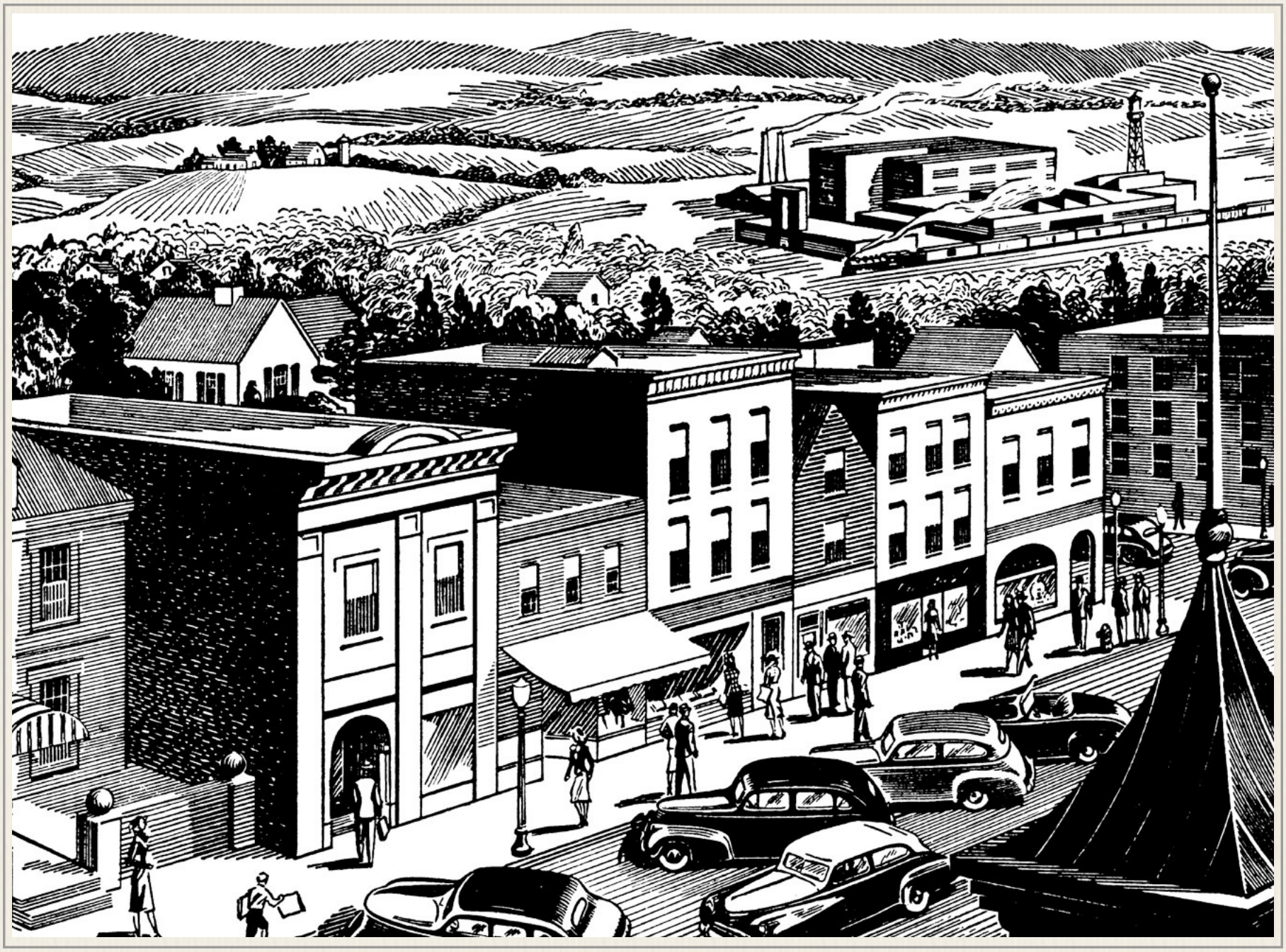
x265入门教程

张芳涛



CHAPTER 1

简介

“随着对高清晰度和超高清视频的需求不断增加，以及对视频点播的需求日益增长，导致对带宽和存储需求的需求呈指数增长。 新的高效视频编码（HEVC）标准（也称为H.265）可以满足这些挑战。 x265 HEVC编码器项目由MulticoreWare于2013年推出，旨在提供最高效，最高性能的HEVC视频编码器。

1.1：关于HEVC

高效视频编码（HEVC）是由ISO / IEC运动图像专家组（MPEG）和ITU-T 视频编码专家组（VCEG）通过其视频编码联合协作组（JCT-VC）开发。 HEVC也称为ISO / IEC 23008-2 MPEG-H第2部分和ITU-T H.265。 HEVC可以提供卓越的视频质量和高达比先前标准（H.264 / MPEG-4 AVC）提高两倍的数据压缩技术。 不仅如此，HEVC还可支持8K超高清视 频，最大支持高达8192x4320像素尺寸的图像。

1.2：关于X265

x265的主要目标是成为可在任何平台都可以使用的性能最佳的H.265 / HEVC编码器，在各种硬件平台上提供最高的压缩效率和最高性能。 x265编 码器现在是开源的，以GPLv2许可证发布。 X265还可以获得商业许可，使 商业公司能够在其解决方案中使用和分发x265，而不受GPL许可的限制。

x265由高性能软件解决方案的领导者MulticoreWare开发，该软件得到了包括像Telestream和Doremi Labs（以及其他不希望公开身份的公司）等领先视频技术提供商的大力支持，同时也的到了很多开源开发人员的大力贡献。 x265从x265利用x264 AVC编码器项目身上借鉴了很多许多出色的视频编码功能并且在此基础之上做了很多优化。

x265软件可通过https://bitbucket.org/multicoreware/x265 GNU GPL 2许可之后免费获得。对于希望分发x265但不希望接受GPL 2许可证的开源要求约束的商业公司，商业许可证具有竞争性条款。 请联系许可证@ x265.com以查询商业许可条款。

目前x265主要做为视频编码器软件库，但提供命令行可执行文件对于测试和开发来说是很方便的。 我们预计x265将在未来几个月内将会在许多先进的视频硬件和软件产品和服务方面得到广泛的应用。

1.3： 法律声明

x265软件属于MulticoreWare，Inc。并且版权归该公司所有。.MulticoreWare致力于在GNU GPL v2许可下提供x265软件。 不希望根据GPL许可条款将x265软件集成到其产品中的公司可以联系MulticoreWare（许可@ x265.com）以获得商业许可协议。 在GPL下使用x265的公司也可能希望与MulticoreWare合作， 或签订支持合同 ，以加快特定功能的开发或优化对特定硬件或软件平台的支持。

GNU GPL v2许可证或x265商业许可协议约束您访问受版权保护的x265软件源代码的权利，但不包括可能适用于从x265源代码创建的二进制可执行软件功能的任何专利。 您有责任了解您所在国家/地区的法律，并对使用或分发从x265源代码创建的软件应用程序所需的所有适用专利权进行许可。

x265是MulticoreWare，Inc的注册商标。x265徽标是MulticoreWare的商标，只能在明确书面许可的情况下使用。 版权所有。

CHAPTER 2

命令行选项

请注意，除非选项仅作为CLI列出，否则**x265_param_parse**（）也支持该选项。 CLI使用getopt来解析命令行选项，因此可以使用简称版本或全称版本，并且可以将全称选项截断为最短的明确缩写。 API的用户必须传递**x265_param_parse**（）完整的选项名称。

预设和调整具有特殊含义。 在调用**x265_param_parse**（）以设置任何其他字段之前，API用户必须使用他们希望使用的预设和调优参数调用**x265_param_default_preset**（）。 CLI隐式为用户执行此操作，因此所有CLI选项都在用户的预设和调整选项之后应用，而不管命令行上的参数顺序如何。

如果存在额外的命令行参数（既不是选项也不是选项值），CLI将把它视为输入文件名。 这有效地使 - 输入特定输入文件的输入特定。 如果有两个额外的参数，则第二个被视为输出比特流文件名，如果隐含输入文件名，则--output也是可选的。 这使得x265 in.y4m out.hevc成为有效的命令行。 如果有两个以上的额外参数，CLI将认为这是一个错误并中止。

可执行选项

命令行解释	
--help, -h	显示帮助文本
	CLI ONLY
--version,-v	显示版本的详细信息
	CLI ONLY

命令行可执行返回码	
0	编码成功
1	命令行无法解析
2	编码器打开失败
3	无法生成流的头信息
4	终止编码

记录/统计选项

--log-level <integer|string>

控制控制台上显示的信息级别。 调试级别启用每帧QP，度量标准和比特率日志记录。 完整级别启用散列和重量记录。 -1禁用所有日志记录，但某些致命错误除外，并且可以通过字符串“none”指定。

log-level	
0	error
1	warning
2	info(默认值)
3	debug
4	full

--no-progress

来自于CLI的定期进度报告是不允许使用的

CLI ONLY”

--csv <filename>

将编码统计信息写入逗号分隔值日志文件。 如果文件尚不存在，则创建文件。 如果--csv-log-level为0，则每次运行添加一行。 如果--csv-log-level大于0，则每帧写入一行。 默认是none

当--csv-log-level大于或等于1时，可以使用以下统计信息：

可用的统计信息	
Encode Order	编码器编码的顺序
Type	帧的切片类型
POC	图片顺序技术，帧的显示顺序
QP	量化参数决定帧
Bits	消耗帧的位数
Scenecut	如果帧是场景切换，则为1，否则为0
RateFactor	仅在启用CRF时适用。 速率因子取决于用户给出的CRF。 这用于确定QP以便以某种质量为目标。
BufferFill	可用于下一帧的位。 包括从当前帧携带的比特。

当--csv-log-level大于或等于2时，可以获得有关编码比特流和编码器性能的若干统计信息：

I/P cost ratio	
I/P cost ratio	当帧被确定为I帧时的成本与将其确定为P帧时的成本之间的比率，该P帧是从向前预测帧的四分之一分辨率计算的。 这与其他参数（例如GOP中帧的位置）的组合用于决定场景转换。

统计分析	
CU Statistics	CU模式所占的百分比
Distortion	平均亮度和色度失真。 计算为SSE在fenc和recon（量化后）完成
Psy Energy	平均psy能量计算为源能量和再能量之间的绝对差值之和。 能量通过sa8d减去SAD得到。
Residual Energy	平均剩余能量。 SSE是在fenc和pred（量化前）上计算的。
Luma/Chroma Values	每帧的最小值，最大值和平均值（按面积平均）源的亮度和色度值。
PU Statistics	每个深度值的PU模式百分比。

性能统计	
DecideWait ms	帧编码器必须等待的毫秒数，因为在给出新帧之前，API线程检索了前一帧。 这是slicetype决策（lookahead）引入的延迟
Row0Wait ms	自从帧编码器在允许第一行CTU开始压缩之前接收到要编码的帧以来的毫秒数。 这是参考帧引入的延迟，为了使重建和过滤行可用
Wall time ms	准备好压缩的第一个CTU与正在压缩的整个帧和输出NAL完成之间相隔的毫秒数
Ref Wait Wall ms	可用的第一个参考行与最后一个参考行可用之间相隔的毫秒数
Total CTU time ms	工作线程压缩和过滤此帧的CTU所花费的总时间（以毫秒为单位）
Stall Time ms	使用0工作线程花费的报告挂起时间的毫秒数，即所有压缩完全停止
Total frame time	编码帧所花费的总时间

--csv-log-level <integer>

控制-csv日志文件的详细程度（和大小）

csv-log-level	
0	摘要信息（默认）
1	帧级日志信息
2	具有性能统计信息的帧级别日志信息

--ssim, --no-ssim

计算并报告结构相似度值。 如果需要测量ssim的值，建议使用--tune ssim， 默认该选项是disabled。

--psnr, --no-psnr

计算并报告峰值信噪比。 如果需要测量PSNR的值，建议使用--tune psnr。 默认该选项是disabled。

性能选项

--asm <integer:false:string>, --no-asm

默认情况下，x265将使用所有检测到的CPU SIMD架构。 可以使用--no-asm禁用所有程序集，也可以指定要使用的逗号分隔的SIMD体系结构列表，匹配如下这些字符串：MMX2，SSE，SSE2，SSE3，SSSE3，SSE4，SSE4.1，SSE4.2， AVX，XOP，FMA4，AVX2，FMA3。

一些较高的体系结构意味着较低的体系结构存在，这是隐式处理的。

也可以直接提供CPU能力位图作为整数。

请注意，通过指定此选项，您将覆盖x265的CPU检测，并且可能会执行此操作。 您可以通过指定CPU不支持的SIMD体系结构来导致编码器崩溃。

默认值：自动检测SIMD架构

--frame-threads, -F <integer>

并发编码帧的数量。 使用单个帧线程可以略微改善压缩，因为整个参考帧始终可用于运动补偿，但它具有严重的性能影响。 默认值是基于CPU核心数以及是否启用WPP的自动检测计数。

帧线程的过度分配不会提高性能，通常只会增加内存使用量。

Values: 自动检测 ，取值范围为0到16之间，默认值为0。

--pools <string>, --numa-pools <string>

逗号分隔的每个NUMA节点的线程列表。 如果为“none”，则不会创建任何工作池，只能实现帧并行性。 如果为NULL或“”（默认值）x265将使用每个NUMA节点上的所有可用线程：

+'	是一个特殊值，表示在节点上检测到的所有核心
*'	是一个特殊值，表示在节点和所有剩余节点上检测到的所有核心
-'	是一个特殊值，表示节点上没有核心，与'0'相同

在4节点系统下的示例字符串：

""	default，unspecified，所有numa节点都用于线程池
" * "	和默认值一样
"none"	没有创建线程池，只能实现帧并行性
"-"	和"none"选项一样
"10"	分配一个线程池，在所有可用节点上最多使用10个核心
"-,+"	使用节点1上的所有核心分配一个线程池
"+,-,+"	分配一个线程池，仅使用节点0和2上的核心
"+,-,+,-"	分配一个线程池，仅使用节点0和2上的核心
"-, "	分配一个线程池，使用节点1,2和3上的所有核心
"8,8,8,8"	分配4个线程池，每一个线程池里面分配最多8个线程
"8,+,+,+"	分配两个池，第一个在节点0上有8个线程，第二个在节点1,2,3上有所有内核

仅当在具有-pools选项的相应位置中明确提到要在该NUMA节点上创建的线程数时，才启用专用于给定NUMA节点的线程池。 否则，所有线程都是从单个线程池中生成的。 线程总数将由分配给该池的已启用NUMA节点的线程数确定。 除非如上所述明确指定，否则工作线程将被赋予该池的所有已启用NUMA节点的可用性，并且可以在它们之间进行迁移。

在任何线程池具有超过64个线程的情况下，线程池可以分解为多个64个线程的线程池; 在32位计算机上，这个数字是32.所有线程池都被授予原始线程池具有相同功能的NUMA节点。 出于性能原因，只有当64位计算机的线程数超过32个，或者32位计算机的线程数超过16时，才会生成最后一个线程池。 如果系统中的线程总数不遵守此约束，那么我们可能产生的线程数少于经过证明性能更好的核心。

如果四个线程池功能： - wpp， - pmode， - pme和--lookahead-slices都被禁用，那么--pools将被忽略，并且不会创建任何线程池。

如果指定“none”，则隐式禁用所有四个线程池功能。

帧编码器分布在可用的线程池之间，编码器永远不会生成比--frame-threads更多的线程池。 这些线程池用于WPP以及分布式分析和运动搜索。

在Windows上，本机API提供了足够的功能来发现NUMA拓扑并强制执行libx265所需的线程功能（只要没有选择以XP或Vista为目标），但在POSIX系统上，它依赖于libnuma来实现此功能。 如果您的目标POSIX系统是单个套接字，那么在没有libnuma的情况下构建是一个非常合理的选择，因为它对运行时行为没有影响。 在多插槽系统上，没有libnuma的libx265 POSIX版本的工作效率会降低。 有关详细信息，请参阅第四章第一节的线程池相关介绍。

默认为“”，在所有可用的NUMA节点上创建一个线程池，每个检测到的硬件线程（逻辑CPU核心）分配一个线程。 如果线程总数超过ATOMIC操作可以处理的最大大小（32位编译为32位，64位编译为64位），则可能会产生多个线程池，但受上述性能限制的影响。

请注意，需要对字符串值进行转义或引用，以防止在许多平台上进行shell扩展

--wpp, --no-wpp

启用Wavefront并行处理。 一旦在其上方的行是编码处理中的至少两个CTU，编码器就可以开始对行进行编码。 这使得并行性增加3-5倍，压缩效率约为1％。

没有线程池时，将隐式禁用此功能。

默认值：Enabled

--pmode, --no-pmode

并行模式决策或分布式模式分析。 启用后，编码器将在多个工作线程中分配每个CU（合并，帧间，帧内）的分析工作。 仅在x265尚未使CPU内核饱和时才建议使用。 在RD处于3级和4级中，如果启用了-rect，则最有效。 在RD处于5级和6级的时候，通常总是有足够的工作来分配以保证开销，前提是CPU尚未处于饱和状态。

