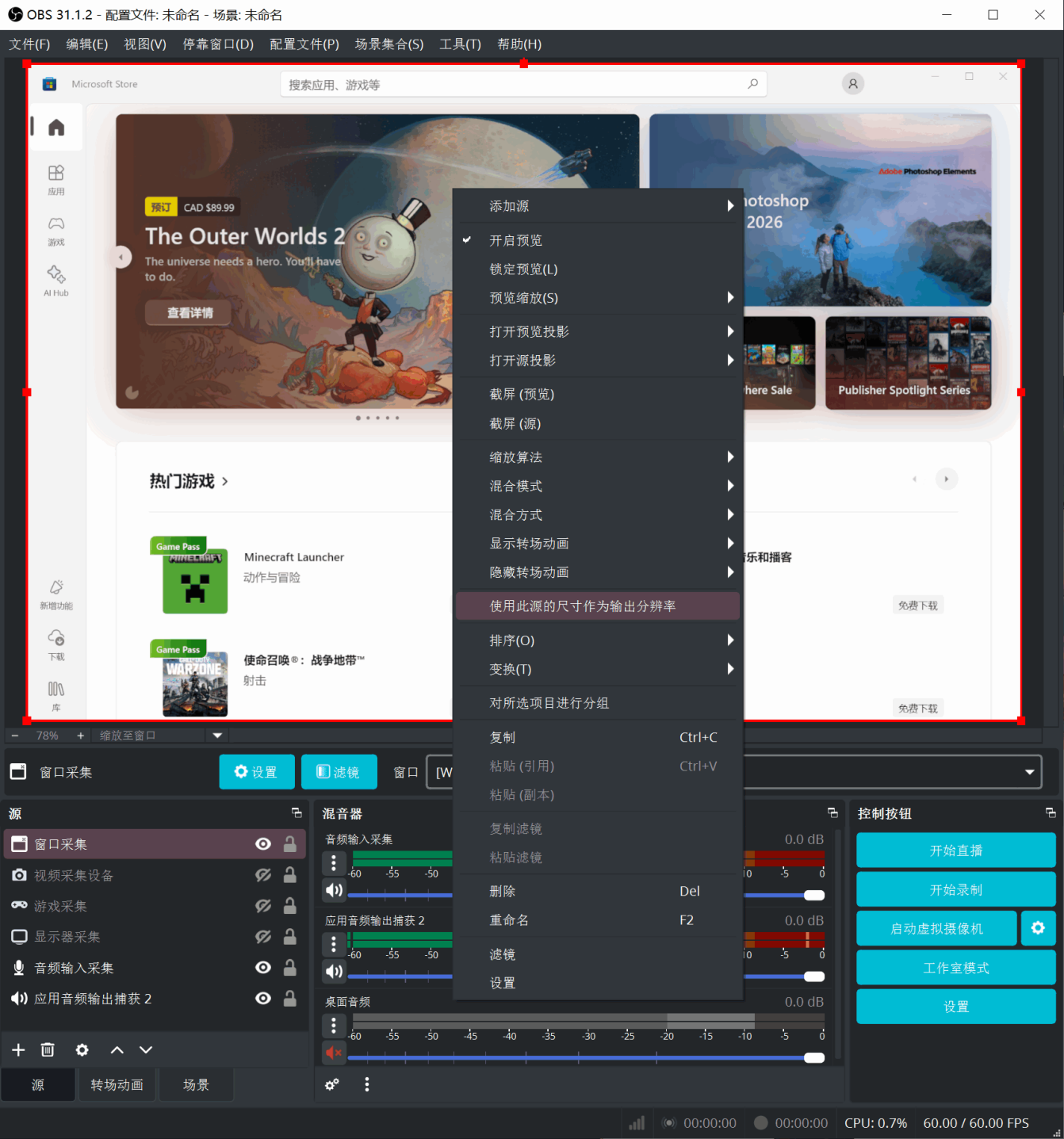
视频

帧率

设置为 60，如果背景移动，则建议 90 或 120，除非录制前已经测量过内容（包括鼠标光标移动）的低帧率可行性。尽管帧率变高会增加文件体积，但录制步骤是先于剪辑和压制的，因此必须预留足够多的信息余量给后续步骤（包括上传到平台后的转码），以避免画质与流畅度受到二次损失。

