

# 《FFmpeg Basics》中文版

---

转载自简书-[张芳涛](#)

原地址：<https://www.jianshu.com/p/835aee0a5a0a>

由loongmonkey整理并排版。如有错漏，欢迎给我写邮件[loongmonkey@qq.com](mailto:loongmonkey@qq.com)，  
或在我的开源项目<https://github.com/loongmonkey/Multimedia-development>上提意见  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## 0.简介

---

### 欢迎

亲爱的读者们，

欢迎来到这本书，它将使您熟悉FFmpeg项目的许多有趣的特性。下面的几个大公司都是FFmpeg使用者：

- Facebook，最大的社交网络，用FFmpeg技术处理用户的视频。
- Google Chrome，流行的web浏览器，使用FFmpeg库支持HTML5音频和视频支持。
- YouTube，最大的视频分享网站，用ffmpeg技术将上传的视频进行转换。

本书的重点是解释诸如调整大小、裁剪、填充、去噪、覆盖等基本的视频编辑功能，但包含了更复杂的处理和实验的指导内容。

章数字音频描述如何转换和创建音频，声音处理是在[批处理文件](#)章节和[先进技术点](#)章节。

### 第一步

第一步是下载FFmpeg二进制文件，如果还没有完成，具体的细节将在[FFmpeg基本介绍](#)或[专门的网站](#)上。许多Linux发行版已经安装了FFmpeg工具，或者高级用户可以编译自己的二进制文件。

第一章包含FFmpeg项目的基本信息，以及如何使用工具简化工作。如果已经熟悉这些数据，或者在开始时看起来太专业，那么您可以进入[显示帮助和功能](#)，开始输入各种ffmpeg命令。

请注意，本书中的许多命令都是简化的，以说明当前解释的特性和一些参数，特别是在转换中，细节在[格式之间的转换](#)章节中。

### 针对这本书专门设立的网站

[ffmpeg.tv](#) 专门为这本书建立了一个网站，该网站包括如下内容：

- 书的索引、目录和书的描述。
- 在视频格式的书中的例子，视频在特定的章节。
- 用户论坛讨论图书主题和各种想法和观点。
- 勘误列表
- 联系方式
- 来自FFmpeg邮件列表里面(不断更新)的最新的40篇文章

## 简介

## 协议

应该在命令行上输入的文本是用衬线的比例字体打印的，例如：

```
ffmpeg -i input.mpg -q 1 output.avi
```

应该用特定文本替换的命令部分用斜体印刷，例如：

```
ffmpeg -i input -vf mp=denoise3d -s vga output
```

控制台输出以无衬线的比例字体打印：

```
Muxer avi [AVI (Audio Video Interleaved)]:  
Common extensions: avi.  
Mime type: video/x-msvideo.  
Default video codec: mpeg4.  
Default audio codec: mp3.
```

蓝色插入符号^表示命令太长被印在书中一行,继续另一个,但在计算机仍然是1条线命令,例如:

```
ffplay -f lavfi -i color=c=white ^  
-vf drawtext=fontfile=/Windows/Fonts/arial.ttf:text=Welcome
```

请注意单词之间的空间在前面的示例中,白色和^空间表明,空间也在命令行上。在批处理文件中需要这种形式的符号,这将在章节批文件中解释。

# Important

Many examples in the book  
are simplified to explain  
the current item, so some  
parameters are omitted  
and used are defaults,  
details are in the chapter

## Conversion Between Formats.

Common omitted options include  
bitrate, codec, frame rate, etc.

简介，看不懂的下面有翻译

- 我把图片内容翻译一下：注意：在这本书里面有很多例子都在解释当前这一项，所以有一些参数被省略掉了并且用的都是默认的值，具体的信息在捕获格式之间的转换里面。(常见的省略选项包括比特率、编解码器、帧速率等)。

为了更好的定位，本书包含了FFmpeg元素的颜色区分，如过滤器、设备、源和其他项目。

与音频和视频有关的设备、过滤器等的颜色区分。	
只有音频	黄色
只有视频	蓝色
既有音频也有视频	粉色

我在简书上画不出来这种表格，用的excel做的



看不懂没关系，下面有翻译

- 我把上面的内容翻译一下：请注意：FFmpeg的工具和文库经常更新并且书上经常使用的命令或者其他信息也会发生变化。请登录[www.ffmpeg.tv](http://www.ffmpeg.tv)查看更新的条目。邮箱地址：[book@ffmpeg.tv](mailto:book@ffmpeg.tv)

## 您的反馈是非常重要的

关于FFmpeg工具的许多选项和参数不能在本书中描述，大约有200页，您的可以改进的意见，在下一版中将会出现。

在通过电子邮件发送查询之前，请访问[www.ffmpeg.tv](http://www.ffmpeg.tv)并在论坛或FAQ上搜索，它将避免重复的问题，在某些情况下它会提供即时帮助。

非常感谢，并致以良好的祝愿。

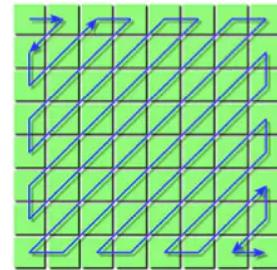


## 01-FFmpeg基本介绍

为了最优化地使用各种FFmpeg组件，需要正确理解FFmpeg的基本属性和特性。如果你基础差了点，看了不知道说的啥，您可以继续下一章，并在遇到需要了解的内容的时候再回来看。

### FFmpeg介绍

FFmpeg是根据GNU通用公共许可证获得许可的多媒体处理自由软件项目的名称。该项目最受欢迎的部分是用于视频和音频编码/解码的ffmpeg命令行工具，其主要特点是速度快，输出质量高和文件大小比较小。 FFmpeg中的“FF”表示 媒体播放器上的表示“快进”的控制按钮，“mpeg”是Moving Pictures Experts Group的缩写。 FFmpeg标志包含Z形图案，这是图片中以8x8块图示的熵编码方案的特征。



知道FFmpeg图标啥意思了吧

FFmpeg命令行工具

<b>ffmpeg</b>	快速音频和视频编码器/解码器
<b>ffplay</b>	媒体播放器
<b>ffprobe</b>	显示媒体文件的特点
<b>ffserver</b>	使用HTTP和RTSP协议进行多媒体流的广播服务器

FFmpeg软件库

<b>libavcodec</b>	各种多媒体编解码器的软件库
<b>libavdevice</b>	软件库的设备
<b>libavfilter</b>	软件库包含过滤器
<b>libavformat</b>	媒体格式的软件库
<b>libavutil</b>	包含各种实用程序的软件库
<b>libpostproc</b>	用于后期处理的软件库
<b>libsamplerate</b>	用于音频重采样的软件库
<b>libswscale</b>	用于媒体扩展的软件库

所有组件的编程语言是C语言，源代码可以在Linux/Unix、Windows、Mac OS X等系统上编译。

本书是使用官方二进制版本在Microsoft Windows上创建的，但几乎所有的指令和示例都应该在其他操作系统上无任何更改的情况下运行。有关启用的选项的详细信息，请参阅[词汇表](#)中的FFmpeg配置条目。

## FFmpeg开发者

该项目由Fabrice Bellard于2000年开始，Fabrice Bellard是QEMU和Tiny C Compiler的创建者，也是一位出色的程序员。现在该项目由FFmpeg团队维护，开发人员来自许多国家，主要的开发人员可以参与合同工作：

姓名	地址	专长
Baptiste Coudurier	美国. 洛杉矶	他在广播codecs (ProRes, DNxHD, IMX/D-10, AVC-Intra), 格式(MXF, GXF, MOV)和用法(Avid, FCP, Interlacing, Time Code, Metadata)中有专业的专业知识
Benjamin Larsson	斯德哥尔摩, 瑞典	他的专业领域是音频编解码器
Diego Biurrun	德国 亚琛	他在许可证法规遵循工程和构建系统方面具有特殊的专长
Jason Garrett-Glaser	洛杉矶. 美国	他是x264的主要开发人员, 在H.264和其他现代有损视频格式以及x86 SIMD装配优化方面具有专业的专业知识
Luca Barbato	意大利都灵	他在流媒体协议方面有特殊专长
Michael Niedermayer	奥地利的维也纳	他是视频编码和x86汇编领域的专家
Stefano Sabatini	卡利亚里. 意大利	他在libavfilter, ff工具的使用和可用性问题上有特别的专长

## 参与FFmpeg开发

任何人都可以通过在网页上加入特定的邮件列表来参与FFmpeg的开发和更新:

<http://www.ffmpeg.org/contact.html>

表中列出了可用的邮件列表:

FFmpeg邮件列表

<b>ffmpeg-user</b>	对于常见的用户问题, 如编译问题、命令行问题和类似的问题
ffserver-user	对于ffserver用户的问题, 比如配置和流媒体问题
libav-user	对于应用程序开发人员有关使用FFmpeg库开发的问题
ffmpeg-devel	对于FFmpeg本身的开发, 不是为了开发使用FFmpeg库的软件, 也不是为了bug报告
ffmpeg-cvslog	对于FFmpeg源/主要git存储库的所有更改
ffmpeg-trac	对于FFmpeg Trac的所有修改

## FFmpeg 下载

主要下载来源位于如下网页:

<http://ffmpeg.org/download.html>

Windows的用户可以从如下网页下载二进制文件(推荐使用静态构建):

<http://ffmpeg.zeranoe.com/builds>

许多Linux发行版已经安装了FFmpeg工具, 否则它们可以被编译, 这在OS X上也是可能的, 或者OS X二进制文件可以从web页面下载:

<http://www-evermeet.cx/ffmpeg> or <http://ffmpegmac.net>

## 命令行语法

ffmpeg命令行工具的语法相对简单, 重要的是在正确的位置键入所需的参数, 而不是在各种输入和输出之间混合选项。ffmpeg命令的一般结构如下, 需要注意的是全局选项影响所有输入和输出:

```
ffmpeg [global options] [input file options] -i input_file [output file options]
output_file
```

**Command Syntax of ffmpeg**

**ffmpeg**

***global options***

***input1\_options -i input1***

***input2\_options -i input2***

***... additional input files, streams, etc.***

***output1\_options output1***

***output2\_options output2***

***... additional output files, streams...***

Example: **ffmpeg -y -i video.avi -s vga video.mp4**

global option	input 1	output 1 option	output 1
---------------	---------	-----------------	----------

**items in blue** - required, **items in green** - optional  
**items in italic style** - will be replaced with actual data  
**input1 ... inputN, output1 ... outputN** - strings specifying files, pipes, network streams, grabbing devices, etc.

FFmpeg语法, 大家能看懂不? 看不懂的话看看Example, 哪一部分是全局的设置选项, 哪一部分是第一个视频的设置选项, 所有的结构都是这种模式, 看多了就可以了

## Windows命令提示符及其替代方法

Windows上的ffmpeg命令行工具是通过命令提示符来管理的, 可以通过Windows ->所有程序->附件->命令提示。它也可以从一个快捷方式Win+R开始, 然后键入cmd, 然后输入。

```

C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Frantisek>ffmpeg
ffmpeg version N-42547-g7543fd8 Copyright (c) 2000-2012 the FFmpeg developers
  built on Jul 15 2012 21:36:07 with gcc 4.7.1
    configuration: --enable-gpl --enable-version3 --disable-w32threads --enable-rudimental --enable-cpudetect --enable-avisynth --enable-bzlib --enable-frei0r --enable-libass --enable-libcelt --enable-libopencore-amrnb --enable-libopencore-amrwb --enable-libfreetype --enable-libgsm --enable-libmp3lame --enable-libnut --enable-libopejpeg --enable-librtmp --enable-libschroedinger --enable-libspeex --enable-libtheora --enable-libutvideo --enable-libvo-aacenc --enable-libvo-amrwbenc --enable-libvorbis --enable-libvpx --enable-libx264 --enable-libxavs --enable-libxvid --enable-zlib
      libavutil      51. 65.100 / 51. 65.100
      libavcodec     54. 40.100 / 54. 40.100
      libavformat    54. 16.104 / 54. 16.104
      libavdevice    54.  1.100 / 54.  1.100
      libavfilter     3.  2.100 / 3.  2.100
      libswscale      2.  1.100 / 2.  1.100
      libswresample   0. 15.100 / 0. 15.100
      libpostproc    52.  0.100 / 52.  0.100
Hyper fast Audio and Video encoder
usage: ffmpeg [options] [[infile options] -i infile]... <[outfile options] outfile>...
Use -h to get full help or, even better, run 'man ffmpeg'

C:\Users\Frantisek>_

```

Windows命令提示符在关闭时不会保存所使用的命令的历史，因为有一些具有额外特性的免费应用程序，如文件管理、编辑、宏、FTP客户端等，建议选择高级的FFmpeg工具程序。下一个表描述了几种免费的替代方案。

#### Windows命令提示符选择

名称	下载	描述
FAR Manager	<a href="http://farmanager.com">farmanager.com</a>	- 文件管理器与shell, 编辑器, ftp客户端 - 命令行完成, 快捷键, 宏, 插件 - 两个窗口, 可定制的界面
PyCmd	<a href="http://sourceforge.net/projects/pycmd">sourceforge.net/projects/pycmd</a>	- 选项卡完成, 持续的历史.....
Console	<a href="http://sourceforge.net/projects/console">sourceforge.net/projects/console</a>	- 多个标签, 可配置
Gregs DOS Shell	<a href="http://gammadyne.com/cmdline.htm#gss">gammadyne.com/cmdline.htm#gss</a>	- 改进编辑, 命令历史, 支持Aero Glass等
TCC/LE	<a href="http://jpssoft.com/all-downloads/downloads.html">jpssoft.com/all-downloads/downloads.html</a>	- 包括111个内部命令, 103个内部变量, 140个变量函数

接下来的行描述了最好的替代FAR管理器，在Linux上它可以用一个类似的应用程序Midnight Commander替代，如果安装了它，它将从控制台的mc命令启动。

FAR Manager是一个受欢迎的文件管理器、编辑器和FTP客户端，它支持宏、插件和其他高级特性。用户界面是高度可定制的，并被翻译成多种语言。下一幅图展示了它的命令历史窗口，它在输入新命令时显示，因此用户可以轻松地选择之前使用的命令并最终编辑它。

```

[Far] [C:\media] - Far 2.0.1666.x86 12:05
Left   Files   Commands   Options   Right
libavdevice      54.  2.100 / 54.  2.100
libavfilter       3.  5.101 / 3.  5.101
libswscale        2.  1.100 / 2.  1.100
libswresample     0. 15.100 / 0. 15.100
libpostproc       52.  0.100 / 52.  0.100
Hyper fast Audio and Video encoder
usage: ffmpeg [options] [infile options] -i infile... <outfile options> outfile...
Use -h to get full help or, even better, run 'man ffmpeg'

C:\media>ffmpeg -f lavfi -i color=c=blue:s=vga:f=blue'
FFmpeg version N-43060-ga85b4a5 Copyright (c) 2000-2012 the FFmpeg developers
  built on Jul 30 2012 13:14:08 with gcc 4.7.1 (GCC)
configuration: --enable-gpl --enable-version3 --disable-w32threads --enable-runtime-cpudetect --enable-avisynth --enable-bzlib
--enable-frei0r --enable-libass --enable-libclt --enable-libopencore-amrnb --enable-libopencore-amrnb --enable-libfreetype --enable-libgnome
--enable-libgsm --enable-libmp3lame --enable-libnut --enable-libopenjpeg --enable-librtmp --enable-libschroedinger --enable-libspeex
--enable-libtheora --enable-libutvideo --enable-libvorbis --enable-libvo-aacenc --enable-libvo-amrwbenc --enable-libvorbis --enable-libvpx --enable
libx264 --enable-libxvid --enable-libxvid
libavutil      54. 6.100 / 54. 6.100
libavcodec     54. 45.100 / 54. 45.100
libavformat    54. 22.100 / 54. 22.100
libavdevice     54.  2.100 / 54.  2.100
libavfilter      3.  5.101 / 3.  5.101
libswscale       2.  1.100 / 2.  1.100
libswresample    0. 15.100 / 0. 15.100
libpostproc      52.  0.100 / 52.  0.100
[lavfi @ 0x2eef20] Estimating duration from bitrate, this may be inaccurate
Input #0, rawvideo, from 'color=c=blue:s=vga:f=blue':
  Duration: N/A, start: 0.000000, bitrate: N/A
    Stream #0:0: Video: rawvideo (I420 / 0x30323449), yuv420p, 320x240 [SAR 1:1 DAR 4:3], 25 tbr, 25 tbn, 25 tbc
[tsd1 @ 0x3952401 v:640 h:480 fnt:yuv420p sar:1/1 -> u:640 h:480]
Output #0:
Metadata:
  encoder         : ffmp
Stream
Stream name: ffmp
Stream name: ffmp
Press [q] to stop, [?] for help
frame=  2  ffmp -f lavfi -i color=c=blue:s=vga:f=blue -t 30 blue.mp4
video:110
ffmp -i blue5.mp4 overlay.suf

```

对于其高级文件编辑器，在创建ffmpeg批处理时，FAR Manager会很有用，这在[批处理文件](#)章节中有介绍。还需要文件编辑器来将媒体文件包含在网页中-[“网络视频”](#)章节的主题。界面的自定义开始于F9键并支持选择选项选项卡。

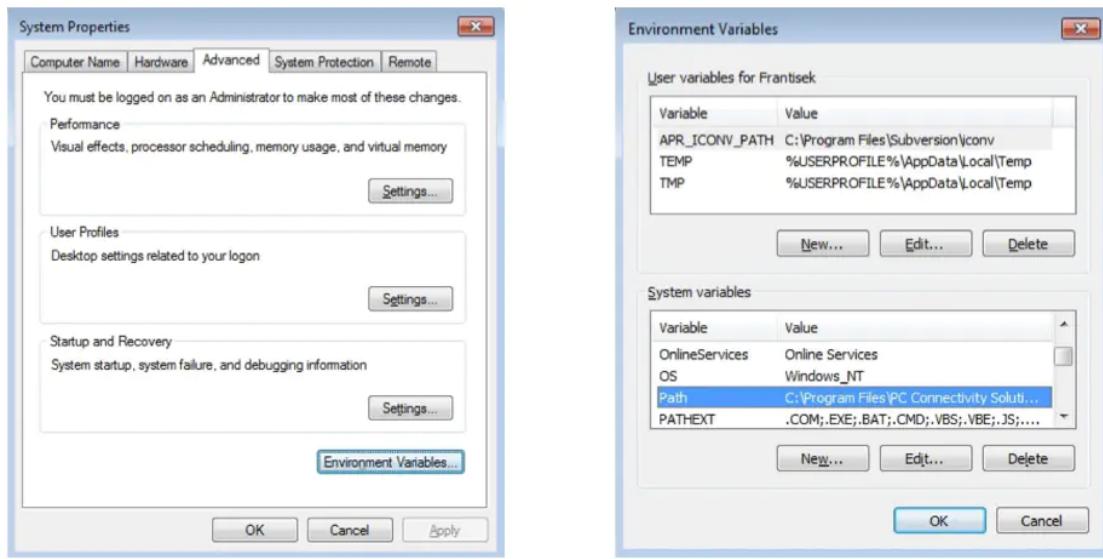
## 路径设置

将下载的FFmpeg命令行实用程序（ffmpeg.exe, ffplay.exe, ffprobe.exe）复制到环境变量Path部分中包含的目录是可以实现的，因此可以从任何目录调用它们而无需编写他们完整的路径。

或者，您可以将FFmpeg程序复制到其他目录，例如C: \ media，然后通过控制面板 ->系统和安全 ->系统 ->高级系统设置将此文件夹添加到系统路径。请点击环境变量按钮，向下滚动系统变量的滚动条，点击路径，然后点击编辑按钮。在弹出的窗口中单击编辑系统变量值字段，将光标移动到行尾，添加文本

```
;C:\media
```

点击OK按钮。分号分隔特定的目录，别复制。



对于命令提示符的当前会话，可以使用命令设置路径：

```
set path=%path%;C:\path_to_ffmpeg.exe
```

例如，如果文件ffmpeg.exe被复制到目录C:\media，命令是：

```
set path=%path%;C:\media
```

## 重命名为缩写形式（懒人专区）

命令名称ffmpeg有6个字符，如果每次用的话都输入"ffmpeg"这六个字符就太麻烦了，想个简单的办法，建议将文件ffmpeg.exe重命名为f.exe（ffplay.exe至fp.exe等）或类似的简写形式以保存时间并防止被误认。在命令提示符中，你可以使用以下命令：

```
ren ffmpeg.exe f.exe
```

为了清晰起见，本书总是使用完整的命令形式ffmpeg。

## 显示输出预览

在各种视频测试中，我们可以通过直接在屏幕上显示命令输出来节省大量时间，而不是将其保存到文件中，也不是在媒体播放器中预览。

## FFplay媒体播放器预览

使用简化的命令，而不是使用ffmpeg工具生成一个新文件。

```
ffmpeg -i input_file ... test_options ... output_file
```

我们可以使用将显示与ffmpeg完全相同的ffplay，使用该命令保存到文件中

```
ffplay -i input_file ... test_options
```

## 使用SDL输出设备预览。

此预览由表中描述的SDL(简单的DirectMedia层)输出设备生成:

输出设备: SDL

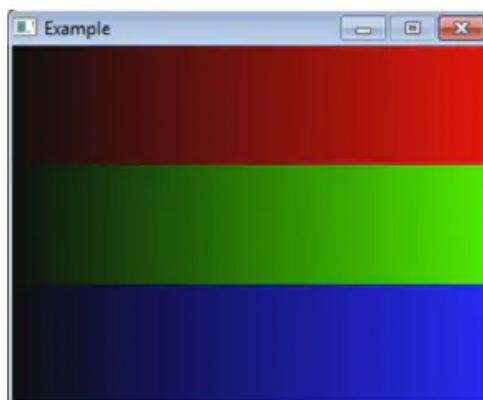
描述	在SDL窗口中显示视频流，需要安装libsdl库
语法	[-icon_title i_title] [-window_size w_size] [-window_title w_title] -f sdl output

描述设备的选择

icon_title	名称为iconified SDL窗口，默认值为window_title值
window_size	SDL窗口大小，widthxheight或缩写，默认是输入视频的大小
window_title	窗口标题，默认为输出设备指定的文件名

请注意，SDL设备只能显示具有yuv420p像素格式的输出，并且使用其他输入类型的选项-pix\_fmt具有一个值yuv420p必须被预先处理，否则会显示一个错误，例如:

```
ffmpeg -f lavfi -i rgbsrc -pix_fmt yuv420p -f sdl Example
```



## 可以在FFmpeg中使用的SI前缀

当指定数值各种ffmpeg选项如比特率或最大文件大小可以使用常见的SI后缀:K:千(103),M为百万(106),G十亿(10^ 9),等下一个示例指定一个新的比特率1.5 mbps的输出文件,所有命令给相同的结果:

```
ffmpeg -i input.avi -b:v 1500000 output.mp4
ffmpeg -i input.avi -b:v 1500K output.mp4
ffmpeg -i input.avi -b:v 1.5M output.mp4
ffmpeg -i input.avi -b:v 0.0015G output.mp4
```

请注意，在FFmpeg文档中，SI前缀被称为后缀，因为它们必须在数值之后立即输入。

后缀B(字节)可以在ffmpeg选项中使用数值，并将数值乘以8。它可以与其他前缀相结合，以表示千字节(KB)、兆字节(MB)等。例如，为输出文件设置10兆字节的最大文件大小，可以使用下一个命令：

```
ffmpeg -i input.mpg -fs 10MB output.mp4
```

SI prefixes available in FFmpeg								
for parts (negative)				for multiples (positive)				
Symbol	Prefix	decimal base 10	binary base 2	Symbol	Prefix	decimal base 10	binary base 2	
y	yocto-	-24	-80	h	hecto-	2		
z	zepto-	-21	-70	k, K	kilo-	3	10	
a	atto-	-18	-60	M	mega-	6	20	
f	femto-	-15	-50	G	giga-	9	30	
p	piko-	-12	-40	T	tera-	12	40	
n	nano-	-9	-30	P	peta-	15	50	
μ	mikro-	-6	-20	E	exa-	18	60	
m	milli-	-3	-10	Z	zetta-	21	70	
c	centi-	-2		Y	yotta-	24	80	
d	deci-	-1						

这种嵌套的表格类型好像在简书上做不出来

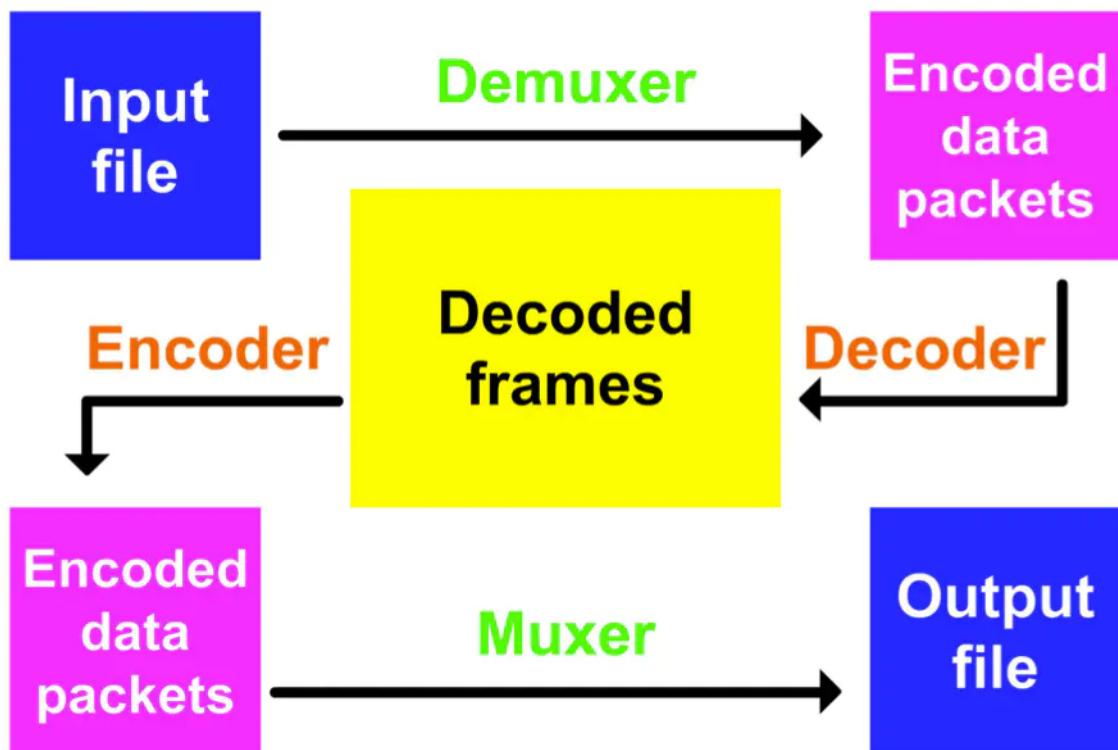
## FFmpeg的代码转换

ffmpeg程序读入内存中指定的任意数量的输入的内容，根据输入的参数或程序的默认值对其进行处理，并将结果写入任意数量的输出。输入和输出可以是计算机文件、管道、网络流、抓取设备等。

在代码转换过程中，ffmpeg在libavformat库中调用demuxers来读取输入，并从数据包中获取编码数据。如果有更多的输入，ffmpeg可以通过跟踪任何活动输入流的最低时间戳来保持它们的同步。然后解码器从编码的数据包中生成未压缩的帧，在可选的过滤后，帧被发送到编码器。编码器产生新的编码包，它被发送到muxer并写入到输出。

FFmpeg工具的重要部分是过滤器，它可以被组织成过滤链和filtergraphs。Filtergraphs可以是简单的或复杂的。在解码源和编码输出之间实现滤波处理。转码过程在下一个图中说明。

# Transcoding in ffmpeg



FFmpeg的代码转换

## Filters, filterchains, filtergraphs

在多媒体处理中，术语过滤器是指在编码到输出之前修改输入的软件工具。过滤器分为音频和视频过滤器(请参阅词汇表中的过滤器)。FFmpeg内置了许多多媒体过滤器，可以通过多种方式组合它们。具有复杂语法的命令根据指定的参数从一个过滤器到另一个过滤器。这简化了媒体处理，因为有损压缩的媒体流的多译码和编码会降低整体质量。FFmpeg的过滤API(应用程序编程接口)是libavfilter软件库，它允许过滤器有多个输入和输出。过滤器包括在输入和输出之间使用-vf选项的视频过滤器和-af选项音频过滤器。例如，下一个命令生成一个测试模式顺时针旋转90°使用转置过滤器(7中描述。章)：

```
ffplay -f lavfi -i testsrc -vf transpose=1
```