-pmode将在不降低压缩效率的情况下提高利用率。 实际上，由于这些模式都是并行测量的，因此某些早期实现不切实际，因此在启用时通常会获得稍微好一些的压缩（以不跳过不可能的模式为代价）。 尤其是在更快的预设时 ，绕过早期输出可能会导致pmode减慢编码速度。

没有线程池时，将隐式禁用此功能。

默认值： disabled

--pme, --no-pme

并行运动估计。 启用后，当两个以上的引用需要对给定CU进行运动搜索时，编码器将在多个工作线程上分配运动估计。 仅在x265尚未使CPU内核饱和时才推荐使用。 --pmode比这个选项更有效，因为它分配的工作量要高得多。 使用-pme，能耗太高。

没有线程池时，将隐式禁用此功能。

-pme将提高许多核心系统的利用率，而不会影响输出比特流。

默认值： disabled

--preset, -p <integer|string>

将参数设置为预选值，将压缩效率与编码速度进行折衷。 这些参数在应用所有其他输入参数之前应用，因此可以覆盖这些值控制的任何参数。 有关详细信息，请参阅第五章第一节的预设选项相关内容。

preset, -p	
0	ultrafast
1	superfast
2	veryfast
3	faster
4	fast
5	medium (default)
6	slow
7	slower
8	veryslow
9	placebo

--tune, -t <string>

调整特定类型的源或情境的设置。 更改将在--preset之后但在所有其他参数之前应用。 默认无。有关详细信息，请参阅第五章第二节的tuning相关内容。

可取的值如下表：

psnr,
ssim
grain
zero-latency
fast-decode

--slices <integer>

将每个输入帧编码为可以独立解码的多个并行切片。 支持仅适用于覆盖图像整个宽度的矩形切片。

仅当帧并行和WPP无法在给定硬件上最大化利用率时，建议用于提高编码器性能。

默认值：每帧1个切片。 该功能目前还处于试验阶段。

--copy-pic, --no-copy-pic

允许编码器将输入x265图像复制到内部帧缓冲区。 禁用时，x265不会生成输入图片的内部副本，并且可以使用应用程序的缓冲区。 虽然这允许更深入的集成，但是应用程序的责任是（a）确保分

配的图片具有额外的填充空间，这将由库完成，并且（b）缓冲区在库具有之前不会被回收 完成对此帧的编码（可以通过跟踪由x265输出的NAL来确定）。

默认值：enabled

输入/输出文件选项

这些选项都描述了输入视频序列，或者，对于编码之前对序列执行的操作的情况。 处理文件的所有选项（名称，格式，偏移或帧计数）仅适用于CLI应用程序。

--input <filename>

输入文件名，仅支持原始YUV或Y4M。 对stdin使用单个破折号。 对于第一个“extra”命令行参数，将隐含此选项名称。

CLI ONLY

--y4m

无论文件扩展名如何，将输入流解析为YUV4MPEG2，主要用于stdin（即：--input - --y4m）。 如果输入文件名具有“.y4m”扩展名，则隐含此选项。

CLI ONLY

--input-depth <integer>

仅适用于YUV：输入文件或流的位深度。

值：取值范围是8到16之间。默认值为internal depth。

CLI ONLY

--frames <integer>

要编码的帧数。 它可能没有特别说明，但是当它被指定时，速率控制可以利用这些信息。 它还用于确定编码是否实际上是静止图像编码（单帧）

--dither

为编码器的内部bitdepth启用高质量缩减。 抖动基于从一行像素到图像中的下一行像素的误差的扩散。 仅在输入位深度大于8位时适用。

默认状态：disabled

CLI ONLY

--input-res <wxh>

仅限YUV：源图片尺寸大小[w x h]

CLI ONLY

--input-csp <integer|string>

色度子采样（仅限YUV）：目前只支持4：0：0（单色），4：2：0,4：2：2和4：4：4。 输入的色度子采样格式必须与所需的输出色度子采样格式匹配（libx265不会执行任何色度子采样转换），并且必须由指定的HEVC配置文件支持。

0	i400（4：0：0单色） - Main或Main10配置文件不支持
1	i420（默认值为4：2：0） - 所有HEVC配置文件均支持
2	i422（4：2：2） - Main，Main10和Main12配置文件不支持
3	i444（4：4：4） - 支持Main 4：4：4，Main 4：4：4 10，Main 4：4：4 12，Main 4：4：4 16Intra profiles
4	nv12
5	nv16

--fps <integer|float|numerator/denominator>

仅支持YUV：源帧速率。

值范围：正整数或浮点数，或num / denom。

--interlace <false|tff|bff>, --no-interlace

interlace	
0	渐进式图片（默认）
1	顶部区域优先
2	底部区域优先

HEVC将隔行内容编码为字段。 必须以正确的时间顺序将字段提供给编码器。 源尺寸必须是现场尺寸，FPS必须以每秒字段为单位。 解码器必须以正确的方向重新组合场以进行显示。

--seek <integer>

输入文件开头要跳过的帧数。 默认值为0

CLI ONLY

--frames, -f <integer>

要编码的输入序列的帧数。 默认为0（全部）

CLI ONLY

--output, -o <filename>

比特流输出文件名。 如果有两个额外的CLI选项，则第一个是隐式输入文件名，第二个是输出文件名，--output选项是可选的。

输出文件将始终包含原始HEVC比特流，CLI不支持任何容器文件格式。

CLI ONLY

--output-depth, -D 8|10|12

输出HEVC比特流的Bitdepth，也是编码器的内部比特深度。 如果请求的位深度不是链接的libx265的位深度，它将尝试绑定libx265_main用于8位编码器，libx265_main10用于10位编码器，或libx265_main12用于12位编码器，其API版本与链接的libx265相同。

如果未指定输出深度但指定了--profile，则输出深度将从配置文件名称中获得。

CLI ONLY

--chunk-start <integer>

块的第一帧。 将按照显示顺序在此之前的帧进行编码，但是，它们将在比特流中被丢弃。 只能在封闭的GOP结构中启用该功能。 默认值为0（禁用）。

--chunk-end <integer>

块的最后一帧。 显示顺序后面的帧将用于做出先行决定，但是，它们不会被编码。 只能在封闭的GOP结构中启用该功能。 默认值为0（禁用）。

Pro fi le，Level，Tier

--profile, -P <string>

强制执行特定配置文件的要求，确保输出流可由支持该配置文件的解码器解码。 如果为编码器选择的编译选项不能支持指定的配置文件，则可以中止编码（高位深度编码器将无法输出符合Main或MainStillPicture的比特流）。

x265支持以下配置文件：

8位profiles:

- main, main-intra, mainstillpicture (或者实用简称：msp)
- main444-8, main444-intra, main444-stillpicture

请参阅下面有关内部和静止图像信息的注释。

10位profiles:

- main10, main10-intra
- main422-10, main422-10-intra
- main444-10, main444-10-intra

12位profiles：

- main12, main12-intra
- main422-12, main422-12-intra
- main444-12, main444-12-intra

CLI ONLY

API用户在配置其param结构后必须调用x265_param_apply_pro文件（）。 在此调用之后对param结构所做的任何更改都可能使编码不符合要求。

如果未指定--output-depth，CLI应用程序将从配置文件名称中获取输出位深度。

--level-idc <integer|float>

最低解码器要求级别。 默认为0，表示编码器自动检测。 如果指定了一个值，编码器将尝试将编码规范置于该指定级别内。 如果编码器无法达到该级别，则会发出警告并中止编码。 如果请求的需求级别高于实际级别，则发信号通知实际需求级别。

请注意，指定解码器级别将迫使编码器启用VBV以进行恒定速率因子编码，这可能会引入非确定性。

值的类型应该为整数或者浮点数类型，级别是10，例如：当级别是5.1的时候，将会被指定为5.1或者51，级别5.0被指定为5.0或者50。

Annex A 级别：1,2,2.1,3,3.1,4,4.1,5,5.1,5.2,6,6.1,6.2,8.5。

--high-tier, --no-high-tier

如果指定了--level-idc，则-high-tier允许在该级别支持高级别。 编码器将首先尝试在指定级别进行编码，首先是主层，仅在必要时启用高层并且在该级别可用。如果您请求的级别不支持高级别，则不支持高级别。 如果指定了-no-high-tier，则编码器将尝试仅在主层进行编码。

默认值： enabled。

--ref <1..16>

允许的最大L0引用数。 此数字对运动搜索中执行的工作量具有线性乘数效应，但通常会对压缩和失真产生有利影响。

请注意，x265最多允许16个L0参考，但HEVC规格仅允许最多8个参考帧。 因此，如果启用了B帧，则只有7个L0 refs有效且如果启用了-b-pyramid（默认情况下在所有预设中都启用），则HEVC规范允许的最大值仅为6 L0 refs。 如果x265检测到总引用计数大于8，它将发出警告，表明结果流不合规，并且它将流发送信号为NONE和NONE，并将中止编码，除非它具体说明了--allow-non-conformance。 兼容的HEVC解码器可以拒绝解码这样的流。

默认值： 3。

--allow-non-conformance, --no-allow-non-conformance

允许libx265生成具有pro fi le和level NONE的比特流。 默认情况下，它将中止任何不符合严格级别合规性的编码。 造成不合规的两个最可能的原因是--ctu太小， - ref太高，或者比特率或分辨率超出规范。

默认值： disabled。

--uhd-bd

启用超高清蓝光格式支持。 如果指定了不兼容的编码选项，编码器将尝试修改/设置正确的编码规范。 如果编码器无法执行此操作，则此选项将关闭。 Highly experimental。

默认值： disabled。

注意：

--profile， - level-idc和--high-tier仅在您针对具有固定资源限制的特定解码器（或解码器）时使用，并且必须在这些限制内约束比特流。 指定配置文件或级别可能会降低编码质量参数以满足这些要求，但永远不会提高它们。 它可以在CRF编码上启用VBV约束。

另请注意，x265分三个步骤确定解码器要求的配置和级别。 首先，用户使用他们建议的编码器选项配置x265_param结构，然后可选地调用x265_param_apply_profile（）来强制执行特定的配置文件（main，main10等）。 其次，从此x265_param实例创建编码器，并使用--level-idc和--high-tier参数来降低比特率或其他功能，以便强制执行目标级别。 最后，编码器重新检查最终的参数集并检测实际的最小解码器需求级别，这是在比特流报头中发出的信号。 如果用户指定了高级别级别，则检测到的解码器级别将仅使用高级别。

信号发送的配置文件将由编码器的内部bitdepth和输入颜色空间决定。 如果--keyint为0或1，则将发信号通知配置文件的内部变体。

如果--total-frames为1，则将发信号通知静态图片变体，但此参数并非始终由应用程序设置，尤其是在CLI使用stdin流式传输或第三方应用程序使用libx265时。

决策/分析模式

--rd <1..6>

模式决策中的RDO级别。 值越高，分析越详尽，使用的失真优化越多。 值越低，编码越快，值越高，比特流越小（通常）。 默认取值是3。

请注意，下面的表格旨在提高准确性，但不一定是每种模式的最终目标行为。

级别	描述
0	sa8d模式和拆分决策，内部w /源像素，目前不支持
1	重新生成（更好的内部），RDO合并/跳过选择
2	RDO拆分和合并/跳过选择
3	RDO模式和拆分决策，用于sa8d的色度残差
4	目前和第3级别相同
5	添加RDO预测决策
6	目前和第5级别相同

取值范围：1~6

影响编码单元四叉树的选项，有时称为预测四叉树。

--ctu, -s <64|32|16>

最大CU尺寸（宽度和高度）。 最大CU尺寸越大，x265在图像区域上编码fl的效率就越高，从而大大降低了比特率。 然而，这导致并行性的损失，可以并行编码的CU行数较少，并且帧并行性也较少。 因为当CU大小32的时候可以更快的预设使用。

默认值：64。

--min-cu-size <32|16|8>

最小CU尺寸（宽度和高度）。 通过使用16或32，编码器将不会分析低于该最小阈值的CU的成本，从而在可预测的比特率增加的情况下节省大量计算。 此设置对更快预设的性能有很大影响。

默认值：8（HEVC最低8x8 CU，最佳压缩效率）

- 注意：单个过程中的所有编码器必须对CU大小范围使用相同的设置。 --ctu和--min-cu-size必须对所有这些都一致，因为编码器根据此范围配置了几个关键的全局数据结构。

--limit-refs <0|1|2|3>

当设置为X265_REF_LIMIT_DEPTH（1）时，x265将根据用于编码下一个深度处的4个子块的参考来限制在当前深度处分析的参考。 例如，16x16 CU仅使用用于编码其四个8x8 CU的引用。

当设置为X265_REF_LIMIT_CU（2）时，矩形和非对称分区将仅使用由2Nx2N运动搜索选择的参考（包括在最低深度，否则不受深度限制影响）。

当设置为3（X265_REF_LIMIT_DEPTH && X265_REF_LIMIT_CU）时，每个深度处的2Nx2N运动搜索将仅使用来自分割CU的参考值，并且在该深度处的矩形/放大器运动搜索将仅使用由2Nx2N选择的参考值。

对于limit-ref的所有非零值，当前深度将评估帧内模式（在片间），仅当帧内模式被选择为4个子块中的至少一个的最佳模式时。

如果启用这些标记中的一个或两个，通常可以增加正在使用的引用数（在解码器级别限制内）。

默认值：3。

--limit-modes, --no-limit-modes

启用后，限制模式将使用来自4个子CU的成本度量来限制针对每个CU分析的模式。 当启用多个帧间模式（如--rect和/或--amp）时，此功能将使用来自4个子CU的运动成本启发法来绕过不太可能是最佳选择的模式。 当以最小的压缩效率损失启用rect和/或--amp时，这可以显着提高性能。

--rect, --no-rect

启用矩形运动分区Nx2N和2NxN（50/50分割，两个方向）的分析。 默认禁用。

--amp, --no-amp

启用非对称运动分区的分析（75/25分割，四个方向）。 在RD级别0到4，AMP分区仅在CU大小32x32及以下时考虑。 在RD级别5和6，它将仅考虑AMP分区作为64x64处的合并候选（无运动搜索），以及作为64x64以下的合并或帧间候选。

搜索到的AMP分区是从当前最佳的分区间派生的。 如果Nx2N（垂直矩形）是最佳当前预测，则将评估左和右不对称分裂。 如果2NxN（水平矩形）是最佳当前预测，则将评估顶部和底部非对称分裂，如果2Nx2N是最佳预测，并且该块不是合并/跳过，则评估所有四个AMP分区。

如果禁用矩形分区，则此设置无效。 默认禁用。

--early-skip, --no-early-skip

首先测量2Nx2N合并候选者; 如果没有发现残差，则不分析该深度处的附加模式。 默认禁用。

--rskip, --no-rskip

此选项确定提前退出CU深度递归。 当找到跳过CU时，使用附加的启发式（取决于rd级）来决定是否终止递归。 在rdlevels 5和6中，使用与inter2Nx2N的比较，而在rdlevels 4和邻居成本用于跳过递归。 启用时，在性能提升时提供最低质量降级。

默认值：启用，使用--tune grain方法可以禁用。

--fast-intra, --no-fast-intra

对每个第五个角度内模式执行初始扫描，然后检查距离最佳模式+/- 2距离的模式，然后距离最佳模式+/- 1距离，有效地执行梯度下降。 启用后，将检查总共10种模式。 禁用时，将检查所有33种角度模式。 仅适用于--rd level 4及以下（中等预设和更快）。

--b-intra, --no-b-intra

允许评估B切片中的帧内模式。 默认禁用。

--cu-lossless, --no-cu-lossless

对于每个CU，评估最佳非无损模式选项的无损（变换和量子旁路）编码作为潜在的速率失真优化。 如果已指定全局选项--lossless，则无论是否启用此选项，所有CU都将无条件地编码为无损。 默认禁用。

仅在RD3级及以上有效，执行RDO模式决策。

--tskip-fast, --no-tskip-fast

仅评估NxN帧内预测的变换跳过（4x4块）。 仅适用于启用变换跳过的情况。 对于色度，仅评估luma使用tskip。 Inter block tskip分析未经修改。 默认禁用。

--rd-refine, --no-rd-refine

对于每个分析的CU，在一系列QP值的最佳分区模式下计算R-D成本，以找到最佳舍入效应。 默认禁用。

仅在RD级别是5和6的时候有效。

分析重用选项，以便在多次编码相同序列时提高性能（可能是在不同的比特率情况下）。 如果切片类型参数不匹配，编码器将不重用分析。

--analysis-save <filename>

编码器输出每帧的分析信息。 保存模式下的分析数据将写入指定的文件中。 需要cutree，pmode关闭。 默认禁用。

--analysis-save <filename>

编码器重用来自指定文件的分析信息。 通过读取由相同序列的早期编码所写的分析数据，可以避免大量的冗余工作。 需要cutree，pmode关闭。 默认禁用。

存储/重用的分析数据量由--analysis-reuse-level确定。

--analysis-reuse-file <filename>

为multi-pass-opt-analysis和multi-pass-opt-distortion指定文件名。 使用x265_analysis.dat。

--analysis-reuse-level <1..10>

在--analysis-reuse-mode中存储/重用的信息量分布在各个级别。 值越高，存储/重用的信息越高，编码越快。 默认5。

请注意，分析 - 重用级别必须与分析 - 重用 - 模式配对。

级别	描述
1	Lookahead信息
2，3，4	Level 1 + intra/inter modes, ref’s
5，6	Level 2 + rect-amp
7	Level 5 + AVC size CU refinement
8，9	Level 5 + AVC size Full CU analysis-info
10	Level 5 + Full CU analysis-info