分辨率

窗口录制：主页中新建“窗口采集”并选中需录制的窗口，画面中右键 → 使用此源的输出作为分辨率。



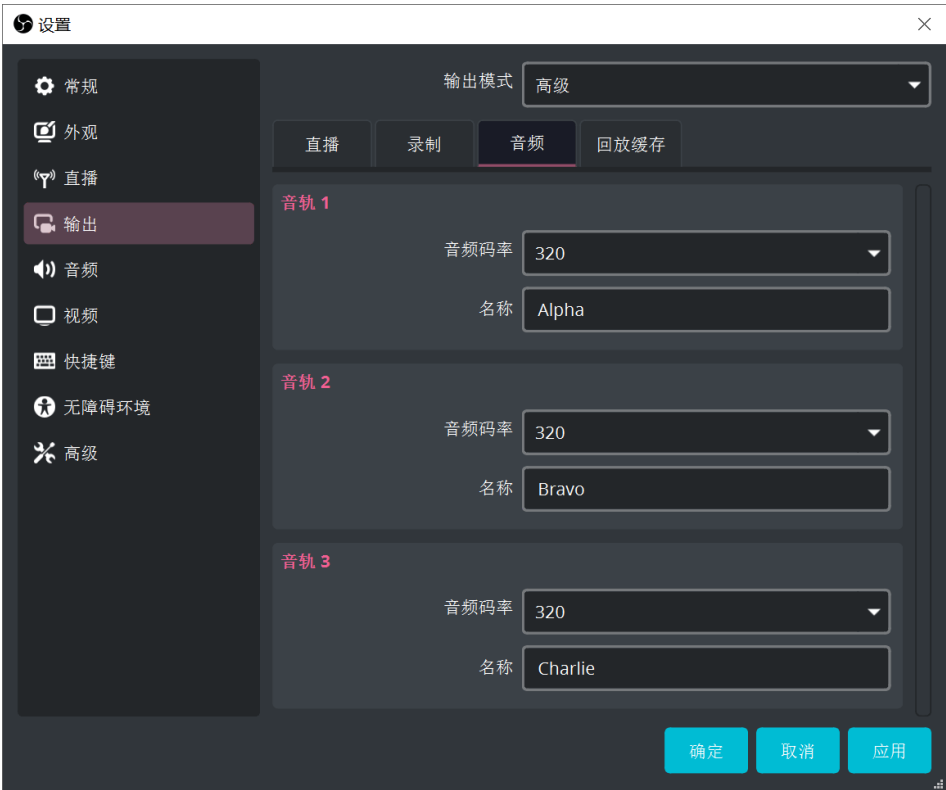
全屏录制：根据是否要录制桌面底部的任务栏来决定视频高，如分辨率为 1920x1080 时，避开任务栏使用 1920x1040；3840x2160 则使用 3840x2080。



音频

输出→音频→码率

建议选择 320kbps (不要低于 256kbps)



主页→源

建议安装 [win-capture-audio](#) 插件录制应用程序的音轨（降低延迟），原理是录制的对象从 Windows 音频输出改为应用程序进程，但可能会与部分程序不兼容。只要 win-capture-audio 能用，就不建议使用延迟更高的音频输出采集（添加源列表中）或桌面音频（默认开启）。开启多个音源时，注意这些音源采集的目标不同。

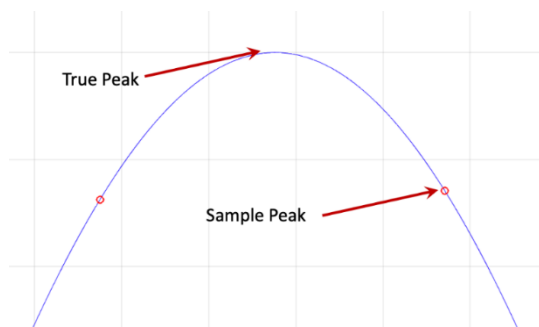
主页→混音器→音源 [:]→音频插件

不建议在录制时添加任何滤镜，而是之后使用编辑软件处理（如 Adobe Audition 的杂音降噪器滤镜、清理旁白电平效果组），从而避免录制结果作废，以及额外的 CPU 占用延迟。

音频→采样率

不高于声卡 DAC 默认设置（除非有问题，否则不要更改声卡默认），OBS 中最高 48kHz（48000），这个值代表了最高音高（采样率÷2，以避免混叠失真 aliasing），也就是齿音中最尖锐部分的信号、和“擦”这种乐器所能发出的音高上限。当然，44.1kHz（44100）也是完全足够的。