下一个示例使用atempo音频过滤器将输入音频的速度降低到80%：

```
ffmpeg -i input.mp3 -af atempo=0.8 output.mp3
```

# Filter, filterchain and filtergraph

## Filter syntax

[input\_link\_label1][input\_link\_label2]...  
filter\_name=parameters  
[output\_link\_label1][output\_link\_label2]...

in blue - required  
in green - optional

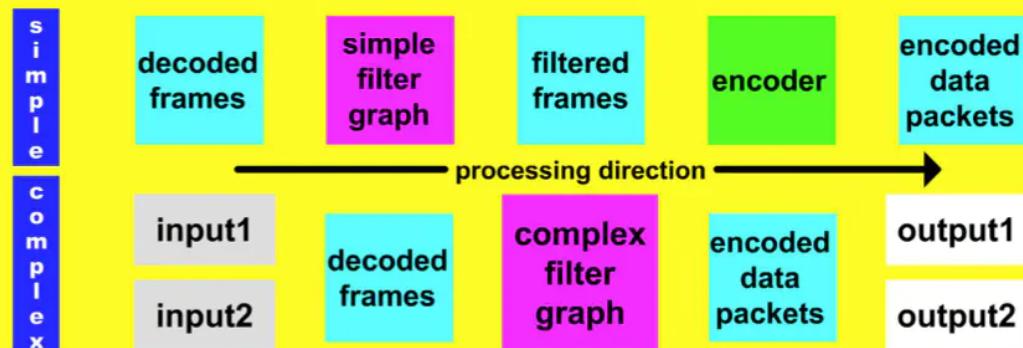
Filterchain = sequence of comma-separated filter descriptions

"filter1,filter2,filter3, ... filterN-2,filterN-1,filterN"

Filtergraph = sequence of semicolon-separated filterchain descriptions

"filterchain1;filterchain2; ... filterchainN-1;filterchainN"

## Diagram of simple and complex filtergraph



Example: ffmpeg -i video.avi -i logo.png -filter\_complex overlay=iw-50:ih-30 out.mp4

过滤器通常用于filterchains(逗号分隔的过滤器序列)和filtergraphs(分号分隔的filterchains序列)。如果使用了任何空格，那么filterchain必须用引号括起来。在filtergraphs中，可以使用表示所选filterchain输出的链接标签，并可以在以下的filtergraphs中使用。例如，我们希望将输入视频与hqdn3d过滤器输出的输出进行比较。如果没有filtergraphs，我们必须至少使用两个命令，例如：

```
ffmpeg -i input.mpg -vf hqdn3d,pad=2*iw output.mp4
ffmpeg -i output.mp4 -i input.mpg -filter_complex overlay=w compare.mp4
```

使用带有链接标签的filtergraph，就只有一个命令：

```
ffplay -i i.mpg -vf split[a][b];[a]pad=2*iw[A];[b]hqdn3d[B];[A][B]overlay=w
```

分割过滤器将输入分为2个输出标签[a]和[b]，然后将[a]链接用作第二个filterchain的输入，它为标记[a]的比较创建了一个pad。[b]链接被用作第三个filterchain的输入，它创建一个标记为[b]的输出。最后一个filterchain使用[A]和[B]标签作为覆盖过滤器的输入，从而产生最终的比较。另一个例子是在下一个图中。

## Link labels in filterchains and filtergraphs

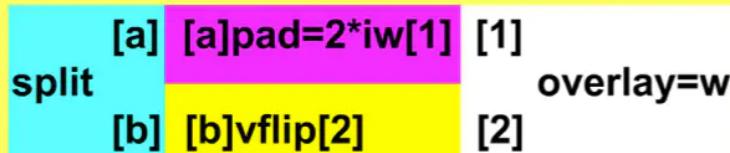
## Filter syntax

**[in\_link1]...[in\_linkN]filter\_name=parameters[out\_link1]...[out\_linkM]**

**Link label** is a name of the link associated with the input or output pad of the filter.

## Example of a complex filtergraph

**split[a][b];[a]pad=2\*iw[1];[b]vflip[2];[1][2]overlay=w**  
filterchain 1      filterchain 2      filterchain 3      filterchain 4



- F1: The split filter creates 2 copies of the input with link labels [a] and [b].
  - F2: Link [a] forms the input for the pad filter that outputs double width to a link [1].
  - F3: Link [b] forms the input for the vflip filter that flips its content to the link [2].
  - F4: Links [1] and [2] are inputs to the overlay filter that places the [2] link content beside the [1] link content.



### **Complete command and displayed result:**

```
ffplay -f lavfi -i rgbtests -vf split[a][b];[a]pad=2*iw[1];[b]vflip[2];[1][2]overlay=w
```

## 选择的媒体流

一些媒体容器如AVI、Matroska、MP4等可以包含多种类型的流，FFmpeg可以识别5种流类型：音频(a)、附件(t)、数据(d)、字幕(s)和视频(v)。

使用-map选项选择流，然后使用流说明符和语法：

file number:stream type[:stream number]

File\_number和stream\_number也表示为file\_index和stream\_index，从0开始计数，这意味着第一个是0，第二个是1，等等。

- `-map 0`选择所有类型的所有流。
  - `-map i:v`从文件中选择所有的视频流，用`i`(index)，`-map i:a`选择所有的音频流，`-map i:s`选择所有字幕流，等等。
  - 特殊选项`--vn`，`-sn`分别排除所有音频、视频或字幕流。

如果输入文件包含更多相同类型的流，而-map选项不被使用，那么选择的是每种类型的1个流。例如，如果文件包含2个视频流，选择的是具有较高分辨率的，对于音频选择的流有更多的通道，等等，细节如下图所示：

## Selection of streams from input(s) to output

### 1. Default: selected is 1 stream only of each type

**audio** - stream with the most channels

**subtitle** - first subtitle stream

**video** - stream with the highest resolution

If more audio or video streams have the same parameters, the first one is selected.

### 2. Manual: with **-map** option

**-map [-]inputfile\_id[:stream\_specifier][,syncfile\_id[:stream\_specifier]]**

file1 streams specifier

1st video 0:v:0

2nd video 0:v:1

1st audio 0:a:0

2nd audio 0:a:1

1st subtitle 0:s:0

2nd subtitle 0:s:1

3rd subtitle 0:s:2

file2 streams specifier

1st video 1:v:0

1st audio 1:a:0

1st subtitle 1:s:0

Example of the command

ffmpeg -i file1 -i file2 **selected\_streams** output

Examples of selected streams

a) all streams from both files

-map 0 -map 1

b) file1: 3rd subtitle, file2: 1st video, 1st audio

-map 0:s:2 -map 1:v:0 -map 1:a:0

c) file1: 2nd video, file2: 1st subtitle, no audio

-map 0:v:1 -map 1:s:0 -an

d) all streams except 1st video and 2nd audio in file1

-map 0 -map 1 -map -0:v:0 -map -0:a:1

Complete example, selected is 1. video from A.mov, 1. audio from B.mov and 1. subtitles from C.mov.

ffmpeg -i A.mov -i B.mov -i C.mov -map 0:v:0 -map 1:a:0 -map 2:s:0 clip.mov

Stream types: a - audio, d - data, s - subtitles, t - attachment, v - video

除了特定的-map选项，流指定符还与许多其他选项一起使用:

流的形式说明符

说明符形式	描述
stream_index	选择该索引的流(编号)
stream_type[:stream_index]	stream_type为字母a(音频)、d(数据)、s(字幕)、t(附件)或v(视频);如果添加了stream_index, 它将选择该类型的流并使用给定的索引, 否则它将选择该类型的所有流
p:program_id[:stream_index]	如果添加了stream_index, 那么使用给定的program_id在程序中选择带有stream_index的流, 否则将选择该程序中的所有流
stream_id	按格式指定的ID选择流

例如, 为了设置音频和视频的使用-b选项的比特率, 我们可以使用以下命令:

```
ffmpeg -i input.mpg -b:a 128k -b:v 1500k output.mp4
```

## Lavfi虚拟设备

在前面的部分中, 我们使用了一个带有lavfi值的-f选项, 其中lavfi是在表中描述的libavfilter虚拟输入设备的名称:

	<b>输入设备:lavfi</b>
描述	从filtergraph的打开的输出pad中处理数据，对于每个输出垫创建一个对应的流，映射到编码。filtergraph由一个-graph选项指定，目前只支持视频输出垫
语法	-f lavfi [-graph[ -graph_file]]
	lavfi选项的描述
-graph	作为输入的filtergraph，每个视频的开放输出必须用一个“outN”形式的唯一字符串标记，其中N是一个数字，从0开始，对应于设备生成的映射的输入流。第一个未标记的输出将自动分配给“out0”标签，但是其他所有的输出都需要显式指定。如果没有指定，则默认为输入设备指定的文件名
-graph_file	filtergraph的文件名被读取并发送到其他过滤器，filtergraph的语法与选项-graph指定的语法相同

Lavfi通常用于显示测试模式，例如带有命令的SMPTE条：

```
ffplay -f lavfi -i smptebars
```

其他经常使用的源是可以用命令显示的颜色源：

```
ffplay -f lavfi -i color=c=blue
```

## 颜色名称

一些视频过滤器和源有一个颜色参数，需要指定需要的颜色，并且有4种颜色规范的方法(默认值为黑色)：

- 1. 颜色被指定为W3C(万维网联盟)标准名称，以其十六进制值按字母顺序排列的标准名称的列表如下图所示。请注意，这里有几个同义词:aqua = cyan, fuchsia = magenta和gray = grey。
- 2. 颜色被指定为十六进制数的形式0 xrrggb @AA, RR红色通道, GG是绿色通道和BB是蓝色通道，例如0 x0000ff是蓝色的,0xffff00是黄色的,等等。(@AA)是一个可选的alpha通道,指定有多少不透明的颜色,颜色在通道与一个@符号字符。Alpha通道值要么是从0x00到0xff的十六进制数字，要么是介于0.0和1.0之间的十进制数，其中0.0 (0x00)是完全透明的，1.0 (0xff)是完全不透明的。例如，半透明度的绿色是0x00ff00@0.5。
- 3. 和之前的方法一样,但表示十六进制数字系统使用#符号而不是0 x前缀,一样在HTML代码中,例如# ff0000是红色的,# ffffff是白色的,等等。请注意,#前缀不能使用alpha通道,这意味着# 0000 ff@0x34是好的,但没有# 0000 ff@ # 34。
- 4. 颜色是用一个特殊的值随机指定的，结果是由计算机产生的随机颜色。

Color name	Hex value	Color name	Hex value	Color name	Hex value
AliceBlue	0xF0F8FF	GhostWhite	0xF8F8FF	Moccasin	0xFFE4B5
AntiqueWhite	0xFAEBD7	Gold	0xFFD700	NavajoWhite	0xFFDEAD
Aqua, Cyan	0x00FFFF	GoldenRod	0xDAA520	Navy	0x000080
Aquamarine	0x7FFF4	Gray	0x808080	OldLace	0xFDF5E6
Azure	0x0FFFFF	Grey	0x808080	Olive	0x808000
Beige	0xF5F5DC	Green	0x008000	OliveDrab	0x6B8E23
Bisque	0xFFE4C4	GreenYellow	0xADFF2F	Orange	0xFFA500
Black	0x000000	HoneyDew	0xF0FFF0	OrangeRed	0xFF4500
BlanchedAlmond	0xFFEBBC	HotPink	0xFF69B4	Orchid	0xDA70D6
Blue	0x0000FF	IndianRed	0xCD5C5C	PaleGoldenRod	0xEEEE8AA
BlueViolet	0x8A2BE2	Indigo	0x4B0082	PaleGreen	0x98FB98
Brown	0xA52A2A	Ivory	0xFFFFF0	PaleTurquoise	0xAFEEEE
BurlyWood	0xDEB887	Khaki	0xF0E68C	PaleVioletRed	0xD87093
CadetBlue	0x5F9EA0	Lavender	0xE6E6FA	PapayaWhip	0xFFEFD5
Chartreuse	0x7FFF00	LavenderBlush	0xFFF0F5	PeachPuff	0xFFDAB9
Chocolate	0xD2691E	LawnGreen	0x7CFC00	Peru	0xCD853F
Coral	0xFF7F50	LemonChiffon	0xFFFFAC	Pink	0xFFC0CB
CornflowerBlue	0x6495ED	LightBlue	0xADD8E6	Plum	0xDDA0DD
Cornsilk	0xFFFF8DC	LightCoral	0xF08080	PowderBlue	0xB0E0E6
Crimson	0xDC143C	LightCyan	0xE0FFFF	Purple	0x800080
DarkBlue	0x00008B	LightGoldenRodYellow	0xFAFAD2	Red	0xFF0000
DarkCyan	0x008B8B	LightGray	0xD3D3D3	RosyBrown	0xBC8F8F
DarkGoldenRod	0xB8860B	LightGrey	0xD3D3D3	RoyalBlue	0x4169E1
DarkGray	0xA9A9A9	LightGreen	0x90EE90	SaddleBrown	0x8B4513
DarkGrey	0xA9A9A9	LightPink	0xFFB6C1	Salmon	0xFA8072
DarkGreen	0x006400	LightSalmon	0xFFA07A	SandyBrown	0xF4A460
DarkKhaki	0xBD876B	LightSeaGreen	0x20B2AA	SeaGreen	0x2E8B57
DarkMagenta	0x8B008B	LightSkyBlue	0x87CEFA	Seashell	0xFFFF5EE
DarkOliveGreen	0x56B2F	LightSlateGrey	0x778899	Sienna	0xA0522D
DarkOrange	0xFF8C00	LightSteelBlue	0xB0C4DE	Silver	0xC0C0C0
DarkOrchid	0x932CC	LightYellow	0xFFFFE0	SkyBlue	0x87CEEB
DarkRed	0x8B0000	Lime	0x0FFF00	SlateBlue	0x6A5ACD
DarkSalmon	0xE9967A	LimeGreen	0x32CD32	SlateGray	0x708090
DarkSeaGreen	0x8FB88F	Linen	0xF0AF0E	SlateGrey	0x708090
DarkSlateBlue	0x483D8B	Magenta	0xFF00FF	Snow	0xFFFFFA
DarkSlateGrey	0x2F4F4F	Maroon	0x800000	SpringGreen	0x00FF7F
DarkTurquoise	0x00CED1	MediumAquaMarine	0x66CDAA	SteelBlue	0x4682B4
DarkViolet	0x9400D3	MediumBlue	0x0000CD	Tan	0xD2B48C
DeepPink	0xFF1493	MediumOrchid	0xBA55D3	Teal	0x008080
DeepSkyBlue	0x00BF00	MediumPurple	0x9370D8	Thistle	0xD8BFD8
DimGray	0x696969	MediumSeaGreen	0x3CB371	Tomato	0xFF6347
DimGrey	0x696969	MediumSlateBlue	0x7B68EE	Turquoise	0x40E0D0
DodgerBlue	0x1E90FF	MediumSpringGreen	0x00FA9A	Violet	0xEE82EE
FireBrick	0xB22222	MediumTurquoise	0x48D1CC	Wheat	0xF5DEB3
FloralWhite	0xFFFFA0	MediumVioletRed	0xC71585	White	0xFFFFFFF
ForestGreen	0x228B22	MidnightBlue	0x191970	WhiteSmoke	0xF5F5F5
Fuchsia	0xFF00FF	MintCream	0xF5FFFA	Yellow	0xFFFF00
Gainsboro	0xDCDCDC	MistyRose	0FFE4E1	YellowGreen	0x9ACD32

Synonyms: aqua = cyan, fuchsia = magenta and gray = grey  
# can replace 0x, for example 0x0000ff = #0000ff

## 02-显示帮助和功能

关于FFmpeg程序的帮助和其他信息都显示在空格和连字符之后输入的各种选项，示例显示了FFmpeg工具的用法，但是相同的选项对于ffplay、ffprobe和ffserver是有效的。参数是区分大小写的。

FFmpeg组件的开发速度很快，从2012年11月开始，一些可用项目列表很快就会不完整，比如X264的支持，不过你可以自己对里面的一些项目列表进行扩展。

### FFmpeg中的文本帮助

FFmpeg工具有一个很大的控制台帮助，可以完整显示或关于特定元素 - 解码器，编码器等。下表介绍了可用选项，斜体文本将替换为要显示的项目。ffplay和ffprobe也有类似的选项。（注意：有一些倾斜的命令行，我这边显示出来之后并不是倾斜的，请大家谅解）

基础的帮助	选中的项目的帮助
ffmpeg -? or ffmpeg -h	ffmpeg -h decoder=decoder_name
额外的帮助/扩展的帮助	ffmpeg -h encoder=encoder_name
ffmpeg -h long or ffmpeg -h full	ffmpeg -h demuxer=demuxer_name
ffmpeg -? topic or ffmpeg -h topic	ffmpeg -h muxer=muxer_name

```
[zhangfangtaodeMacBook-Pro:~ zhangfangtao$ ffmpeg -?
ffmpeg version N-90810-g153e920892 Copyright (c) 2000-2018 the FFmpeg developers
  built with Apple LLVM version 9.1.0 (clang-902.0.39.1)
configuration: --prefix=/usr/local --enable-gpl --enable-nonfree --enable-liba
ss --enable-libfdk-aac --enable-libfreetype --enable-libmp3lame --enable-libopus
--enable-libtheora --enable-libvorbis --enable-libvpx --enable-libx264 --enable
-libxvid --extra-ldflags=-L/usr/local/lib
  libavutil      56. 15.100 / 56. 15.100
  libavcodec     58. 19.100 / 58. 19.100
  libavformat    58. 13.100 / 58. 13.100
  libavdevice     58.  4.100 / 58.  4.100
  libavfilter     7. 19.100 /  7. 19.100
  libswscale      5.  2.100 /  5.  2.100
  libswresample   3.  2.100 /  3.  2.100
  libpostproc    55.  2.100 / 55.  2.100
Hyper fast Audio and Video encoder
usage: ffmpeg [options] [[infile options] -i infile]... {[outfile options] outfi
le}...

Getting help:
  -h      -- print basic options
  -h long -- print more options
  -h full -- print all options (including all format and codec specific option
s, very long)
  -h type=name -- print all options for the named decoder/encoder/demuxer/muxe
r/filter/bsf
      See man ffmpeg for detailed description of the options.

Print help / information / capabilities:
-L          show license
-h topic    show help
-? topic    show help
-help topic show help
--help topic show help
-version   show version
-buildconf show build configuration
-formats   show available formats
-muxers    show available muxers
-demuxers  show available demuxers
-devices   show available devices
-codecs    show available codecs
-decoders  show available decoders
-encoders  show available encoders
```

ffmpeg -? or ffmpeg -h

例如，要显示关于FLV解码器的信息，我们可以使用以下命令：

```
ffmpeg -h decoder=flv
```

控制台的输出是：

```
Decoder flv [FLV / Sorenson Spark / Sorenson H.263 (Flash Video)]: Threading  
capabilities: no Supported pixel formats: yuv420p
```

完整的帮助是非常长的，请参见本章末尾的格式化解决方案，下面我给大家接了一个图，这是在我的电脑上显示的结果：

```
Decoder flv [FLV / Sorenson Spark / Sorenson H.263 (Flash Video)]:  
General capabilities: horizband dr1  
Threading capabilities: none  
Supported pixel formats: yuv420p
```

这是在我电脑上系那是出来的结果

## 可用的比特流过滤器

显示内置的比特流过滤器的命令是：

```
ffmpeg -bsfs
```

### 比特流控制器

text2movsub

remove\_extra

noise

mov2textsub

mp3decomp

mp3comp

mjpegadump

mjpeg2jpeg

imxdump

h264\_mp4toannexb

dump\_extra

chomp

aac\_adtstoasc

我的电脑上显示的如下：

```
Bitstream filters:  
aac_adtstoasc  
chomp  
dump_extra  
dca_core  
eac3_core  
extract_extradata  
filter_units  
h264_metadata  
h264_mp4toannexb  
h264_redundant_pps  
hapqa_extract  
hevc_metadata  
hevc_mp4toannexb  
imxdump  
mjpeg2jpeg  
mjpegadump  
mp3decomp  
mpeg2_metadata  
mpeg4_unpack_bframes  
mov2textsub  
noise  
null  
remove_extra  
text2movsub  
trace_headers  
vp9_raw_reorder  
vp9_superframe  
vp9_superframe_split
```

我电脑上显示出来的结果

## 可用的解码器

可以使用-codecs选项显示可用的解码器，我们可以使用以下命令：

```
ffmpeg -codecs
```

## 解码器

D..... = Decoding supported

.E.... = Encoding supported

..V... = Video codec

..A... = Audio codec

..S... = Subtitle codec

....I.. = Intra frame-only codec

....L. = Lossy compression

.....S = Lossless compression

-----

## Codecs:

D..... = Decoding supported

.E.... = Encoding supported

..V... = Video codec

..A... = Audio codec

..S... = Subtitle codec

....I.. = Intra frame-only codec

....L. = Lossy compression

.....S = Lossless compression

-----

D.V.L. 4xm	4X Movie
D.VI.S 8bps	QuickTime 8BPS video
.EVIL. a64_multi	Multicolor charset for Commodore 64 (encoders: a64multi )
.EVIL. a64_multi5	Multicolor charset for Commodore 64, extended with 5th color (colram) (encoders: a64multi5 )
D.V..S aasc	Autodesk RLE
DEVIL. amv	AMV Video
D.V.L. anm	Deluxe Paint Animation
D.V.L. ansi	ASCII/ANSI art
DEVIL. asv1	ASUS V1
DEVIL. asv2	ASUS V2
D.VIL. aura	Auravision AURA
D.VIL. aura2	Auravision Aura 2
D.V... avrn	Avid AVI Codec
DEVI.. avrp	Avid 1:1 10-bit RGB Packer
D.V.L. avs	AVS (Audio Video Standard) video
DEVI.. avui	Avid Meridien Uncompressed
DEVI.. ayuv	Uncompressed packed MS 4:4:4:4
D.V.L. bethsoftvid	Bethesda VID video
D.V.L. bfi	Brute Force & Ignorance
D.V.L. binkvideo	Bink video
D.VI.. bintext	Binary text
DEVI.S bmp	BMP (Windows and OS/2 bitmap)
D.V..S bmv_video	Discworld II BMV video
D.V.L. c93	Interplay C93
DEV.L. cavs	Chinese AVS (Audio Video Standard) (AVS1-P2, JiZhen profile) (encoders: libxavs )
D.V.L. cdgraphics	CD Graphics video
D.VIL. cdxl	Commodore CDXL video
D.V.L. cinepak	Cinepak
DEVIL. cljr	Cirrus Logic AccuPak
D.VI.S cllc	Canopus Lossless Codec

<b>D.V.L. 4xm</b>	<b>4X Movie</b>
D.V.L. cmv	Electronic Arts CMV video (decoders: eacmv )
D.V... cpia	CPiA video format
D.V..S cscd	CamStudio (decoders: camstudio )
D.VIL. cyuv	Creative YUV (CYUV)
D.V.L. dfa	Chronomaster DFA
DEV.LS dirac	Dirac (decoders: dirac libschroedinger) (encoders: libschroedinger)
DEVIL. dnxhd	VC3/DNxHD
DEVIL. dpx	DPX image
D.V.L. dsicinvideo	Delphine Software International CIN video
DEVIL. dvvideo	DV (Digital Video)
D.V..S dxa	Feeble Files/ScummVM DXA
D.VI.S dxtory	Dxtory
D.V.L. escape124	Escape 124
D.V.L. escape130	Escape 130
D.VILS exr	OpenEXR image
DEV..S ffv1	FFmpeg video codec #1
DEVI.S fvhuff	Huffyuv FFmpeg variant
DEV..S flashsv	Flash Screen Video v1
DEV.L. flashsv2	Flash Screen Video v2
D.V..S flic	Autodesk Animator Flic video
DEV.L. flv1	FLV / Sorenson Spark / Sorenson H.263 (Flash Video) (decoders: flv) (encoders: flv )
D.V..S fraps	Fraps
D.VI.S frwu	Forward Uncompressed
..V... g2m	GoToMeeting
DEV..S gif	GIF (Graphics Interchange Format)
DEV.L. h261	H.261
DEV.L. h263	H.263 / H.263-1996, H.263+ / H.263-1998 / H.263 version 2
D.V.L. h263i	Intel H.263
DEV.L. h263p	H.263+ / H.263-1998 / H.263 version 2

D.V.L. 4xm	4X Movie
DEV.LS h264	H.264/AVC/MPEG-4 AVC/MPEG-4 part 10 (encoders: libx264 libx264rgb )
DEVI.S huffyuv	HuffYUV
D.V.L. idcin	id Quake II CIN video (decoders: idcinvideo )
D.VI.. idf	iCEDraw text
D.V.L. iff_byterun1	IFF ByteRun1
D.V.L. iff_ilbm	IFF ILBM
D.V.L. indeo2	Intel Indeo 2
D.V.L. indeo3	Intel Indeo 3
D.V.L. indeo4	Intel Indeo Video Interactive 4
D.V.L. indeo5	Intel Indeo Video Interactive 5
D.V.L. interplayvideo	Interplay MVE video
DEVILS jpeg2000	JPEG 2000 (decoders: j2k libopenjpeg ) (encoders: j2k libopenjpeg )
DEVILS jpegls	JPEG-LS
D.VIL. jv	Bitmap Brothers JV video
D.V.L. kgv1	Kega Game Video
D.V.L. kmvc	Karl Morton's video codec
D.VI.S lagarith	Lagarith lossless
.EVI.S ljpeg	Lossless JPEG
D.VI.S loco	LOCO
D.V.L. mad	Electronic Arts Madcow Video (decoders: eamad )
D.VIL. mdec	Sony PlayStation MDEC (Motion DECoder)
D.V.L. mimic	Mimic
DEVIL. mjpeg	Motion JPEG
D.VIL. mjpegb	Apple MJPEG-B
D.V.L. mmvideo	American Laser Games MM Video
D.V.L. motionpixels	Motion Pixels video
DEV.L. mpeg1video	MPEG-1 video
DEV.L. mpeg2video	MPEG-1 video (decoders: mpeg2video mpegvideo )
DEV.L. mpeg4	MPEG-4 part 2 (encoders: mpeg4 libxvid )

D.V.L. 4xm	4X Movie
..V.L. mpegvideo_xvmc	MPEG-1/2 video XvMC (X-Video Motion Compensation)
D.V.L. msa1	MS ATC Screen
D.V.L. msmpeg4v1	MPEG-4 part 2 Microsoft variant version 1
DEV.L. msmpeg4v2	MPEG-4 part 2 Microsoft variant version 2
DEV.L. msmpeg4v3	MPEG-4 part 2 Microsoft variant version 3 (decoders: msmpeg4 ) (encoders: msmpeg4 )
D.V..S msrle	Microsoft RLE
D.V.L. mss1	MS Screen 1
D.VIL. mss2	MS Windows Media Video V9 Screen
DEV.L. msvideo1	Microsoft Video 1
D.VI.S mszh	LCL (LossLess Codec Library) MSZH
D.V.L. mts2	MS Expression Encoder Screen
D.V.L. mxpeg	Mobotix MxPEG video
D.V.L. nuv	NuppelVideo/RTJPEG
D.V.L. paf_video	Amazing Studio Packed Animation File Video
DEVI.S pam	PAM (Portable AnyMap) image
DEVI.S pbm	PBM (Portable BitMap) image
DEVI.S pcx	PC Paintbrush PCX image
DEVI.S pgm	PGM (Portable GrayMap) image
DEVI.S pgmyuv	PGMYUV (Portable GrayMap YUV) image
D.VIL. pictor	Pictor/PC Paint
DEV..S png	PNG (Portable Network Graphics) image
DEVI.S ppm	PPM (Portable PixelMap) image
DEVIL. prores	Apple ProRes (iCodec Pro) (decoders: prores prores_lgpl ) (encoders: prores prores_anatoliy prores_kostya )
D.VIL. ptx	V.Flash PTX image
D.VI.S qdraw	Apple QuickDraw
D.V.L. qpeg	Q-team QPEG
DEV..S qtrle	QuickTime Animation (RLE) video
DEVI.S r10k	AJA Kona 10-bit RGB Codec