--refine-mv-type <string>

重用通过API调用接收的MV信息。 当前接收AVC大小的信息，并且接受的字符串输入是“avc”。默认为禁用。

--scale-factor

输入视频按比例缩小以用于分析保存模式的因子。 此选项应与analyze- reuse-mode选项-analysis-reuse-level 10结合使用。ctu加载大小可以与save相同，也可以是save的两倍。 默认值为0。

--refine-intra <0..4>

在当前编码中启用帧内块的重建。

refine-intra	
0	从保存编码强制模式和深度。
1	当前块大小比min-cu-size大1时，评估当前深度（n）和深度（n + 1）处的所有帧内模式。 强制较大块的模式。

默认值是：0。

--refine-inter <0..3>

在当前编码中启用帧间块的重建。

refine-inter	
0	从保存编码强制模式和深度。
1	当前块大小比min-cu-size大1时，评估当前深度（n）和深度（n + 1）处的所有帧间模式。 强制使用较大块的模式。
2	除了级别1的功能外，当通过保存编码将特定模式确定为最佳模式时，限制评估的模式。 保存编码中的2nx2n - 禁用对rect和amp的重新评估。 跳过保存编码 - 仅重新计算跳过，合并和2nx2n模式。
3	在重用保存编码的深度的同时执行帧间模式的分析。

默认值是： 0。

--dynamic-refine, --no-dynamic-refine

根据内容和编码器设置动态切换-refine-inter level 0-3。 建议使用--refine-intra 4和动态改进。默认是禁用状态。

--refine-mv

为缩放视频启用运动矢量的重建。 通过搜索周围的八个整数和子面像素来评估最佳运动矢量

位置

影响变换单元四叉树的选项，有时称为残差四叉树（RQT）。

--rdoq-level <0|1|2>, --no-rdoq-level

指定在量化中使用的帧率 - 失真分析量：

在0级时，帧率失真成本不考虑在量子中。

在1级帧率 - 失真成本用于找到每个级别的最佳舍入值（并允许psy-rdoq有效）。 它从预定量系数中消除了系数的信号成本与其反后量子失真之间的关系。 启用--psy-rdoq时，此公式偏向于残差中更多的能量（更大的系数绝对水平）。

在级别2，帧率失真成本用于对每个4x4编码组进行抽取决策，包括在组位图内发信号通知组的成本。 如果不发信号通知整个编码组的总失真小于速率成本，则该块被抽取。 接下来，它将速率 - 失真

成本分析应用于最后的非零系数，这可能导致许多（或所有）编码组被抽取。 当RDOQ处于2级时，Psy-rdoq在保存能量方面效果较差，因为它只会影响水平失真成本。

--tu-intra-depth <1..4>

变换单元（残差）四叉树以与编码单元四叉树相同的深度开始，但是如果编码单元树提高了压缩效率，则编码器可以决定进一步分割变换单元树。 此设置限制了可以针对帧内编码单元尝试的额外递归深度的数量。 默认值：1，表示残差四叉树始终与编码单位四叉树处于相同深度。

注意，当CU帧内预测是NxN（仅可能具有8x8 CU）时，暗示TU分裂，因此残余四叉树以4x4开始并且不能分割任何后者。

--tu-inter-depth <1..4>

变换单元（残差）四叉树以与编码单元四叉树相同的深度开始，但是如果编码单元树提高了压缩效率，则编码器可以决定进一步分割变换单元树。 此设置限制了可以针对帧间编码单元尝试的额外递归深度的数量。 默认值：1。这意味着残差四叉树总是与编码单元四叉树处于相同的深度，除非CU用矩形或AMP分区编码，在这种情况下暗示TU分裂，因此残差四叉树 从CU四叉树下面开始一层。

--limit-tu <0..4>

对于帧间编码块，允许提前退出TU深度递归。

limit-tu	
1	根据全尺寸TU和分割TU的成本比较，决定递归到下一个更高的深度。
2	基于第一次拆分subTU的深度，限制其他拆分子TX的递归。
3	基于共址的平均深度和相邻CU的TU深度，限制当前CU的递归。
4	使用相邻/共同定位的CU TU深度的深度来限制第一subTU深度。 第一个subTU深度作为其他子组的限制深度。

启用级别3或4可能会导致-–analysis-save选项和 option:–analysis-load选项之间的输出比特流不匹配，因为所有相邻的CU TU深度可能在选项中不可用：–analysis-load仅作为可用信息模式的最佳选项运行。

默认值： 0。

--nr-intra <integer>, --nr-inter <integer>

降噪 - 在量化之前应用DCT（从DCT系数中减去）后应用的自适应盲区。 它没有像素级滤波，不跨越DCT块边界，没有重叠，强度值参数越高，降低噪声能力越强。

启用降噪将使输出在不同数量的帧线程之间发生分歧。 输出将是确定性的，但-F2的输出将不再匹配-F3等的输出。

取值范围：0～2000，默认值0（禁用）。

--tskip, --no-tskip

启用对4x4 TU编码块的变换跳过（旁路DCT但仍使用量化）编码的评估。

仅在RD级别3及以上有效，执行RDO模式决策。 默认禁用。

--rdpenalty <0..2>

当设置为1时，与较小变换单元相比，在P或B切片中的帧内编码CU中，大小为32×32的变换单元被给予4x比特成本代价。

设置为2时，除非最大递归深度另有要求，否则甚至不会尝试大小为32x32的变换单元。 要使此选项对32x32内部CU有效，-tu-intra-depth必须至少为2.为使其对64x64内部CU有效，-tu-intra-depth必须至少为3。

注意，在HEVC中，帧内变换单元（残余四叉树的块）也是预测单元，意味着针对每个TU块生成帧内预测信号，减去残差然后编码。 编码单元简单地提供将在预测CU内的所有变换单元时使用的预测模式。 这意味着当您阻止32x32帧内变换单元时，您将阻止32x32帧内预测。

默认值：0（禁用）。

取值范围：0：禁用，1：4x cost penalty，2：强制拆分。

--max-tu-size <32|16|8|4>

最大TU尺寸（宽度和高度）。 当最大TU尺寸较大时，可以通过DCT变换更有效地压缩残差，但是以更多计算为代价。 变换单元四叉树在编码树单元的相同深度处开始，但是如果最大TU尺寸小于CU尺寸，则变换QT开始于max-tu-size的深度。 默认值：32。

--dynamic-rd <0..4>

由于VBV速率控制实施而导致质量下降的点处的RD级别增加。 根据强度确定重新配置RD的CU的数量。 强度为1时呈现最佳FPS，强度为4时呈现最佳SSIM。 强度为0时关闭该功能。 默认值：0。

对RD为4级及以下有效。

--ssim-rd, --no-ssim-rd

启用/禁用SSIM RDO。 与MSE相比，SSIM是一种更好的感知质量评估方法。 基于SSIM的RDO计算基于残差分裂归一化方案。 该归一化与人类视觉系统的亮度和对比度掩蔽效果一致。 它用于在CTU分析期间进行模式选择，并且可以在客观质量度量SSIM和PSNR方面获得显着增益。 它仅对使用基于RDO的模式决策（--rd 3及更高版本）的预设有影响。

时间/动作搜索选项

--max-merge <1..5>

编码器可以考虑用于合并运动预测的最大相邻（空间和时间）候选块的数量。 如果合并候选块没有残差，则立即将其选为“skip”。 否则，在搜索最低成本的inter选项时，将合并候选块作为运动估计的一部分进行测试。 最大候选编号在SPS中编码并确定信令合并CU的比特成本。 默认值是2。

--me <integer|string>

动作搜索方法。 通常情况下是数字越大，ME方法将尝试找到最佳匹配的难度越大。 Diamond搜索是最简单的。 Hexagon搜索更好一点。 Uneven Multi-Hexegon是x264用于较慢预设的搜索方法的改编。 Star是从HM编码器改编的三步搜索：星型搜索，然后是可选的基数扫描，然后是可选的星形搜索改进。 Full是一个详尽的搜索; 比所有其他搜索速度慢一个数量级，但不比umh或star好多少。 SEA类似于全搜索; 从x264采用的三步运动搜索：DC计算，然后是ADS计算，然后是通过的运动矢量候选的SAD，因此比全搜索更快。

me	
0	dia
1	hex (default)
2	umh
3	star
4	sea
5	full

--subme, -m <0..7>

要执行的subpel改进量。 数字越大，执行的子主题迭代和步骤越多。 默认是2。

-m	HPEL iters	HPEL dirs	QPEL iters	QPEL dirs	HPEL SATD
0	1	4	0	4	FALSE
1	1	4	1	4	FALSE
2	1	4	1	4	TRUE
3	2	4	1	4	TRUE
4	2	4	2	4	TRUE
5	1	8	1	8	TRUE
6	2	8	1	8	TRUE

在-subme值大于2时，色度残差成本包含在所有子复制步骤中，色度残差包含在所有运动估计决策中（选择每个列表中的最佳参考图像，并在合并，单向运动和双向运动之间选择） 定向运动）。“slow”预设是第一个预设，可以使用色度残差。

--merange <integer>

动作搜索范围。 默认是57。

默认值来自默认CTU大小（64）减去亮度内插半长度（4）减去最大子位置距离（2）减去一个额外的像素值，以防使用十六进制搜索方法。 如果搜索范围大于此值，则参考帧将需要另一个CTU行延迟。

取值范围： 0~32768之间的整数。

--temporal-mvp, --no-temporal-mvp

在P和B切片中启用时间运动矢量预测值。 这使得能够使用来自前一帧中的并置块的运动矢量作为预测器。 默认值已启用

--weightp, -w, --no-weightp

在P切片中启用加权预测。 这使得前瞻中的加权分析能够影响切片决策，并且能够在主编码器中进行加权分析，其允许P参考样本在将它们用于运动补偿之前应用权重函数。 在具有照明变化的视频中，它可以大大提高压缩效率。

默认值：enabled。

--weightb, --no-weightb

在B切片中启用加权预测。

默认值： disabled。

--analyze-src-pics, --no-analyze-src-pics

启用源帧像素的运动估计，在此模式下，可以独立计算运动估计。

默认值：disabled。

空间/内部选项

--strong-intra-smoothing, --no-strong-intra-smoothing

为32x32内部块启用强内部平滑。 这种方法执行角落参考样本的双线性插值，以获得强大的平滑效果。 目的是防止在AC系数很小/零的区域中出现阻塞或带状伪影。

默认值： enabled。

--constrained-intra, --no-constrained-intra

约束帧内预测。 当为帧间片段中的块生成帧内预测时，仅使用帧内编码的参考像素。 帧间编码的参考像素被帧内编码的相邻像素或默认值替换。 一般的想法是阻止可能由有损信号引起的参考误差的传播。

默认值： disabled。

心理视觉选择

由编码器根据自己的设备进行模式决策，基于简单的速率失真公式，对比特率进行交易失真。 除了测量这种失真的方式之外，这通常是有效的。 它往往倾向于在具有错误运动的块上模糊重建块。 人眼通常喜欢模糊的错误运动，因此x265为速率失真算法提供心理视觉调整。

--psy-rd将为重建的块添加额外的成本，这些块与源块的视觉能量不匹配。 -psy-rd的强度越高，它就越有利于相似的能量而不是模糊，它会更加强烈地忽略速率失真。 如果它太高，它将引入视觉伪像并增加足够的比特率以进行速率控制，从而全局增加量化，降低整体质量。 psy-rd将倾向于减少模糊预测模式的使用，如DC和平面帧内和双向帧间预测。

--psy-rdoq将调整速率失真优化量化（RDO量子）中使用的失真成本，由-rdoq-level 1或2启用，有利于保留重建图像中的能量。 --psy-rdoq可防止RDOQ模糊psy-rd必须选择的所有编码选项。 在低强度水平下，psy-rdoq将影响量化水平决策，有利于重建图像中更高的AC能量。 随着psy-rdoq强度的增加，增加了更多的非零系数水平，并且RDOQ的速率失真分析将更少的系数归零。 高水平的psy-rdoq会使比特率加倍，这会对速率控制产生严重影响，从而导致更高的整体QP，并且可能导致振铃伪像。 psy-rdoq不如psy-rd准确，它一般偏向于能量，而psy-rd偏向于源图像的能量。 但非常大的psy-rdoq值有时可能是有益的。

作为一般规则，当两个心理视觉特征都被禁用时，编码器将倾向于在难以运动的区域中模糊块。 打开少量的psy-rd和psy-rdoq可以改善感知的视觉质量。 增加心理视觉强度将进一步提高质量并开始引入伪影并增加比特率，这可能迫使速率控制增加全球QP。 找到给定视频的最佳心理视觉参数需要实验。 我们推荐的默认值（两者均为1.0）通常位于频谱的低端。

比特率越低，最佳心理视觉设置越低。 如果心理视觉设置的比特率太低，将开始看到时间瑕疵（运动抖动）。 当编码器被迫在困难运动区域中编码跳过块（无残留）时会引起这种情况，因为它是心理视觉上的最佳选择（它们具有大量能量且没有剩余成本）。 当颤动发生时，可以降低心理设置，并允许编码器在这些高运动区域使用一些模糊。

--psy-rd <float>

在影响率失真的情况下，优化模式决定以牺牲压缩效率为代价来保持编码图像中的源图像的能量。 它仅对使用基于RDO的模式决策（--rd 3及更高版本）的预设有影响。 1.0是典型值。

默认值：2.0。

取值范围： 0~5.0。

--psy-rdoq <float>

在影响率失真的情况下，通过在重建图像中支持更高的能量来优化量化。 这通常以较低质量度量分数为代价改善感知视觉质量。 它仅在--rdoq-level为1或2时有效。高值可以保留高频细节。

默认值：0.0（1.0为预设慢，慢，veryslow）。

取值范围： 0~50.0。

切片决策选项

--open-gop, --no-open-gop

启用开放GOP，允许I-slice为非IDR。

默认值：enabled。

--keyint, -I <integer>

帧内最大帧内周期。 可以使用参数-1触发fi -ite-gop（流的开头的单个关键帧）的特殊情况。 使用1强制全内部。 启用内部刷新时，它会指定刷新扫描发生的间隔。

默认值：250。

--min-keyint, -i <integer>

最低GOP大小。 超出此间隔的场景切换编码为IDR并开始新的关键帧，而更靠近的场景切换编码为I或P.对于固定的关键帧间隔，将值设置为等于keyint。

取值范围：大于等于0（0：自动）

--scenecut <integer>, --no-scenecut

I帧的密度。 阈值越高，I帧放置越频繁。 --scenecut 0或--no-scenecut禁用自适应I帧放置。

默认值： 40。

--scenecut-bias <0..100.0>

该值表示场景检测中使用的帧的帧间成本和帧内成本之间的百分比差异。 例如，值5表示，如果帧的帧间成本大于或等于帧的帧内成本的95％，则将该帧检测为场景切换。 建议值介于5和15之间。

默认值：5。

--radl <integer>

允许在IDR前面的RADL图片数量。 需要固定的关键帧间隔。 建议值是2-3。 默认值为0（禁用）。

取值范围： 0～–bframes。

--ctu-info <0, 1, 2, 4, 6>

该值可以异步接收CTU信息，并确定对CTU信息的反应。 默认值0。

1：如果存在CTU信息，则强制分区。

2：（1）的功能，如果CTU信息已经改变，则减少qp。

4：CTU信息改变时（1）和强制帧间模式的功能，否则合并/跳过。 仅当计划调用API函数x265_encoder_ctu_info以异步复制ctu-info时，才应启用此选项。 如果在未调用API函数的情况下启用，编码器将无限期地等待。

--intra-refresh

启用定期帧内刷新（PIR）而不是关键帧插入。 PIR可以通过在非关键帧中插入一列帧内块来替换关键帧，这些帧通过视频从一侧移动到另一侧，从而刷新图像，但是在一段多帧而不是一个关键帧上。

--rc-lookahead <integer>

切片类型决策前瞻的帧数（编码器延迟的关键决定因素）。 前瞻缓冲区越长，场景切割决策就越准确，cuTree在提高自适应量化方面的效果就越好。 预测大于最大关键帧间隔是没有用的。

默认值：20。

取值范围：最大连续bframe计数（--bframes）～ 250。

--gop-lookahead <integer>

GOP边界决策前瞻的帧数。 如果在-keyint设置的gop边界内找到一个场景切割帧，则GOP将被延伸到这一点，否则GOP将被-keyint设置终止。

默认值：0。

取值范围： 0 ～ （-rc-lookahead - mini-GOP长度）。

建议-gop-lookahaed小于-min-keyint，因为-min-keyint以外的场景切换已被编码为关键帧。

--lookahead-slices <0..16>

使用多个工作线程来测量前瞻中每个帧的估计成本。 帧被分成指定数量的切片，每个切片启动一个线程。 当-b-adapt为2时，大多数帧成本估算将以批处理模式执行（同时进行多次成本估算），并且对于批量估算忽略前瞻切片; 它仍可用于单一成本估算。 该参数越高，帧成本就越不准确（因为上下文在切片边界上丢失），这将导致不准确的B帧和场景切换决策。 对性能的影响可能非常显着，特别是在具有许多线程的系统上。