音频→电平表



涉及专业混音监听显示，但 OBS 只负责录制，分工上不应搀和混音操作，因此选择低占用（采样峰值）的电平表显示方法即可。真峰值 True Peak 的作用是通过超采样（二倍放大）得出每个采样点之间的真实音频响度，这在混音，尤其是多轨混音中起到避免响

度超标触发平台/标准惩罚机制的作用。然而，[OBS 具备多音轨导出能力](#)，因此多轨混音仍是应该对症下药地交给混音软件去处理（如 Adobe Audition、FL Studio 的多轨混音）。图片来源：[docs.dolby.io](#)

输出→录制→音频编码器

音频格式\封装格式

.flv .mkv .mp4 .mov 混合 MP4 分片 MP4 分片 MOV .ts .m3u8+.ts

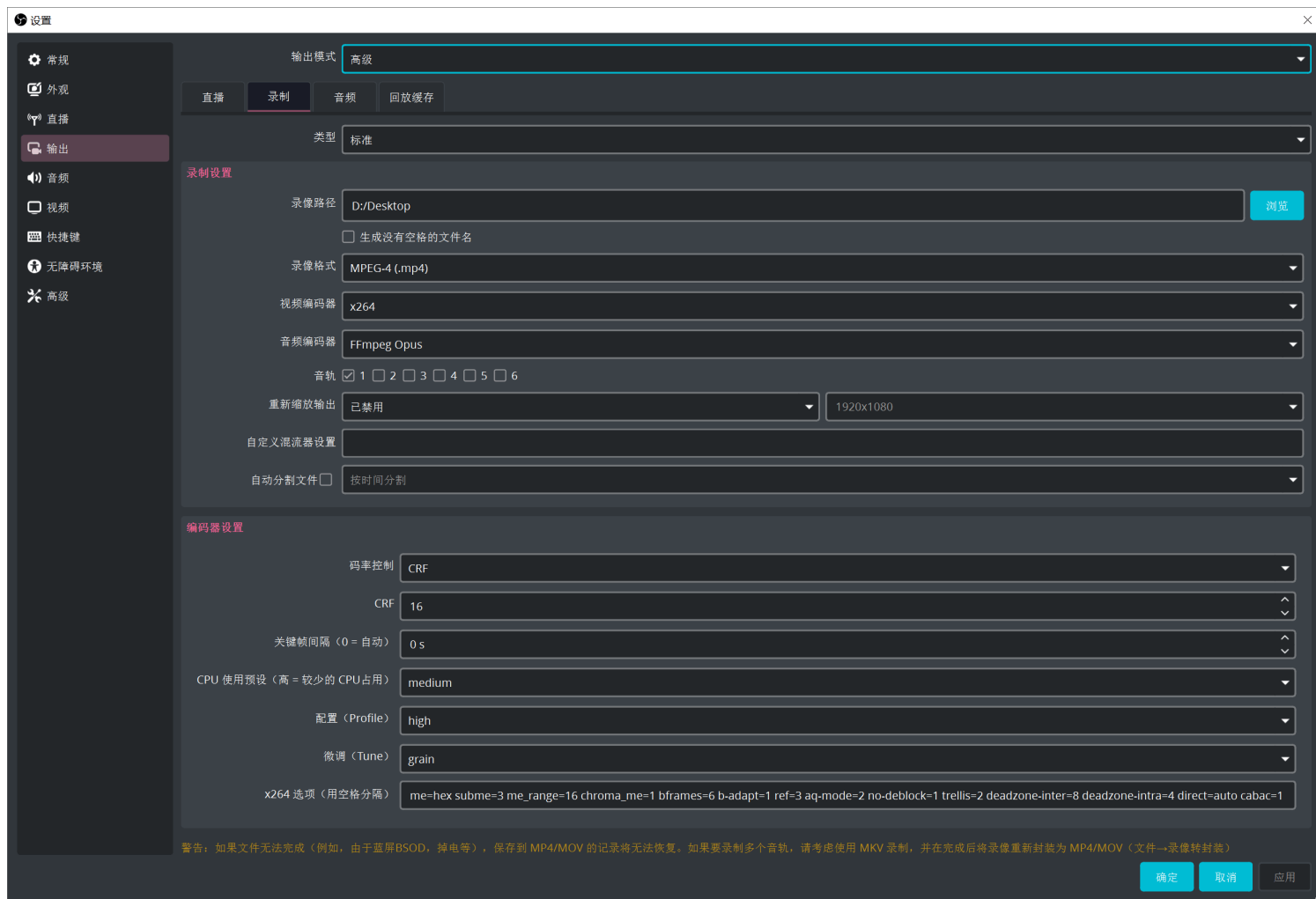
FFmpeg AAC 有损，音质最差，高兼容	✓	✓	✓	✓	⚠	⚠	⚠	✓	✓
FFmpeg ALAC 无损，24bit	✗	✓	✓	✓	⚠	⚠	⚠	✗	✗
FFmpeg FLAC 无损，16bit	✗	✓	⚠	⚠	⚠	✗	✗	✗	✗
FFmpeg OPUS 有损，音质好，兼容一般	✗	✓	✓	⚠	⚠	⚠	✗	✗	✗
FFmpeg PCM 无损，16/24/32bit	✗	✓	⚠	✓	⚠	⚠	⚠	✓	✓