<b>D.V.L. 4xm</b>	<b>4X Movie</b>
DEVI.S r210	Uncompressed RGB 10-bit
DEVI.S rawvideo	raw video
D.VIL. rl2	RL2 video
DEV.L. roq	id RoQ video (decoders: roqvideo ) (encoders: roqvideo )
D.V.L. rpza	QuickTime video (RPZA)
DEV.L. rv10	RealVideo 1.0
DEV.L. rv20	RealVideo 1.0 (我怀疑这儿书上是不是错了)
D.V.L. rv30	RealVideo 3.0\
D.V.L. rv40	RealVideo 4.0
D.V.L. sanm	LucasArts SMUSH video
DEVIL. sgi	SGI image
D.V.L. smackvideo	Smacker video (decoders: smackvid )
D.V.L. smc	QuickTime Graphics (SMC)
DEV.LS snow	Snow
D.VIL. sp5x	Sunplus JPEG (SP5X)
DEVI.S sunrast	Sun Rasterfile image
DEV.L. svq1	Sorenson Vector Quantizer 1 / Sorenson Video 1 / SVQ1
D.V.L. svq3	Sorenson Vector Quantizer 3 / Sorenson Video 3 / SVQ3
DEVI.S targa	Truevision Targa image
D.VI.. targa_y216	Pinnacle TARGA CineWave YUV16
D.V.L. tgq	Electronic Arts TGQ video (decoders: eatgq )
D.V.L. tgv	Electronic Arts TGV video (decoders: eatgv )
DEV.L. theora	Theora (encoders: libtheora )
D.VIL. thp	Nintendo Gamecube THP video
D.V.L. tierTEXseqvideo	Tiertex Limited SEQ video
DEVI.S tiff	TIFF image
D.VIL. tmv	8088flex TMV
D.V.L. tqi	Electronic Arts TQI video (decoders: eatqi )
D.V.L. truemotion1	Duck TrueMotion 1.0
D.V.L. truemotion2	Duck TrueMotion 2.0

<b>D.V.L. 4xm</b>	<b>4X Movie</b>
D.V..S tscc	TechSmith Screen Capture Codec (decoders: camtasia )
D.V.L. tscc2	TechSmith Screen Codec 2
D.VIL. txd	Renderware TXD (TeXture Dictionary) image
D.V.L. ulti	IBM UltiMotion (decoders: ultimotion )
DEVI.S utvideo	Ut Video (decoders: utvideo libutvideo) (encoders: utvideo libutvideo)
DEVI.S v210	Uncompressed 4:2:2 10-bit
D.VI.S v210x	
DEVI.. v308	Uncompressed packed 4:4:4
DEVI.. v408	Uncompressed packed QT 4:4:4:4
DEVI.S v410	Uncompressed 4:4:4 10-bit
D.V.L. vb	Beam Software VB
D.VI.S vble	VBLE Lossless Codec
D.V.L. vc1	SMPTE VC-1
D.V.L. vc1image	Windows Media Video 9 Image v2
D.VIL. vcr1	ATI VCR1
D.VIL. vixl	Miro VideoXL (decoders: xl )
D.V.L. vmdvideo	Sierra VMD video
D.V..S vmnc	VMware Screen Codec / VMware Video
D.V.L. vp3	On2 VP3
D.V.L. vp5	On2 VP5
D.V.L. vp6	On2 VP6
D.V.L. vp6a	On2 VP6 (Flash version, with alpha channel)
D.V.L. vp6f	On2 VP6 (Flash version)
DEV.L. vp8	On2 VP8 (decoders: vp8 libvpx ) (encoders: libvpx )
DEV.L. wmv1	Windows Media Video 7
DEV.L. wmv2	Windows Media Video 8
D.V.L. wmv3	Windows Media Video 9
D.V.L. wmv3image	Windows Media Video 9 Image
D.VIL. wnv1	Winnov WNV1

D.V.L. 4xm	4X Movie
D.V.L. ws_vqa	Westwood Studios VQA (Vector Quantiz. Animation) video (decoders:vqavideo)
D.V.L. xan_wc3	Wing Commander III / Xan
D.V.L. xan_wc4	Wing Commander IV / Xxan
D.VI.. xbin	eXtended BINary text
DEVI.S xbm	XBM (X BitMap) image
DEV... xface	X-face image
DEVI.S xwd	XWD (X Window Dump) image
DEVI.. y41p	Uncompressed YUV 4:1:1 12-bit
D.V.L. yop	Psygnosis YOP Video
DEVI.. yuv4	Uncompressed packed 4:2:0
D.V..S zerocodec	ZeroCodec Lossless Video
DEVI.S zlib	LCL (LossLess Codec Library) ZLIB
DEV..S zmbv	Zip Motion Blocks Video
D.A.L. 8svx_exp	8SVX exponential
D.A.L. 8svx_fib	8SVX fibonacci
..A... 8svx_raw	8SVX raw
DEA.L. aac	AAC (Advanced Audio Coding) (encoders: aac libvo_aacenc )
D.A.L. aac_latm	AAC LATM (Advanced Audio Coding LATM syntax)
DEA.L. ac3	ATSC A/52A (AC-3) (encoders: ac3 ac3_fixed )
D.A.L. adpcm_4xm	ADPCM 4X Movie
DEA.L. adpcm_adx	SEGA CRI ADX ADPCM
D.A.L. adpcm_ct	ADPCM Creative Technology
D.A.L. adpcm_ea	ADPCM Electronic Arts
D.A.L. adpcm_ea_maxis_xa	ADPCM Electronic Arts Maxis CDROM XA
D.A.L. adpcm_ea_r1	ADPCM Electronic Arts R1
D.A.L. adpcm_ea_r2	ADPCM Electronic Arts R2
D.A.L. adpcm_ea_r3	ADPCM Electronic Arts R3
D.A.L. adpcm_ea_xas	ADPCM Electronic Arts XAS

<b>D.V.L. 4xm</b>	<b>4X Movie</b>
DEA.L. adpcm_g722	G.722 ADPCM (decoders: g722 ) (encoders: g722 )
DEA.L. adpcm_g726	G.726 ADPCM (decoders: g726 ) (encoders: g726 )
D.A.L. adpcm_ima_amv	ADPCM IMA AMV
D.A.L. adpcm_ima_apc	ADPCM IMA CRYO APC
D.A.L. adpcm_ima_dk3	ADPCM IMA Duck DK3
D.A.L. adpcm_ima_dk4	ADPCM IMA Duck DK4
D.A.L. adpcm_ima_ea_eacs	ADPCM IMA Electronic Arts EACS
D.A.L. adpcm_ima_ea_sead	ADPCM IMA Electronic Arts SEAD
D.A.L. adpcm_ima_iss	ADPCM IMA Funcom ISS
DEA.L. adpcm_ima_qt	ADPCM IMA QuickTime
D.A.L. adpcm_ima_smjpeg	ADPCM IMA Loki SDL MJPEG
DEA.L. adpcm_ima_wav	ADPCM IMA WAV
D.A.L. adpcm_ima_ws	ADPCM IMA Westwood
DEA.L. adpcm_ms	ADPCM Microsoft
D.A.L. adpcm_sbpro_2	ADPCM Sound Blaster Pro 2-bit
D.A.L. adpcm_sbpro_3	ADPCM Sound Blaster Pro 2.6-bit
D.A.L. adpcm_sbpro_4	ADPCM Sound Blaster Pro 4-bit
DEA.L. adpcm_swf	ADPCM Shockwave Flash
D.A.L. adpcm_thp	ADPCM Nintendo Gamecube THP
D.A.L. adpcm_xa	ADPCM CDROM XA
DEA.L. adpcm_yamaha	ADPCM Yamaha

D.V.L. 4xm	4X Movie
DEA..S alac	ALAC (Apple Lossless Audio Codec)
DEA.L. amr_nb	AMR-NB (Adaptive Multi-Rate NarrowBand) (decoders: amrnrb libopencore_amrnb ) (encoders: libopencore_amrnb )
DEA.L. amr_wb	AMR-WB (Adaptive Multi-Rate WideBand) (decoders: amrwb libopencore_amrwb ) (encoders: libvo_amrwbenc )
D.A..S ape	Monkey's Audio
D.A.L. atrac1	Atrac 1 (Adaptive TRansform Acoustic Coding)
D.A.L. atrac3	Atrac 3 (Adaptive TRansform Acoustic Coding 3)
..A.L. atrac3p	Sony ATRAC3+
D.A.L. binkaudio_dct	Bink Audio (DCT)
D.A.L. binkaudio_rdft	Bink Audio (RDFT)
D.A.L. bmv_audio	Discworld II BMV audio
..A.L. celt	Constrained Energy Lapped Transform (CELT)
DEA.L. comfortnoise	RFC 3389 Comfort Noise
D.A.L. cook	Cook / Cooker / Gecko (RealAudio G2)
D.A.L. dsicinaudio	Delphine Software International CIN audio
DEA.LS dts	DCA (DTS Coherent Acoustics) (decoders: dca ) (encoders: dca ) ..A.L. dvaudio
DEA.L. eac3	ATSC A/52B (AC-3, E-AC-3)
DEA..S flac	FLAC (Free Lossless Audio Codec)
DEA.L. g723_1	G.723.1
D.A.L. g729	G.729
DEA.L. gsm	GSM (decoders: gsm libgsm ) (encoders: libgsm )
DEA.L. gsm_ms	GSM Microsoft variant (decoders:gsm_ms libgsm_ms) (encoders: libgsm_ms)
D.A.L. iac	IAC (Indeo Audio Coder)
..A.L. ilbc	iLBC (Internet Low Bitrate Codec)
D.A.L. imc	IMC (Intel Music Coder)
D.A.L. interplay_dpcm	DPCM Interplay
D.A.L. mace3	MACE (Macintosh Audio Compression/Expansion) 3:1
D.A.L. mace6	MACE (Macintosh Audio Compression/Expansion) 6:1

D.V.L. 4xm	4X Movie
D.A..S mlp	MLP (Meridian Lossless Packing)
D.A.L. mp1	MP1 (MPEG audio layer 1) (decoders: mp1 mp1float )
DEA.L. mp2	MP2 (MPEG audio layer 2) (decoders: mp2 mp2float )
DEA.L. mp3	MP3 (MPEG audio layer 3) (decoders:mp3 mp3float) (encoders: libmp3lame)
D.A.L. mp3adu	ADU (Application Data Unit) MP3 (MPEG audio layer 3) (decoders: mp3adu mp3adufloat )
D.A.L. mp3on4	MP3onMP4 (decoders: mp3on4 mp3on4float )
D.A..S mp4als	MPEG-4 Audio Lossless Coding (ALS) (decoders: als )
D.A.L. musepack7	Musepack SV7 (decoders: mpc7 )
D.A.L. musepack8	Musepack SV8 (decoders: mpc8 )
DEA.L. nellymoser	Nellymoser Asao
DEA.L. opus	Opus (Opus Interactive Audio Codec) (decoders:libopus) (encoders: libopus)
D.A.L. paf_audio	Amazing Studio Packed Animation File Audio
DEA... pcm_alaw	PCM A-law
D.A..S pcm.bluray	PCM signed 16/20/24-bit big-endian for Blu-ray media
D.A..S pcm_dvd	PCM signed 20/24-bit big-endian
DEA..S pcm_f32be	PCM 32-bit floating point big-endian
DEA..S pcm_f32le	PCM 32-bit floating point little-endian
DEA..S pcm_f64be	PCM 64-bit floating point big-endian
DEA..S pcm_f64le	PCM 64-bit floating point little-endian
D.A..S pcm_lxf	PCM signed 20-bit little-endian planar
DEA... pcm_mulaw	PCM mu-law
DEA..S pcm_s16be	PCM signed 16-bit big-endian
DEA..S pcm_s16le	PCM signed 16-bit little-endian
D.A..S pcm_s16le_planar	PCM 16-bit little-endian planar
DEA..S pcm_s24be	PCM signed 24-bit big-endian
DEA..S pcm_s24daud	PCM D-Cinema audio signed 24-bit
DEA..S pcm_s24le	PCM signed 24-bit little-endian

D.V.L. 4xm	4X Movie
DEA..S pcm_s32be	PCM signed 32-bit big-endian
DEA..S pcm_s32le	PCM signed 32-bit little-endian
DEA..S pcm_s8	PCM signed 8-bit
D.A..S pcm_s8_planar	PCM signed 8-bit planar
DEA..S pcm_u16be	PCM unsigned 16-bit big-endian
DEA..S pcm_u16le	PCM unsigned 16-bit little-endian
DEA..S pcm_u24be	PCM unsigned 24-bit big-endian
DEA..S pcm_u24le	PCM unsigned 24-bit little-endian
DEA..S pcm_u32be	PCM unsigned 32-bit big-endian
DEA..S pcm_u32le	PCM unsigned 32-bit little-endian
DEA..S pcm_u8	PCM unsigned 8-bit
D.A.L. pcm_zork	PCM Zork
D.A.L. qcelp	QCELP / PureVoice
D.A.L. qdm2	QDesign Music Codec 2
..A.L. qdmc	QDesign Music
DEA.L. ra_144	RealAudio 1.0 (14.4K) (decoders: real_144 ) (encoders: real_144 )
D.A.L. ra_288	RealAudio 2.0 (28.8K) (decoders: real_288 )
D.A..S ralf	RealAudio Lossless
DEA.L. roq_dpcm	DPCM id RoQ
D.A.L. s302m	SMPTE 302M
D.A..S shorten	Shorten
D.A.L. sipr	RealAudio SIPR / ACELP.NET
D.A.L. smackaudio	Smacker audio (decoders: smackaud )
D.A.L. sol_dpcm	DPCM Sol
DEA... sonic	Sonic
.EA... sonicls	Sonic lossless
DEA.L. speex	Speex (decoders: libspeex ) (encoders: libspeex )
D.A..S tak	TAK (Tom's lossless Audio Kompressor)
D.A..S truehd	TrueHD
D.A.L. truespeech	DSP Group TrueSpeech

<b>D.V.L. 4xm</b>	<b>4X Movie</b>
D.A..S tta	TTA (True Audio)
D.A.L. twinvq	VQF TwinVQ
D.A.L. vima	LucasArts VIMA audio
D.A.L. vmdaudio	Sierra VMD audio
DEA.L. vorbis	Vorbis (decoders: vorbis libvorbis ) (encoders: vorbis libvorbis )
..A.L. voxware	Voxware RT29 Metasound
Voxware RT29 Metasound	Wave synthesis pseudo-codec
D.A.LS wavpack	WavPack
D.A.L. westwood_snd1	Westwood Audio (SND1) (decoders: ws_snd1 )
D.A..S wmalossless	Windows Media Audio Lossless
D.A.L. wmapro	Windows Media Audio 9 Professional
DEA.L. wmav1	Windows Media Audio 1
DEA.L. wmav2	Windows Media Audio 2
D.A.L. wmavoice	Windows Media Audio Voice
D.A.L. xan_dpcm	DPCM Xan
DES... dvb_subtitle	DVB subtitles (decoders: dvbsub ) (encoders: dvbsub )
..S... dvb_teletext	DVB teletext
DES... dvd_subtitle	DVD subtitles (decoders: dvdsb ) (encoders: dvdsb )
..S... eia_608	EIA-608 closed captions
D.S... hdmv_pgs_subtitle	HDMV Presentation Graphic Stream subtitles (decoders: pgssub )
D.S... jacosub	JACOsub subtitle
D.S... microdvd	MicroDVD subtitle
DES... mov_text	MOV text
D.S... realtext	RealText subtitle
D.S... sami	SAMI subtitle
DES... srt	SubRip subtitle with embedded timing
DES... ssa	SSA (SubStation Alpha)/ ASS (Advanced SSA) subtitle (decoders: ass) (encoders: ass )
DES... subrip	SubRip subtitle

D.V.L. 4xm	4X Movie
D.S... subviewer	SubViewer subtitle
D.S... text	raw UTF-8 text
D.S... webvtt	WebVTT subtitle
DES... xsub	XSUB

- 写的我想吐。。妹的！这么多。 . . . .

```
-----
D.VI.S 012v          Uncompressed 4:2:2 10-bit
D.V.L. 4xm           4X Movie
D.VI.S 8bps          QuickTime 8BPS video
.EVIL. a64_multi     Multicolor charset for Commodore 64 (encoders: a64m
ulti )
.EVIL. a64_multi5    Multicolor charset for Commodore 64, extended with
5th color (colram) (encoders: a64multi5 )
D.V..S aasc          Autodesk RLE
D.VIL. aic           Apple Intermediate Codec
DEVI.S alias_pix    Alias/Wavefront PIX image
DEVIL. amv           AMV Video
D.V.L. anm           Deluxe Paint Animation
D.V.L. ansi          ASCII/ANSI art
DEV..S apng          APNG (Animated Portable Network Graphics) image
DEVIL. asv1          ASUS V1
DEVIL. asv2          ASUS V2
D.VIL. aura          Auravision AURA
D.VIL. aura2         Auravision Aura 2
..V.L. av1           Alliance for Open Media AV1
D.V... avrn          Avid AVI Codec
DEVI.S avrp          Avid 1:1 10-bit RGB Packer
D.V.L. avs           AVS (Audio Video Standard) video
DEVI.S avui          Avid Meridien Uncompressed
DEVI.S ayuv          Uncompressed packed MS 4:4:4:4
D.V.L. bethsoftvid   Bethesda VID video
D.V.L. bfi           Brute Force & Ignorance
D.V.L. binkvideo     Bink video
D.VI.. bintext        Binary text
D.VI.S bitpacked     Bitpacked
DEVI.S bmp           BMP (Windows and OS/2 bitmap)
D.V..S bmv_video     Discworld II BMV video
D.VI.S brender_pix   BRRender PIX image
D.V.L. c93           Interplay C93
D.V.L. cavs          Chinese AVS (Audio Video Standard) (AVS1-P2, JiZhen
profile)
D.V.L. cdgraphics    CD Graphics video
D.VIL. cdxl          Commodore CDXL video
D.V.L. cfhd          Cineform HD
DEV.L. cinepak        Cinepak
D.V.L. clearvideo    Iterated Systems ClearVideo
DEVIL. cljr          Cirrus Logic AccuPak
D.VI.S cllc          Canopus Lossless Codec
D.V.L. cmv           Electronic Arts CMV video (decoders: eacmv )
D.V... cpia          CPIA video format
D.V..S cscd          CamStudio (decoders: camstudio )
D.VIL. cyuv          Creative YUV (CYUV)
```

image.png

## 可用的编码器

要显示内置的ffmpeg编码器的列表，我们可以使用以下命令：

```
ffmpeg -encoders
```

编码器
V..... = Video
A..... = Audio
S..... = Subtitle
.F.... = Frame-level multithreading
..S... = Slice-level multithreading
...X.. = Codec is experimental
....B. = Supports draw_horiz_band
.....D = Supports direct rendering method 1
-----

### Encoders:

V..... = Video  
 A..... = Audio  
 S..... = Subtitle  
 .F.... = Frame-level multithreading  
 ..S... = Slice-level multithreading  
 ...X.. = Codec is experimental  
 ....B. = Supports draw\_horiz\_band  
 .....D = Supports direct rendering method 1  
 -----  
 -----

V..... a64multi	<b>Multicolor charset for Commodore 64 (codec a64_multi)</b>
V..... a64multi5	Multicolor charset for Commodore 64, extended with 5th color (colram) (codec a64_multi5)
V..... amv	AMV Video
V..... asv1	ASUS V1
V..... asv2	ASUS V2
V..... avrp	Avid 1:1 10-bit RGB Packer
V..X.. avui	Avid Meridien Uncompressed
V..... ayuv	Uncompressed packed MS 4:4:4:4
V..... bmp	BMP (Windows and OS/2 bitmap)
V..... libxavs	libxavs Chinese AVS (Audio Video Standard) (codec cavs)
V..... cljr	Cirrus Logic AccuPak
V..... libschroedinger	libschroedinger Dirac 2.2 (codec dirac)
V.S... dnxhd	VC3/DNxHD
V..... dpx	DPX image
V.S... dvvideo	DV (Digital Video)
V.S... ffv1	FFmpeg video codec #1
V..... ffvhuff	Huffyuv FFmpeg variant
V..... flashsv	Flash Screen Video
V..... flashsv2	Flash Screen Video Version 2
V..... flv	FLV / Sorenson Spark / Sorenson H.263 (Flash Video) (codec flv1)
V..... gif	GIF (Graphics Interchange Format)
V..... h261	H.261
V..... h263	H.263 / H.263-1996
V.S... h263p	H.263+ / H.263-1998 / H.263 version 2
V..... libx264	libx264 H.264 / AVC / MPEG-4 AVC / MPEG-4 part 10 (codec h264)
V..... libx264rgb	libx264 H.264 /AVC / MPEG-4 AVC / MPEG-4 part 10 RGB (codec h264)
V..... huffyuv	Huffyuv / HuffYUV
V.X.. j2k	JPEG 2000 (codec jpeg2000)
V..... libopenjpeg	OpenJPEG JPEG 2000 (codec jpeg2000)

V..... a64multi	<b>Multicolor charset for Commodore 64 (codec a64_multi)</b>
V..... jpegls	JPEG-LS
V..... ljpeg	Lossless JPEG
VFS... mjpeg	MJPEG (Motion JPEG)
V..... mpeg1video	MPEG-1 video
V.S... mpeg2video	MPEG-2 video
V.S... mpeg4	MPEG-4 part 2
V..... libxvid	libxvidcore MPEG-4 part 2 (codec mpeg4)
V..... msmpeg4v2	MPEG-4 part 2 Microsoft variant version 2
V..... msmpeg4	MPEG-4 part 2 Microsoft variant version 3 (codec msmpeg4v3)
V..... msvideo1	Microsoft Video-1
V..... pam	PAM (Portable AnyMap) image
V..... pbm	PBM (Portable BitMap) image
V..... pcx	PC Paintbrush PCX image
V..... pgm	PGM (Portable GrayMap) image
V..... pgmyuv	PGMYUV (Portable GrayMap YUV) image
VF.... png	PNG (Portable Network Graphics) image
V..... ppm	PPM (Portable PixelMap) image
VF.... prores	Apple ProRes
VF.... prores_anatoliy	Apple ProRes (codec prores)
V.S... prores_kostya	Apple ProRes (iCodec Pro) (codec prores)
V..... qtrle	QuickTime Animation (RLE) video
V..... r10k	AJA Kona 10-bit RGB Codec
V..... r210	Uncompressed RGB 10-bit
V..... rawvideo	raw video
V..... roqvideo	id RoQ video (codec roq)
V..... rv10	RealVideo 1.0
V..... rv20	RealVideo 2.0
V..... sgi	SGI image
V..... snow	Snow
V..... sunrast	Sun Rasterfile image

V..... a64multi	<b>Multicolor charset for Commodore 64 (codec a64_multi)</b>
V..... svq1	Sorenson Vector Quantizer 1 / Sorenson Video 1 / SVQ1
V..... targa	Truevision Targa image
V..... libtheora	libtheora Theora (codec theora)
V..... tiff	TIFF image
V..... utvideo	Ut Video
V..... libutvideo	Ut Video (codec utvideo)
V..... v210	Uncompressed 4:2:2 10-bit
V..... v308	Uncompressed packed 4:4:4
V..... v408	Uncompressed packed QT 4:4:4:4
V..... v410	Uncompressed 4:4:4 10-bit
V..... libvpx	libvpx VP8 (codec vp8)
V..... wmv1	Windows Media Video 7
V..... wmv2	Windows Media Video 8
V..... xbm	XBM (X BitMap) image
V..... xface	X-face image
V..... xwd	XWD (X Window Dump) image
V..... y41p	Uncompressed YUV 4:1:1 12-bit
V..... yuv4	Uncompressed packed 4:2:0
V..... zlib	LCL (LossLess Codec Library) ZLIB
V..... zmbv	Zip Motion Blocks Video
A..X.. aac	AAC (Advanced Audio Coding)
A..... libvo_aacenc	Android VisualOn AAC (Advanced Audio Coding) (codec aac)
A..... ac3	ATSC A/52A (AC-3)
A..... ac3_fixed	ATSC A/52A (AC-3) (codec ac3)
A..... adpcm_adx	SEGA CRI ADX ADPCM
A..... g722	G.722 ADPCM (codec adpcm_g722)
A..... g726	G.726 ADPCM (codec adpcm_g726)
A..... adpcm_ima_qt	ADPCM IMA QuickTime
A..... adpcm_ima_wav	ADPCM IMA WAV
A..... adpcm_ms	ADPCM Microsoft

V..... a64multi	<b>Multicolor charset for Commodore 64 (codec a64_multi)</b>
A..... adpcm_swf	ADPCM Shockwave Flash
A..... adpcm_yamaha	ADPCM Yamaha
A..... alac	ALAC (Apple Lossless Audio Codec)
A..... libopencore_amrnb	OpenCORE AMR-NB (Adaptive Multi-Rate Narrow-Band) (codec amr_nb)
A..... libvo_amrwbenc	Android VisualOn AMR-WB (Adaptive Multi-Rate WideBand) (codec amr_wb)
A..... comfortnoise	RFC 3389 comfort noise generator
A..X.. dca	DCA (DTS Coherent Acoustics) (codec dts)
A..... eac3	ATSC A/52 E-AC-3
A..... flac	FLAC (Free Lossless Audio Codec)
A..... g723_1	G.723.1
A..... libgsm	libgsm GSM (codec gsm)
A..... libgsm_ms	libgsm GSM Microsoft variant (codec gsm_ms)
A..... mp2	MP2 (MPEG audio layer 2)
A..... libmp3lame	libmp3lame MP3 (MPEG audio layer 3) (codec mp3)
A..... nellymoser	Nellymoser Asao
A..... libopus	libopus Opus (codec opus)
A..... pcm_alaw	PCM A-law / G.711 A-law
A..... pcm_f32be	PCM 32-bit floating point big-endian
A..... pcm_f32le	PCM 32-bit floating point little-endian
A..... pcm_f64be	PCM 64-bit floating point big-endian
A..... pcm_f64le	PCM 64-bit floating point little-endian
A..... pcm_mulaw	PCM mu-law / G.711 mu-law
A..... pcm_s16be	PCM signed 16-bit big-endian
A..... pcm_s16le	PCM signed 16-bit little-endian
A..... pcm_s24be	PCM signed 24-bit big-endian
A..... pcm_s24daud	PCM D-Cinema audio signed 24-bit
A..... pcm_s24le	PCM signed 24-bit little-endian
A..... pcm_s32be	PCM signed 32-bit big-endian

V..... a64multi	Multicolor charset for Commodore 64 (codec a64_multi)
A..... pcm_s32le	PCM signed 32-bit little-endian
A..... pcm_s8	PCM signed 8-bit
A..... pcm_u16be	PCM unsigned 16-bit big-endian
A..... pcm_u16le	PCM unsigned 16-bit little-endian
A..... pcm_u24be	PCM unsigned 24-bit big-endian
A..... pcm_u24le	PCM unsigned 24-bit little-endian
A..... pcm_u32be	PCM unsigned 32-bit big-endian
A..... pcm_u32le	PCM unsigned 32-bit little-endian
A..... pcm_u8	PCM unsigned 8-bit
A..... real_144	RealAudio 1.0 (14.4K) (codec ra_144)
A..... roq_dpcm	id RoQ DPCM
A..X.. sonic	Sonic
A..X.. sonicls	Sonic lossless
A..... libspeex	libspeex Speex (codec speex)
A..X.. vorbis	Vorbis
A..... libvorbis	libvorbis (codec vorbis)
A..... wma1	Windows Media Audio 1
A..... wma2	Windows Media Audio 2
S..... dvbsub	DVB subtitles (codec dvb_subtitle)
S..... dvdsub	DVD subtitles (codec dvd_subtitle)
S..... mov_text	3GPP Timed Text subtitle
S..... srt	SubRip subtitle with embedded timing
S..... ass	SSA (SubStation Alpha) subtitle (codec ssa)
S..... subrip	SubRip subtitle
S..... xsub	DivX subtitles (XSUB)

## 可用的过滤器

要显示内置过滤器列表，我们可以使用下一个命令：

```
ffmpeg -filters
```

過濾器		
aconvert	A->A	Convert the input audio to sample_fmt:channel_layout.
afifo	A->A	Buffer input frames and send them when they are requested.
aformat	A->A	Convert the input audio to one of the specified formats.
amerge	->A	Merge two audio streams into a single multi-channel stream.
amix	->A	Audio mixing.
anull	A->A	Pass the source unchanged to the output.
aresample	A->A	Resample audio data.
asendcmd	A->A	Send commands to filters.
asetnsamples	A->A	Set the number of samples for each output audio frames.
asetpts	A->A	Set PTS for the output audio frame.
asettb	A->A	Set timebase for the audio output link.
ashowinfo	A->A	Show textual information for each audio frame.
asplit	A->	Pass on the audio input to N audio outputs.
astreamsync	AA->AA	Copy two streams of audio data in a configurable order.
atempo	A->A	Adjust audio tempo.
channelmap	A->A	Remap audio channels.
channelsplit	A->	Split audio into per-channel streams
earwax	A->A	Widen the stereo image.