编码器可以在内部降低切片数量或禁用。

切片以确保每个切片编码至少10个16x16行的低位块，以最小化对质量的影响。 例如，对于720p和1080p视频，切片数量的上限分别为4和6。 对于小于720p的分辨率，切片会自动禁用。

如果在前瞻中使用切片，则它们将作为lslices记录在工具列表中。

Values: 0 - 不可用。 1 和0选项一样。最大是16。

默认值：8 for ultrafast, superfast, faster, fast, medium 4 for slow, slower disabled for veryslow, slower。

--lookahead-threads <integer>

使用专用于执行前瞻的多个工作线程，而不是使用帧编码器共享工作线程。 使用指定数量的工作线程创建专用的lookahead线程池。 这可以是0到可用于编码的硬件线程的一半。 使用过多的线程进行lookahead可能会使帧编码器的资源匮乏并且可能会损害性能。

默认值为：0 - disabled, Lookahead。

与其他帧编码器共享工作线程。

values:0 - 禁用（默认）。 Max - 可用硬件线程的一半。

--b-adapt <integer>

设置确定B帧放置的工作量。

使用b-adapt 0，GOP结构基于--keyint和--bframes的值进行固定。

使用b-adapt 1，使用光前瞻选择B帧放置。

利用b-adapt 2（trellis），执行维特比B路径选择

Values: 0:none; 1:fast; 2:full(trellis) 默认值。

--bframes, -b <0..16>

最大连续b帧数。 使用--bframes 0强制所有P / I低延迟编码。 默认值是4.该参数对分配的内存量和--b-adapt lookahead的full trellis版本执行的工作量具有二次效应。

--bframe-bias <integer>

在slicetype决策中B帧的倾向性。 偏差越高，x265使用B帧的可能性越大。 可以是-90到100之间的任何值，并且可以剪裁到该范围。

默认值：0。

--b-pyramid, --no-b-pyramid

尽可能使用B帧作为参考帧。

默认值：enabled。

--force-flush <integer>

强制编码器使用帧。 默认值为0。

Values：

0 - 只有在所有输入图像结束时才使用编码器。

1 - 即使输入未结束，也可以显示所有帧。

slicetype决定可能会随此选项而改变。

2 - 只使用slicetype决定帧。

质量，速率控制和速率失真选项

--bitrate <integer>

启用单次ABR速率控制。 以kbps为单位指定目标比特率。 默认值为0（CRF）。

取值范围：大于0的整数。

--crf <0..51.0>

质量控制的可变比特率。 CRF是默认的速率控制方法; 它不会尝试达到任何特定的比特率目标，而是尝试实现给定的统一质量，并且比特流的大小由源视频的复杂性决定。 速率因子越高，量化越高，质量越低。 默认比率因子是28.0。

--crf-max <0..51.0>

指定可以分配给任何给定帧的速率因子的上限（确保最大QP）。 当CRF与VBV结合使用时，这很危险，因为它可能导致缓冲器欠载。默认是禁用的。

--crf-min <0..51.0>

指定可分配给任何给定帧的速率因子的下限（确保最小压缩因子）。

--vbv-bufsize <integer>

指定VBV缓冲区的大小（kbits）。 在ABR模式下启用VBV。 在CRF模式下，还必须指定--vbv-maxrate。 默认值为0（禁用vbv）。

--vbv-maxrate <integer>

最大局部比特率（kbits / sec）。 仅在vbv-bufsize也为非零时使用。 在CRF模式下启用VBV都需要vbv-bufsize和vbv-maxrate。 默认值是0（禁用）。

请注意，当启用VBV（使用有效的--vbv-bufsize）时，将打开VBV紧急去噪。 当帧QP> QP_MAX_SPEC（51）时，这将在帧级开启积极的去噪，大大降低了比特率并允许速率控制为后续帧分配较低的QP。 视觉效果模糊，但删除了重要的阻塞/位移瑕疵。

--vbv-init <float>

初始缓冲区占用率 在解码器开始解码之前必须填满的解码缓冲区部分。 确定绝对最大帧大小。 可以指定为0到1之间的小数值，或者以kbits为单位。 换句话说，这两个选项对是等价的：

--vbv-bufsize 1000 --vbv-init 900
--vbv-bufsize 1000 --vbv-init 0.9

默认值：0.9。

取值范围：小数：0 - 1.0，或kbits：2 .. bufsize。

--vbv-end <float>

最终缓冲emptiness。 在将所有指定帧插入解码缓冲区后，解码缓冲区中必须可用的部分。 指定为介于0和1之间的小数值，或以千位为单位。 默认值0（禁用）。

这实现了对块并行编码的基本支持，其中每个段可以指定VBV缓冲器的开始和结束状态，以便在块被独立编码和拼接在一起时可以保持VBV兼容性。

--vbv-end-fr-adj <float>

必须调整qp的帧以实现最终的解码缓冲区emptiness。 指定为总帧数的一部分。 仅当已知总帧数时才支持分数> 0。 默认值为0。

--qp, -q <integer>

为恒定QP速率控制指定基本量化参数。 使用此选项可启用恒定QP速率控制。 指定的QP被分配给P切片。 使用param-> rc.ipFactor和param-> rc.pbFactor给I和B切片相对于P切片给出QP，除非指定QP 0，在这种情况下QP 0用于所有切片类型。 请注意，QP 0不会导致无损编码，它只会禁用量化。 默认是禁用的。

取值范围：0~51之间的整数。

--lossless, --no-lossless

通过绕过缩放，变换，量化和环路滤波器过程实现真正的无损编码。 这用于超高比特率，零质量损失。 重建的输出图像对输入图像是精确的。 隐式无损编码没有速率控制，所有速率控制选项都被忽略。 较慢的预设通常会实现更好的压缩效率（并产生更小的比特流）。 默认是禁用状态。

--aq-mode <0|1|2|3>

自适应量化操作模式。 基于源图像的复杂度分析来提高或降低每块量化。 块越复杂，使用的量化就越多。 这抵消了编码器在复杂区域上花费太多比特而在区域中花费不足的趋势。

0	禁用
1	启用AQ（默认）
2	AQ启用了自动差异

--aq-strength <float>

调整自适应量化偏移的强度。 将-aq-strength设置为0将禁用AQ。 在aq-模式2和3中，高aq强度将导致高QP偏移，导致达到的比特率差异很大。

默认值：1.0.

取值范围： 0.0~3.0。

--aq-motion, --no-aq-motion

根据每个块相对于帧运动的相对运动来调整AQ偏移。 块的相对运动越多，使用的量化越多。 默认禁用。 这是一个处于实验阶段的特性。

--qg-size <64|32|16|8>

为子CTU启用自适应量化。 该参数指定可以调整QP的最小CU大小: 量化集团规模。 允许的值范围是64,32,16,8，只要它在包含范围[maxCUSize，minCUSize]内。 默认值：与maxCUSize相同。

--cutree, --no-cutree

允许使用先行的低运动矢量场来确定每个块的重用量以调整自适应量化因子。 被重度用作后续帧的运动参考的CU块被给予较低的QP（更多位），而快速改变且未被参考的CU块被给予较少的位。 这倾向于改善视频背景中的细节，而在高运动区域中细节较少。 默认是开启状态。

--pass <integer>

启用多次通过率控制模式。 输入被多次编码，将每个通道的编码信息存储在统计文件中，连续通道从中调整每帧的qp以改善输出的质量。 默认是禁用状态。

1	First pass, creates stats file
2	Last pass, does not overwrite stats file
3	Nth pass, overwrites stats file

取值范围： 1，2，3。

--stats <filename>

指定多次通过统计文件的文件名称。 如果未指定，编码器将使用x265_2pass.log。

--slow-firstpass, --no-slow-firstpass

使用指定的确切设置启用第一次传递编码。 当设置在每次通过时匹配时，后续多次通过编码的质量更好（与第一次通过相比）。 默认是启用状态。

禁用慢速首次通过时，使用具有以下快速选项的turbo编码来提高性能：

- --fast-intra
- --no-rect
- --no-amp
- --early-skip
- --ref = 1
- --max-merge = 1
- --me = DIA
- --subme = MIN(2, --subme)
- --rd = MIN(2, --rd)

--multi-pass-opt-analysis, --no-multi-pass-opt-analysis

启用/禁用多通道分析功能以及多通道速率控制。 基于存储在第1遍中的信息，在后续通过中，分析数据被重新定义，并且还跳过冗余步骤。 在第1遍中，为每个CTU存储分析信息，如最佳CTU分区的最终CTU分区的运动矢量，深度，参考和预测模式。 启用“分析 - 保存/分析 - 加载”选项时，无法启用多通道分析改进，并且当同时启用时，两者都将被禁用。 此功能需要禁用'pmode / pme'，因此当同时启用时，pmode / pme将被禁用。

默认状态：禁用。

--multi-pass-opt-distortion, --no-multi-pass-opt-distortion

根据失真数据和多通道速率控制启用/禁用qp的多通道改进。 在第1遍中，存储了最佳CTU分区的失真。 具有高失真的CTU在通过2中对于低失真CTU获得较低（负）qp偏移，反之亦然。这有助于改善主观质量。 启用“分析 - 保存/分析 - 加载”选项时，无法启用qp的多通道修改，并且当同时启用时，两者都将被禁用。 'multi-pass-opt-distortion'需要禁用'pmode / pme'，因此当与它一起启用时，pmode / pme将被禁用。

默认值： 禁用。

--strict-cbr, --no-strict-cbr

在ABR模式下，允许更严格的条件来控制目标比特率的比特率偏差。 比特率依从性优先于质量。速率容差降低到50％。

默认值： 禁用。

此选项适用于要求最终平均比特率在目标的非常严格的限制范围内的用例; 防止过冲，同时保持比特率在目标设置的5％以内，特别是在短段编码中。 通常，编码器保持保守，等待直到在编码帧方面有足够的反馈来控制QP。 strict-cbr允许编码器在达到目标比特率时更加积极，即使对于短片段视频也是如此。

--cbqpoffs <integer>

通过速率控制选择的亮度QP偏移Cb色度QP。 这是在色度通道上花费更多或更少位的一般方法。 默认值为0。

取值范围：-12~12之间的整数。

--crqpoffs <integer>

通过速率控制选择的亮度QP偏移Cr色度QP。 这是在色度通道上花费更多或更少位的一般方法。 默认值为0。

取值范围：-12~12之间的整数。

--ipratio <float>

I和P切片之间的QP比率因子。 该比率用于所有速率控制模式。 某些--tune选项可能会更改默认值。 它通常不是手动指定的。

默认值：1.4。

--pbratio <float>

P和B切片之间的QP比率因子。 该比率用于所有速率控制模式。 某些--tune选项可能会更改默认值。 它通常不是手动指定的。

默认值：1.3。

--qcomp <float>

qComp设置量化器曲线压缩因子。 它基于残差的复杂性（通过先行测量）对帧量化器进行加权。 默认值为0.6。 将其增加到1将有效地生成CQP。

--qpstep <integer>

QP中的最大单次调整允许速率控制。 默认值是4。

--qpmin <integer>

设定允许速率控制的QP的硬下限。 默认值为0。

--qpmax <integer>

设定允许进行速率控制的QP的硬上限。 默认值是69。

--rc-grain, --no-rc-grain

为电影grain内容启用专门的速率控制算法。 该参数严格地最小化帧内和帧之间的QP波动并消除grain的脉冲。 默认禁用。 启用时间：选项：' - tune'grain应用。 强烈建议通过调整纹理功能使用此选项，其中使用参数选项的组合来提高视觉质量。

--const-vbv, --no-const-vbv

使VBV算法在运行期间保持一致。 默认禁用。 启用时间：选项：' - tune'grain应用。

--qblur <float>

暂时模糊量子。 默认值为0.5。

--cplxblur <float>

暂时模糊复杂性。 默认20。

--zones <zone0>/<zone1>/...

调整视频区域的比特率。 每个区域采用以下形式：

<start frame>,<end frame>,<option> where <option> is either q=<integer> (force QP) or b=<float> (bitrate multiplier).

如果区域重叠，则列表中较晚的位置优先。 默认值是：none。

量化选项

注意，速率 - 失真优化量化（RDOQ）在--rd 4,5和6处隐式启用，并在所有其他级别隐式禁用。

--signhide, --no-signhide

隐藏每个TU（rdo）一个coeff的符号位。 暗示最后的标志。 这需要分析所有系数以确定是否必须切换符号，然后确定哪一个可以以最少的失真切换。 默认是启用状态。

--qpfile <filename>

指定包含某些或所有帧的帧类型和QP的文本文件。 每行的格式是:

framenumber frametype QP

Frametype可以是[I，i，K，P，B，b]之一。 B是参考的B帧，b是未参考的B帧。 我是一个关键帧（随机访问点），而我是一个不是关键帧的I帧（引用没有被破坏）。 如果启用了closed_gop选项，则K表示我，否则为i。

指定QP（整数）是可选的，如果指定，它们在编码器中被钳位到qpmin / qpmax。

--scaling-list <filename>

量化缩放列表。 HEVC支持定义6个量化缩放列表; 每个用于Y，Cb，Cr用于帧内预测，每个用于帧间预测。

默认情况下，x265不使用缩放列表，但也可以通过--scaling-list off使其显式化。

HEVC指定一组默认的缩放列表，可以在不要求它们在SPS中发信号的情况下启用。 可以通过--scaling-list default启用这些缩放列表。

所有其他字符串表示包含HM格式的自定义缩放列表的文件名。 如果未正确解析文件，则编码将中止。 必须在SPS中发出自定义列表信号。

--lambda-file <filename>

指定包含x265_lambda_tab和x265_lambda2_tab值的文本文件。 MAX_MAX_QP + 1（70）浮点值。

文本文件语法很简单。 逗号被认为是空白空间。 所有空白空间都被忽略。 行的长度必须小于2k字节。 哈希（＃）字符后面的内容被忽略。 从文件中读取的值记录在--log-level debug中。

请注意，lambda表是进程全局的，因此新值会影响在同一进程中运行的所有编码器。

Lambda值影响编码器模式决策，lambda越低，它将尝试在信令信息（运动矢量和分裂）上花费的位越多，残差越少。 此功能旨在用于实验。

--max-ausize-factor <float>

它控制规格中定义的最大AU尺寸。 它表示使用的最大AU大小的百分比。 默认值为1，取值范围是0.5到1。

循环过滤器

--deblock=<int>:<int>, --no-deblock

切换解块循环滤波器，可选择指定解块强度偏移。

<int>：<int> - 解析为tC偏移和Beta偏移<int>，<int> - 解析为tC偏移和Beta偏移<int> - tC和Beta偏移分配相同的值。

如果未指定，则偏移默认为0.偏移量必须在-6（最低强度）到6（最高强度）的范围内。

要完全禁用解块滤镜，请使用-no-deblock或-deblock = false。 默认启用，两个偏移默认为0。

如果禁用去块效应，或者偏移量不为零，则默认配置中的这些更改将在PPS中发出信号。

--sao, --no-sao

切换样本自适应偏移环路滤波器，默认是启用状态。

--sao-non-deblock, --no-sao-non-deblock

指定如何处理SAO和解块滤波器之间的依赖性。 启用后，非解锁像素用于SAO分析。 禁用时，SAO分析会跳过右/下边界区域。 默认是禁用状态。

--limit-sao, --no-limit-sao

通过基于帧间预测模式，CTU空间域相关以及亮度和色度之间的关系提前终止SAO过程来限制SAO滤波器计算。 默认是禁用状态。

VUI（视频可用性信息）选项

默认情况下，x265仅发出具有时序信息的VUI。 如果SAR是指定的（或从Y4M头读取），它也包括在内。 所有其他VUI字段必须手动指定。

--sar <integer|w:h>

样本纵横比，单个样本（像素）的宽高比。 用户可以明确地提供宽度和高度，或者从HEVC规范中定义的预定义的纵横比列表中指定一个整数。 默认未定义（未发信号）

1	1:1 (square)
2	12:11
3	10:11
4	16:11
5	40:33
6	24:11
7	20:11
8	32:11
9	80:33
10	18:11
11	15:11
12	64:33
13	160:99
14	4:3
15	3:2
16	2:1

--display-window <left,top,right,bottom>

定义不包含信息的图像（过扫描）区域，因为它是为了达到某种分辨率或纵横比而添加的（区域通常是黑条）。 可以指示解码器在通过--overcan选项显示图像之前裁剪掉该区域。 默认未定义（未发信号）。

请注意，这与编码器内部添加的填充无关，以确保图片大小是最小编码单位（4x4）的倍数。 该填充在单独的“一致性窗口”中发出信号，并且不是用户可配置的。

--overscan <show|crop>

指定解码器是否适合显示或裁剪过扫描区域。 默认未指定（未发出信号）。

--videoformat <integer|string>

在数字化和编码之前指定原始模拟视频的源格式。 默认未定义（未发信号）。

0	component
1	pal
2	ntsc
3	secam
4	mac
5	undefined

--range <full|limited>

指定黑色级别的输出范围以及亮度和色度信号的范围。 默认未定义（未发信号）。

--colorprim <integer|string>

指定转换为RGB时要使用的颜色原色。 默认未定义（未发信号）。

1	bt709
2	unknown
3	reserved
4	bt470m
5	bt470bg
6	smpte170m
7	smpte240m
8	film
9	bt2020
10	smpte428
11	smpte431
12	smpte432