根据视频封装格式、[剪辑软件](#)及[视频平台](#)的限制选择。如果既要高压缩，又要兼容工作流，可以考虑录制为 OPUS (.ogg)，之后用如 `ffmpeg -i "视频.mp4" -vn -c:a pcm_s16le "导出波形.wav"` 的命令无损转换。



编码

输出→录制→编码器设置



录屏 10fps 高压 - 音乐/助眠/棋牌/B-roll

码率控制:	CRF	关键帧间隔:	0 (自动)
CPU 使用预设:	slow	CRF:	20
Profile:	high	tune:	stillimage
设定-高级-色彩空间-色度采样:	i420		
参数:	me=hex subme=7 chroma_me=0 bframes=12 b-adapt=1 ref=2 aq-mode=0 aq-strength=0 deblock=0,-1 trellis=2 chroma_qp_offset=-4 cabac=1 psy=0 opencl=1 fgo=15		

- 由于画面变化极小也没有突发情况，所以用不到 VBR 模式（VBV 参数）

录屏 28fps 高压 - 聊天/演奏/小游戏/装机/发布会

码率控制：	VBR	关键帧间隔：	0（自动）
比特率：	9000kbps	CPU 使用预设：	medium

缓冲大小:	默认	Profile:	main
CRF:	19	tune:	film
设定-高级-色彩空间-色度采样:			i420

参数: me=umh subme=6 chroma_me=1 bframes=5 b-adapt=1 ref=2 aq-mode=1 aq-strength=0.7
deblock=0,0 trellis=2 direct=temporal opencl=1 fgo=15

- 画面有动态, 且以上情况里的背景一定是静态的, 所以动搜增强了很多
- 画面可能有突发情况, 所以用了 VBR 模式
- 增加了 IDR 帧时间来给码率让位

录屏 55fps 低压 - FPS-STG-赛车-3A 游戏/演唱会

码率控制:	VBR/ABR	关键帧间隔:	0 (自动)
比特率:	35000kbps	CPU 使用预设:	veryfast
缓冲大小:	3500	Profile:	无
CRF:	18	tune:	film

参数: me=hex subme=4 me_range=12 chroma_me=0 bframes=3 b-adapt=1 ref=3 aq-mode=2 aq-strength=0.9 deblock=0,0 trellis=0 deadzone-inter=8 deadzone-intra=5 direct=temporal cabac=0 opencl=1 nr=10 fgo=15

- deadzone-inter=8 deadzone-intra=5 这两个参数必须在 trellis<2 时使用, 否则画面会变得很脏
- 为进一步限制码率, 使用了降噪 nr 功能
- 本方案没有考虑游戏的突发高占用情况, 可能会在大量 AI, 大量效果等情况下造成卡顿; 虽然这是游戏本身就把 CPU 占满的情况

录屏 100FPS - x264 低压

码率控制:	CRF/ABR	关键帧间隔:	0 (自动)
CPU 使用预设:	fast	CRF/比特率:	18/9000 kbps (或直播平台/网速上限)
Profile:	无	tune:	无

参数: me=hex me_range=12 subme=3 chroma_me=0 bframes=3 b-adapt=1 ref=3 aq-mode=0 psy=0 mbtree=0 cabac=1 mixed_refs=0 deadzone-inter=8 deadzone-intra=5 deblock=0,0 trellis=0 direct=temporal ref=3 opencl=1 fgo=15

- 如果以上参数录制的视频仍然卡顿, CPU 占用高的话, 则逐渐降低关键帧间隔