過濾器		
ebur128	A->	EBU R128 scanner.
join	->A	Join multiple audio streams into multi-channel output
pan	A->A	Remix channels with coefficients (panning).
silencedetect	A->A	Detect silence.
volume	A->A	Change input volume.
volumedetect	A->A	Detect audio volume.
aevalsrc	->A	Generate an audio signal generated by an expression
anullsrc	->A	Null audio source, return empty audio frames.
anullsink	A->	Do absolutely nothing with the input audio.
alphaextract	V->V	Extract an alpha channel as a grayscale image component.
alphamerge	VV->V	Copy the luma value of the 2nd input to the alpha channel of the 1st input.
ass	V->V	Render subtitles onto input video using the libass library.
bbox	V->V	Compute bounding box for each frame.
blackdetect	V->V	Detect video intervals that are (almost) black.
blackframe	V->V	Detect frames that are (almost) black.
boxblur	V->V	Blur the input.
colormatrix	V->V	Color matrix conversion
copy	V->V	Copy the input video unchanged to the output.
crop	V->V	Crop the input video to width:height×y.

過濾器		
cropdetect	V->V	Auto-detect crop size.
decimate	V->V	Remove near-duplicate frames.
delogo	V->V	Remove logo from input video.
deshake	V->V	Stabilize shaky video.
drawbox	V->V	Draw a colored box on the input video.
drawtext	V->V	Draw text on top of video frames using libfreetype library.
edgedetect	V->V	Detect and draw edge.
fade	V->V	Fade in/out input video.
field	V->V	Extract a field from the input video.
fieldorder	V->V	Set the field order.
fifo	V->V	Buffer input images and send them when they are requested.
format	V->V	Convert the input video to one of the specified pixel formats.
fps	V->V	Force constant framerate
framestep	V->V	Select one frame every N frames.
gradfun	V->V	Debands video quickly using gradients.
hflip	V->V	Horizontally flip the input video.
hqdn3d	V->V	Apply a High Quality 3D Denoiser.
hue	V->V	Adjust the hue and saturation of the input video.
idet	V->V	Interlace detect Filter.

過濾器		
lut	V->V	Compute and apply a lookup table to the RGB/YUV input video.
lutrgb	V->V	Compute and apply a lookup table to the RGB input video.
lutyuv	V->V	Compute and apply a lookup table to the YUV input video.
mp	V->V	Apply a libmpcodecs filter to the input video.
negate	V->V	Negate input video.
noformat	V->V	Force libavfilter not to use any of the specified pixel formats for the input to the next filter.
null	V->V	Pass the source unchanged to the output.
overlay	VV->V	Overlay a video source on top of the input.
pad	V->V	Pad input image to width:height[xy[:color]] (default x and y: 0, default color: black).
pixdesctest	V->V	Test pixel format definitions.
removelogo	V->V	Remove a TV logo based on a mask image.
scale	V->V	Scale the input video to width:height size and/or convert the image format.
select	V->V	Select frames to pass in output.
sendcmd	V->V	Send commands to filters.
setdar	V->V	Set the frame display aspect ratio.
setfield	V->V	Force field for the output video frame.
setpts	V->V	Set PTS for the output video frame.
setsar	V->V	Set the pixel sample aspect ratio.
settb	V->V	Set timebase for the video output link.

过滤器		
showinfo	V->V	Show textual information for each video frame.
slicify	V->V	Pass the images of input video on to next video filter as multiple slices.
smartblur	V->V	Blur the input video without impacting the outlines.
split	V->	Pass on the input video to N outputs.
super2xsai	V->V	Scale the input by 2x using the Super2xSal pixel art algorithm.
swapuv	V->V	Swap U and V components.
thumbnail	V->V	Select the most representative frame in a sequence of consecutive frames
tile	V->V	Tile several successive frames together.
tinterlace	V->V	Perform temporal field interlacing.
transpose	V->V	Transpose input video.
unsharp	V->V	Sharpen or blur the input video.
vflip	V->V	Flip the input video vertically.
yadif	V->V	Deinterlace the input image.
cellauto	->V	Create pattern generated by an elementary cellular automaton.
color	->V	Provide an uniformly colored input.
life	->V	Create life.
mandelbrot	->V	Render a Mandelbrot fractal.
mptestsrc	->V	Generate various test pattern.
nullsrc	->V	Null video source, return unprocessed video frames.

過濾器		
rgbtessrc	->V	Generate RGB test pattern.
smptebars	->V	Generate SMPTE color bars.
testssrc	->V	Generate test pattern.
nullsink	V->	Do absolutely nothing with the input video.
concat	->	Concatenate audio and video streams.
showspectrum	A->V	Convert input audio to a spectrum video output.
showwaves	A->V	Convert input audio to a video output.
amovie	->	Read audio from a movie source.
movie	->	Read from a movie source.
ffbuffersink	V->	Buffer video frames and make them available to the end of the filter graph.
ffabuffersink	A->	Buffer audio frames and make them available to the end of the filter graph.
buffersink	V->	Buffer video frames and make them available to the end of the filter graph.
abuffersink	A->	Buffer audio frames and make them available to the end of the filter graph.
buffer	->V	Buffer video frames and make them accessible to the filterchain.
abuffer	->A	Buffer audio frames and make them accessible to the filterchain.
buffersink_old	V->	Buffer video frames and make them available to the end of the filter graph.
abuffersink_old	A->	Buffer audio frames and make them available to the end of the filter graph.

```
-----
V..... a64multi          Multicolor charset for Commodore 64 (codec a64_multi)
i)
V..... a64multis         Multicolor charset for Commodore 64, extended with
5th color (colram) (codec a64_multi5)
V..... alias_pix         Alias/Wavefront PIX image
V..... amv               AMV Video
V..... apng              APNG (Animated Portable Network Graphics) image
V..... asv1              ASUS V1
V..... asv2              ASUS V2
V..... avrp              Avid 1:1 10-bit RGB Packer
V..X.. avui              Avid Meridien Uncompressed
V..... ayuv              Uncompressed packed MS 4:4:4:4
V..... bmp               BMP (Windows and OS/2 bitmap)
V..... cinepak           Cinepak
V..... cljr              Cirrus Logic AccuPak
V.S... vc2               SMPTE VC-2 (codec dirac)
VFS... dnxhd             VC3/DNxHD
V..... dpx               DPX (Digital Picture Exchange) image
VFS... dvvideo            DV (Digital Video)
V.S... ffv1              FFmpeg video codec #1
VF.... ffvhuff            Huffyuv FFmpeg variant
V..... fits              Flexible Image Transport System
V..... flashsv            Flash Screen Video
V..... flashsv2           Flash Screen Video Version 2
V..... flv               FLV / Sorenson Spark / Sorenson H.263 (Flash Video)
(codec flv1)
V..... gif               GIF (Graphics Interchange Format)
```

## 可用的格式

要显示内置的音频和视频格式，下一个命令是：

```
ffmpeg -formats
```

下面的内容我打算直接截图，大家将就着看吧：

```
File formats:
D. = Demuxing supported
.E = Muxing supported
--
E 3g2               3GP2 (3GPP2 file format)
E 3gp               3GP (3GPP file format)
D 4xm              4X Technologies
E a64              a64 - video for Commodore 64
D aac              raw ADTS AAC (Advanced Audio Coding)
DE ac3              raw AC-3
D act               ACT Voice file format
D adf               Artworx Data Format
E adts              ADTS AAC (Advanced Audio Coding)
DE adx              CRI ADX
```

D aea	MD STUDIO audio
DE aiff	Audio IFF
DE alaw	PCM A-law
DE amr	3GPP AMR
D anm	Deluxe Paint Animation
D apc	CRYO APC
D ape	Monkey's Audio
DE asf	ASF (Advanced / Active Streaming Format)
E asf_stream	ASF (Advanced / Active Streaming Format)
DE ass	SSA (SubStation Alpha) subtitle
DE au	Sun AU
DE avi	AVI (Audio Video Interleaved)
E avm2	SWF (ShockWave Flash) (AVM2)
D avr	AVR (Audio Visual Resarch)
D avs	AVISynth
D bethsoftvid	Bethesda Softworks VID
D bfi	Brute Force & Ignorance
D bin	Binary text
D bink	Bink
DE bit	G.729 BIT file format
D bmv	Discworld II BMV
D c93	Interplay C93
DE caf	Apple CAF (Core Audio Format)
DE cavsvideo	raw Chinese AVS (Audio Video Standard) video
D cdg	CD Graphics
D cdxl	Commodore CDXL video
E crc	CRC testing
DE daud	D-Cinema audio
D dfa	Chronomaster DFA
DE dirac	raw Dirac
DE dnxhd	raw DNxHD (SMPTE VC-3)
D dshow	DirectShow capture
D dsicin	Delphine Software International CIN
DE dts	raw DTS
D dtshd	raw DTS-HD
DE dv	DV (Digital Video)
E dvd	MPEG-2 PS (DVD VOB)
D dxä	DXA
D ea	Electronic Arts Multimedia
D ea_cdata	Electronic Arts cdata
DE eac3	raw E-AC-3
DE f32be	PCM 32-bit floating-point big-endian
DE f32le	PCM 32-bit floating-point little-endian
E f4v	F4V Adobe Flash Video
DE f64be	PCM 64-bit floating-point big-endian
DE f64le	PCM 64-bit floating-point little-endian
DE ffm	FFM (FFserver live feed)
DE ffmetadata	FFmpeg metadata in text
D film_ckp	Sega FILM / CPK
DE filmstrip	Adobe Filmstrip
DE flac	raw FLAC
D flic	FLI/FLC/FLX animation
DE flv	FLV (Flash Video)
E framecrc	framecrc testing
E framemd5	Per-frame MD5 testing

```
DE g722           raw G.722
DE g723_1         raw G.723.1
D  g729          G.729 raw format demuxer
E  gif           GIF Animation
D  gsm           raw GSM
DE gxf            GXF (General eXchange Format)
DE h261          raw H.261
DE h263          raw H.263
DE h264          raw H.264 video
D  hls,applehttp Apple HTTP Live Streaming
DE ico            Microsoft Windows ICO
D  idcin          id Cinematic
D  idf            iCE Draw File
D  iff             IFF (Interchange File Format)
DE ilbc            iLBC storage
DE image2          image2 sequence
DE image2pipe      piped image2 sequence
D  ingenient      raw Ingenient MJPEG
D  ipmovie          Interplay MVE
E  ipod            iPod H.264 MP4 (MPEG-4 Part 14)
E  ismv             ISMV/ISMA (Smooth Streaming)
D  iss              Funcom ISS
D  iv8              IndigoVision 8000 video
DE ivf              On2 IVF
DE jacosub          JACOsub subtitle format
D  jv               Bitmap Brothers JV
DE latm             LOAS/LATM
D  lavfi            Libavfilter virtual input device
DE libnut            nut format
D  lmlm4            raw lmlm4
D  loas             LOAS AudioSyncStream
D  lvf              LVF
D  lxf              VR native stream (LXF)
DE m4v              raw MPEG-4 video
E  matroska          Matroska
D  matroska,webm    Matroska / WebM
E  md5              MD5 testing
D  mgsts             Metal Gear Solid: The Twin Snakes
DE microdvd          MicroDVD subtitle format
DE mjpeg             raw MJPEG video
E  mkvtimestamp_v2  extract pts as timecode v2 format, as defined by mkvtoolnix
DE mlip              raw MLP
D  mm               American Laser Games MM
DE mmf              Yamaha SMAF
E  mov              QuickTime / MOV
D  mov,mp4,m4a,3gp,3g2,mj2 QuickTime / MOV
E  mp2              MP2 (MPEG audio layer 2)
DE mp3              MP3 (MPEG audio layer 3)
E  mp4              MP4 (MPEG-4 Part 14)
D  mpc              Musepack
D  mpc8             Musepack SV8
DE mpeg             MPEG-1 Systems / MPEG program stream
E  mpeg1video        raw MPEG-1 video
E  mpeg2video        raw MPEG-2 video
DE mpegs            MPEG-TS (MPEG-2 Transport Stream)
```

D	mpegtsraw	raw MPEG-TS (MPEG-2 Transport Stream)
D	mpegvideo	raw MPEG video
E	mpjpeg	MIME multipart JPEG
D	msnwctcp	MSN TCP Webcam stream
D	mtv	MTV
DE	mulaw	PCM mu-law
D	mvi	Motion Pixels MVI
DE	mxf	MXF (Material eXchange Format)
E	mxf_d10	MXF (Material eXchange Format) D-10 Mapping
D	mxg	MxPEG clip
D	nc	NC camera feed
D	nsv	Nullsoft Streaming Video
E	null	raw null video
DE	nut	NUT
D	nuv	NuppelVideo
DE	ogg	Ogg
DE	oma	Sony OpenMG audio
D	paf	Amazing Studio Packed Animation File
D	pmp	Playstation Portable PMP
E	ppsp	PSP MP4 (MPEG-4 Part 14)
D	psxstr	Sony Playstation STR
D	pva	TechnoTrend PVA
D	qcp	QCP
D	r3d	REDCODE R3D
DE	rawvideo	raw video
E	rcv	VC-1 test bitstream
D	realtext	RealText subtitle format
D	rl2	RL2
DE	rm	RealMedia
DE	roq	raw id RoQ
D	rpl	RPL / ARMovie
DE	rso	Lego Mindstorms RSO
DE	rtp	RTP output
DE	rtsp	RTSP output
DE	s16be	PCM signed 16-bit big-endian
DE	s16le	PCM signed 16-bit little-endian
DE	s24be	PCM signed 24-bit big-endian
DE	s24le	PCM signed 24-bit little-endian
DE	s32be	PCM signed 32-bit big-endian
DE	s32le	PCM signed 32-bit little-endian
DE	s8	PCM signed 8-bit
D	sami	SAMI subtitle format
DE	sap	SAP output
D	sbg	SBaGen binaural beats script
E	sdl	SDL output device
D	sdp	SDP
E	segment	segment
D	shn	raw Shorten
D	siff	Beam Software SIFF
DE	smjpeg	Loki SDL MJPEG
D	smk	Smacker
E	smoothstreaming	Smooth Streaming Muxer
D	smush	LucasArts Smush
D	sol	Sierra SOL
DE	sox	SoX native

```
DE spdif           IEC 61937 (used on S/PDIF - IEC958)
DE srt             SubRip subtitle
E stream_segment,segment streaming segment muxer
D subviewer        SubViewer subtitle format
E svcd            MPEG-2 PS (SVCD)
DE swf             SWF (ShockWave Flash)
D tak             raw TAK
D thp             THP
D tiertexseq      TierteX Limited SEQ
D tmv             8088flex TMV
DE truehd         raw TrueHD
D tta             TTA (True Audio)
D tty             Tele-typewriter
D txd             Renderware TeXture Dictionary
DE u16be          PCM unsigned 16-bit big-endian
DE u16le          PCM unsigned 16-bit little-endian
DE u24be          PCM unsigned 24-bit big-endian
DE u24le          PCM unsigned 24-bit little-endian
DE u32be          PCM unsigned 32-bit big-endian
DE u32le          PCM unsigned 32-bit little-endian
DE u8              PCM unsigned 8-bit
D vc1             raw VC-1
D vc1test         VC-1 test bitstream
E vcd             MPEG-1 Systems / MPEG program stream (VCD)
D vfwcap          VFW video capture
D vmd             Sierra VMD
E vob             MPEG-2 PS (VOB)
DE voc            Creative Voice
D vqf             Nippon Telegraph and Telephone Corporation (NTT) TwinVQ
D w64             Sony Wave64
DE wav             WAV / WAVE (Waveform Audio)
D wc3movie         Wing Commander III movie
E webm            WebM
D webvtt          WebVTT subtitle
D wsaud            Westwood Studios audio
D wsvqa            Westwood Studios VQA
DE wtv             Windows Television (WTV)
DE wv              WavPack
D xa               Maxis XA
D xbin             eXtended BINary text (XBIN)
D xmv              Microsoft XMV
D xwma             Microsoft xWMA
D yop              Psygnosis YOP
DE yuv4mpegpipe   YUV4MPEG pipe
```

## 可用的音频通道布局

要显示可用的音频通道布局列表，我们可以使用以下命令：

```
ffmpeg -layouts
```

个性化通道：

名称	分解（描述）
FL	front left
FR	front right
FC	front center
LFE	low frequency
BL	back left
BR	back right
FLC	front left-of-center
FRC	front right-of-center
BC	back center
SL	side left
SR	side right
TC	top center
TFL	top front left
TFC	top front center
TFR	top front right
TBL	top back left
TBC	top back center
TBR	top back right
DL	downmix left
DR	downmix right
WL	wide left
WR	wide right
SDL	surround direct left
SDR	surround direct right
LFE2	low frequency 2

### Individual channels:

NAME	DESCRIPTION
FL	front left
FR	front right
FC	front center
LFE	low frequency
BL	back left
BR	back right
FLC	front left-of-center
FRC	front right-of-center
BC	back center
SL	side left
SR	side right
TC	top center
TFL	top front left
TFC	top front center
TFR	top front right
TBL	top back left
TBC	top back center
TBR	top back right
DL	downmix left
DR	downmix right
WL	wide left
WR	wide right
SDL	surround direct left
SDR	surround direct right
LFE2	low frequency 2

个性化通道

标准通道布局:

名称	分解（描述）
mono	FC
stereo	FL+FR
2.1	FL+FR+LFE
3.0	FL+FR+FC
3.0(back)	FL+FR+BC
4.0	FL+FR+FC+BC
quad	FL+FR+BL+BR
quad(side)	FL+FR+SL+SR
3.1	FL+FR+FC+LFE
5.0	FL+FR+FC+BL+BR
5.0(side)	FL+FR+FC+SL+SR
4.1	FL+FR+FC+LFE+BC
5.1	FL+FR+FC+LFE+BL+BR
5.1(side)	FL+FR+FC+LFE+SL+SR
6.0	FL+FR+FC+BC+SL+SR
6.0(front)	FL+FR+FLC+FRC+SL+SR
hexagonal	FL+FR+FC+BL+BR+BC
6.1	FL+FR+FC+LFE+BC+SL+SR
6.1	FL+FR+FC+LFE+BL+BR+BC
6.1(front)	FL+FR+LFE+FLC+FRC+SL+SR
7.0	FL+FR+FC+BL+BR+SL+SR
7.0(front)	FL+FR+FC+FLC+FRC+SL+SR
7.1	FL+FR+FC+LFE+BL+BR+SL+SR
7.1(wide)	FL+FR+FC+LFE+FLC+FRC+SL+SR
octagonal	FL+FR+FC+BL+BR+BC+SL+SR
downmix	DL+DR

```
Standard channel layouts:
NAME           DECOMPOSITION
mono          FC
stereo         FL+FR
2.1            FL+FR+LFE
3.0            FL+FR+FC
3.0(back)     FL+FR+BC
4.0            FL+FR+FC+BC
quad           FL+FR+BL+BR
quad(side)    FL+FR+SL+SR
3.1            FL+FR+FC+LFE
5.0            FL+FR+FC+BL+BR
5.0(side)     FL+FR+FC+SL+SR
4.1            FL+FR+FC+LFE+BC
5.1            FL+FR+FC+LFE+BL+BR
5.1(side)     FL+FR+FC+LFE+SL+SR
6.0            FL+FR+FC+BC+SL+SR
6.0(front)    FL+FR+FLC+FRC+SL+SR
hexagonal     FL+FR+FC+BL+BR+BC
6.1            FL+FR+FC+LFE+BC+SL+SR
6.1(back)     FL+FR+FC+LFE+BL+BR+BC
6.1(front)    FL+FR+LFE+FLC+FRC+SL+SR
7.0            FL+FR+FC+BL+BR+SL+SR
7.0(front)    FL+FR+FC+FLC+FRC+SL+SR
7.1            FL+FR+FC+LFE+BL+BR+SL+SR
7.1(wide)     FL+FR+FC+LFE+BL+BR+FLC+FRC
7.1(wide-side) FL+FR+FC+LFE+FLC+FRC+SL+SR
octagonal     FL+FR+FC+BL+BR+BC+SL+SR
hexadecagonal FL+FR+FC+BL+BR+BC+SL+SR+TFL+TFC+TFR+TBL+TBC+TBR+WL+WR
downmix       DL+DR
zhangfangtaodeMacBook-Pro:~ zhangfangtao$
```

标准化通道

## FFmpeg许可证

关于FFmpeg许可的信息可以用大写L作为参数显示:

```
ffmpeg -L
```

```
This version of ffmpeg has nonfree parts compiled in.
Therefore it is not legally redistributable.
```

```
ffmpeg -L
```

ffmpeg是自由软件;您可以根据自由软件基金会发布的GNU通用公共许可证的条款重新分配和/或修改它;任何版本3的许可,或(在您的选择)以后的版本。

ffmpeg是分布式的,希望它是有用的,但是没有任何保证;甚至没有对适销性或适合某一特定用途的适用性的默示保证。有关更多细节,请参见GNU通用公共许可证。您应该已经收到了一份GNU通用公共许可证和ffmpeg的副本。如果不是,见<http://www.gnu.org/licenses>

## 可用的像素格式

该命令可以显示内置的像素格式列表:

```
ffmpeg -pix_fmts
```

像素格式:

```
-----  
I.... = Supported Input format for conversion  
.O... = Supported Output format for conversion  
.H.. = Hardware accelerated format  
.P. = Palettized format  
.B = Bitstream format
```

标记名称	NB_COMPONENTS (不知道该咋翻译)	BITS_PER_PIXEL
IO... yuv420p	3	12
IO... yuyv422	3	16
IO... rgb24	3	24
IO... bgr24	3	24
IO... yuv422p	3	16

标记名称	NB_COMPONENTS (不知道该咋翻译)	BITS_PER_PIXEL
IO... yuv444p	3	24
IO... yuv410p	3	9
IO... yuv411p	3	12
IO... gray	1	8
IO..B monow	1	1
IO..B monob	1	1
I..P. pal8	1	8
IO... yuvj420p	3	12
IO... yuvj422p	3	16
IO... yuvj444p	3	24
..H.. xvcmc	0	0
..H.. xvmcidct	0	0
IO... uyvy422	3	16
..... uyyvyy411	3	12
IO... bgr8	3	8
.O..B bgr4	3	4
IO... bgr4_byte	3	4
IO... rgb8	3	8
.O..B rgb4	3	4
IO... rgb4_byte	3	4
IO... nv12	3	12
IO... nv21	3	12
IO... argb	4	32
IO... rgba	4	32
IO... abgr	4	32
IO... bgra	4	32
IO... gray16be	1	16
IO... gray16le	1	16
IO... yuv440p	3	16
IO... yuvj440p	3	16

标记名称	NB_COMPONENTS (不知道该咋翻译)	BITS_PER_PIXEL
IO... yuva420p	4	20
..H.. vdpaу_h264	0	0
..H.. vdpaу_mpeg1	0	0
..H.. vdpaу_mpeg2	0	0
..H.. vdpaу_wmv3	0	0
..H.. vdpaу_vc1	0	0
IO... rgb48be	3	48
IO... rgb48le	3	48
IO... rgb565be	3	16
IO... rgb565le	3	16
IO... rgb555be	3	15
IO... rgb555le	3	15
IO... bgr565be	3	16
IO... bgr565le	3	16
IO... bgr555be	3	15
IO... bgr555le	3	15
..H.. vaapi_moco	0	0
..H.. vaapi_idct	0	0
..H.. vaapi_vld	0	0
IO... yuv420p16le	3	24
IO... yuv420p16be	3	24
IO... yuv422p16le	3	32
IO... yuv422p16be	3	32
IO... yuv444p16le	3	48
IO... yuv444p16be	3	48
..H.. vdpaу_mpeg4	0	0
..H.. dxva2_vld	0	0
IO... rgb444le	3	12
IO... rgb444be	3	12
IO... bgr444le	3	12

标记名称	NB_COMPONENTS (不知道该咋翻译)	BITS_PER_PIXEL
IO... bgr444be	3	12
I.... gray8a	2	16
IO... bgr48be	3	48
IO... bgr48le	3	48
IO... yuv420p9be	3	13
IO... yuv420p9le	3	13
\IO... yuv420p10be	3	15
IO... yuv420p10le	3	15
IO... yuv422p10be	3	20
IO... yuv422p10le	3	20
IO... yuv444p9be	3	27
IO... yuv444p9le	3	27
IO... yuv444p10be	3	30
IO... yuv444p10le	3	30
IO... yuv422p9be	3	18
IO... yuv422p9le	3	18
..H.. vda_vld	0	0
I.... gbrp	3	24
I.... gbrp9be	3	27
I.... gbrp9le	3	27
I.... gbrp10be	3	30
I.... gbrp10le	3	30
I.... gbrp16be	3	48
I.... gbrp16le	3	48
IO... yuva420p9be	4	22
IO... yuva420p9le	4	22
IO... yuva422p9be	4	27
IO... yuva422p9le	4	27
IO... yuva444p9be	4	36
IO... yuva444p9le	4	36

标记名称	NB_COMPONENTS (不知道该咋翻译)	BITS_PER_PIXEL
IO... yuva420p10be	4	25
IO... yuva420p10le	4	40
IO... yuva422p10be	4	48
IO... yuva422p10le	4	48
IO... yuva444p10be	4	64
IO... yuva444p10le	4	64
IO... yuva420p16be	4	40
IO... yuva420p16le	4	40
IO... yuva422p16be	4	48
IO... yuva422p16le	4	48
IO... yuva444p16be	4	64
IO... yuva444p16le	4	64
I.... rgba64be	4	64
I.... rgba64le	4	64
..... bgra64be	4	64
..... bgra64le	4	64
IO... 0rgb	3	24
IO... rgb0	3	24
IO... 0bgr	3	24
IO... bgr0	3	24
IO... yuva444p	4	32
IO... yuva422p	4	24
IO... yuv420p12be	3	18
IO... yuv420p12le	3	18
IO... yuv420p14be	3	21
IO... yuv420p14le	3	21
IO... yuv422p12be	3	24
IO... yuv422p12le	3	24
IO... yuv422p14be	3	28
IO... yuv422p14le	3	28

标记名称	NB_COMPONENTS (不知道该咋翻译)	BITS_PER_PIXEL
IO... yuv444p12be	3	36
IO... yuv444p12le	3	36
IO... yuv444p14be	3	42
IO... yuv444p14le	3	42
I.... gbrp12be	3	36
I.... gbrp12le	3	36
I.... gbrp14be	3	42
I.... gbrp14le	3	42

FLAGS	NAME	NB_COMPONENTS	BITS_PER_PIXEL
<hr/>			
IO...	yuv420p	3	12
IO...	yuyv422	3	16
IO...	rgb24	3	24
IO...	bgr24	3	24
IO...	yuv422p	3	16
IO...	yuv444p	3	24
IO...	yuv410p	3	9
IO...	yuv411p	3	12
IO...	gray	1	8
IO..B	monow	1	1
IO..B	monob	1	1
I..P.	pal8	1	8
IO...	yuvj420p	3	12
IO...	yuvj422p	3	16
IO...	yuvj444p	3	24
IO...	uyvy422	3	16
.....	uyyvyy411	3	12
IO...	bgr8	3	8
.O..B	bgr4	3	4
IO...	bgr4_byte	3	4
IO...	rgb8	3	8
.O..B	rgb4	3	4
IO...	rgb4_byte	3	4
IO...	nv12	3	12
IO...	nv21	3	12
IO...	argb	4	32
IO...	rgba	4	32
IO...	abgr	4	32
IO...	bgra	4	32
IO...	gray16be	1	16
IO...	gray16le	1	16
IO...	yuv440p	3	16
IO...	yuvj440p	3	16
IO...	yuva420p	4	20
IO...	rgb48be	3	48
IO...	rgb48le	3	48
IO...	rgb565be	3	16
IO...	rgb565le	3	16
IO...	rgb555be	3	15
IO...	rgb555le	3	15
IO...	bgr565be	3	16
IO...	bgr565le	3	16
IO...	bgr555be	3	15
IO...	bgr555le	3	15

## 可用的协议

对于显示可用的文件协议，下一个命令是：

```
ffmpeg -protocols
```

下面是支持的文件协议：

**Input:** (输入)

applehttp

cache

concat

crypto

file

gopher

hls

http

httpproxy

mms

mmst

pipe

rtp

tcp

udp

rtmp

rtmpe

rtmps

rtmpt

rtmpte

**Output:**(输出)

file

gopher

http

httpproxy

md5

pipe

rtp

tcp

udp

**Input:** (输入)

rtmp

rtmpe

rtmps

rtmpt

rtmpte

Supported file protocols:

Input:

async  
cache  
concat  
crypto  
data  
ffrtmphttp  
file  
ftp  
gopher  
hls  
http  
httpproxy  
https  
mmsh  
mmst  
pipe  
rtmp  
rtmps  
rtmpt  
rtmpts  
rtp  
srtp  
subfile  
tcp  
tls  
udp  
udplite  
unix

Output:

crypto  
ffrtmphttp  
file  
ftp  
gopher  
http

```
http proxy  
https  
icecast  
md5  
pipe  
prompeg  
rtmp  
rtmps  
rtmpt  
rtmpts  
rtp  
srtp  
tee  
tcp
```

## 可用的音频样本格式

FFmpeg中包含的音频样本格式可以通过命令显示:

```
ffmpeg -sample_fmts
```

名称	位深度
u8	8
s16	16
s32	32
flt	32
dbl	64
u8p	8
s16p	16
s32p	32
fltp	32
dblfp	64

我电脑上显示出来的样本格式多了两个

<b>name</b>	<b>depth</b>
<b>u8</b>	<b>8</b>
<b>s16</b>	<b>16</b>
<b>s32</b>	<b>32</b>
<b>flt</b>	<b>32</b>
<b>dbl</b>	<b>64</b>
<b>u8p</b>	<b>8</b>
<b>s16p</b>	<b>16</b>
<b>s32p</b>	<b>32</b>
<b>fltp</b>	<b>32</b>
<b>dblfp</b>	<b>64</b>
<b>s64</b>	<b>64</b>
<b>s64p</b>	<b>64</b>

## FFmpeg版本

版本的ffmpeg可以显示为 -version 选项，下一个结果是在2012年11月25日创建的官方构建窗口的显示结果。

```
ffmpeg -version
```

```
ffmpeg version N-47062-g26c531c
built on Nov 25 2012 12:21:26 with gcc 4.7.2 (GCC)
configuration: --enable-gpl --enable-version3 --disable-pthreads --enable-runtime-cpudetect --enable-avisynth --enable-bzlib --enable-frei0r --enable-libass --enable-libopencore-amrnb --enable-libopencore-amrwb --enable-libfreetype --enable-libgsm --enable-libmp3lame --enable-libnut --enable-libopenjpeg --enable-libopus --enable-librtmp --enable-libschroedinger --enable-libspeex --enable-libtheora --enable-libutvideo --enable-libvo-aacenc --enable-libvo-amrwbenc --enable-libvorbis --enable-libvpx --enable-libx264 --enable-libxavs --enable-libxvid --enable-zlib
libavutil      52. 9.100 / 52. 9.100
libavcodec     54. 77.100 / 54. 77.100
libavformat    54. 37.100 / 54. 37.100
libavdevice    54.  3.100 / 54.  3.100
libavfilter     3. 23.102 /  3. 23.102
libswscale      2.  1.102 /  2.  1.102
libswresample   0. 17.101 /  0. 17.101
libpostproc    52.  2.100 / 52.  2.100
```

FFmpeg版本信息

有关ffmpeg配置的详细信息，请参见[词汇表](#)。

## 使用MORE命令来实现输出格式。

由于help命令、可用过滤器命令、格式命令等的输出很长并且通常不适合一个屏幕，所以可以使用更多命令来显示从一开始就按顺序将输出文本进行格式化，下一个屏幕由按下空格键，按Enter键后显示下一行(只会多显示一行)，按Q或q将退出预览。语法是

```
ffmpeg -help | more
```

---

-version	show version
-buildconf	show build configuration
-formats	show available formats
-muxers	show available muxers
-demuxers	show available demuxers
-devices	show available devices
-codecs	show available codecs
-decoders	show available decoders
-encoders	show available encoders
-bsfs	show available bit stream filters
-protocols	show available protocols
-filters	show available filters
-pix_fmts	show available pixel formats
-layouts	show standard channel layouts
-sample_fmts	show available audio sample formats
-colors	show available color names
-sources device	list sources of the input device
-sinks device	list sinks of the output device
-hwaccels	show available HW acceleration methods

Global options (affect whole program instead of just one file:

```
-loglevel loglevel set logging level  
-v loglevel set logging level
```

```
:
```

想看更多信息的，敲Enter键就可以

或使用更短的形式：

```
ffmpeg -h|more
```

还可以使用更多的命令来显示文本文件，内容再次被划分为适合一个屏幕，语法是：

```
more filename.txt
```

可以通过输入显示更多命令的附加参数：

```
help more
```

## 重定向输出到文件

有时需要精确研究帮助命令，可用过滤器命令，格式命令等，为了将这些信息保存到文本文件中，可以使用下一个命令：

```
ffmpeg -help > help.txt
```

该命令将在当前目录中创建一个名为help.txt的新文件，并将其保存到ffmpeg help的内容中。如果具有相同名称的文件已经存在，它将被覆盖。要将输出文本附加到现有文件而不覆盖其内容，请使用两个大于符号">":

```
ffmpeg -help > data.txt  
ffmpeg -filters >> data.txt
```

现在文件data.txt包含ffmpeg帮助，后面跟着可用过滤器列表。

下图是我打印出来的data.txt信息：



The screenshot shows a terminal window titled "data.txt" displaying the output of the ffmpeg -help command. The output is a detailed list of command-line options and their descriptions, organized into sections such as "Getting help", "Print help / information / capabilities", and "Global options (affect whole program instead of just one file)". The text is in black font on a white background, with some options highlighted in red.

```
Hyper fast Audio and Video encoder  
usage: ffmpeg [options] [[infile options] -i infile]... {[outfile options] outfile}...  
  
Getting help:  
  -h      -- print basic options  
  -h long -- print more options  
  -h full -- print all options (including all format and codec specific options, very  
long)  
  -h type=name -- print all options for the named decoder/encoder/demuxer/muxer/filter/  
bsf  
  See man ffmpeg for detailed description of the options.  
  
Print help / information / capabilities:  
-L                  show license  
-h topic            show help  
-? topic            show help  
-help topic         show help  
--help topic        show help  
-version           show version  
-buildconf          show build configuration  
-formats            show available formats  
-muxers             show available muxers  
-demuxers           show available demuxers  
-devices            show available devices  
-codecs             show available codecs  
-decoders           show available decoders  
-encoders           show available encoders  
-bsfs               show available bit stream filters  
-protocols          show available protocols  
-filters             show available filters  
-pix_fmts           show available pixel formats  
-layouts             show standard channel layouts  
-sample_fmts         show available audio sample formats  
-colors              show available color names  
-sources device     list sources of the input device  
-sinks device       list sinks of the output device  
-hwaccel             show available HW acceleration methods  
  
Global options (affect whole program instead of just one file):  
-loglevel loglevel   set logging level  
-v loglevel          set logging level  
-report              generate a report  
-max_alloc bytes    set maximum size of a single allocated block  
-y                  overwrite output files  
-n                  never overwrite output files  
-ignore_unknown     Ignore unknown stream types  
-filter_threads     number of non-complex filter threads  
-filter_complex_threads number of threads for -filter_complex  
-stats              print progress report during encoding  
-max_error_rate     maximum error rate ratio of errors (0.0: no errors, 1.0: 100% errors)  
above which ffmpeg returns an error instead of success.  
-bits per raw sample number  set the number of bits per raw sample
```

## 03-比特率/帧率/文件大小

比特率和帧速率是视频的基本特征，它们的正确设置对整体视频质量非常重要。如果我们知道所有包含的媒体流的比特率和持续时间，我们可以计算输出文件的最终大小。由于在使用FFmpeg工具时对帧速率和比特率的理解很重要，因此包含每个术语的简短描述。

## 帧率(频率)的介绍

帧速率是编码成视频文件的每秒帧数（FPS或fps），人眼需要至少约15 fps来观看连续运动。帧率也称为帧频，其单位是赫兹（Hz），LCD显示器通常具有60 Hz的频率。

有两种帧速率 - 隔行（在FPS编号后表示为i）和逐行（在FPS编号后表示为p）。

在电视中使用隔行帧率：

\*NTSC标准使用60i fps，意味着每秒隔行扫描60次（30帧）

- PAL和SECAM标准使用50i fps，这意味着50隔行场，相当于每秒25帧24p, 25p和30p的逐行帧率被用于电影行业。高端HDTV产品使用较高的帧频50p / 60p。

常见的视频帧率

### Common video frame rates

蓝色的，，我弄不出来，就用图片代替了

FPS i=interlaced p=progressive	描述
24p or 23.976	从20世纪20年代开始，电影行业的标准帧速率，所有的电影都是以这个频率拍摄的。当这些电影被采用到NTSC电视广播时，帧速率降低到 $24 \times 1000 / 1001 = 23.976$ 值，但是对于PAL / SECAM电视，电影的帧速率增加到25帧/秒。
25p	由于25个逐行扫描视频可轻松转换为50个隔行扫描电视场，因此电影频率为50赫兹（PAL和SECAM标准）的国家中的电影和电视的标准帧频。
30p	常见的视频帧速率，常用于数码相机和摄像机。它可用于60赫兹（NTSC）隔行场的电视广播。
50i	PAL和SECAM电视的标准场率（隔行帧率）。
60ior 59.94	NTSC电视的标准场频率，在彩电发明之后，帧速率被降低到 $60 * 1000 / 1001 = 59.94$ 的值，以防止色度副载波和声音载波之间的干扰。
50p/60p	HDTV（高清晰度电视）的通用帧频。
48p	提议的帧速率，目前经过测试了
72p	提议的帧速率，目前经过测试了
120p	为UHDTV（超高清电视）标准化的渐进式格式，计划成为UHDTV的单一全球“双精度”帧速率（而不是使用PAL标准的100 Hz和NTSC标准的119.88 Hz）

## 帧率设置

### 使用- r 选项

要设置视频帧速率，我们在输出文件之前使用-r选项，语法是：

```
ffmpeg -i input -r fps output
```

例如改变电影的帧率。avi文件从25到30 fps值，我们使用命令：

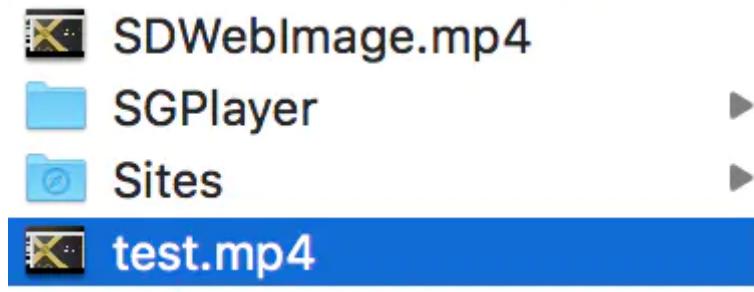
```
ffmpeg -i input.avi -r 30 output.mp4
```

我给大家演示一下，我把我的根目录下面的一个SDWebImage.mp4文件输出为30fps的test.mp4文件使用的命令如下：（你用自己的视频文件做测试，别瞎模仿）

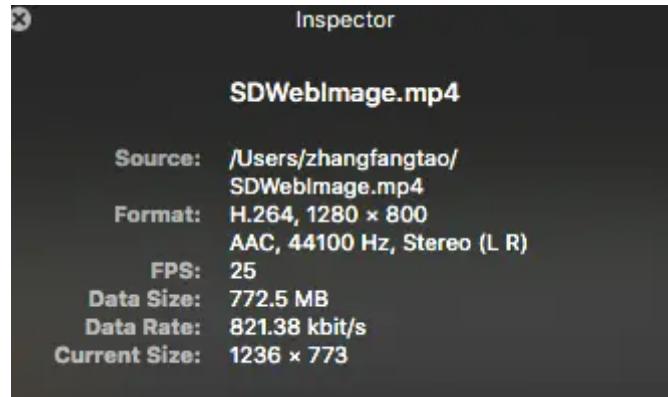
```
ffmpeg -i /Users/zhangfangtao/SDWebImage.mp4 -r 30 test.mp4

  creation_time      : 2018-02-17T14:57:18.000000Z
Stream mapping:
  Stream #0:0 -> #0:0 (h264 (native) -> h264 (libx264))
  Stream #0:1 -> #0:1 (aac (native) -> aac (native))
Press [q] to stop, [?] for help
[libx264 @ 0x7fac2100ea00] using cpu capabilities: MMX2 SSE2Fast SSSE3 SSE4.2 AVX FMA3 AVX2 LZCNT BMI2
[libx264 @ 0x7fac2100ea00] profile High, level 3.2
[libx264 @ 0x7fac2100ea00] 264 - core 148 r2795 aaa9aa8 - H.264/MPEG-4 AVC codec
  - Copyleft 2003-2017 - http://www.videolan.org/x264.html - options: cabac=1 ref=3 deblock=1:0:0 analyse=0x3:0x113 me=hex subme=7 psy=1 psy_rd=1.00:0.00 mixed_rf=1 me_range=16 chroma_me=1 trellis=1 8x8dct=1 cqm=0 deadzone=21,11 fast_pskip=1 chroma_qp_offset=-2 threads=6 lookahead_threads=1 sliced_threads=0 nr=0 decimal=1 interlaced=0 bluray_compat=0 constrained_intra=0 bframes=3 b_pyramid=2 b_adapt=1 b_bias=0 direct=1 weightb=1 open_gop=0 weightp=2 keyint=250 keyint_min=25 scenecut=40 intra_refresh=0 rc_lookahead=40 rc=crf mbtree=1 crf=23.0 qcomp=0.60 qpmin=0 qpmax=69 qpstep=4 ip_ratio=1.40 aq=1:1.00
Output #0, mp4, to 'test.mp4':
  Metadata:
    major_brand     : mp42
    minor_version   : 512
    compatible_brands: isomiso2avc1mp41
    encoder         : Lavf58.13.100
  Stream #0:0(und): Video: h264 (libx264) (avc1 / 0x31637661), yuv420p(progressive), 1280x800, q=-1--1, 30 fps, 15360 tbn, 30 tbc (default)
  Metadata:
    creation_time   : 2018-02-17T14:57:18.000000Z
    encoder         : Lavc58.19.100 libx264
  Side data:
    cpb: bitrate max/min/avg: 0/0/0 buffer size: 0 vbv_delay: -1
  Stream #0:1(und): Audio: aac (LC) (mp4a / 0x6134706D), 44100 Hz, stereo, fltp, 128 kb/s (default)
  Metadata:
    creation_time   : 2018-02-17T14:57:18.000000Z
    encoder         : Lavc58.19.100 aac
frame=  68 fps=0.0 q=29.0 size=          0kB time=00:00:01.99 bitrate=  0.2kbits/
frame= 139 fps=135 q=29.0 size=          0kB time=00:00:04.04 bitrate=  0.1kbits/
frame= 203 fps=131 q=29.0 size=          0kB time=00:00:06.10 bitrate=  0.1kbits/
```

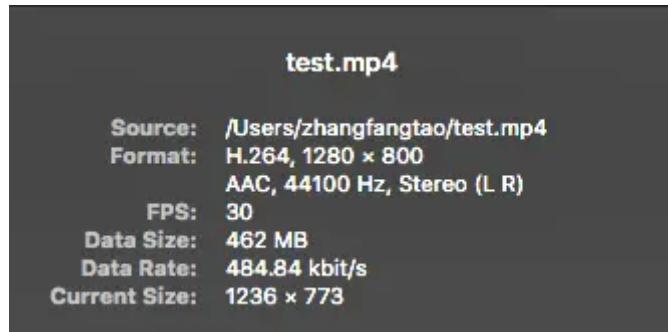
看到没，下面真的就出来了输出的test.mp4：



先让你们看一下之前的视频信息是什么：



再让你们看一下现在的视频信息是什么：



- 有没有发现，重新编码以后不仅仅fps变化了，Data Rate也改变了？因为我没有设置，FFmpeg使用了默认的比特率，重新编码之后，自动降码了。所以，建议各位做转码的时候把参数填的全面一些，要不然会有意外存在。

当使用原始输入格式时，-r选项也可以在输入之前使用。

## 使用fps过滤器

另一种设置帧速率的方法是使用fps过滤器，这在过滤链中尤其有用。

描述	将视频帧速率更改为指定的值。
Syntax	fps=fps=number_of_frames
fps	指定输出帧速率的数字或预定义缩写。

例如，要更改剪辑的输入帧速率。mpg文件到值25，我们使用命令。

```
ffmpeg -i clip.mpg -vf fps=fps=25 clip.webm
```

我亲自给大家测试一下，我用的是我刚才转码出来的test.mp4，转码成为test.webm,具体的命令行如下：

```
ffmpeg -i /Users/zhangfangtao/test.mp4 -vf fps=fps=25 test.webm
```

```

Last login: Fri Apr 27 14:10:35 on ttys001
[zhangfangtaodeMacBook-Pro:~ zhangfangtao$ ffmpeg -i /Users/zhangfangtao/test.mp4
-vf fps=fps=25 test.webm
ffmpeg version N-90810-g153e920892 Copyright (c) 2000-2018 the FFmpeg developers
built with Apple LLVM version 9.1.0 (clang-902.0.39.1)
configuration: --prefix=/usr/local --enable-gpl --enable-nonfree --enable-liba
ss --enable-libfdk-aac --enable-libfreetype --enable-libmp3lame --enable-libopus
--enable-libtheora --enable-libvorbis --enable-libvpx --enable-libx264 --enable
-libxvid --extra-ldflags=-L/usr/local/lib
libavutil      56. 15.100 / 56. 15.100
libavcodec     58. 19.100 / 58. 19.100
libavformat    58. 13.100 / 58. 13.100
libavdevice     58.  4.100 / 58.  4.100
libavfilter     7. 19.100 / 7. 19.100
libswscale      5.  2.100 / 5.  2.100
libswresample   3.  2.100 / 3.  2.100
libpostproc    55.  2.100 / 55.  2.100
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from '/Users/zhangfangtao/test.mp4':
Metadata:
  major_brand     : isom
  minor_version   : 512
  compatible_brands: isomiso2avc1mp41
  encoder         : Lavf58.13.100
Duration: 02:04:48.65, start: 0.000000, bitrate: 493 kb/s
  Stream #0:0(und): Video: h264 (High) (avc1 / 0x31637661), yuv420p, 1280x800,
353 kb/s, 30 fps, 30 tbr, 15360 tbn, 60 tbc (default)
  Metadata:
    handler_name   : VideoHandler
  Stream #0:1(und): Audio: aac (LC) (mp4a / 0x6134706D), 44100 Hz, stereo, flt
p, 131 kb/s (default)
  Metadata:
    handler_name   : SoundHandler
Stream mapping:
  Stream #0:0 -> #0:0 (h264 (native) -> vp9 (libvpx-vp9))
  Stream #0:1 -> #0:1 (aac (native) -> opus (libopus))
Press [q] to stop, [?] for help
[libopus @ 0x7ff7e6805000] No bit rate set. Defaulting to 96000 bps.
[libvpx-vp9 @ 0x7ff7e6803e00] v1.6.1
Output #0, webm, to 'test.webm':
Metadata:
  major_brand     : isom
  minor_version   : 512
  compatible_brands: isomiso2avc1mp41
  encoder         : Lavf58.13.100
  Stream #0:0(und): Video: vp9 (libvpx-vp9), yuv420p, 1280x800, q=-1--1, 200 k
b/s, 25 fps, 1k tbn, 25 tbc (default)
  Metadata:
    handler_name   : VideoHandler
    encoder        : Lavc58.19.100 libvpx-vp9
  Side data:
    cpb: bitrate max/min/avg: 0/0/0 buffer size: 0 vbv_delay: -1
  Stream #0:1(und): Audio: opus (libopus), 48000 Hz, stereo, flt, 96 kb/s (def
ault)
  Metadata:
    handler_name   : SoundHandler
    encoder        : Lavc58.19.100 libopus
frame=  25 fps= 13 q=0.0 size=      1kB time=00:00:01.15 bitrate=  7.4kbits/s
frame=  28 fps= 10 q=0.0 size=      1kB time=00:00:01.25 bitrate=  6.8kbits/s
frame=  29 fps=8.1 q=0.0 size=      1kB time=00:00:01.31 bitrate=  6.5kbits/s
frame=  30 fps=6.7 q=0.0 size=      1kB time=00:00:01.35 bitrate=  6.3kbits/s
frame=  31 fps=5.7 q=0.0 size=      1kB time=00:00:01.37 bitrate=  6.2kbits/s
frame=  32 fps=5.0 q=0.0 size=      1kB time=00:00:01.41 bitrate=  6.0kbits/s
frame=  33 fps=4.6 q=0.0 size=      1kB time=00:00:01.45 bitrate=  5.8kbits/s

```

因为没有设置bit rate, 所以给了一个默认值96000bps

## 帧速率的预定义值。

除了数值, 设置帧率的两种方法都接受下一个预定义的文本值:

帧速率的预定义缩写。

缩写	精确值	相应的FPS (相应的帧)
ntsc-film	24000/1001	23.97
film	24/1	24
pal, qpal, spal	25/1	25
ntsc, qntsc, sntsc	30000/1001	29.97

例如，设置帧速率为29.97 fps，接下来的3个命令给出了相同的结果：这个就不用我给大家示范了吧。。。

```
ffmpeg -i input.avi -r 29.97 output.mpg
ffmpeg -i input.avi -r 30000/1001 output.mpg
ffmpeg -i input.avi -r ntsc output.mpg
```

## 位(数据)率的介绍

比特率（也是比特率或数据率）是决定整体音频或视频质量的参数。它规定了每时间单位处理的位数，在FFmpeg中，位速率以每秒位数表示。

### Types of bit rate

类型的比特率

类型	缩写	描述
平均比特率	ABR	平均每秒处理的位数，该值也用于VBR编码，需要时是输出的某个文件大小
恒定比特率	CBR	每秒处理的比特数是恒定的，这对于存储是不实际的，因为具有快速运动的部分需要比静态比特更多的比特，CBR主要用于多媒体流
可变比特率	VBR	每秒处理的比特数是可变的，复杂的场景或声音被编码更多的数据并与CBR进行比较，相同尺寸的文件的VBR质量比CBR更好（VBR编码比CBR需要更多的时间和CPU功率，但最近的媒体播放器可以充分解码VBR。）

## 设置比特率

比特率决定了存储1秒编码流的位数，它使用-b选项设置，以区分推荐使用-b: a或-b: v格式的音频和视频流。例如，要设置总体1.5 Mbit / s的比特率，我们可以使用以下命令：

```
ffmpeg -i film.avi -b 1.5M film.mp4
```

我是用的是我电脑桌面上的一个视频，使用的代码如下：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -b 1.5M newTest.mp4
```

原来的视频信息是：

# Video

<i>ID</i> :	1
<i>Format</i> :	AVC
<i>Format/Info</i> :	Advanced Video Codec
<i>Format profile</i> :	High@L4
<i>Format settings</i> :	CABAC / 4 Ref Frames
<i>Format settings, CABAC</i> :	Yes
<i>Format settings, RefFrames</i> :	4 frames
<i>Format settings, GOP</i> :	M=3, N=75
<i>Codec ID</i> :	avc1
<i>Codec ID/Info</i> :	Advanced Video Coding
<i>Duration</i> :	12 min 48 s
<i>Source duration</i> :	12 min 48 s
<i>Bit rate</i> :	198 kb/s
<i>Width</i> :	1 024 pixels
<i>Height</i> :	768 pixels
<i>Display aspect ratio</i> :	4:3
<i>Frame rate mode</i> :	Constant
<i>Frame rate</i> :	15.000 FPS
<i>Color space</i> :	YUV
<i>Chroma subsampling</i> :	4:2:0
<i>Bit depth</i> :	8 bits
<i>Scan type</i> :	Progressive
<i>Bits/(Pixel*Frame)</i> :	0.017
<i>Stream size</i> :	18.1 MiB (73%)
<i>Source stream size</i> :	19.7 MiB (79%)
<i>Language</i> :	English
<i>Encoded date</i> :	UTC 2012-08-22 19:08:48
<i>Tagged date</i> :	UTC 2012-08-22 19:08:48
<i>Color range</i> :	Limited
<i>Color primaries</i> :	BT.709
<i>Transfer characteristics</i> :	BT.709
<i>Matrix coefficients</i> :	BT.709
<i>mdhd_Duration</i> :	768388

转码之后的视频信息是：

# Video

<i>ID :</i>	1
<i>Format :</i>	AVC
<i>Format/Info :</i>	Advanced Video Codec
<i>Format profile :</i>	High@L3.1
<i>Format settings :</i>	CABAC / 4 Ref Frames
<i>Format settings, CABAC :</i>	Yes
<i>Format settings, RefFrames :</i>	4 frames
<i>Codec ID :</i>	avc1
<i>Codec ID/Info :</i>	Advanced Video Coding
<i>Duration :</i>	12 min 48 s
<i>Bit rate :</i>	1 422 kb/s
<i>Nominal bit rate :</i>	1 500 kb/s
<i>Width :</i>	1 024 pixels
<i>Height :</i>	768 pixels
<i>Display aspect ratio :</i>	4:3
<i>Frame rate mode :</i>	Constant
<i>Frame rate :</i>	15.000 FPS
<i>Color space :</i>	YUV
<i>Chroma subsampling :</i>	4:2:0
<i>Bit depth :</i>	8 bits
<i>Scan type :</i>	Progressive
<i>Bits/(Pixel*Frame) :</i>	0.121
<i>Stream size :</i>	130 MiB (91%)
<i>Writing library :</i>	x264 core 148 r2795 aaa9aa8 cabac=1 / ref=3 / deblock=1:0:0 / analyse=0x3:0x113 / me=hex / subme=7 / psy=1 / psy_rd=1.00:0.00 / mixed_ref=1 / me_range=16 / chroma_me=1 / trellis=1 / 8x8dct=1 / cqm=0 / deadzone=21,11 / fast_pskip=1 / chroma_qp_offset=-2 / threads=6 / lookahead_threads=1 / sliced_threads=0 / nr=0 / decimate=1 / interlaced=0 / bluray_compat=0 / constrained_intra=0 / bframes=3 / b_pyramid=2 / b_adapt=1 / b_bias=0 / direct=1 / weightb=1 / open_gop=0 / weightp=2 / keyint=250 / keyint_min=15 / scenecut=40 / intra_refresh=0 / rc_lookahead=40 / rc=abr / mbtree=1 / bitrate=1500 / ratetol=1.0 / qcomp=0.60 / qpmin=0 / qpmax=69 / qpstep=4 / ip_ratio=1.40 / aq=1:1.00
<i>Encoding settings :</i>	
<i>Language :</i>	English

如果可能的话，ffmpeg使用一个可变比特率（VBR），并对比具有快速运动的部分具有更少比特的静态部分进行编码。ffmpeg通常用于使用高级编解码器来降低输出文件的比特率和相应的文件大小，例如：

```
ffmpeg -i input.avi -b:v 1500k output.mp4
```

该命令将输入比特率更改为每秒1500千比特。

在我电脑上测试的命令行是：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -b:v 1500k  
/Users/zhangfangtao/Desktop/newTest.mp4
```

转码之后的结果如下图：

# Video

<i>ID :</i>	1
<i>Format :</i>	AVC
<i>Format/Info :</i>	Advanced Video Codec
<i>Format profile :</i>	High@L3.1
<i>Format settings :</i>	CABAC / 4 Ref Frames
<i>Format settings, CABAC :</i>	Yes
<i>Format settings, RefFrames :</i>	4 frames
<i>Codec ID :</i>	avc1
<i>Codec ID/Info :</i>	Advanced Video Coding
<i>Duration :</i>	12 min 48 s
<i>Bit rate :</i>	1 422 kb/s
<i>Nominal bit rate :</i>	1 500 kb/s
<i>Width :</i>	1 024 pixels
<i>Height :</i>	768 pixels
<i>Display aspect ratio :</i>	4:3
<i>Frame rate mode :</i>	Constant
<i>Frame rate :</i>	15.000 FPS
<i>Color space :</i>	YUV
<i>Chroma subsampling :</i>	4:2:0
<i>Bit depth :</i>	8 bits
<i>Scan type :</i>	Progressive
<i>Bits/(Pixel*Frame) :</i>	0.121
<i>Stream size :</i>	130 MiB (91%)
<i>Writing library :</i>	x264 core 148 r2795 aaa9aa8 cabac=1 / ref=3 / deblock=1:0:0 / analyse=0x3:0x113 / me=hex / subme=7 / psy=1 / psy_rd=1.00:0.00 / mixed_ref=1 / me_range=16 / chroma_me=1 / trellis=1 / 8x8dct=1 / cqm=0 / deadzone=21,11 / fast_pskip=1 / chroma_qp_offset=-2 / threads=6 / lookahead_threads=1 / sliced_threads=0 / nr=0 / decimate=1 / interlaced=0 / bluray_compat=0 / constrained_intra=0 / bframes=3 / b_pyramid=2 / b_adapt=1 / b_bias=0 / direct=1 / weightb=1 / open_gop=0 / weightp=2 / keyint=250 / keyint_min=15 / scenecut=40 / intra_refresh=0 / rc_lookahead=40 / rc=abr / mbtree=1 / bitrate=1500 / ratetol=1.0 / qcomp=0.60 / qpmin=0 / qpmax=69 / qpstep=4 / ip_ratio=1.40 / aq=1:1.00
<i>Encoding settings :</i>	
<i>Language :</i>	English