--transfer <integer|string>

指定传输特性。 默认未定义（未发信号）。

1	bt709
2	unknown
3	reserved
4	bt470m
5	bt470bg
6	smpte170m

--colormatrix <integer|string>

指定颜色矩阵设置，即设置用于导出亮度和色度的矩阵系数。 默认未定义（未发信号）。

0	GBR
1	bt709
2	undef
3	reserved
4	fcc
5	bt470bg
6	smpte170m
7	smpte240m
8	YCgCo
9	bt2020nc
10	bt2020c
11	smpte2085
12	chroma-derived-nc
13	chroma-derived-c
14	ictcp

--chromaloc <0..5>

为4：2：0输入指定色度样本位置。 有关这些值的说明，请参阅HEVC规范。 默认未定义（未发信号）。

--master-display <string>

SMPTE ST 2086掌握显示颜色体积SEI信息，指定为在发射流标题SEI时解析的字符串。 字符串格式为“G（%hu，%hu）B（%hu，%hu）R（%hu，%hu）WP（%hu，%hu）L（%u，%u）”其中%hu是 无符号16位整数和%u是无符号32位整数。 SEI包括用于RGB通道的X，Y显示原色和以0.00002为单位的白点（WP）以及以每平方米0.0001坎德拉为单位的最大，最小亮度（L）值。 适用于HDR内容。

P3D65 1000尼特监视器示例，其中G（x = 0.265，y = 0.690），B（x = 0.150，y = 0.060），R（x = 0.680，y = 0.320），WP（x = 0.3127，y = 0.3290），L（max = 1000，min = 0.0001）：

G(13250,34500)B(7500,3000)R(34000,16000)WP(15635,16450)L(10000000,1)

请注意，此字符串值需要进行转义或引用，以防止在许多平台上进行shell扩展。

该选项没有默认值。

--max-cll <string>

消费电子协会861.3规范要求的最大内容光水平（MaxCLL）和最大帧平均光水平（MaxFALL）。

指定为在发出流标头SEI时解析的字符串。 字符串格式为“%hu，%hu”，其中%hu是无符号16位整数。 第一个值是最大内容光级别（如果没有指示最大值，则为0），第二个值是最大图像平均光级别（或0）。 适用于HDR内容。

MaxCLL =每平方米1000坎德拉，MaxFALL =每平方米400坎德拉的示例：

–max-cll “1000,400”

请注意，此字符串值需要进行转义或引用，以防止在许多平台上进行shell扩展。

没有默认值。

--hdr, --no-hdr

在SEI包中强制发送HDR参数。 指定--master-display或--max-cll时自动启用。 当需要为max-cll和max-fall发出0值时这样设置很有用。 默认是禁用状态。

--hdr-opt, --no-hdr-opt

为HDR / WCG内容添加亮度和色度偏移。 输入视频应为10 bit 4：2：0。 适用于HDR内容。建议启用AQ模式以及此功能。 默认是禁用状态。

--dhdr10-info <filename>

将色调映射信息作为SEI消息插入。 它将包含Creative Intent元数据的JSON文件的路径作为输入，将其编码为比特流中的动态色调映射。

--dhdr10-opt, --no-dhdr10-opt

限制插入色调映射信息的帧作为SEI消息。 仅为IDR帧和色调映射信息已更改的帧插入SEI。

--min-luma <integer>

输入图片允许的最小亮度值。 低于min-luma的任何值都会被剪裁。 没有默认值。

--max-luma <integer>

输入图片允许的最大亮度值。 任何高于max-luma的值都会被剪裁。 没有默认值。

--nalu-file <filename>

以POC顺序包含userSEI的文本文件：<POC> <space> <PREFIX> <space> <NAL UNIT TYPE> / <SEI TYPE> <space> <SEI Payload>解析指定的输入文件并将SEI消息插入比特流。 目前，我们仅支持PREFIX SEI消息。 这是一个“仅限应用程序”的功能。

--atc-sei <integer>

发出备选传输特性SEI消息，其中整数是优选的传输特性。 HLG（混合日志Gamma）信令需要。 默认情况下未发出信号。

--pic-struct <integer>

设置图像结构并在图像时序SEI消息中发出它。 值范围为0~12。 详细解释见HEVC规范的D.3.3。 HLG（混合日志Gamma）信令需要。 默认情况下未发出信号。

比特流选项

--annexb, --no-annexb

如果启用，x265将生成附件B比特流格式，该格式在NAL之前放置起始码。 如果禁用，x265将生成文件格式，在NAL之前放置长度。 x265 CLI将根据输出格式选择正确的选项。 默认是启用状态。

API ONLY

--repeat-headers, --no-repeat-headers

如果启用，x265将为每个关键帧发出VPS，SPS和PPS标头。 当没有容器来保存流标题并且您希望关键帧是随机访问点时，可以使用该选项。

默认是禁用状态。

--aud, --no-aud

在每个切片访问单元的开始处发出访问单元定界符NAL。 如果未启用--repeat-headers（表示用户将在流的开头手动编写标题），则将跳过第一个AUD，因为它不能放置在它所属的访问单元的开头。 默认是禁用状态。

--hrd, --no-hrd

启用HRD参数到解码器的信令。 HRD参数由缓冲周期SEI消息和图像定时SEI消息携带，其向解码器提供定时信息。

默认是禁用状态。

--info, --no-info

使用描述编码器版本，构建信息和编码参数的流标头发出信息性SEI。 这对于调试目的非常有用，但编码版本号和构建信息可能会使比特流发散并干扰回归测试。

默认是启用状态。

--hash <integer>

发射解码图像散列SEI，因此解码器可以验证重建的图像并检测数据丢失。 也可用作验证编码器状态的调试功能。

默认值：None。

1	MD5
2	CRC
3	Checksum

--temporal-layers,--no-temporal-layers

启用时间子图层。 所有参考的I / P / B帧都在基础层中，并且所有未参考的B帧被放置在时间增强层中。 解码器可以选择丢弃增强层并且仅解码和显示基础层切片。

如果使用固定GOP（--b-adapt 0）和--bframes 3，则两个层均匀地分割帧速率，并以节流的PbBbP。 您可能还需要--no-scenecut和一个4的倍数的关键帧间隔。

--log2-max-poc-lsb <integer>

图片顺序计数的最大值。 默认是8。

--vui-timing-info, --no-vui-timing-info

在比特流中发出VUI时序信息。 默认是启用状态。

--vui-hrd-info, --no-vui-hrd-info

在比特流中发出VUI HRD信息。 启用--hrd时默认启用。

--opt-qp-pps, --no-opt-qp-pps

基于在最后GOP中观察到的QP值，优化PPS中的QP（而不是默认值26）。 默认是禁用状态。

--opt-ref-list-length-pps, --no-opt-ref-list-length-pps

基于在最后一个GOP中观察到的长度，优化PPS中的L0和L1 ref列表长度（而不是默认值0）。 默认是禁用状态。

--multi-pass-opt-rps, --no-multi-pass-opt-rps

允许在多通道模式下在SPS中存储常用的RPS。 默认是禁用状态。

--opt-cu-delta-qp, --no-opt-cu-delta-qp

通过将较低的QP拉至接近meanQP的值来优化CU水平QP，从而最小化deltaQP信号传导中的波动。 默认是禁用状态。

仅在RD级别5和6有效。

--idr-recovery-sei, --no-idr-recoveery-sei

Emit RecoveryPoint信息作为每个IDR帧的比特流中的sei。 默认是禁用状态。

--single-sei, --no-single-sei

Emit 单个NAL单元中的SEI消息而不是多个NAL。 默认是禁用状态。

When HRD SEI启用HM解码器将发出警告。

DCT近似值

--lowpass-dct

如果启用，x265将使用低通子带dct近似而不是16x16和32x32块的标准dct。 这种近似计算密集度较低，但它为变换后的块生成截断的系数矩阵。 经验分析显示，压缩和性能增益的边际损失高达10％，特别是中等比特率。

对于具有性能和时间限制的平台，应考虑这种近似。

默认是禁用状态。 这是处于实验阶段的功能。

调试选项

--recon, -r <filename>

以显示顺序输出包含重建图像的文件。 如果文件扩展名为“.y4m”，文件将包含YUV4MPEG2流标题和帧标题。 否则它将是编码器内部位深度的原始YUV文件。

CLI ONLY

--recon-depth <integer>

输出文件的位深度。 此值默认为内部位深度，目前无法修改。

CLI ONLY

--recon-y4m-exec <string>

如果应用程序可以播放在stdin上接收的Y4MPEG流，则x265 CLI可以按显示顺序为其重建图片。 显然，图片没有定时信息，因此图像定时将主要通过编码经过时间和延迟来确定，但是预览编码器输出的图片以验证输入设置和速率控制参数可能是有用的。

ffplay的示例命令（假设它在PATH中）：

-recon-y4m-exec“ffplay -i pipe：0 -autoexit”

CLI ONLY

CHAPTER 3

应用程序接口

SECTION 1

简介

x265主要用C ++和x86汇编语言编写，但面向公众的编程接口是C，以实现最广泛的可移植性。此C接口完全在x265.h中定义在源树的源/文件夹中。 最终用户使用的所有函数，变量和枚举都出现在此标题中。

在可能的情况下，x265已尽力使其公共API尽可能接近x264的公共API。 所以那些熟悉通过C接口使用x264的人会发现x265非常熟悉。

该文件旨在按顺序阅读; 叙述线性地贯穿各个部分。

构建注意事项

Main或Main10 pro fi le编码的选择是在编译时进行的; 内部像素深度会影响大量可变大小，因此8和10位像素作为不同的构建选项进行处理（主要是为了保持8位构建的性能）。 libx265导出一个变量**x265_max_bit_depth**，它指示库的编译方式（它将包含值8或10）。 此外，**x265_version_str**是指向已编译的x265版本的字符串的指针，x265_build_info_str是指向标识编译器和构建选项的字符串的指针。

- **x265_version_str**仅在cmake运行时更新。 如果要为其他人创建二进制文件，建议在构建脚本中在make之前运行cmake。

无论内部像素（8或10）的深度如何，x265都将接受8到16位之间任意深度的输入像素。 它将根据需要移动和屏蔽输入像素以达到内部深度。 如果使用我们的CLI应用程序（至8位）执行降档，则可以启用--dither选项以减少绑定。 C接口无法使用此功能。

编码器

x265中的主要对象是编码器对象，它在公共API中表示为**opaque typedef x265_encoder**。 这种指针传递给大多数编码器功能。

单个编码器从一系列原始输入图像生成单个输出比特流。 因此，如果需要多个输出比特流，则必须分配多个编码器。 可以将相同的输入图像传递给多个编码器，编码功能不会修改输入图像结构（图像作为编码的第一步复制到编码器中）。

编码器分配是一种可重入的功能，因此可以在单个进程中安全地分配多个编码器。 编码器访问函数对于单个编码器是不可重入的，因此建议的用例是为每个编码器实例分配一个客户端线程（对于所有编码器实例，可以使用一个线程，但是某些编码器访问功能是阻塞的，因此效率较低）。

- 在单个进程中有多个编码器有一点需要注意。 所有编码器必须使用相同的最大CTU大小，因为许多全局变量都是根据此大小配置的。 如果尝试不匹配的CTU大小，则编码器分配将失败。 如果没有打开编码器，可以调用**x265_cleanup**（）来重置配置的CTU大小，以便可以使用新的大小。

通过调用**x265_encoder_open**（）来分配编码器：

```
/* x265_encoder_open:
*      create a new encoder handler, all parameters from x265_param are copied * /
x265_encoder * x265_encoder_open(x265_param * );
```

然后将返回的指针传递给与此编码有关的所有函数。 在此函数调用期间分配了大量内存，但编码器将继续分配内存，因为第一张图片将传递给编码器; 直到它的图片结构池大到足以处理它必须保留在内部的所有图片。 池大小由lookahead深度，帧线程数和最大引用数确定。

如注释中所示，**x265_param**在内部复制，因此用户可以在分配编码器后释放其副本。 对其编码结构的副本所做的更改在分配后对编码器没有影响。

Param

x265_param结构描述了编码器需要了解输入图像和输出比特流以及介于两者之间的所有内容的所有内容。

处理这些param结构的推荐方法是通过以下方式从libx265分配它们：

```
/* x265_param_alloc:
*      Allocates an x265_param instance. The returned param structure is not
*      special in any way, but using this method together with x265_param_free()
*      and x265_param_parse() to set values by name allows the application to treat
*      x265_param as an opaque data struct for version safety * / x265_param *
  x265_param_alloc();
```

通过这种方式，应用程序不需要知道param结构的确切大小（x265的构建可能比应用程序编译的x265.h的副本更新一些）。

接下来，通过选择性能预设和可选的调谐因子x265_preset_names和x265_tune_names分别保存预设和调谐因子的字符串名称来执行编码器的初始粗略配置（有关预设和调谐因子的详细信息，请参阅第五章第一节的预设部分）：

```
/* returns 0 on success, negative on failure (e.g. invalid preset/tune name). * / int
x265_param_default_preset(x265_param * , const char * preset, const char * tune);
```

现在可以选择指定一个配置文件。 **x265_profile_names**包含此函数接受的字符串名称：

```
/* (can be NULL, in which case the function will do nothing)
*      returns 0 on success, negative on failure (e.g. invalid profile name). * /
int x265_param_apply_profile(x265_param * , const char * profile);
```

最后，使用重复调用来按名称配置任何剩余选项：

```
/* x265_param_parse:
*      set one parameter by name.
*      returns 0 on success, or returns one of the following errors.
*      note: BAD_VALUE occurs only if it can't even parse the value,
*      numerical range is not checked until x265_encoder_open().
*      value=NULL means "true" for boolean options, but is a BAD_VALUE for non-booleans. ,!
  * /
#define X265_PARAM_BAD_NAME (-1)
#define X265_PARAM_BAD_VALUE (-2)
int x265_param_parse(x265_param * p, const char * name, const char * value);
```

有关可由**x265_param_parse**（）设置的选项列表（及其说明），请参阅字符串选项。

创建编码器后，可以释放param结构：

```
/* x265_param_free:
*      Use x265_param_free() to release storage for an x265_param instance
*      allocated by x265_param_alloc() * /
void x265_param_free(x265_param * );
```

注意：使用这些方法来分配和释放param结构有助于以多种方式对代码进行面向未来的验证，但是x265 API的版本化方式使得我们可以防止与x265的构建链接，而x265的构建与您的头部版本不匹配编译（除非你使用x265_api_query（）来获取库的接口）。 这是X265_BUILD宏的功能。

x265_encoder_parameters（）可用于在打开后从编码器获取param结构的副本，以便查看对参数所做的更改以进行自动检测以及其他原因：

```
/* x265_encoder_parameters:
*      copies the current internal set of parameters to the pointer provided
*      by the caller. useful when the calling application needs to know
*      how x265_encoder_open has changed the parameters.
*      note that the data accessible through pointers in the returned param struct
*     (e.g. filenames) should not be modified by the calling application. * /
void x265_encoder_parameters(x265_encoder * , x265_param * );
```

Paramx265_encoder_reconfig() 可用于重新编码中间编码的编码器参数：

```
/* x265_encoder_reconfig:
*      used to modify encoder parameters.
*      various parameters from x265_param are copied.
*      this takes effect immediately, on whichever frame is encoded next;
*      returns negative on parameter validation error, 0 on successful reconfigure
*      and 1 when a reconfigure is already in progress.
```

```
*
*      not all parameters can be changed; see the actual function for a
*      detailed breakdown. since not all parameters can be changed, moving
*      from preset to preset may not always fully copy all relevant parameters,
*      but should still work usably in practice. however, more so than for
*      other presets, many of the speed shortcuts used in ultrafast cannot be
*      switched out of; using reconfig to switch between ultrafast and other
*      presets is not recommended without a more fine-grained breakdown of
*      parameters to take this into account. * /
int x265_encoder_reconfig(x265_encoder * , x265_param * );
```

x265_get_slicetype_poc_and_scenecut（）可用于获取切片类型，poc和场景剪切信息mid-encode：

```
/*  x265_get_slicetype_poc_and_scenecut:
*      get the slice type, poc and scene cut information for the current frame,
*      returns negative on error, 0 on success.
*      This API must be called after(poc >= lookaheadDepth + bframes + 2) condition ,!check. *
   /
int x265_get_slicetype_poc_and_scenecut(x265_encoder * encoder, int * slicetype, int ,! *
poc, int * sceneCut);
```

x265_get_ref_frame_list（）可用于获取前向和后向引用列表：

```
/* x265_get_ref_frame_list:
*       returns negative on error, 0 when access unit were output.
*       This API must be called after(poc >= lookaheadDepth + bframes + 2) condition ,!check * /
int x265_get_ref_frame_list(x265_encoder * encoder, x265_picyuv ** , x265_picyuv ** , int,
,! int, int * , int * );
```

x265_encoder_ctu_info可用于向编码器提供额外的CTU特定信息：

```
/* x265_encoder_ctu_info:
*      Copy CTU information such as ctu address and ctu partition structure of all
*      CTUs in each frame. The function is invoked only if "–ctu-info" is enabled and
*      the encoder will wait for this copy to complete if enabled. * /
int x265_encoder_ctu_info(x265_encoder * encoder, int poc, x265_ctu_info_t ** ctu);
```

x265_set_analysis_data()可用于从外部应用程序接收分析信息：

```
/* x265_set_analysis_data:
*      set the analysis data. The incoming analysis_data structure is assumed to be ,!AVC-sized
   blocks.
*      returns negative on error, 0 access unit were output. * /
int x265_set_analysis_data(x265_encoder * encoder, x265_analysis_data * analysis_data, ,!int
poc, uint32_t cuBytes);
```

x265_alloc_analysis_data()可用于为**x265_analysis_data**分配内存：

```
/* x265_alloc_analysis_data:
*      Allocate memory for the x265_analysis_data object's internal structures. * / void
x265_alloc_analysis_data(x265_param * param, x265_analysis_data * analysis);
```

x265_free_analysis_data()可用于释放**x265_analysis_data**的内存：

```
/* x265_free_analysis_data:
*      Free the allocated memory for x265_analysis_data object's internal structures. ,! * /
void x265_free_analysis_data(x265_param * param, x265_analysis_data * analysis);
```

Pictures

原始图片通过x265_picture结构传递给编码器。 就像param结构一样，我们建议从编码器中分配此结构，以避免出现大小不匹配的可能：

```
/* x265_picture_alloc:
*      Allocates an x265_picture instance. The returned picture structure is not
*      special in any way, but using this method together with x265_picture_free()
*      and x265_picture_init() allows some version safety. New picture fields will
*      always be added to the end of x265_picture * /
x265_picture * x265_picture_alloc();
```

无论以这种方式分配图片结构还是只是在堆栈中进行声明，下一步是通过以下方式初始化结构：

```
/***
*      Initialize an x265_picture structure to default values. It sets the pixel
*      depth and color space to the encoder's internal values and sets the slice
*      type to auto - so the lookahead will determine slice type.
* /
void x265_picture_init(x265_param * param, x265_picture * pic);
```

x265不执行任何色彩空间转换，因此原始图片的色彩空间（色度采样）必须与用于分配编码器的参数结构中指定的色彩空间匹配。 **x265_picture_init**将该字段初始化为内部颜色空间，最好不要对其进行随意修改。

图像位深度初始化为编码器的内部位深度，但该值应更改为传递给编码器的像素的实际深度。 如果图像位深度大于8，则编码器假设使用两个字节来表示每个样本（小端短路）。

用户负责设置平面指针和平面步幅（以字节为单位，而不是像素）。 演示时间戳（**pts**）是可选的，具体取决于是否需要输出上的准确解码时间戳（**dts**）。

如果希望覆盖给定图片的前瞻或速率控制，可以指定除**X265_TYPE_AUTO**之外的切片类型，或者指定0以外的forceQP值。

x265不会修改作为输入提供的图片结构，因此可以为传递给单个编码器的所有图片重复使用单个**x265_picture**，甚至可以将所有图片重复传递给多个编码器。

最终应该发布从库中分配的结构：

```
/* x265_picture_free:
*      Use x265_picture_free() to release storage for an x265_picture instance
*      allocated by x265_picture_alloc() * /
void x265_picture_free(x265_picture * );
```

Analysis Buffers

可以保存分析信息并将其重用于相同视频序列的编码之间（通常用于多比特率编码）。 通过保存最高比特率编码的分析信息并在较低比特率编码中重用它来获得最佳结果。

保存或加载分析数据时，必须为传递到编码器的每个图像分配缓冲区：

```
/* x265_alloc_analysis_data:
*      Allocate memory to hold analysis meta data, returns 1 on success else 0 * /
int x265_alloc_analysis_data(x265_picture * );
```

请注意，这与**x265_picture**的典型语义非常不同，后者可以多次重复使用。 必须为每个输入图像重新分配分析缓冲区。

传递给编码器的分析缓冲区由编码器拥有，直到它们通过输出**x265_picture**传回缓冲区。 用户通过以下方法释放缓冲区：

```
/* x265_free_analysis_data:
*      Use x265_free_analysis_data to release storage of members allocated by
*      x265_alloc_analysis_data * /
void x265_free_analysis_data(x265_picture * );
```

编码过程

编码器的输出是一系列NAL数据包，它们总是在连续的内存中连接返回。 HEVC流具有SPS和PPS以及VPS报头，其描述了如何解码以下分组。 如果指定了--repeat-headers，那么这些标题将与每个关键帧一起输出。 否则，必须使用以下方法显式查询。

```
/* x265_encoder_headers:
*      return the SPS and PPS that will be used for the whole stream.
*      * pi_nal is the number of NAL units outputted in pp_nal.
*      returns negative on error, total byte size of payload data on success
*      the payloads of all output NALs are guaranteed to be sequential in memory. * /
int x265_encoder_headers(x265_encoder * , x265_nal ** pp_nal, uint32_t * pi_nal);
```

现在我们进入主编码循环。 原始输入图像通过以下显示顺序传递给编码器：

```
/* x265_encoder_encode:
*      encode one picture.
*      * pi_nal is the number of NAL units outputted in pp_nal.
*      returns negative on error, zero if no NAL units returned.
*      the payloads of all output NALs are guaranteed to be sequential in memory. * /
int x265_encoder_encode(x265_encoder * encoder, x265_nal ** pp_nal, uint32_t * pi_nal,
,!x265_picture * pic_in, x265_picture * pic_out);
```

这些图片排队等待直到前瞻为止，然后帧编码器依次填充，然后最后你开始接收输出NAL（对应于单个输出图像），每个输入图像都传入编码器。

管道完全填满后，**x265_encoder_encode**（）将阻塞，直到下一个输出图片完成。

注意：可选地，如果提供了第二个**x265_picture**结构的指针，则编码器将使用与输出NAL相对应的输出图像的数据来填充它，包括重构图像，POC和解码时间戳。 这些图片将按编码（或解码）顺序排列。 编码器还会将相应的帧编码统计信息写入**x265_frame_stats**。

当最后一个原始输入图像被发送到编码器时，仍然必须使用*pic_in*参数0重复调用**x265_encoder_encode**（），指示管道flush，直到该函数返回小于或等于0的值（表示 输出比特流完成）。

在此过程中的任何时候，应用程序都可以从编码器查询运行的统计信息：

```
/* x265_encoder_get_stats:
*     returns encoder statistics * /
void x265_encoder_get_stats(x265_encoder * encoder, x265_stats * , uint32_t
,!statsSizeBytes);
```

Cleanup

在编码结束时，如果指定了*--csv*，应用程序将希望触发最终编码统计信息的记录：

```
/* x265_encoder_log:
*      write a line to the configured CSV file. If a CSV filename was not
*      configured, or file open failed, this function will perform no write. * /
void x265_encoder_log(x265_encoder * encoder, int argc, char ** argv);
```

最后，必须关闭编码器以释放其所有资源。 已经过闪存的编码器无法重新启动和重复使用。 一旦调用了**x265_encoder_close**（），就必须丢弃编码器句柄：

```
/* x265_encoder_close:
*      close an encoder handler * /
void x265_encoder_close(x265_encoder * );
```

当应用程序完成所有编码时，它应调用**x265_cleanup**（）以释放进程全局，特别是在使用内存泄漏检测工具时。 **x265_cleanup**（）还会重置已保存的CTU大小，以便可以创建具有不同CTU大小的新编码器：

```
/* x265_cleanup:
*      release library static allocations, reset configured CTU size * /
void x265_cleanup(void);
```

多库接口

如果您的应用程序可能希望进行运行时位深度选择，则需要使用这些位深度介绍接口之一，该接口返回包含公共函数入口点和常量的API结构。

您不是直接使用上面记录的所有**x265_**方法，而是从**libx265**查询**x265_api**结构，然后在该结构中使用相同名称的函数指针（减去**x265_ prefix**）。 例如，**x265_param_default**（）变为**api-> param_default**（）。

x265_api_get

第一个位深度的instrospecton方法是**x265_api_get**（）。 它专为可能与**libx265**静态链接的应用程序而设计，或者至少与特定的SONAME或API版本相关联：

```
/* x265_api_get:
*      Retrieve the programming interface for a linked x265 library.
*      May return NULL if no library is available that supports the
*      requested bit depth. If bitDepth is 0, the function is guarunteed
*      to return a non-NULL x265_api pointer from the system default
*      libx265 * /
const x265_api * x265_api_get(int bitDepth);
```

与**x265_encoder_encode**（）一样，此函数具有通过宏自动附加到函数名称的内部版本号。这将您的应用程序绑定到libx265的特定二进制API版本（您编译的那个）。 如果尝试使用具有不同API版本号的libx265进行链接，会发生连接失败的情况。

很明显，这对静态链接到libx265的应用程序没有任何有意义的影响。

x265_api_query

第二种深度内省方法适用于需要在API版本中更灵活的应用程序。 如果使用**x265_api_query**（）并在运行时动态链接到libx265（在Windows上使用POSIX或LoadLibrary（）上的**dlopen**（）），应用程序将不再直接与其编译的API版本相关联：

```
/* x265_api_query:
*      Retrieve the programming interface for a linked x265 library, like
*      x265_api_get(), except this function accepts X265_BUILD as the second
```

```
*      argument rather than using the build number as part of the function name.
*      Applications which dynamically link to libx265 can use this interface to
*      query the library API and achieve a relative amount of version skew
*      flexibility. The function may return NULL if the library determines that
*      the apiVersion that your application was compiled against is not compatible
*      with the library you have linked with.
*
*      api_major_version will be incremented any time non-backward compatible
*      changes are made to any public structures or functions. If
*      api_major_version does not match X265_MAJOR_VERSION from the x265.h your
*      application compiled against, your application must not use the returned
*      x265_api pointer.
*
*      Users of this API * must * also validate the sizes of any structures which
*      are not treated as opaque in application code. For instance, if your
*      application dereferences a x265_param pointer, then it must check that
*      api->sizeof_param matches the sizeof(x265_param) that your application
*      compiled with. * /
const x265_api * x265_api_query(int bitDepth, int apiVersion, int * err);
```

必须对返回的API结构执行许多验证，以确定应用程序是否可以安全使用。 如果不执行这些检查，应用程序可能会崩溃：

```
if (api->api_major_version != X265_MAJOR_VERSION) / * do not use * /
if (api->sizeof_param != sizeof(x265_param)) / * do not use * /
if (api->sizeof_picture != sizeof(x265_picture)) / * do not use * /
if (api->sizeof_stats != sizeof(x265_stats)) / * do not use * /
if (api->sizeof_zone != sizeof(x265_zone)) / * do not use * /
etc.
```

请注意，如果您的应用程序没有直接分配或取消引用其中一个结构，如果它将结构视为不透明或根本不使用它，那么它可以跳过该结构的大小检查。

特别需要注意的是，如果应用程序使用**api-> param_alloc**（），**api-> param_free**（），**api-> param_parse**（）等，并且永远不会直接访问任何**x265_param**字段，那么它可以跳过对sizeof的检查（**x265_parm**））从而忽略对该结构的改变（占X265_BUILD碰撞的很大比例）。

Build Implications

默认情况下，libx265会将其所有内部C ++类和函数放在x265命名空间中，并导出此文件中记录的所有C函数。 显然，这可以防止libx265的8位和10位构建静态链接到单个二进制文件中，所有这些符号都会发生冲突。

但是，如果将**EXPORT_C_API** cmake选项设置为OFF，则libx265将使用位深度特定命名空间和pre fi x作为其汇编函数（x265_8bit，x265_10bit或x265_12bit）并且不导出C函数。

通过这种方式，您可以构建一个或多个libx265库而无需任何导出的C接口，并将它们链接到导出C接口的libx265构建中。 导出C函数的构建成为组合库的默认位深度，其他位深度可通过位深度内省方法获得。

注意：将**EXPORT_C_API** cmake选项设置为OFF时，建议还将**ENABLE_SHARED**和**ENABLE_CLI**设置为OFF以防止生成问题。 我们只需要这些构建中的静态库。

如果应用程序请求默认库或任何其他链接库不支持的位深度，则内省方法将回退到尝试使用适合于请求的位深度的名称动态绑定共享库：

```
8-bit: libx265_main
10-bit: libx265_main10
12-bit: libx265_main12
```

如果找不到命名文件库，它将尝试绑定一个通用的libx265，希望它是一个具有所有位深度的multilib库。

Packaging and Distribution

我们建议打包器分发一个组合的共享/静态库构建，其中包括链接在一起的所有位深度库。 有关如何影响这些组合库构建的示例，请参阅我们的构建/子目录中的multilib脚本。 打包程序可以自行决定是否导出公共C函数，从而成为组合库的默认位深度。

注意：Windows打包程序可能希望在启用**WINXP_SUPPORT**的情况下构建libx265。 这使得生成的二进制文件在XP和Vista上运行。 如果没有这个版本，支持的最小主机O / S就是Windows 7.另请注意，使用**WINXP_SUPPORT**构建的二进制文件不会支持NUMA，性能会稍差。

还建议使用**STATIC_LINK_CRT**，以便最终用户无需安装任何其他MSVC C运行时库。

CHAPTER 4

Threading

线程池

x265为每个编码器创建一个或多个线程池，每个NUMA节点一个池（通常是CPU套接字）。 *--pools*指定编码器将分配的池数和每个池的线程数。 默认情况下，x265在每个NUMA节点上为每个（超线程）CPU核心分配一个线程。

如果在具有多个NUMA节点的系统上运行多个编码器，建议将每个编码器隔离到单个节点，以避免远程内存访问的NUMA开销。

工作分配是基于工作的。 空闲工作线程扫描分配给其线程池的作业提供程序以执行作业。 当没有可用的作业时，空闲工作线程阻塞并且不消耗CPU周期。

希望将工作分配给工作线程的对象称为作业提供者（它们派生自JobProvider类）。 线程池有一种方法可以唤醒被阻塞的空闲线程，并建议作业提供者在创建新作业时调用此方法。

除非绝对有必要进行数据锁定，否则不允许阻止工作程序作业。 如果作业被阻止，则工作函数应该删除该作业，以便工作线程可以返回池并找到更多工作。

在Windows上，本机API提供了足够的功能来发现NUMA拓扑并强制执行libx265所需的线程功能（只要您没有选择以XP或Vista为目标），但在POSIX系统上，它依赖于libnuma来实现此功能。 如果您的目标POSIX系统是单个套接字，那么在没有libnuma的情况下构建是一个非常合理的选择，因为它对运行时行为没有影响。 在多插槽系统上，没有libnuma的libx265的POSIX版本工作效率会降低，但仍能正常工作。 您失去了工作隔离效果，使每个帧编码器仅使用单个套接字的线程，因此会导致更大的上下文切换成本。

Wavefront Parallel Processing

新的HEVC，波前并行处理允许每行CTU并行编码，只要每行在其上方的行后面至少保留两个CTU，以确保上方和右上方的块的帧内引用和其他数据 可用。 WPP对每个CTU的分析和压缩几乎没有影响，因此它对压缩效率相对于切片或瓷砖的影响非常小。 在我们的大多数测试中，发现WPP的压缩损失小于1％。

WPP有三种可影响效率的效果。 第一个是行开始必须在切片报头中发信号，第二个是每行必须填充到偶数字节的长度，第三个是熵编码器的状态从每行的第二个CTU转移到 它下面一行的第一个CTU。 在某些情况下，这种状态转移实际上改善了压缩，因为右上角状态可能具有比前一行末端更好的局部性。

Parabola Research发布了一款出色的HEVC动画，可以很好地显示WPP。 它甚至可以正确地显示一些WPP的主要缺点，例如：

1	每帧开始和结束时的低线程利用率
2	一个困难的区块可能会阻止wave-front，wave-front恢复需要一段时间
3	64x64 CTU比较大！ 与H.264和类似的编解码器相比，行数要少得多

由于这些stall问题，很少能获得行线程所期望的完全并行化优势。 理论上的典型状态是完美线程的30％到50％。

在x265中，WPP默认启用，因为它不仅提高了编码性能，而且还使解码器可以进行线程化。

如果*-no-wpp*禁用WPP，则帧将按扫描顺序编码，并且将避免熵开销。 如果未禁用帧线程，编码器将更改默认帧线程计数，使其高于启用WPP的情况。 确切的公式将在下一节中介绍。

Bonded Task Groups

如果工作线程作业具有可由许多线程并行执行的工作，则它可以分配绑定的任务组并从同一线程池中获取其他空闲工作线程的帮助。 这些线程将合作完成绑定任务组的工作，然后返回其空闲状态。 这些任务越大越均匀，绑定任务组的性能就越好。

Parallel Mode Analysis

启用--pmode时，每个CU（从64x64到8x8的所有深度）都将通过绑定的任务组将其分析工作分发到线程池。 每个分析工作将测量CU的一个预测的成本：合并，跳过，帧内，帧间（2Nx2N，Nx2N，2NxN和AMP）。