## 固定比特率(CBR)设置

例如视频会议之类的实时视频流，可以使用固定的比特率，因为传输的数据不能被缓冲。为了设置输出的恒定比特率，三个参数必须具有相同的值:比特率(-b选项)、最小速率(-minrate)和最大速率(-maxrate)。对于minrate和maxrate选项可以添加一个流指示符，maxrate选项需要设置一个-bufsize选项(比特的速率控制缓冲区大小)。例如，要设置0.5 Mbit/s的CBR，我们可以使用以下命令：

```
ffmpeg -i in.avi -b 0.5M -minrate 0.5M -maxrate 0.5M -bufsize 1M out.mkv
```

我自己测试的命令行是：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -b 0.5M -minrate 0.5M -maxrate 0.5M -bufsize 1M /Users/zhangfangtao/Desktop/newTest.mp4
```

输出的视频信息是：

## Video

<i>ID</i> :	1
<i>Format</i> :	AVC
<i>Format/Info</i> :	Advanced Video Codec
<i>Format profile</i> :	High@L3.1
<i>Format settings</i> :	CABAC / 4 Ref Frames
<i>Format settings, CABAC</i> :	Yes
<i>Format settings, RefFrames</i> :	4 frames
<i>Codec ID</i> :	avc1
<i>Codec ID/Info</i> :	Advanced Video Coding
<i>Duration</i> :	12 min 48 s
<i>Bit rate</i> :	416 kb/s
<i>Nominal bit rate</i> :	500 kb/s
<i>Width</i> :	1 024 pixels
<i>Height</i> :	768 pixels
<i>Display aspect ratio</i> :	4:3
<i>Frame rate mode</i> :	Constant
<i>Frame rate</i> :	15.000 FPS
<i>Color space</i> :	YUV
<i>Chroma subsampling</i> :	4:2:0
<i>Bit depth</i> :	8 bits
<i>Scan type</i> :	Progressive
<i>Bits/(Pixel*Frame)</i> :	0.035
<i>Stream size</i> :	38.1 MiB (76%)
<i>Writing library</i> :	x264 core 148 r2795 aaa9aa8 cabac=1 / ref=3 / deblock=1:0:0 / analyse=0x3:0x113 / me=hex / subme=7 / psy=1 / psy_rd=1.00:0.00 / mixed_ref=1 / me_range=16 / chroma_me=1 / trellis=1 / 8x8dct=1 / cqm=0 / deadzone=21,11 / fast_pskip=1 / chroma_qp_offset=-2 / threads=6 / lookahead_threads=1 / sliced_threads=0 / nr=0 / decimate=1 / interlaced=0 / bluray_compat=0 / constrained_intra=0 / bframes=3 / b_pyramid=2 / b_adapt=1 / b_bias=0 / direct=1 / weightb=1 / open_gop=0 / weightp=2 / keyint=250 / keyint_min=15 / scenecut=40 / intra_refresh=0 / rc_lookahead=40 / rc=cbr / mbtree=1 / bitrate=500 / ratetol=1.0 / qcomp=0.60 / qpmin=0 / qpmax=69 / qpstep=4 / vbv_maxrate=500 / vbv_bufsize=1000 / nal_hrd=none / filler=0 / ip_ratio=1.40 / aq=1:1.00
<i>Encoding settings</i> :	
<i>Language</i> :	English

设置输出文件的最大尺寸

为了使输出文件的大小保持一定的值，我们使用`-fs`选项（文件大小的缩写），期望值以字节为单位。例如，要指定10兆字节的最大输出文件大小，我们可以使用以下命令：

```
ffmpeg -i input.avi -fs 10MB output.mp4
```

给大家看一下我自己的测试命令行：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -fs 1MB  
/Users/zhangfangtao/Desktop/newTest.mp4
```

结果可能要让大家失望了，我设置的1MB的大小，结果输出的文件将近8MB(一一` )我去。。。

## General

<i>Complete name :</i>	/Users/zhangfangtao/Desktop/newTest.mp4
<i>Format :</i>	MPEG-4
<i>Format profile :</i>	Base Media
<i>Codec ID :</i>	isom (isom/iso2/avc1/mp41)
<i>File size :</i>	7.79 MiB
<i>Duration :</i>	2 min 38 s
<i>Overall bit rate :</i>	411 kb/s
<i>Writing application :</i>	Lavf58.13.100

我把大小的控制设置成1024K，就会精确很多，新的命令行：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -fs 1024K  
/Users/zhangfangtao/Desktop/newTest.mp4
```

再看一下结果：

## General

<i>Complete name :</i>	/Users/zhangfangtao/Desktop/newTest.mp4
<i>Format :</i>	MPEG-4
<i>Format profile :</i>	Base Media
<i>Codec ID :</i>	isom (isom/iso2/avc1/mp41)
<i>File size :</i>	1.24 MiB
<i>Duration :</i>	18 s 934 ms
<i>Overall bit rate :</i>	551 kb/s
<i>Writing application :</i>	Lavf58.13.100

## 文件的大小计算

编码输出的最终文件大小是音频和视频流大小的总和。以字节为单位的视频流大小的方程是(由比特到字节的转换为8):

```
video_size = video_bitrate * time_in_seconds / 8
```

如果音频未压缩，其大小由公式计算:

```
audio_size = sampling_rate * bit_depth * channels * time_in_seconds / 8
```

要计算压缩音频流的文件大小，我们需要知道它的比特率和方程。

```
audio_size = bitrate * time_in_seconds / 8.
```

例如，用1500 kbytes/s的视频比特率和128 kbytes/s音频比特率计算10分钟视频剪辑的最终大小，我们可以使用这些公式:

```
file_size = video_size + audio_size  
file_size = (video_bitrate + audio_bitrate) * time_in_seconds / 8  
file_size = (1500 kbit/s + 128 kbit/s) * 600 s  
file_size = 1628 kbit/s * 600 s  
file_size = 976800 kb = 976800000 b / 8 = 122100000 B / 1024 = 119238.28125 KB  
file_size = 119238.28125 KB / 1024 = 116.443634033203125 MB ≈ 116.44 MB
```

- 1 byte (B) = 8 bits (b)
- 1 kilobyte (kB or KB) = 1024 B
- 1 megabyte (MB) = 1024 kB, 等。

最终文件的大小比计算的要大一些，因为包含了一个muxing开销和文件元数据。

都是计算题，我就不发表什么个人的测试数据了。。。

## 04-调整和伸缩视频

在FFmpeg中调整视频的大小意味着可以通过一个选项改变其宽度和高度，而缩放则意味着使用一个具有高级功能的scale filter来改变帧的大小。

### 调整视频

输出视频的宽度和高度可以在输出文件名之前设置-s选项。视频分辨率以wxh格式输入，其中w为像素宽度，h为像素高度。例如，要将初始分辨率的输入调整为320x240，我们可以使用以下命令:

```
ffmpeg -i input_file -s 320x240 output_file
```

给大家看看我的测试命令行:

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -s 320x480  
/Users/zhangfangtao/Desktop/newtest.mp4
```

原来的视频信息如下图：

# Video

<i>ID :</i>	1
<i>Format :</i>	AVC
<i>Format/Info :</i>	Advanced Video Codec
<i>Format profile :</i>	High@L3.1
<i>Format settings :</i>	CABAC / 4 Ref Frames
<i>Format settings, CABAC :</i>	Yes
<i>Format settings, RefFrames :</i>	4 frames
<i>Codec ID :</i>	avc1
<i>Codec ID/Info :</i>	Advanced Video Coding
<i>Duration :</i>	18 s 934 ms
<i>Bit rate :</i>	418 kb/s
<i>Width :</i>	1 024 pixels
<i>Height :</i>	768 pixels
<i>Display aspect ratio :</i>	4:3
<i>Frame rate mode :</i>	Constant
<i>Frame rate :</i>	15.000 FPS
<i>Color space :</i>	YUV
<i>Chroma subsampling :</i>	4:2:0
<i>Bit depth :</i>	8 bits
<i>Scan type :</i>	Progressive
<i>Bits/(Pixel*Frame) :</i>	0.035
<i>Stream size :</i>	966 KiB (76%)
<i>Writing library :</i>	x264 core 148 r2795 aaa9aa8 cabac=1 / ref=3 / deblock=1:0:0 / analyse=0x3:0x113 / me=hex / subme=7 / psy=1 / psy_rd=1.00:0.00 / mixed_ref=1 / me_range=16 / chroma_me=1 / trellis=1 / 8x8dct=1 / cqm=0 / deadzone=21,11 / fast_pskip=1 / chroma_qp_offset=-2 / threads=6 / lookahead_threads=1 / sliced_threads=0 / nr=0 / decimate=1 / interlaced=0 / bluray_compat=0 / constrained_intra=0 / bframes=3 / b_pyramid=2 / b_adapt=1 / b_bias=0 / direct=1 / weightb=1 / open_gop=0 / weightp=2 / keyint=250 / keyint_min=15 / scenecut=40 / intra_refresh=0 / rc_lookahead=40 / rc=crf / mbtree=1 / crf=23.0 / qcomp=0.60 / qpmin=0 / qpmax=69 / qpstep=4 / ip_ratio=1.40 / aq=1:1.00
<i>Encoding settings :</i>	
<i>Language :</i>	English

重新编码之后如下图：

# Video

<i>ID :</i>	1
<i>Format :</i>	AVC
<i>Format/Info :</i>	Advanced Video Codec
<i>Format profile :</i>	High@L2.1
<i>Format settings :</i>	CABAC / 4 Ref Frames
<i>Format settings, CABAC :</i>	Yes
<i>Format settings, RefFrames :</i>	4 frames
<i>Codec ID :</i>	avc1
<i>Codec ID/Info :</i>	Advanced Video Coding
<i>Duration :</i>	12 min 48 s
<i>Bit rate :</i>	38.3 kb/s
<i>Width :</i>	320 pixels
<i>Height :</i>	480 pixels
<i>Display aspect ratio :</i>	4:3
<i>Frame rate mode :</i>	Constant
<i>Frame rate :</i>	15.000 FPS
<i>Color space :</i>	YUV
<i>Chroma subsampling :</i>	4:2:0
<i>Bit depth :</i>	8 bits
<i>Scan type :</i>	Progressive
<i>Bits/(Pixel*Frame) :</i>	0.017
<i>Stream size :</i>	3.51 MiB (22%)
<i>Writing library :</i>	x264 core 148 r2795 aaa9aa8 cabac=1 / ref=3 / deblock=1:0:0 / analyse=0x3:0x113 / me=hex / subme=7 / psy=1 / psy_rd=1.00:0.00 / mixed_ref=1 / me_range=16 / chroma_me=1 / trellis=1 / 8x8dct=1 / cqm=0 / deadzone=21,11 / fast_pskip=1 / chroma_qp_offset=-2 / threads=6 / lookahead_threads=1 / sliced_threads=0 / nr=0 / decimate=1 / interlaced=0 / bluray_compat=0 / constrained_intra=0 / bframes=3 / b_pyramid=2 / b_adapt=1 / b_bias=0 / direct=1 / weightb=1 / open_gop=0 / weightp=2 / keyint=250 / keyint_min=15 / scenecut=40 / intra_refresh=0 / rc_lookahead=40 / rc=crf / mbtree=1 / crf=23.0 / qcomp=0.60 / qpmin=0 / qpmax=69 / qpstep=4 / ip_ratio=1.40 / aq=1:1.00
<i>Encoding settings :</i>	
<i>Language :</i>	English

确实发生了改变。

## 预定义的视频帧大小

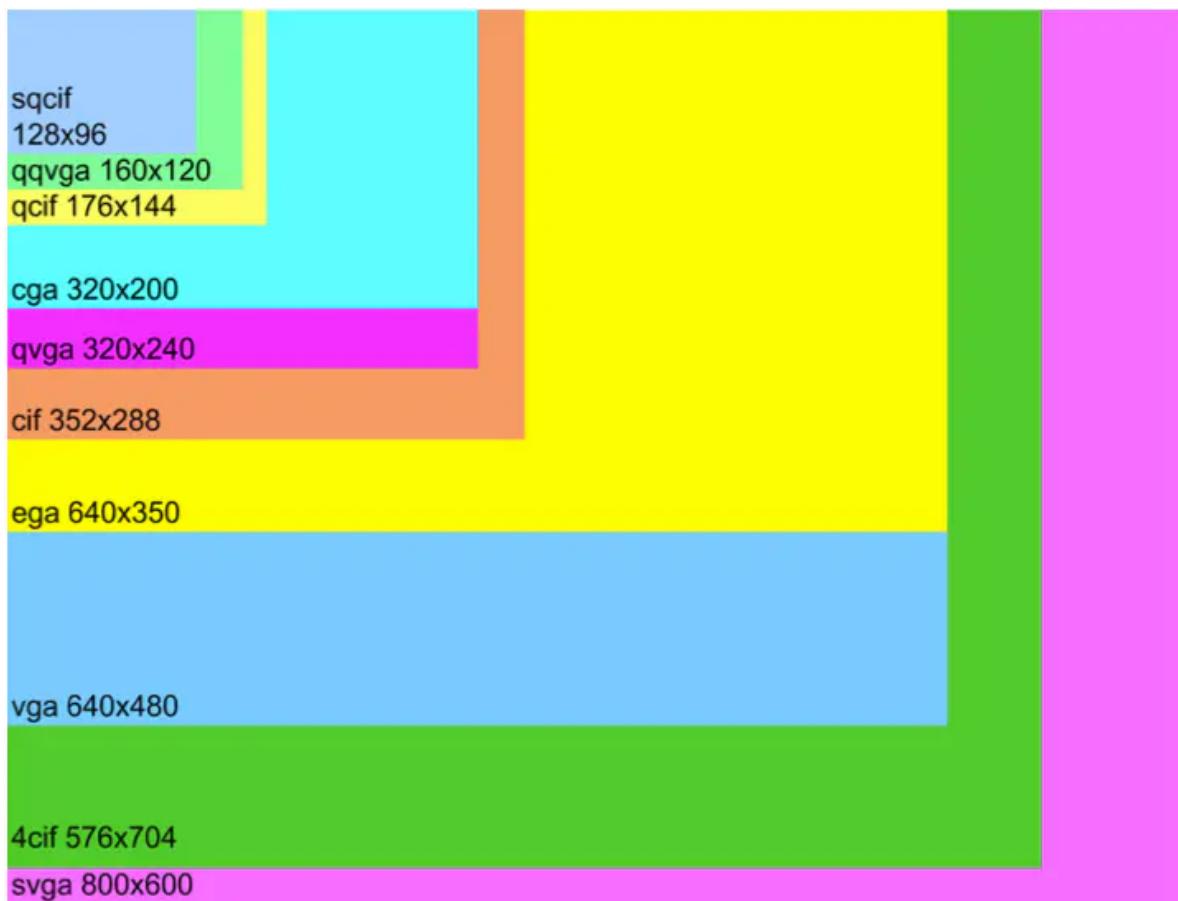
FFmpeg工具没有输入视频宽度和高度的精确数字，而是提供了在下一页的表中列出的预定义视频大小。下面两个命令的结果相同：

```
ffmpeg -i input.avi -s 640x480 output.avi  
ffmpeg -i input.avi -s vga output.avi
```

输入vga参数之后显示的如下图：（确实是640: 480）

# Video

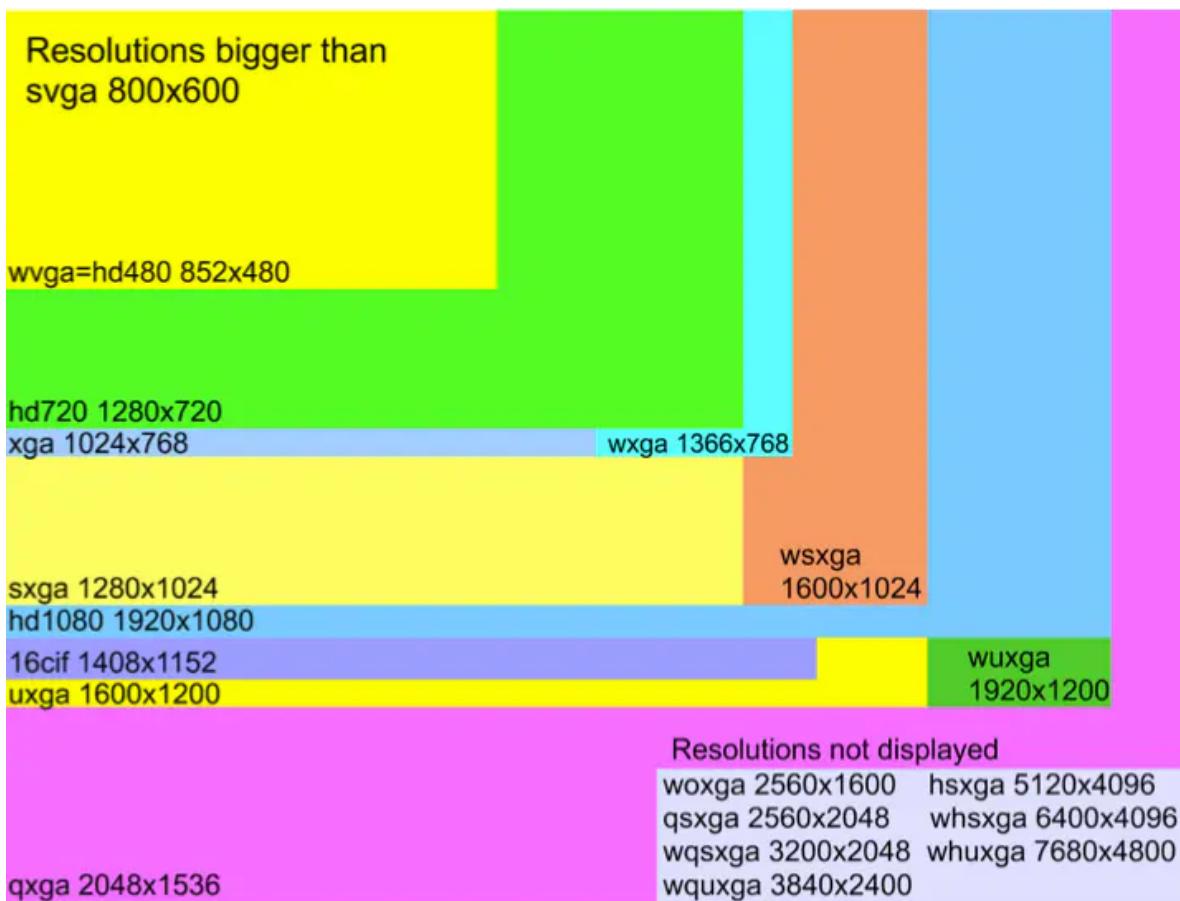
<i>ID :</i>	1
<i>Format :</i>	AVC
<i>Format/Info :</i>	Advanced Video Codec
<i>Format profile :</i>	High@L2.2
<i>Format settings :</i>	CABAC / 4 Ref Frames
<i>Format settings, CABAC :</i>	Yes
<i>Format settings, RefFrames :</i>	4 frames
<i>Codec ID :</i>	avc1
<i>Codec ID/Info :</i>	Advanced Video Coding
<i>Duration :</i>	12 min 48 s
<i>Bit rate :</i>	87.1 kb/s
<i>Width :</i>	640 pixels
<i>Height :</i>	480 pixels
<i>Display aspect ratio :</i>	4:3
<i>Frame rate mode :</i>	Constant
<i>Frame rate :</i>	15.000 FPS
<i>Color space :</i>	YUV
<i>Chroma subsampling :</i>	4:2:0
<i>Bit depth :</i>	8 bits
<i>Scan type :</i>	Progressive
<i>Bits/(Pixel*Frame) :</i>	0.019
<i>Stream size :</i>	7.98 MiB (40%)
<i>Writing library :</i>	x264 core 148 r2795 aaa9aa8 cabac=1 / ref=3 / deblock=1:0:0 / analyse=0x3:0x113 / me=hex / subme=7 / psy=1 / psy_rd=1.00:0.00 / mixed_ref=1 / me_range=16 / chroma_me=1 / trellis=1 / 8x8dct=1 / cqm=0 / deadzone=21,11 / fast_pskip=1 / chroma_qp_offset=-2 / threads=6 / lookahead_threads=1 / sliced_threads=0 / nr=0 / decimate=1 / interlaced=0 / bluray_compat=0 / constrained_intra=0 / bframes=3 / b_pyramid=2 / b_adapt=1 / b_bias=0 / direct=1 / weightb=1 / open_gop=0 / weightp=2 / keyint=250 / keyint_min=15 / scenecut=40 / intra_refresh=0 / rc_lookahead=40 / rc=crf / mbtree=1 / crf=23.0 / qcomp=0.60 / qpmin=0 / qpmax=69 / qpstep=4 / ip_ratio=1.40 / aq=1:1.00
<i>Encoding settings :</i>	
<i>Language :</i>	English



Abbreviations for video sizes in FFmpeg

FFmpeg中视频大小的缩写

大小	缩写	典型用法
128x96	sqcif	手机
160x120	qqvga	手机
176x144	qcif	手机
320x200	cga	旧的CRT显示器
320x240	qvga	手机、摄像头
352x288	cif	手机
640x350	ega	旧的CRT显示器
640x480	vga	显示器,摄像头
704x576	4cif	官方数字视频大小的电视。
800x600	svga	显示器
852x480	hd480, wvga	摄像机
1024x768	xga	显示器,摄像头
1280x720	hd720	高清电视,摄像机
1280 x1024	sxga	显示器
1366x768	wxga	显示器
1408x1152	16cif	设备使用CIF
1600x1024	wsxga	显示器
1600x1200	uxga	显示器, 摄像机
1920x1080	hd1080	高清电视,摄像机
1920x1200	wuxga	宽屏显示器
2048x1536	qxga	显示器
2560x1600	woxga	显示器
2560x2048	qsxga	显示器
3200x2048	wqsxga	显示器
3840x2400	wquxga	显示器
5120x4096	hsxga	显示,显微镜相机
6400x4096	whsxga	显示器
7680x4800	whuxga	显示器



## 调整大小时的注意事项-奈奎斯特采样定理。

视频通常被调整为比来源更小的分辨率，这被称为下采样，主要用于便携式设备，通过互联网流媒体等。重要的是要考虑，在较小的尺寸中，一些细节将会丢失，这一事实解释了奈奎斯特 -Shannon采样定理。它的一般形式与任何信号有关，并告知为了完全重构采样信号，我们必须使用比信源频率高至少2倍的频率。这意味着要将小细节保留在缩小的视频中，它们的原始尺寸必须高于缩放比例除以2。

例如，800x600（SVGA）分辨率的视频包含2像素宽的细节。当缩放到640x480（VGA）分辨率时，缩放比率为0.8，并且2像素再缩放为2像素：

$$640 \text{ pixels} / 800 \text{ pixels} = 0.8$$

$$2 \text{ pixels} * 0.8 = 1.6 \approx 2 \text{ pixels}$$

但是当这个视频被缩放到160x120 (QQVGA)分辨率时，细节就丢失了：

$$160 \text{ pixels} / 800 \text{ pixels} = 0.2$$

$$2 \text{ pixels} * 0.2 = 0.4 \approx 0 \text{ pixels}$$

这意味着在向下采样后，可见的只有输入大小至少3个像素的细节。（采用的是四舍五入的向下取整的方式）

## 专业的扩大滤波器

将视频调整为更大的帧大小比较少见，因为该功能几乎可以提供所有媒体播放器，但由此产生的图像有时并不清晰，特别是当源解析度非常小时。用于平滑放大的源的特殊滤波器是super2xsai滤波器：

## Video filter: super2xsai

视频过滤器:super2xsai

视频过滤器:super2xsai

描述	在不减少锐度的情况下，使用像素艺术缩放算法放大源帧的大小2倍。“2xSal”的意思是“2倍尺度和插值”。
Syntax	-vf super2xsai

例如，将128x96视频从移动电话放大到分辨率256x192像素，可以使用下一个命令：

```
ffmpeg -i phone_video.3gp -vf super2xsai output.mp4
```

给大家看一下我的测试指令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf super2xsai  
/Users/zhangfangtao/Desktop/newTest.mp4
```

原始的视频信息如下图：

# Video

<i>ID</i> :	1
<i>Format</i> :	AVC
<i>Format/Info</i> :	Advanced Video Codec
<i>Format profile</i> :	High@L4
<i>Format settings</i> :	CABAC / 4 Ref Frames
<i>Format settings, CABAC</i> :	Yes
<i>Format settings, RefFrames</i> :	4 frames
<i>Format settings, GOP</i> :	M=3, N=75
<i>Codec ID</i> :	avc1
<i>Codec ID/Info</i> :	Advanced Video Coding
<i>Duration</i> :	12 min 48 s
<i>Source duration</i> :	12 min 48 s
<i>Bit rate</i> :	198 kb/s
<i>Width</i> :	1 024 pixels
<i>Height</i> :	768 pixels
<i>Display aspect ratio</i> :	4:3
<i>Frame rate mode</i> :	Constant
<i>Frame rate</i> :	15.000 FPS
<i>Color space</i> :	YUV
<i>Chroma subsampling</i> :	4:2:0
<i>Bit depth</i> :	8 bits
<i>Scan type</i> :	Progressive
<i>Bits/(Pixel*Frame)</i> :	0.017
<i>Stream size</i> :	18.1 MiB (73%)
<i>Source stream size</i> :	19.7 MiB (79%)
<i>Language</i> :	English
<i>Encoded date</i> :	UTC 2012-08-22 19:08:48
<i>Tagged date</i> :	UTC 2012-08-22 19:08:48
<i>Color range</i> :	Limited
<i>Color primaries</i> :	BT.709
<i>Transfer characteristics</i> :	BT.709
<i>Matrix coefficients</i> :	BT.709
<i>mdhd_Duration</i> :	768388

重新编码之后的视频信息：

# Video

<i>ID :</i>	1
<i>Format :</i>	AVC
<i>Format/Info :</i>	Advanced Video Codec
<i>Format profile :</i>	High 4:4:4 Predictive@L5
<i>Format settings :</i>	CABAC / 4 Ref Frames
<i>Format settings, CABAC :</i>	Yes
<i>Format settings, RefFrames :</i>	4 frames
<i>Codec ID :</i>	avc1
<i>Codec ID/Info :</i>	Advanced Video Coding
<i>Duration :</i>	12 min 48 s
<i>Bit rate :</i>	862 kb/s
<i>Width :</i>	2 048 pixels
<i>Height :</i>	1 536 pixels
<i>Display aspect ratio :</i>	4:3
<i>Frame rate mode :</i>	Constant
<i>Frame rate :</i>	15.000 FPS
<i>Chroma subsampling :</i>	4:4:4
<i>Bit depth :</i>	8 bits
<i>Scan type :</i>	Progressive
<i>Bits/(Pixel*Frame) :</i>	0.018
<i>Stream size :</i>	79.0 MiB (87%)
<i>Writing library :</i>	x264 core 148 r2795 aaa9aa8 cabac=1 / ref=3 / deblock=1:0:0 / analyse=0x3:0x113 / me=hex / subme=7 / psy=1 / psy_rd=1.00:0.00 / mixed_ref=1 / me_range=16 / chroma_me=1 / trellis=1 / 8x8dct=1 / cqm=0 / deadzone=21,11 / fast_pskip=1 / chroma_qp_offset=4 / threads=6 / lookahead_threads=1 / sliced_threads=0 / nr=0 / decimate=1 / interlaced=0 / bluray_compat=0 / constrained_intra=0 / bframes=3 / b_pyramid=2 / b_adapt=1 / b_bias=0 / direct=1 / weightb=1 / open_gop=0 / weightp=2 / keyint=250 / keyint_min=15 / scenecut=40 / intra_refresh=0 / rc_lookahead=40 / rc=crf / mbtree=1 / crf=23.0 / qcomp=0.60 / qpmin=0 / qpmax=69 / qpstep=4 / ip_ratio=1.40 / aq=1:1.00
<i>Encoding settings :</i>	
<i>Language :</i>	English

## 高级缩放技能

当使用-s选项更改视频帧大小时，会在相关滤镜图片的末尾插入缩放视频滤镜。要管理缩放过程开始的位置，可以直接使用缩放过滤器。

### Video filter: scale

视频过滤器:缩放

描述	通过更改输出样本宽高比来缩放源，显示宽高比保持不变。
语法	scale=width:height[:interl={1  -1}]
**	变量表示宽度和高度参数。
iw or in_w	输入的宽度
ih or in_h	输入的高度
ow or out_w	输出的宽度
oh or out_h	输出的高度
a	纵横比，与iw/ih相同。
sar	输入样本纵横比，与dar/a相同。
dar	输入显示纵横比，与*sar相同。
hsub	水平色度子样本值，为yuv422p像素格式为2。
vsub	垂直色度子样本值，为yuv422p像素格式为1。
**	可选的interl参数的可用值。
1	应用交错感知扩展。
-1	如果源被标记为交错的，应用是交错的意识扩展。

例如，下面两个命令的结果相同：