在较慢的预设中，来自pmode的增加的并行度通常足以能够在实现相同的总体CPU利用率的同时减少或禁用帧并行性。 减少帧线程通常对ABR和VBV速率控制有利。

Parallel Motion Estimation

启用--pme时，对参考帧执行运动搜索的所有分析函数都会通过绑定的任务组将这些运动搜索分配给其他工作线程（如果需要两次以上的运动搜索）。

帧线程

帧线程是同时编码多个帧的行为。 这是一个挑战，因为每个帧通常将一个或多个先前编码的帧用作运动参考，并且那些帧可能仍处于自身编码的过程中。

以前的编码器（如x264）通过将这些参考帧内的运动搜索区域限制为正在编码的重合行下方的一个宏块行来解决此问题。 因此，帧可以与其参考帧同时编码，只要它在其参考的编码进度之后保持一行（掩盖几个细节）。

x265具有相同的帧线程机制，但由于我们的CTU行的大小，我们通常比x264具有更少的帧并行性。 例如，对于1080p视频，x264每帧有68个16x16宏块行，而x265只有17个64x64 CTU行。

第二个情有可原的情况是循环过滤器。 用于运动参考的像素必须由循环滤波器处理，并且循环滤波器在编码完整行之前不能运行，并且它必须在编码过程后面运行整行，以便可以使用过滤行下方的像素。 除此之外，HEVC还有两个循环过滤器：去块和SAO，它们必须串联运行，它们之间存在行滞后。 当你将所有行滞后加起来时，每个帧最终在其参考帧后面有3个CTU行（相当于x264的12个宏块行）。 并牢记波前进展模式; 到参考帧结束第三行CTU时，帧中几乎一半的CTU可以被压缩（取决于显示宽高比）。

第三种情有可原的情况是，当编码的帧被可用的参考帧行阻挡时，该帧的波前变得完全失效，当该行再次可用时，如果重新启动该波可能需要相当长的时间才能重新启动，如果 它曾经。 这使得在使用帧并行性时WPP效率较低。

*--merange*会对帧并行性产生负面影响。 如果范围太大，则必须添加更多行的CTU滞后以确保参考帧中的这些像素可用。

注意：即使使用merange来确定参考帧中必须可用的参考像素的数量，实际的运动搜索也不一定以重合块为中心。 运动搜索实际上以运动预测器为中心，但可用的像素区域（mvmin，mvmax）由merange和插值滤波器半高确定。

当禁用帧线程时，所有参考帧的全部始终完全可用（通过定义），因此可用的像素区域根本不受限制，这有时可以提高压缩效率。 因此，禁用帧并行性的编码输出将与启用帧并行性的编码输出不匹配; 但是在启用时，帧线程的数量应该对输出比特流没有影响，除非使用ABR或VBV速率控制或降噪。

当启用--nr时，每个帧线程的输出将是确定性的，但它们都不匹配，因为每个帧编码器保持累积降噪状态。

无论帧并行度如何，VBV在此时都会在编码器中引入非确定性。

默认情况下，帧并行性和WPP一起启用。 使用的帧线程数是从（超线程）CPU核心数自动检测的，但可以通过--frame-threads手动指定

Cores	Frames
> 32	6..8
>= 16	5
>= 8	3
>= 4	2

如果禁用WPP，则帧线程计数默认为**min（cpuCount，ctuRows / 2）**。

过度分配帧线程可能会适得其反。 它们各自分配大量内存，并且由于CTU行数量和参考滞后数量有限，通常会增加帧编码器超出自动检测计数的好处，并且额外的帧编码器通常会降低性能。

考虑到这些因素，您可以理解为什么更快的预设会将最大CTU大小降低到32x32（使WPP可用的CTU行数增加一倍，并实现细粒度帧并行性）并减少--merange。

每个帧编码器在其自己的线程中运行（与工作池分开分配）。 此框架线程具有一些预处理职责和每个框架的一些后处理职责，但它花费大量时间来管理波前处理，方法是在解决依赖关系时使CTU行可供工作线程使用。 帧编码器线程几乎将所有时间都锁定在4个可能位置之一：

1	阻止，等待帧被处理
2	在参考帧上被阻塞，等待CTU行的重建和环路滤波参考像素变得可用
3	阻止等待wave-front完成
4	阻止等待主线程使用编码帧

SECTION 5

Lookahead

x265的先行模块（确定场景切换和切片类型的低预编码）使用线程池将低成本分析分配给工作线程。 它将使用绑定的任务组来执行批量的帧成本估算，并且可以选择使用绑定的任务组来使用切片来测量单帧成本估算。 （具体内容见*--lookahead-slices*部分）。

如果编码器有一个线程池，那么主要的**slicetypeDecide（）**函数本身也由一个工作线程执行，否则它在调用**x265_encoder_encode（）**的线程的上下文中运行。

SECTION 6

SAO

采样自适应偏移环路滤波器对编码性能有很大影响，因为必须对其进行分析和编码。

在CTU本身编码之前，SAO文件和数据在CTU级别进行编码，但是在完全分析CTU（重建像素可用）以及之后，不能执行SAO分析（决定是否启用SAO以及使用什么参数） 右侧和下方的货运单元。因此，编码器必须在CTU压缩波阵面后面的至少一整行的波前执行SAO分析。

这种额外的等待时间迫使编码器保存每个CTU的编码数据，直到整个帧被分析为止，此时功能可以利用所确定的SAO标记对最终的切片比特流进行编码，并且在每个CTU之间交织数据。 第二次通过CTU可能很耗费性能，特别是在高分辨率和高比特率的情况下。

CHAPTER 5

预设选项

Presets

x265有十个预定义的--preset选项，可以优化编码速度（每秒编码帧数）和压缩效率（比特流中每比特的质量）之间的权衡。 默认预设为中等。 它可以很好地找到最好的质量，而不需要花费过多的CPU周期来寻找实现这种质量的绝对最有效的方法。 当您使用更快的预设时，编码器会采用快捷方式来提高性能，但会降低质量和压缩效率。 当您使用较慢的预设时，x265会测试更多编码选项，使用更多计算以所选比特率获得最佳质量（或者在-crf速率控制的情况下，所选质量的最低比特率）。

预设调整编码器参数，如下表所示。 在命令行中指定的以下任何参数将从预设指定的值更改。

0. ultrafast
1. superfast
2. veryfast
3. faster
4. fast
5. medium (**default**)
6. slow
7. slower
8. veryslow
9. placebo

preset	0	1	2	3	4	5	6	7	8	9
ctu	32	32	64	64	64	64	64	64	64	64
min-cu-size	16	8	8	8	8	8	8	8	8	8
bframes	3	3	4	4	4	4	4	8	8	8
b-adapt	0	0	0	0	0	2	2	2	2	2
rc-lookahead	5	10	15	15	15	20	25	30	40	60
lookahead-slices	8	8	8	8	8	8	4	4	1	1
scenecut	0	40	40	40	40	40	40	40	40	40
ref	1	1	2	2	3	3	4	4	5	5
limit-refs	0	0	3	3	3	3	3	2	1	0
me	dia	hex	hex	hex	hex	hex	star	star	star	star
merange	57	57	57	57	57	57	57	57	57	92

preset	0	1	2	3	4	5	6	7	8	9
subme	0	1	1	2	2	2	3	3	4	5
rect	0	0	0	0	0	0	1	1	1	1
amp	0	0	0	0	0	0	0	1	1	1
limit-modes	0	0	0	0	0	0	1	1	1	0
max-merge	2	2	2	2	2	2	3	3	4	5
early-skip	1	1	1	1	0	0	0	0	0	0
recursion-skip	1	1	1	1	1	1	1	1	0	0
fast-intra	1	1	1	1	1	0	0	0	0	0
b-intra	0	0	0	0	0	0	0	1	1	1
sao	0	0	1	1	1	1	1	1	1	1
signhide	0	1	1	1	1	1	1	1	1	1
weightp	0	0	1	1	1	1	1	1	1	1
weightb	0	0	0	0	0	0	0	1	1	1
aq-mode	0	0	1	1	1	1	1	1	1	1
cuTree	1	1	1	1	1	1	1	1	1	1
rdLevel	2	2	2	2	2	3	4	6	6	6
rdoq-level	0	0	0	0	0	0	2	2	2	2
tu-intra	1	1	1	1	1	1	1	2	3	4
tu-inter	1	1	1	1	1	1	1	2	3	4
limit-tu	0	0	0	0	0	0	0	4	4	0

Tuning

有一些--tune选项可用，它们在预设后应用。

注意：psnr和ssim调整选项会禁用所有针对感知视觉质量的度量标准得分的优化（也称为心理视觉优化）。 默认情况下，x265始终会调整最高感知视觉质量，但如果打算使用PSNR或SSIM测量编码以进行基准测试，我们强烈建议配置x265以针对该特定指标进行调整。

–tune	影响
psnr	adaptive quant, psy-rd, 和cutree处于disable状态
ssim	启用自适应定量自动模式，禁用psy-rd
grain	提高film grain的保留率，下面会有详细介绍
fastdecode	没有循环过滤器，没有加权pred，B帧没有intra
zerolatency	没有 lookahead,没有 B frames, 没有cutree

Film Grain

--tune grain旨在以最佳视觉质量编码粒状内容。 此选项的目的既不是保留也不是消除谷物，而是防止由谷物分布不均匀引起的明显伪影。 --tune grain强烈限制在帧内和帧之间改变量化参数的算法。 调整谷物也会偏向于保留更多高频成分的决策。

- --aq-mode 0
- --cutree 0
- --ipratio 1.1
- --pbratio 1.0
- --qpstep 1
- --sao 0
- --psy-rd 4.0
- --psy-rdoq 10.0
- --recursion-skip 0

Fast Decode

*--tune fastdecode*禁用编码器功能，这些功能往往是解码器的瓶颈。 它适用于高比特率的4K内容，这会导致解码器挣扎。 它禁用了两个HEVC循环过滤器，这往往是流程瓶颈：

- --no-deblock
- --no-sao

它禁用加权预测，这往往是带宽瓶颈：

- --no-weightp
- --no-weightb

并且它通过-no-b-intra禁用B帧中的帧内块，因为帧内预测的块导致解码器中的串行依赖性。

Zero Latency

延迟问题有两半。 解码器有延迟，编码器有延迟。 --tune zerolatency消除了双方的延迟。 解码器延迟通过以下方式删除：

- --bframes 0

编码器延迟通过以下方式删除：

- --b-adapt 0
- --rc-lookahead 0
- --no-scenecut
- --no-cutree
- --frame-threads 1

通过所有这些设置，**x265_encoder_encode**（）将同步运行，作为pic_in传递的图片将被编码并作为NAL返回。 这些设置禁用帧并行性，这是x265性能的重要组成部分。 如果您可以容忍编码器的任何延迟，则可以通过增加帧线程数来提高性能。 每个额外的帧线程都会增加一帧延迟。

CHAPTER 6

无损

无损编码

x265可以使用--lossless选项对完全无损的HEVC比特流进行编码（重建的图像对源图像是精确的）。 无损操作在理论上很简单。 通过定义，速率控制被禁用，编码器禁用所有质量指标，因为它们只会浪费CPU周期。 相反，x265仅在编码结束时报告压缩因子。

在HEVC中，无损编码意味着绕过DCT变换和绕过量化（通常称为跨频率旁路）。 仍然允许正常预测，因此编码器将找到最佳的帧间或帧内预测，然后无损地编码残差（使用transquant bypass）。

所有--preset选项都能够生成无损视频流，但通常预置越慢，压缩率越高（编码越慢）。 这里有些例子：

```
./x265 ../test-720p.y4m o.bin –preset ultrafast –lossless
... <snip> ...
encoded 721 frames in 238.38s (3.02 fps), 57457.94 kb/s

./x265 ../test-720p.y4m o.bin –preset faster –lossless
... <snip> ...
x265 [info]: lossless compression ratio 3.11::1
encoded 721 frames in 258.46s (2.79 fps), 56787.65 kb/s

./x265 ../test-720p.y4m o.bin –preset slow –lossless
... <snip> ...
x265 [info]: lossless compression ratio 3.36::1
encoded 721 frames in 576.73s (1.25 fps), 52668.25 kb/s

./x265 ../test-720p.y4m o.bin –preset veryslow –lossless
x265 [info]: lossless compression ratio 3.76::1
encoded 721 frames in 6298.22s (0.11 fps), 47008.65 kb/s
```

注意：在HEVC中，只有QP = 4才是真正的无损量化，因此当编码损耗时，x265在其RDO决策中内部使用QP = 4。

近无损编码

近无损的条件更有趣。 正常的ABR速率控制将允许人们将比特率缩放到完全绕过量化的点（QP <= 4），但即使在这一点上由于DCT变换而在表上留下了大量的SSIM，这是 不是无损的：

```
./x265 ../test-720p.y4m o.bin –preset medium –bitrate 40000 –ssim
encoded 721 frames in 326.62s (2.21 fps), 39750.56 kb/s, SSIM Mean Y: 0.9990703 (30. ,!317 dB)

./x265 ../test-720p.y4m o.bin –preset medium –bitrate 50000 –ssim
encoded 721 frames in 349.27s (2.06 fps), 44326.84 kb/s, SSIM Mean Y: 0.9994134 (32. ,!316 dB)

./x265 ../test-720p.y4m o.bin –preset medium –bitrate 60000 –ssim
encoded 721 frames in 360.04s (2.00 fps), 45394.50 kb/s, SSIM Mean Y: 0.9994823 (32. ,!859 dB)
```

为了使编码器克服这个质量平台，必须在CU级实现*--cu-lossless*。 它告诉编码器将跨量子旁路评估为每个CU的编码选项，并选择具有最佳速率 - 失真特性的选项。

*-cu-lossless*选项在计算上非常昂贵，并且当QP极低时它只有正面效果，允许RDO花费大量的比特来对质量进行小的改进。 因此，只有在编码近无损比特流时才应启用此选项：

```
./x265 ../test-720p.y4m o.bin –preset medium –bitrate 40000 –ssim –cu-lossless
encoded 721 frames in 500.51s (1.44 fps), 40017.10 kb/s, SSIM Mean Y: 0.9997790 (36. ,!557 dB)

./x265 ../test-720p.y4m o.bin –preset medium –bitrate 50000 –ssim –cu-lossless
encoded 721 frames in 524.60s (1.37 fps), 46083.37 kb/s, SSIM Mean Y: 0.9999432 (42. ,!456 dB)

./x265 ../test-720p.y4m o.bin –preset medium –bitrate 60000 –ssim –cu-lossless
encoded 721 frames in 523.63s (1.38 fps), 46552.92 kb/s, SSIM Mean Y: 0.9999489 (42. ,!917 dB)
```

注意：随着增加无损编码，比特率下降并不罕见。 具有“完美编码”的参考块减少了后续帧中的残差。 近无损编码很可能比无损编码花费更多的比特。

实现心理视觉速率失真将改善无损编码。 --psy-rd影响RDO决策，支持通过比特成本保存能量（细节），并导致更多块被无损编码。 我们的psy-rd功能尚未进行程序集优化，因此这使得编码运行速度更慢：

```
./x265 ../test-720p.y4m o.bin –preset medium –bitrate 40000 –ssim –cu-lossless – ,!psy-rd 1.0
encoded 721 frames in 581.83s (1.24 fps), 40112.15 kb/s, SSIM Mean Y: 0.9998632 (38. ,!638 dB)
./x265 ../test-720p.y4m o.bin –preset medium –bitrate 50000 –ssim –cu-lossless – ,!psy-rd 1.0
encoded 721 frames in 587.54s (1.23 fps), 46284.55 kb/s, SSIM Mean Y: 0.9999663 (44. ,!721 dB)

./x265 ../test-720p.y4m o.bin –preset medium –bitrate 60000 –ssim –cu-lossless – ,!psy-rd 1.0
encoded 721 frames in 592.93s (1.22 fps), 46839.51 kb/s, SSIM Mean Y: 0.9999707 (45. ,!334 dB)
```

-cu-lossless在更慢的预设中也会更有效，这些预设在更多级别执行RDO，因此可以找到更小的无损编码块：

```
./x265 ../test-720p.y4m o.bin –preset veryslow –bitrate 40000 –ssim –cu-lossless
encoded 721 frames in 12969.25s (0.06 fps), 37331.96 kb/s, SSIM Mean Y: 0.9998108 (37. ,!231 dB)

./x265 ../test-720p.y4m o.bin –preset veryslow –bitrate 50000 –ssim –cu-lossless
encoded 721 frames in 46217.84s (0.05 fps), 42976.28 kb/s, SSIM Mean Y: 0.9999482 (42. ,!856 dB)

./x265 ../test-720p.y4m o.bin –preset veryslow –bitrate 60000 –ssim –cu-lossless
encoded 721 frames in 13738.17s (0.05 fps), 43864.21 kb/s, SSIM Mean Y: 0.9999633 (44. ,!348 dB)
```

并且通过psy-rd和慢速预设，可以实现非常高的SSIM：

```
./x265 ../test-720p.y4m o.bin –preset veryslow –bitrate 40000 –ssim –cu-lossless - ,!-psy-rd 1.0
encoded 721 frames in 11675.81s (0.06 fps), 37819.45 kb/s, SSIM Mean Y: 0.9999181 (40. ,!867 dB)

./x265 ../test-720p.y4m o.bin –preset veryslow –bitrate 50000 –ssim –cu-lossless - ,!-psy-rd 1.0
encoded 721 frames in 12414.56s (0.06 fps), 42815.75 kb/s, SSIM Mean Y: 0.9999758 (46. ,!168 dB)

./x265 ../test-720p.y4m o.bin –preset veryslow –bitrate 60000 –ssim –cu-lossless - ,!-psy-rd 1.0
```

```
encoded 721 frames in 11684.89s (0.06 fps), 43324.48 kb/s, SSIM Mean Y: 0.9999793 (46.
,!835 dB)
```

最后要注意的是，编码器对视频进行无损编码比对无损编码进行编码更容易（工作量更少）。 如果编码器预先知道编码必须是无损的，则不需要评估任何有损编码方法。 编码器只需要为每个块找到最有效的预测，然后对残差进行熵编码。

当编码器确定它正在编码接近无损的比特流时（即：当速率控制几乎禁用所有量化时），因此该功能需要在流标头中启用fl ag，因此无法自动开启。。 在编写流标题时，我们不知道--cu-lossless是一种帮助还是一种阻碍。 如果很少或没有块最终被编码为无损，则启用该功能是压缩效率的净损失，因为它增加了必须为每个CU编码的标记。 因此，即使忽略功能的性能方面，如果在未使用的情况下启用也可能是压缩损失。 因此，用户只能在接近无损质量的编码时启用此功能。

SECTION 3

Transform Skip

一个有点相关的功能--tskip告诉编码器在编码小的4x4变换块时评估变换跳过（旁路DCT但仍然启用量化）。 此功能旨在提高屏幕内容的编码效率（又名：屏幕上的文本），并非真正用于无损编码。只有当内容中有很多非常尖锐的边缘时，才应启用此功能，并且该功能与无损编码大多无关。

CHAPTER 7

发行说明

Version 2.8

发布日期 - 21/05/2018

新功能

新功能	
1	--asm avx512用于在x265中启用AVX-512。 默认禁用。 对于4K main10高质量编码，效果还是不错的; 对于其他分辨率和预设，我们暂时不建议使用该设置。
2	--dynamic-refine在不同的间隔级别之间动态切换。 默认禁用。 建议使用：选项：' - re fi ne-intra 4'与动态改进，以便在编码效率和性能之间进行更好的权衡，而不是使用静态改进。
3	--single-sei在单个NAL单元中编码SEI消息，而不是多个NAL单元。 默认禁用。
4	--max-ausize-factor控制HEVC规范中定义的最大AU尺寸。 它表示使用的最大AU大小的百分比。 默认值为1。
5	VMAF（视频多方法评估融合）为视频序列的客观质量测量添加了VMAF支持。 启用cmake选项ENABLE_LIBVMAF以报告每帧并聚合VMAF分数。 帧级别VMAF分数不包括时间分数。 目前仅支持linux。

编码器增强功能

Encoder enhancements	
1	引入re-ne-intra level 4以提高质量。
2	支持SEI消息中的HLG分级内容和pic_struct。

Bug修复

Bug Fixes	
1	在Linux中修复32位构建错误（使用CMAKE GUI）。
2	修复asm原语的32位构建错误。
3	修复mac OS上的构建错误。
4	修复分析加载中的VBV Lookahead以达到目标比特率。

Version 2.7

发布日期 - 2018年2月21日。

新功能

	新功能
1	--gop-lookahead可用于扩展gop边界（由-keyint设置）。 如果在这么多帧内找到场景剪切帧，则将扩展GOP。
2	支持在x265中添加的RADL图片。 --radl可用于决定IDR图片之前的RADL图片的数量。

编码器增强功能

	Encoder enhancements
1	从YASM迁移到NASM汇编程序。 支持NASM汇编程序版本2.13及更高版本。
2	在单次运行中启用分析保存和加载。 介绍两个新的cli选项-analysis-save <fi lename>和-analysis-load <fi lename>。
3	符合HDR10 + LLC规范。
4	通过重新分解ip fi lter.asm，将x265构建时间减少50％以上。

Bug修复

	Bug Fixes
1	修复了deblock过滤器和-const-vbv中不一致的输出问题。
2	修复了Mac OS构建警告。
3	修复了启用weightp和cutree时pass-2的不一致性。
4	修复了由于BREF帧丢失导致的死锁问题，同时通过qp文件强制切片类型。

Version 2.6

发布日期 - 2018年5月21日。

新功能

新功能	
1	x265现在可以从先前的HEVC编码（使用选项--refine-inter和--refine-intra）或先前的AVC编码（使用选项--refine-mv-type）进行重新分析。 可以使用x265_picture对象中提供的x265_analysis_data_t数据字段打包先前的代码信息。
2	对使用--vbv-end添加的分段（或分块）编码的基本支持，可以指定段末尾的CPB状态。 将此字符串与--vbv-init串在一起，将标题编码为块，同时保持VBV合规性！
3	--force-flush可用于触发编码器的过早使用。 当已知输入是突发性时，此选项是有益的，并且可能比编码器慢。
4	实验特征 - 使用截断DCT进行变换的lowpass-dct。

编码器增强功能

Encoder enhancements	
1	切片并行模式在性能方面得到了显着提升，特别是在低延迟模式下。
2	x265现在在VS2017上得到了支持。
3	x265现在支持从mono0到mono16的所有深度，用于Y4M格式。

API变化

API changes	
1	现在，默认情况下禁用动态修改PPS的选项（--opt-qp-pps和--opt-ref-list-length-pps），以允许用户通过不发送标头来保存位。 如果启用了这些选项，则必须为每个GOP重复标题。
2	速率控制和分析参数可以通过x265_encoder_recon fi g API同时动态重新配置。
3	现在可以使用新的API函数来提取诸如切片类型，场景切换信息，参考帧等中间信息。 该信息对于集成试图执行内容自适应编码的应用程序可能是有益的。 有关更多详细信息和建议用法，请参阅有关x265_get_slicetype_poc_and_scenecut和x265_get_ref_frame_list的文档。
4	增加了一个新的API，用于将补充CTU信息传递给x265以影响分析决策。 有关更多详细信息，请参阅有关x265_encoder_ctu_info的文档。

Bug修复

Bug Fixes	
1	当--slices与VBV设置一起使用时，会出现错误。
2	针对HDR10 +版本的次要内存泄漏，以及指定池选项时的默认x265。
3	HDR10 + bug fi x去除对poc计数器的依赖以选择元数据信息。

Version 2.5

发布日期 - 2017年7月13日。

编码器增强功能

Encoder enhancements	
1	通过限制VBV操作来限制QP跳跃，使用--tune grain选项改进了grain处理。
2	帧线程现在根据--pools中指定的线程数决定，而不是可用的硬件线程数。 还对映射进行了调整，以提高编码质量，同时对性能的影响最小。
3	CSV日志记录功能（由--csv启用）现在是库的一部分; 它以前是x265应用程序的一部分。 集成libx265的应用程序现在可以通过在库中执行此选项来提取其编码的帧级统计信息。
4	跟踪最小和最大CU大小，切片数量和其他参数的全局变量现已转移到实例特定变量中。 因此，调用多个x265库实例的应用程序不再受限于在多个实例中对这些参数选项使用相同的设置。
5	x265现在可以生成一个单独的库，用于导出HDR10 +解析API。 其他希望使用此API的库可以通过链接此库来实现。 在CMake选项中启用ENABLE_HDR10_PLUS并生成以生成此库。
6	SEA运动搜索从其内核的AVX2优化中获得10％的性能提升。
7	CSV日志现在更加精细，包括PU统计，平均最小 - 最大亮度和色度值等附加字段。有关所有字段的详细信息，请参阅--csv文档。
8	清理了x86inc.asm以改进指令处理。

API变化

API changes	
1	引入了新的API x265_encoder_ctu_info（）以指定帧中各种CTU的建议分区大小。 与--ctu-info一起使用以适当地响应指定的分区。
2	现在，编码器使用传入帧的x265_picture对象传递的速率控制统计数据。
3	对于使用--analysis-reuse-mode load的运行，通过x265_picture中的x265_analysis_data字段共享的传入分析的扩展，重用和重新分析选项; 使用选项--scale， - refine-mv， - refine-inter和--refine-intra来探索。
4	VBV现在具有确定性模式。 使用--const-vbv。

Bug修复

Bug fixes	
1	HDR10 +解析代码的几个问题，包括与用户特定SEI的不兼容，删除警告，链接linux中的问题等。
2	当HDR选项（--hdr-opt， - master-display）指定时，HDR10的SEI消息重复每个keyint。

Version 2.4

发布日期 - 2017年4月22日。

编码器增强功能

Encoder enhancements	
1	支持HDR10 +。 动态元数据可以通过x265_picture的userSEI字段作为比特流提供，也可以作为json jile提供，可以由x265解析并插入到比特流中; 使用--dhdr10-info指定json文件名，使用--dhdr10-opt指定仅在IDR帧处插入色调映射信息的优化，或者当色调映射信息改变时。
2	修改了8,10和12位编码的Lambda表，从而显着提高了主观视觉质量。
3	增强的HDR10编码，具有HDR特定的QP优化功能，可启用WCG内容的色度和亮度平面; 使用--hdr-opt来激活。
4	能够接受来自其他先前编码（可能是或可能不是x265）的分析信息，并选择性地重用和重新分析以编码使用--refine-level选项启用的后续传递。
5	通过启用--limit-tu选项，慢速和veryslow预设可在iso质量下获得20％的速度提升。
6	现在可以通过重新配置的API动态重新配置x265的比特率目标。
7	通过--limit-sao选项引入的性能优化SAO算法; 在更快的预设下看到10％的速度好处。

API变化

API changes	
1	x265_recon图片API现在也接受动态重新配置的速率控制参数。
2	在x265_analysis中添加了几个数据以支持--refine-level：有关更多详细信息，请参阅x265.h。

Bug修复

Bug fixes	
1	在启用SAO的情况下，避免x265 lambda2表中的负偏移。
2	修复mingw32构建错误。
3	除了基于文件的输入之外，现在可以启用管道输入
4	修复静态链接core-utils无法在linux中工作的问题。
5	用VBV修复--multi-pass-opt-distortion的视觉伪像。
6	修复csv中报告的bufferFill统计信息。

Version 2.3

发布日期 - 2017年2月15日。

编码器增强功能

Encoder enhancements	
1	新的基于SSIM的RD成本计算，可提高视觉质量和效率; 使用--ssim-rd进行练习。
2	多通道编码现在可以共享先前通过的分析信息（除了速率控制信息之外），以提高后续通过的性能和质量; 使用--pass选项的多次传递命令行，添加--multi-pass-opt-distortion以共享失真信息，以及--multi-pass-opt-analysis以共享其他分析信息。
3	现在可以使用--lookahead-threads指定用于预测的专用线程池。
4	选项：-dynamic-rd动态增加比特率被VBV限制的区域的分析; 适用于使用VBV设置的CRF和ABR编码。
5	可以使用--opt-cu-delta-qp选项优化用于发信号通知delta-QP的位数; 发现在低比特率目标的某些情况下有用。
6	实验特征选项：-aq-motion根据块相对于帧移动的相对运动添加新的QP偏移。

API变化

API changes	
1	Recon fi aure API现在支持发信号通知新的缩放列表。
2	x265应用程序的csv功能现在报告对每个帧进行编码所花费的时间（以毫秒为单位）。
3	当达到的比特率低于目标时，--strict-cbr通过添加填充比特来实现更严格的比特率依从性; 早些时候，它只是在实现的速度更高时作出反应。
4	--hdr可用于确保始终发信号通知max-cll和max-fall值（即使为0,0）。

Bug修复

Bug fixes	
1	修复了MacOS平台上不正确的HW线程计数问题。
2	固定缩放列表支持4：4：4视频。
3	通过从成本计算中删除最后一个切片的QP，输出fi x for -opt-qp-pss不一致。
4	VTune pro fi ling（使用ENABLE_VTUNE CMake选项启用）现在也适用于2017 VTune版本。

Version 2.2

发布日期 - 2016年12月26日。

编码器增强功能

Encoder enhancements	
1	增强TU选择算法，提前提高速度; 使用--limit-tu来锻炼身体。
2	现在支持新的运动搜索方法SEA（连续消除算法）：选项：-me 4
3	通过--opt-qp-pps， - opt-ref-list-length-pps和--multi-pass-opt-rps，比特流优化可以改善PPS和SPS中的字段以节省比特率。
4	在没有WPP的情况下编码时使用VBV约束启用。
5	选择信息时，所有param选项都会在比特流中转储到SEI数据包中。
6	x265现在支持基于POWERPC的系统。 几个关键功能还具有优化的ALTIVEC内核。

API变化

API changes	
1	从比特流禁用SEI和可选-VUI消息的选项更具描述性。
2	新选项--scenecut-bias可以控制偏差以通过cli标记场景切换。
3	支持单声道和单声道16色彩空间，用于y4m输入。
4	- 由于视觉质量的原因，不再支持64分钟的min-cu大小（反正早先崩溃了。）
5	CSV的API现在需要版本字符串，以便更好地将x265集成到其他应用程序中。

Bug修复

Bug fixes	
1	几个基于切片的编码。
2	--log2-max-poc-lsb的范围根据HEVC规范限制。
3	在编码时将MV限制在合法边界内。

Version 2.1

发布日期 - 2016年9月27日

编码器增强功能

Encoder enhancements	
1	支持qg-size为8
2	支持在场景切换时插入非IDR I帧以及使用固定GOP设置运行时（min-keyint = max-keyint）
3	切片平行度的实验支持。

API变化

API changes	
1	编码通过x265_picture对象传入的用户定义的SEI消息。
2	禁用比特流中的SEI和VUI消息
3	指定qpmin和qpmax
4	控制编码POC的位数。

Bug修复

Bug fixes	
1	QP波动fi x用于mini-GOP中的第一个B帧，用于具有调谐纹理的2遍编码。
2	从dct_sse4以32位的崩溃装配fix。
3	在Windows平台中创建线程池。

Version 2.0

发布日期 - 2016年7月13日

新功能

New Features	
1	uhd-bd：支持Ultra-HD Bluray
2	rskip：允许跳过递归以使用不同rd级别的启发式分析较低的CU大小。 在最高质量的预设下提供良好的视觉质量增益。
3	rc-grain：为粒状内容特定地启用新的速率控制模式。 严格防止帧内和帧之间的QP振荡，以避免晶粒波动。
4	tune grain：一种完全重构和改进的选项，用于编码薄膜颗粒内容，包括QP控制和分析选项。
5	asm：ARM程序集现在默认启用，armv6及更高版本系统支持本机或交叉编译的版本。

Misc

1.更正了SSIM计算错误。

Version 1.9

发布日期 - 2016年1月29日。

新功能

New Features	
1	量子偏移：此功能允许为每个帧指定块级量化偏移。 仅限API的功能。
2	-intra-refresh：关键帧可以由非关键帧中的移动的内部块列替换。
3	-limit-modes：智能地限制模式分析。
4	用于亮度限幅的-max-luma和-min-luma，可用于HDR用例
5	现在默认以非常低的比特率VBV编码启用紧急去噪

API变化

API changes	
1	x265_frame_stats返回许多其他字段：maxCLL，maxFALL，剩余能量，场景切换和延迟记录
2	-qp文件现在支持frametype'K“
3	x265现在允许CRF速率控制在通过N（N大于或等于2）
4	现在完全支持和测试色度子采样格式YUV 4：0：0

Presets 和 Performance

Presets and Performance	
1	最近添加的功能前瞻切片，限制模式，限制参考已默认启用适用的预设。
2	默认的心理强度已增加到2.0
3	多套接口机器现在使用可以跨套接字工作的单个线程池。

Version 1.8

发布日期 - 2015年8月10日

API变化

API changes	
1	现在启用了对Main12的实验支持。 存在部分装配支持。
2	现在支持Main12和Intra / Still图片配置文件。 基于x265_param :: totalFrames检测静态图片配置文件。
3	现在可以通过API获得三类编码统计信息。 a）x265_stats - 包含编码统计信息，可通过x265_encoder_get_stats（）获得b）x265_frame_stats和x265_cu_stats - 包含帧编码统计信息，可通过recon x265_picture获得4. -csv a）x265_encoder_log（）现已弃用b）x265_param :: csvfn也是 已弃用5. -log-level现在仅控制控制台日志记录，已删除帧级控制台日志记录。 6.增加了对新颜色传输特性ARIB STD-B67的支持

新功能

New Features	
1	limit-refs：此功能限制为单个CUS分析的参考。 提供效率和性能之间的良好折衷。
2	aq-mode 3：一种新的aq模式，可为低光条件提供额外的偏置。
3	改进的场景剪切检测逻辑，允许速率控制更好地管理淡入和淡出的视觉质量。

预设和调整选项

Preset and Tune Options	
1	tune grain：将psyRdoq强度增加到10.0，将rdoq-level增加到2。
2	qg-size：默认值更改为32。