```
ffmpeg -i input.mpg -s 320x240 output.mp4
ffmpeg -i input.mpg -vf scale=320:240 output.mp4
```

scale filter的优点是，对于框架设置，可以使用上面表中描述的其他参数。

## 按比例缩放视频输入

如果不知道输入框的大小，可以使用scale filter的ih和iw参数相应地改变其分辨率，例如创建一个半大小的视频，我们可以使用下一个命令：

```
ffmpeg -i input.mpg -vf scale=iw/2:ih/2 output.mp4
```

我的测试指令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf scale=iw/2:ih/2
/Users/zhangfangtao/Desktop/newTest.mp4
```

转换之前的视频信息：

# Video

<i>ID</i> :	1
<i>Format</i> :	AVC
<i>Format/Info</i> :	Advanced Video Codec
<i>Format profile</i> :	High@L4
<i>Format settings</i> :	CABAC / 4 Ref Frames
<i>Format settings, CABAC</i> :	Yes
<i>Format settings, RefFrames</i> :	4 frames
<i>Format settings, GOP</i> :	M=3, N=75
<i>Codec ID</i> :	avc1
<i>Codec ID/Info</i> :	Advanced Video Coding
<i>Duration</i> :	12 min 48 s
<i>Source duration</i> :	12 min 48 s
<i>Bit rate</i> :	198 kb/s
<i>Width</i> :	1 024 pixels
<i>Height</i> :	768 pixels
<i>Display aspect ratio</i> :	4:3
<i>Frame rate mode</i> :	Constant
<i>Frame rate</i> :	15.000 FPS
<i>Color space</i> :	YUV
<i>Chroma subsampling</i> :	4:2:0
<i>Bit depth</i> :	8 bits
<i>Scan type</i> :	Progressive
<i>Bits/(Pixel*Frame)</i> :	0.017
<i>Stream size</i> :	18.1 MiB (73%)
<i>Source stream size</i> :	19.7 MiB (79%)
<i>Language</i> :	English
<i>Encoded date</i> :	UTC 2012-08-22 19:08:48
<i>Tagged date</i> :	UTC 2012-08-22 19:08:48
<i>Color range</i> :	Limited
<i>Color primaries</i> :	BT.709
<i>Transfer characteristics</i> :	BT.709
<i>Matrix coefficients</i> :	BT.709
<i>mdhd_Duration</i> :	768388

转换之后的视频信息：

# Video

<i>ID</i> :	1
<i>Format</i> :	AVC
<i>Format/Info</i> :	Advanced Video Codec
<i>Format profile</i> :	High@L2.1
<i>Format settings</i> :	CABAC / 4 Ref Frames
<i>Format settings, CABAC</i> :	Yes
<i>Format settings, RefFrames</i> :	4 frames
<i>Codec ID</i> :	avc1
<i>Codec ID/Info</i> :	Advanced Video Coding
<i>Duration</i> :	12 min 48 s
<i>Bit rate</i> :	51.9 kb/s
<i>Width</i> :	512 pixels
<i>Height</i> :	384 pixels
<i>Display aspect ratio</i> :	4:3
<i>Frame rate mode</i> :	Constant
<i>Frame rate</i> :	15.000 FPS
<i>Color space</i> :	YUV
<i>Chroma subsampling</i> :	4:2:0
<i>Bit depth</i> :	8 bits
<i>Scan type</i> :	Progressive
<i>Bits/(Pixel*Frame)</i> :	0.018
<i>Stream size</i> :	4.75 MiB (28%)
<i>Writing library</i> :	x264 core 148 r2795 aaa9aa8

90%大小的视频命令：

```
ffmpeg -i input.mpg -vf scale=iw*0.9:ih*0.9 output.mp4
```

这个就不需要测试了吧。

用黄金比例缩放输入值= 1.61803398874989484820...用PHI来表示这个数值：

```
ffmpeg -i input.mpg -vf scale=iw/PHI:ih/PHI output.mp4
```

## 扩展到预定义的宽度或高度。

当输出视频应该有一定的宽度或一定的高度时，输入视频的大小和纵横比都是未知的，第二个维度可以通过一个方面参数来指定，比如下面的例子。要将输出宽度设置为400像素，高度按比例设置，我们可以使用以下命令：

```
ffmpeg -i input.avi -vf scale=400:400/a
```

我的测试指令：宽度确实成为了400.高度是等比缩放的，300

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf scale=400:400/a  
/Users/zhangfangtao/Desktop/newTest.mp4
```

输出文件的视频信息：

# Video

<i>ID :</i>	1
<i>Format :</i>	AVC
<i>Format/Info :</i>	Advanced Video Codec
<i>Format profile :</i>	High@L2.1
<i>Format settings :</i>	CABAC / 4 Ref Frames
<i>Format settings, CABAC :</i>	Yes
<i>Format settings, RefFrames :</i>	4 frames
<i>Codec ID :</i>	avc1
<i>Codec ID/Info :</i>	Advanced Video Coding
<i>Duration :</i>	12 min 48 s
<i>Bit rate :</i>	31.4 kb/s
<i>Width :</i>	400 pixels
<i>Height :</i>	300 pixels
<i>Display aspect ratio :</i>	4:3
<i>Frame rate mode :</i>	Constant
<i>Frame rate :</i>	15.000 FPS
<i>Color space :</i>	YUV
<i>Chroma subsampling :</i>	4:2:0
<i>Bit depth :</i>	8 bits
<i>Scan type :</i>	Progressive
<i>Bits/(Pixel*Frame) :</i>	0.017
<i>Stream size :</i>	2.88 MiB (19%)
<i>Writing library :</i>	x264 core 148 r2795 aaa9aa8

若要将输出高度更改为300像素，宽度按比例改变，命令可以是:(注意到了么，如果需要指定高度，需要用\*，不是/，我试了一下，貌似用除号不行，因为那个参数a意思是纵横比，下面我有说明)

```
ffmpeg -i input.avi -vf scale=300*a:300
```

看看我的测试指令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf scale=300*a:300  
/Users/zhangfangtao/Desktop/newTest.mp4
```

输出视频信息：

# Video

<i>ID :</i>	1
<i>Format :</i>	AVC
<i>Format/Info :</i>	Advanced Video Codec
<i>Format profile :</i>	High@L2.1
<i>Format settings :</i>	CABAC / 4 Ref Frames
<i>Format settings, CABAC :</i>	Yes
<i>Format settings, RefFrames :</i>	4 frames
<i>Codec ID :</i>	avc1
<i>Codec ID/Info :</i>	Advanced Video Coding
<i>Duration :</i>	12 min 48 s
<i>Bit rate :</i>	31.4 kb/s
<i>Width :</i>	400 pixels
<i>Height :</i>	300 pixels
<i>Display aspect ratio :</i>	4:3
<i>Frame rate mode :</i>	Constant
<i>Frame rate :</i>	15.000 FPS
<i>Color space :</i>	YUV
<i>Chroma subsampling :</i>	4:2:0
<i>Bit depth :</i>	8 bits
<i>Scan type :</i>	Progressive
<i>Bits/(Pixel*Frame) :</i>	0.017
<i>Stream size :</i>	2.88 MiB (19%)

如果指令里面这样写：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf scale=300/a:300  
/Users/zhangfangtao/Desktop/newTest.mp4
```

会报下面的错误：

```
[libx264 @ 0x7ff29280cc00] width not divisible by 2 (225x300)  
Error initializing output stream 0:0 -- Error while opening encoder for output s  
tream #0:0 - maybe incorrect parameters such as bit_rate, rate, width or height  
Conversion failed!
```

# 05-裁剪视频

裁剪视频意味着从输入到输出中选择想要的矩形区域而没有余数。裁剪通常用于调整大小，填充和其他编辑。

## 裁剪基础知识

较老的FFmpeg版本有`cropbottom`、`cropleft`、`cropright`和`croptop`选项，但现在已弃用，并使用下表中描述的裁剪操作。

### Video filter: crop

视频过滤器:裁剪

描述	将输入视频帧的宽度和高度从x和y值表示的位置裁剪到指定的宽度和高度;x和y是输出的左上角坐标，协调系统的中心是输入视频帧的左上角。如果使用了可选的 <b>keep_aspect</b> 参数，将会改变输出SAR(样本宽比)以补偿新的DAR(显示长宽比)
语法	<code>crop=ow[:oh[:x[:y[:keep_aspect]]]]</code>
*****	用于ow和oh参数的表达式中的可用变量
x, y	对x的计算值(从左上角水平方向的像素个数)和y(垂直像素的数量)，对每个帧进行评估，x的默认值为(iw - ow)/2, y的默认值为(ih - oh)/2
in_w, iw	输入的宽度
in_h, ih	输入的高度
out_w, ow	输出(裁剪)宽度，默认值= iw
out_h, oh	输出(裁剪)高度，默认值= ih
a	纵横比，与iw/ih相同
sar	输入样本比例
dar	输入显示宽比，等于表达式a*sar
hsub, vsub	水平和垂直的色度子样本值，对于像素格式yuv422p, hsub的值为2,vsub为1
n	输入框的数目，从0开始
pos	位置在输入框的文件中，如果不知道NAN
t	时间戳以秒表示，如果输入时间戳未知

`ow`的值可以从`oh`得到，反之亦然，但不能从`x`和`y`中得到，因为这些值是在`ow`和`oh`之后进行的。`x`的值可以从`y`的值中得到，反之亦然。例如，在输入框的左三、中三和右三，我们可以使用命令：

```
ffmpeg -i input -vf crop=iw/3:ih:0:0 output  
ffmpeg -i input -vf crop=iw/3:ih:iw/3:0 output  
ffmpeg -i input -vf crop=iw/3:ih:iw/3*2:0 output
```

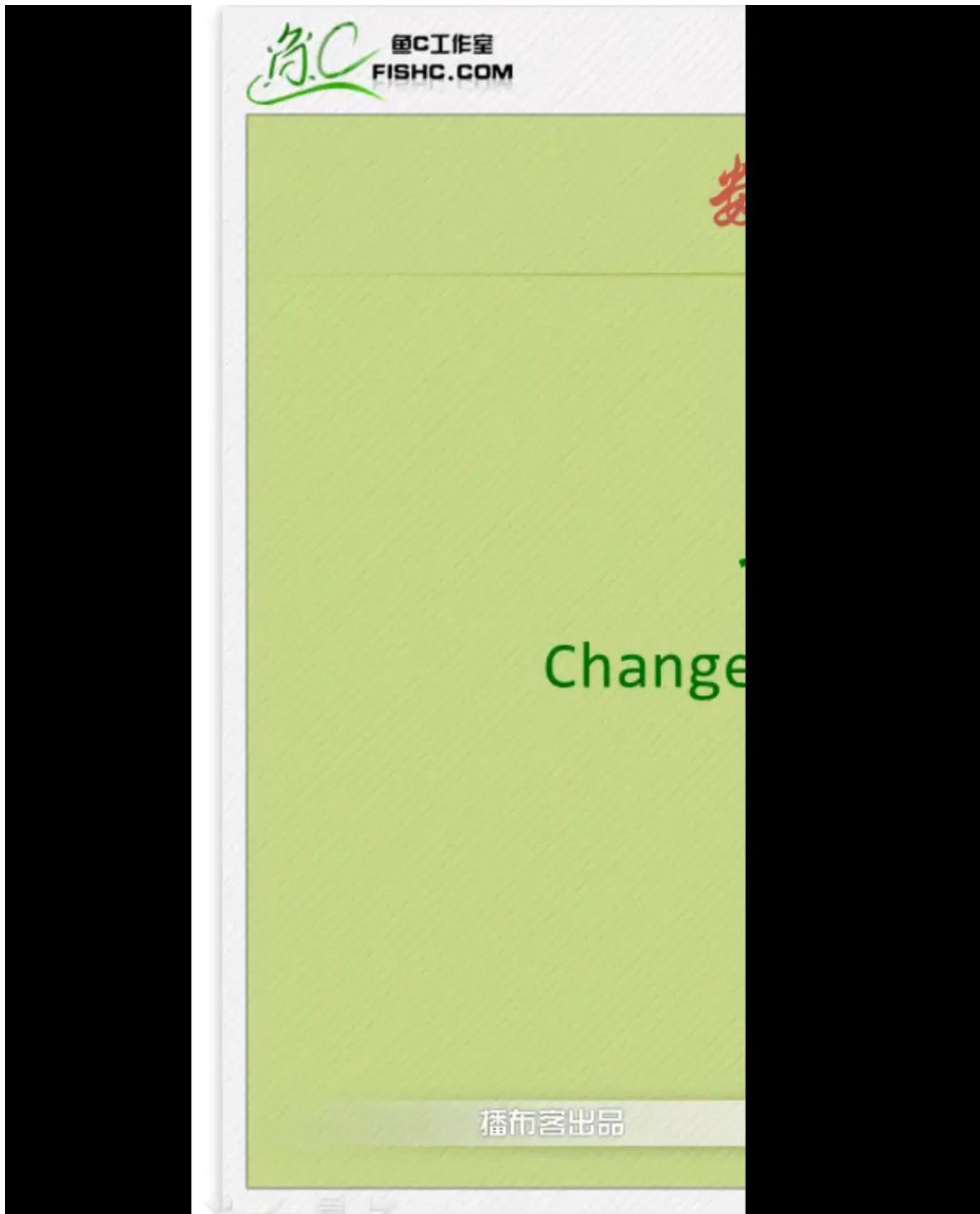
原视频这样：（就当给人家打广告了）



经过如下的测试指令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf crop=iw/3:ih:0:0  
/Users/zhangfangtao/Desktop/newTest.mp4
```

让我重新编码之后长这样：（惊不惊喜？意不意外）



再用下面的测试指令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf crop=iw/3:ih:iw/3:0  
/Users/zhangfangtao/Desktop/newTest.mp4
```

生成的视频如下图：



这次显示的中间的三分之一*w/3:ih*

使用如下指令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf crop=iw/3:ih:iw/3*2:0  
/Users/zhangfangtao/Desktop/newTest.mp4
```

生成的视频如下图：（最右边的三分之一）

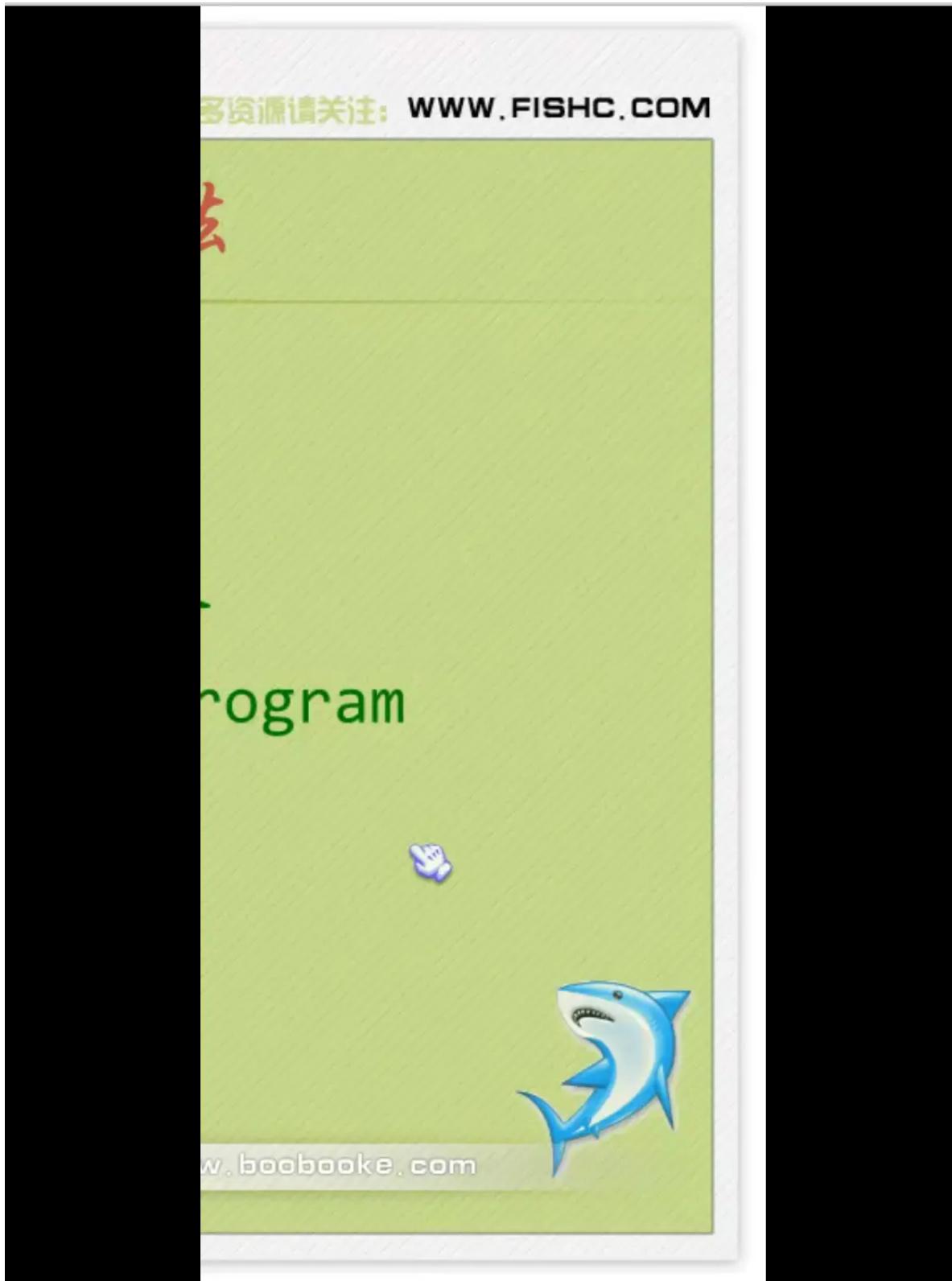


image.png

下图是相关参数的示意图：



# Crop filter

`crop=ow[:oh[:x[:y[:keep_aspect]]]]`

`iw = input width, ih = input height`

`ow = output width, oh = output height`

## 裁剪框中心

当我们想要裁剪框架中心的区域时，作物过滤器的设计可以跳过x和y参数的输入。x和y的默认值是。

$$\begin{aligned}x_{\text{default}} &= (\text{input width} - \text{output width})/2 \\y_{\text{default}} &= (\text{input height} - \text{output height})/2\end{aligned}$$

这意味着默认值被设置为自动裁剪输入中心的区域。该命令语法对w宽度和h高度的矩形中心区域进行裁剪。

```
ffmpeg -i input_file -vf crop=w:h output_file
```

我做了一个100\*100的裁剪，指令如下：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf crop=100:100  
/Users/zhangfangtao/Desktop/newTest.mp4
```

输出的视频长这样：



为了裁剪中间的半帧，我们可以使用以下命令：

```
ffmpeg -i input.avi -vf crop=iw/2:ih/2 output.avi
```

我的测试指令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf crop=iw/2:ih/2  
/Users/zhangfangtao/Desktop/newTest.mp4
```

裁剪之后的视频：



## 自动检测裁剪区域

为了自动检测出裁剪的非黑色区域，我们可以使用crop检测过滤器，如下表所示。当输入视频中包含一些黑条时，这种自动裁剪是有用的，通常是在从第4:3到16:9的转换之后，反之亦然。

### Video filter: cropdetect

描述	检测作物过滤器的作物大小，结果是由参数确定的输入帧的非黑色区域
语法	cropdetect[=limit[:round[:reset]]] all parameters are optional
****	参数的描述
limit	阈值，从0(无)到255 (all)，默认值= 24
round	-即使是整数，宽度和高度也必须是可分割的 - 4:2 2视频需要一个2的值，它只给出了维度 -偏移量自动更改为中心帧-默认值为16，它是许多编解码器的最佳值
reset	计数器决定了多少帧crop探测将重置之前检测到的最大视频区域并重新开始检测当前最优的作物区域。默认值为0。当通道标识扭曲了视频区域时，这是很有用的。0表示永远不会重置和返回在回放期间遇到的最大区域

limit参数指定了选择了多少深颜色的输出，零值意味着只有完整的黑色被裁剪。例如，要裁剪非黑输出，我们可以使用以下命令：（我这边也没有存在黑色边框的视频，测试了几个，没发现有啥用）

```
ffmpeg -i input.mpg -vf cropdetect=limit=0 output.mp4
```

## 时间的裁剪

媒体播放器通常有一个进度条，显示经过的秒数，但大多数只有在鼠标指针停止并在特定持续时间后隐藏时才会显示。 FFmpeg包含一个包含定时器的tests src视频源，我们可以使用以下命令显示它：

这里面大家可能会遇到一个问题，你编译出来的ffmpeg里面可能没有ffplay，编译FFmpeg之后没有ffplay？这里大家可以去官网下载编译好的<http://ffmpeg.org/download.html>

```
ffplay -f lavfi -i testsrc
```

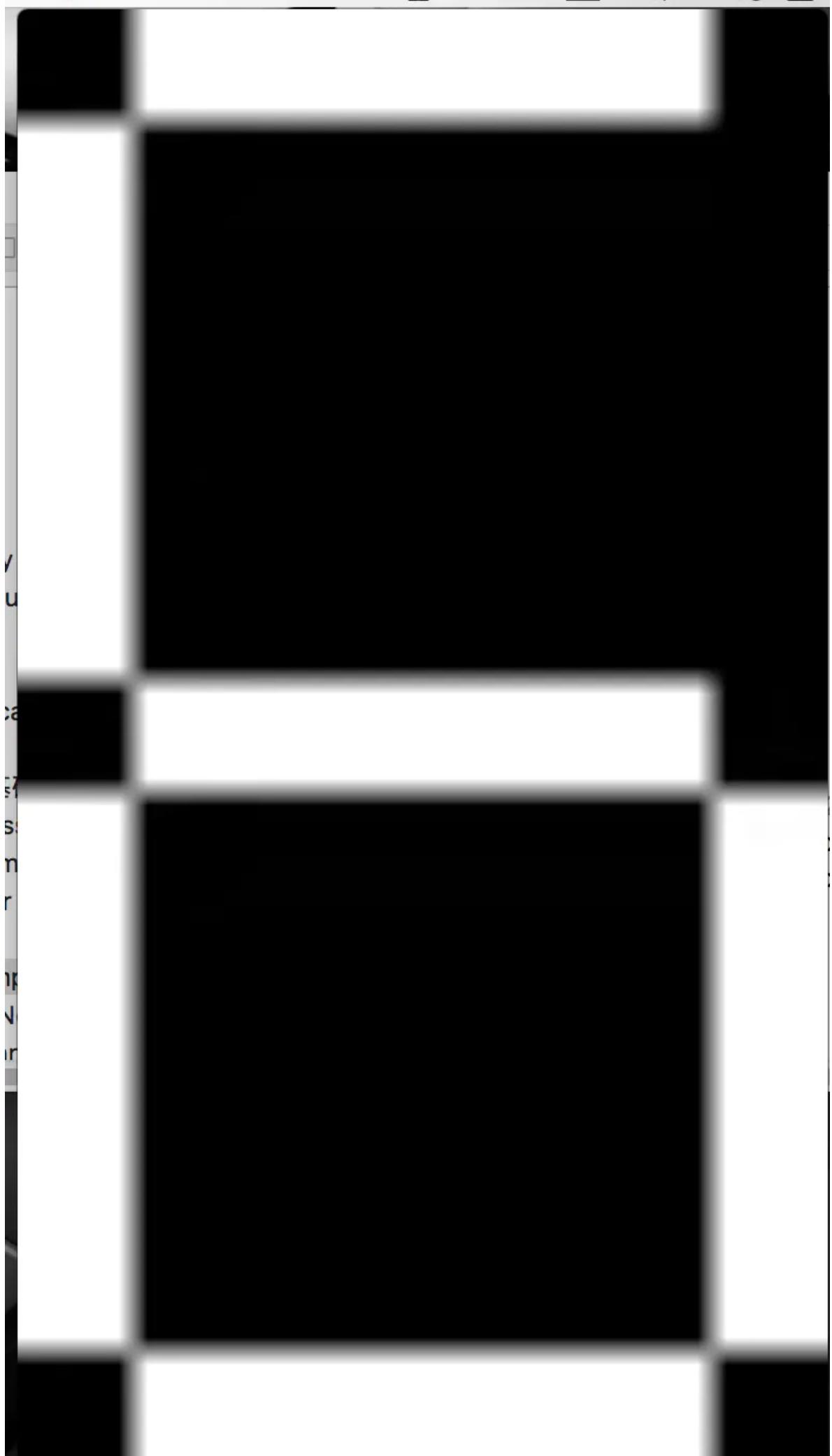
我在自己电脑上测试的，指令和上面的一样，显示的效果如下图所示：



tests src的默认大小是320x240像素，初始计时器的数字0有29x52像素大小，它在左上角的位置是256个像素，垂直于94像素。要裁剪一个数字的面积，我们可以使用以下命令：

```
ffmpeg -f lavfi -i testsrc -vf crop=29:52:256:94 -t 10 timer1.mpg
```

我在自己的电脑上测试了一下，显示的效果如下图(一个10秒钟，20K的小视频)：



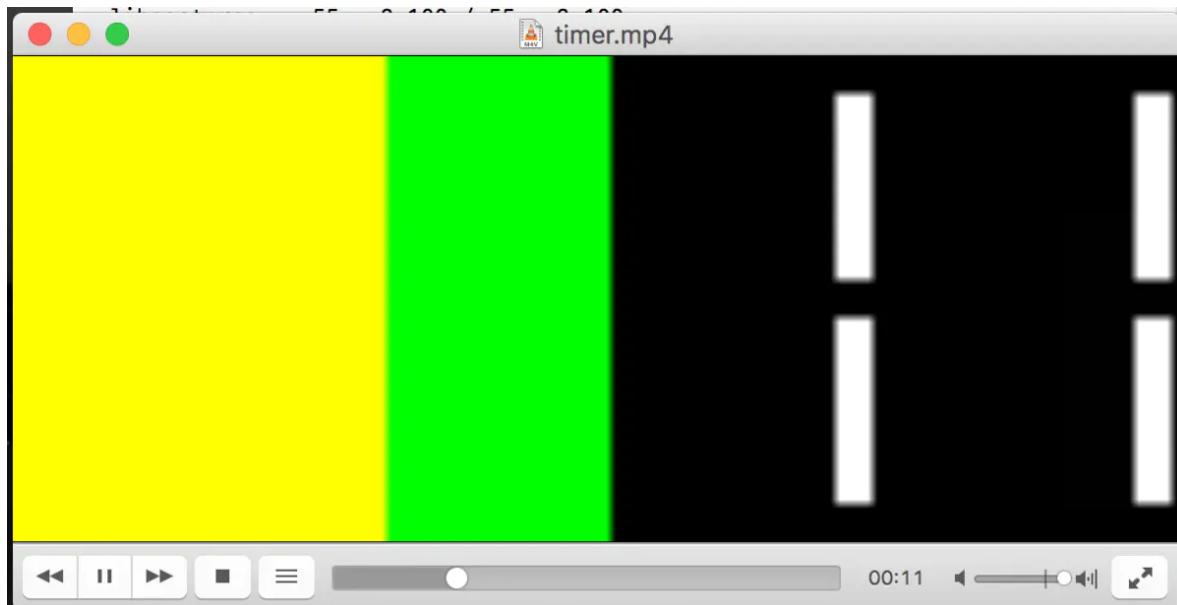
我们想用这个命令来创建一个有1、2、3和4位数字的定时器，每个数字的筛选器和时间期限的规格都在表中：

数的位数	数字过滤器规范	时长	图片
1	crop=29:52:256:94	9 seconds 0 min : 9 sec 00:00:09	
2	crop=61:52:224:94	99 seconds 1 min : 39 sec 00:01:39	
3	crop=93:52:192:94	999 seconds 16 min : 39 sec 00:16:39	
4	crop=125:52:160:94	9999 seconds 2 hours : 46 min : 39 sec 02:46:39	

给大家示范一下下面的指令：

```
ffmpeg -f lavfi -i testsrc -vf crop=125:52:160:94 -t 50 timer.mp4
```

显示的效果如下：



如果我们想要比52像素高的更大的数字，我们可以用一个size参数指定更大的testsrc输出(例如:i testsrc=size=vga)，然后相应地调整作物区域。



上面的那个完整的指令如下：（我自己试了一下，视频有了，没有文字。。。）

```
ffmpeg -f lavfi -i testsrc=size=vga -vf crop=125:52:160:94 -t 50 timer.mp4
```

为了改变数字和背景的颜色，我们可以使用一个lut过滤器，这是在[颜色修正](#)的章节中描述的。创建的计时器将用于视频覆盖的示例。

## 06-填充视频

填充视频意味着向视频帧添加额外的区域以包含额外的内容。当输入应在具有不同宽高比的显示器上播放时，通常需要填充视频。

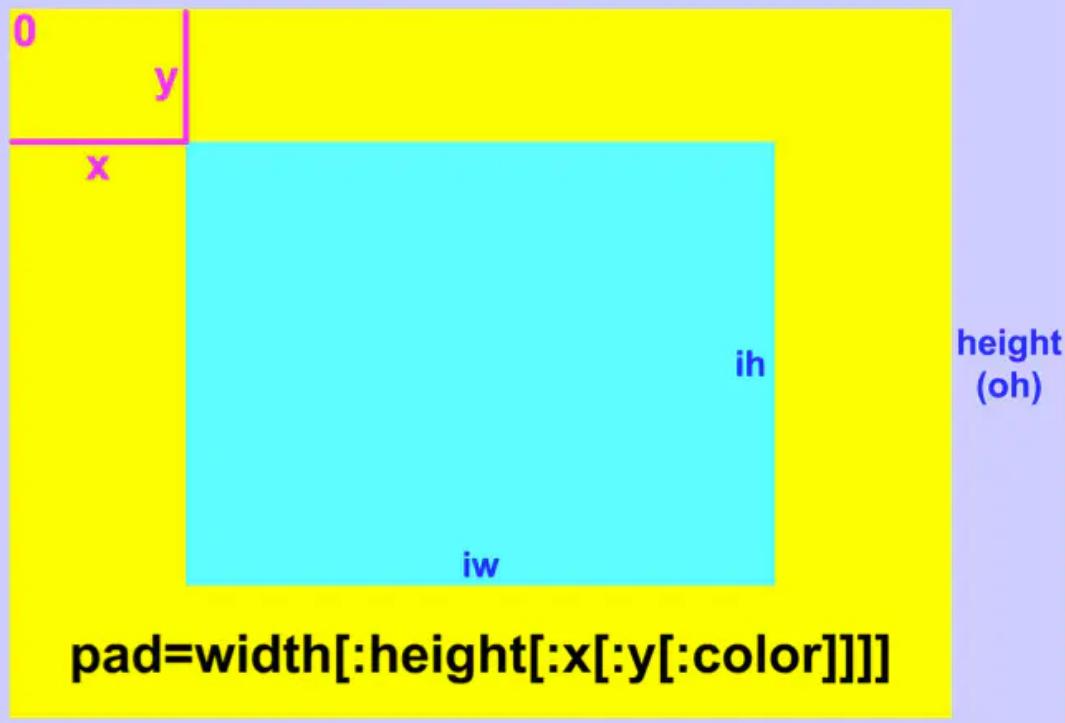
### 填充视频基础知识

对于视频填充，我们使用表格中描述的填充过滤器。

Video filter: pad

<b>描述</b>	在输入视频帧中添加彩色填充，该帧位于协调系统中的[x, y]点，其中输出帧的左上角是[0,0]。输出的大小由宽度和高度参数设置。
<b>语法</b>	pad=width[:height[:x[:y[:color]]]] 中括号里面的参数都是可选的
<b>***</b>	参数的描述
<b>color</b>	十六进制形式的RGB颜色值: 0xRRGGBB [@AA], 其中AA的范围是 (0,1) 中的十进制值或任何有效的颜色名称, 如白色, 蓝色, 黄色等, 默认值为黑色, 请参见颜色有关详细信息, 请参阅 <a href="#">FFmpeg基本介绍</a> 章节中的名称部分
<b>width, height</b>	带填充的输出帧的宽度和高度, 宽度的值可以从高度导出, 反之亦然, 两个参数的默认值都是0
<b>x, y</b>	输入左上角的坐标 (偏移量) 与输出帧的左上角有关, 两个参数的默认值均为0
<b>***</b>	参数的高度, 宽度, x, y的表达式的可用变量
<b>a</b>	纵横比, 与iw/ih相同
<b>dar</b>	输入显示宽比, 与*sar相同
<b>hsub, vsub</b>	水平和垂直的色度子样本值, 对于像素格式yuv422p, hsub的值为2,vsub为1
<b>in_h, ih</b>	输入的高度
<b>in_w, iw</b>	输入的宽度
<b>n</b>	输入框的数目, 从0开始
<b>out_h, oh</b>	输出高度, 默认值=高度
<b>out_w, ow</b>	输出宽度, 默认值=宽度
<b>pos</b>	位置在输入框的文件中, 如果不知道NAN
<b>sar</b>	输入样本比例
<b>t</b>	时间戳以秒表示, 如果输入时间戳未知
<b>x, y</b>	x和y的偏移量由x和y表示, 或者NAN如果没有指定

# Pad filter



例如，为了在一个svga大小的照片周围创建一个30像素宽的粉红色框架，我们可以使用以下命令：

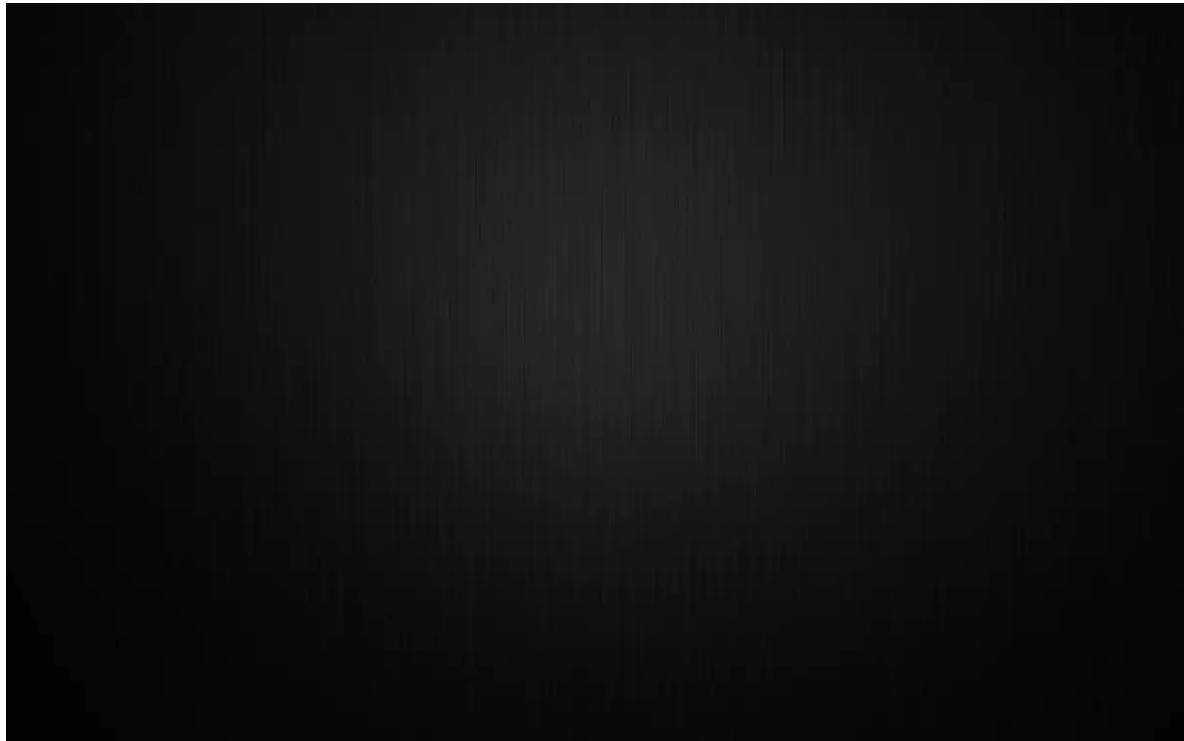
```
ffmpeg -i photo.jpg -vf pad=860:660:30:30:pink framed_photo.jpg
```



- 这里面给大家说一下，我自己测试的结果是，我首先用了一张1920\*1200大小的图片，一点问题都没有，命令如下：

```
ffmpeg -i /Users/zhangfangtao/Desktop/Work/002.jpg -vf  
pad=1980:1260:30:30:pink /Users/zhangfangtao/Desktop/Work/003.jpg
```

原来的图片如下图：



原来的图片

新的图片如下图：



新的图片

后来我又用了一下一个尺寸是250\*250的图片，做实验，结果发现乱码了：

指令如下：

```
ffmpeg -i /Users/zhangfangtao/Desktop/Work/001.jpg -vf pad=310:310:30:30:pink  
/Users/zhangfangtao/Desktop/Work/003.jpg
```

原图如下：



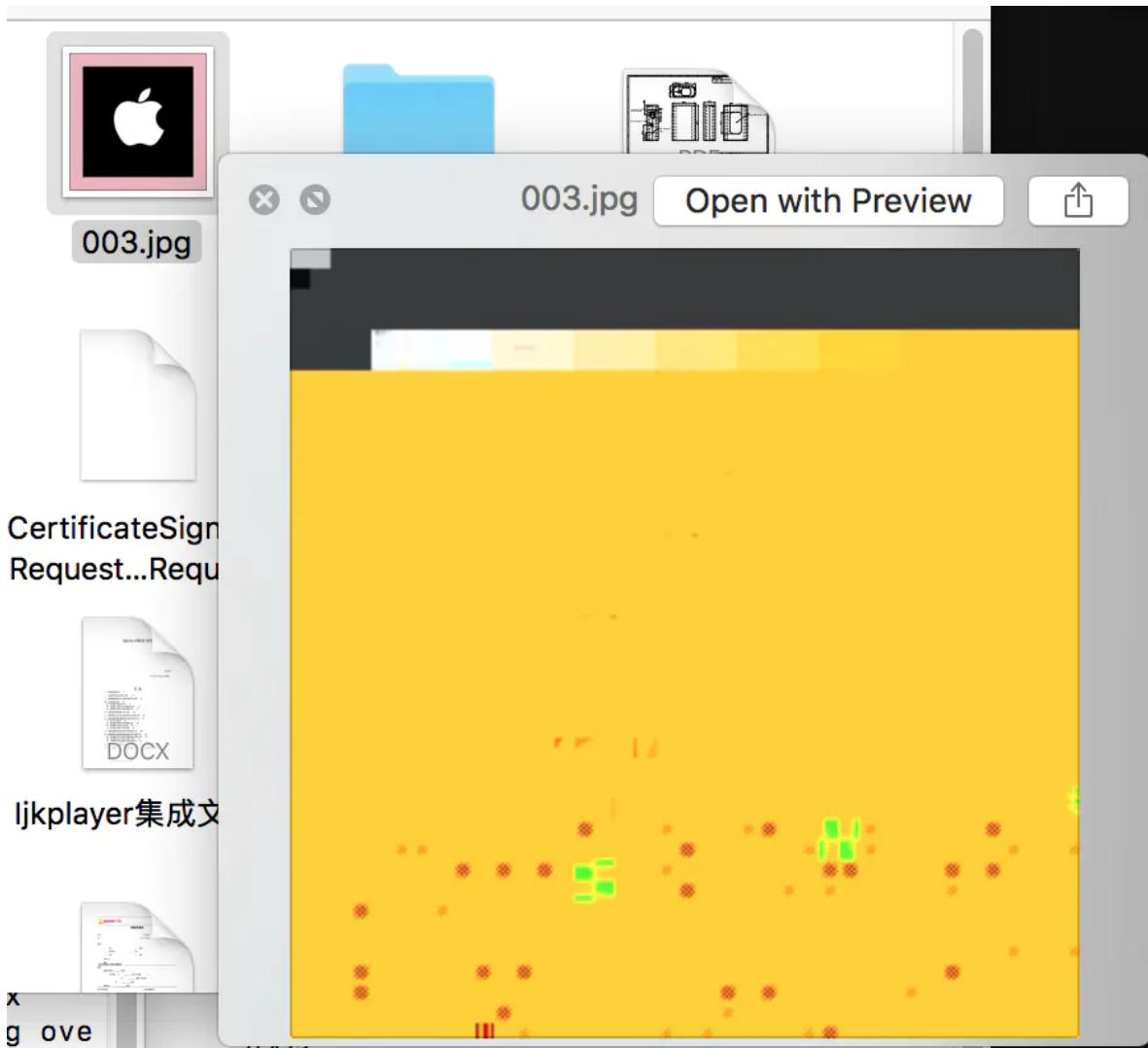
001.jpg

生成的图片如下：（虽然你们看到的没花屏，不过在我这儿确实是花的）



003.jpg

我看到的，其实这是一张图，我上传上去就正常了



见了鬼的图片。。。

## 从4:3到16:9的填充视频

有些设备只能以16:9的宽高比播放视频，而4:3宽高比的视频必须在两种尺寸的水平方向上进行填充。在这种情况下，高度保持不变，宽度等于高度值乘以16/9。x值（输入视频帧水平偏移量）从表达式  $(\text{output\_width} - \text{input\_width}) / 2$  开始计数，因此填充的语法为：

```
ffmpeg -i input -vf pad=ih*16/9:ih:(ow-iw)/2:0:color output
```

例如，不知道电影的确切分辨率。mpg文件有4:3的长宽比，我们可以在默认的黑色中添加所谓的“pillarbox”命令：

```
ffmpeg -i film.mpg -vf pad=ih*16/9:ih:(ow-iw)/2:0 film_wide.avi
```



## 从16:9到4:3的填充视频

为了显示在4:3高宽比的显示中所创建的视频，我们应该垂直地填充两个大小的输入。因此，宽度保持不变，高度为宽\*  $3/4$ 。y值(输入视频帧垂直偏移)是从表达式 $(\text{output\_height} - \text{input\_height})/2$ 中计算的，填充的语法为：

```
ffmpeg -i input -vf pad=iw:iw*3/4:0:(oh-ih)/2:color output
```

- 为啥我自己测试的时候都是失败的呢%>\_<%。。

```
zhangfataodeMBP:~ zhangfangtao$ ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4  
-vf pad=iw:iw*3/4:0:(oh-ih)/2:color /Users/zhangfangtao/Desktop/newTest.mp4  
-bash: syntax error near unexpected token `'  
zhangfataodeMBP:~ zhangfangtao$
```

语法错误

例如，如果不知道输入文件的确切分辨率，我们可以在hd\_video中添加默认为黑色的“letterbox”。avi文件在16/9宽比与命令：

```
ffmpeg -i hd_video.avi -vf pad=iw:iw*3/4:0:(oh-ih)/2 video.avi
```

\*同样是语法错误。。。我测试不了%>\_<%



## 填充从和到不同的纵横比

描述了4:3和16:9的填充率是最常见的，但例如，电视广告在14:9的长宽比中创建，一些电影的比例比16/9大。

### Pillarboxing - adding boxes horizontally

为了调整较小的宽高宽比，我们需要增加输出宽度，它的值将是高度值乘以一个新的纵横比(ar)。通用公式是：

```
ffmpeg -i input -vf pad=ih*ar:ih:(ow-iw)/2:0:color output
```

### Letterboxing - adding boxes vertically

为了调整更大的宽高宽比，我们需要增加输出高度，它的值将是宽度值除以新的纵横比(ar)。通用公式是：

```
ffmpeg -i input -vf pad=iw:iw*ar:0:(oh-ih)/2:color output
```

## 07-翻转和旋转视频

视频帧的翻转和旋转是常见的视觉操作，可以用来创建各种有趣的效果，比如输入的镜像版本。

### 水平翻转

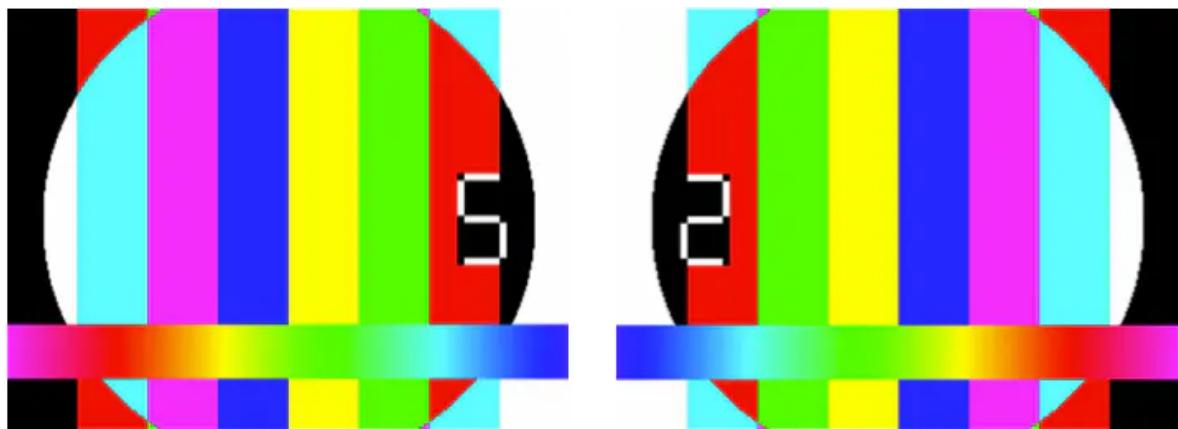
水平镜像的视频版本-水平翻转是用一个在表格中描述的hflip过滤器创建的。

### Video filter: hflip

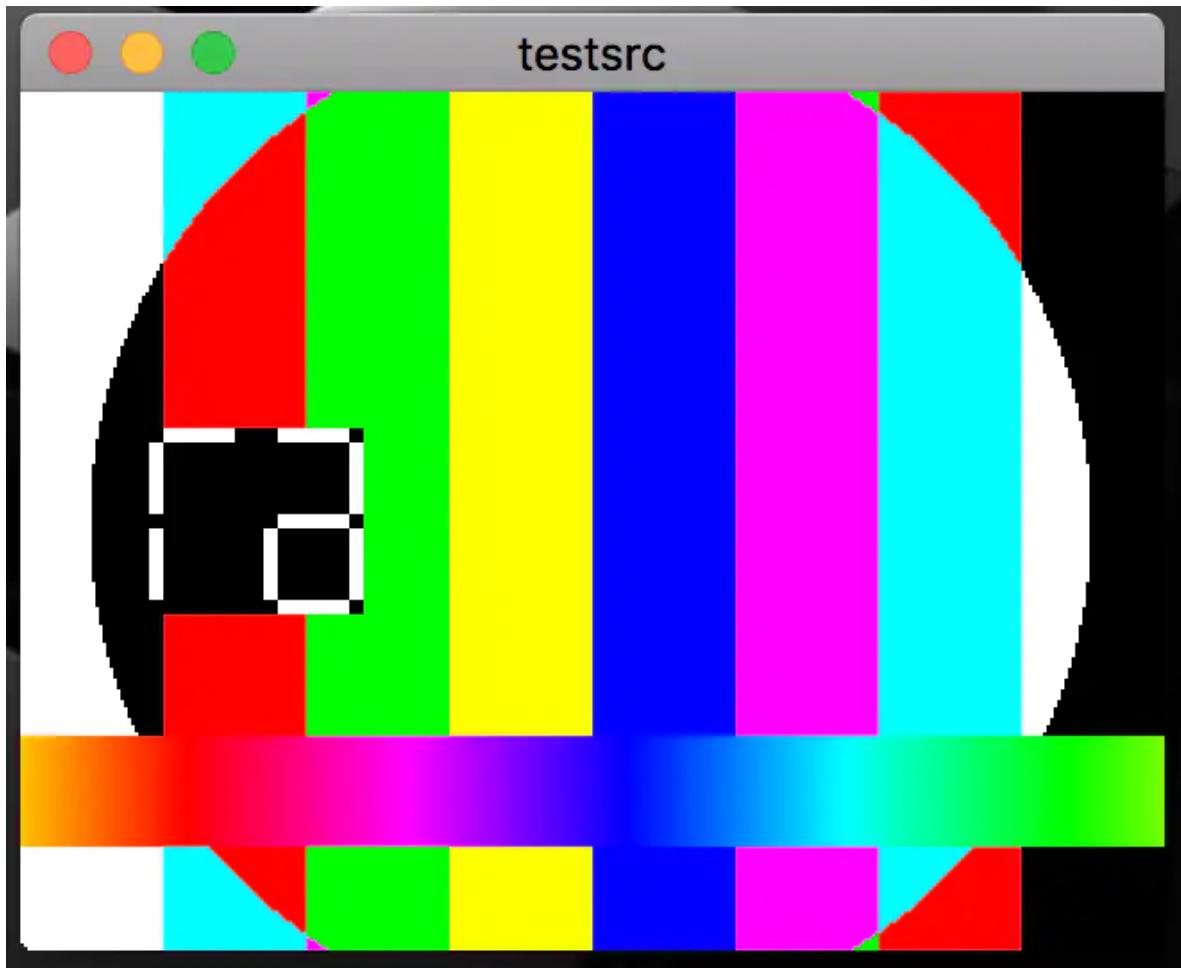
描述	水平翻转输入视频，因此输出看起来像是从侧面镜像。过滤器没有参数
语法	-vf hflip

为了测试tests src视频源的水平翻转，我们可以使用以下命令：

```
ffplay -f lavfi -i testsrc -vf hflip
```



我的测试命令和书里面的一样，显示效果如下图：



左右是倒着的

## 垂直翻转

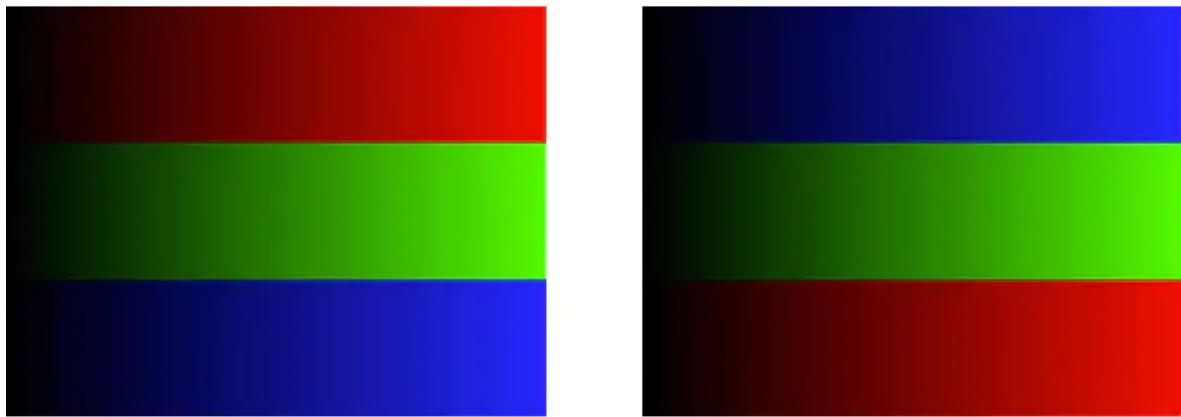
对于垂直方向的输入帧，我们可以使用在表格中描述的vflip过滤器。

### Video filter: vflip

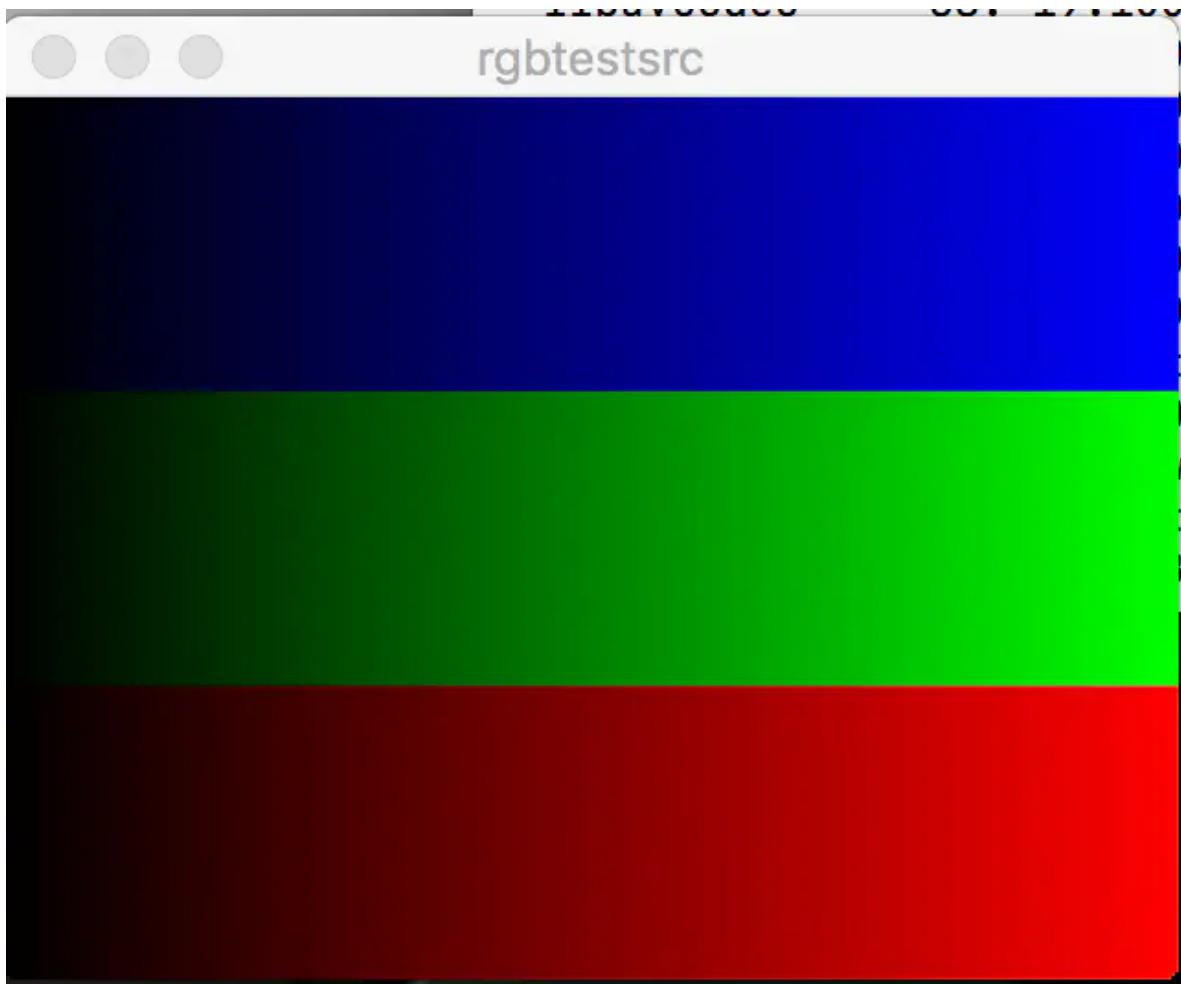
描述	垂直翻转输入视频，因此输出看起来像从顶部或底部镜像
语法	-vf vflip

左下方的图像是在第25章中描述的rgbrestsrc模式。为了得到它的垂直翻转版本，我们可以使用下一个命令：

```
ffplay -f lavfi -i rgbtestsrc -vf vflip
```



我自己的测试命令和书上的一样，效果图如下：



## 介绍旋转

之前的FFmpeg版本包含特殊的过滤器旋转，使视频旋转可以进入角度值。这个过滤器现在已被弃用，取而代之的是一个转置过滤器，它允许旋转并且可以选择立即翻转输入。表中描述了转置滤波器。

**Video filter: transpose**

描述	将行与输入的列进行转置, 如果选择, 也会翻转结果
语法	transpose={0, 1, 2, 3} one from the values 0 - 3 is used
***	描述可用的值
0	输入由90°逆时针旋转,垂直翻转
1	输入是顺时针旋转90°
2	输入是逆时针旋转90°
3	输入是顺时针旋转90°,垂直翻转

请注意, 转置滤波器的值0和3在视频帧上同时提供两个操作——旋转和垂直翻转。这意味着值0的使用包括两个过滤器的效果, 下面两个命令的结果相同:

```
ffplay -f lavfi -i smptebars -vf transpose=0
ffplay -f lavfi -i smptebars -vf transpose=2,vflip
```

看看我自己的测试效果:

如果使用如下命令行:

```
ffplay -f lavfi -i smptebars -vf transpose=0
```

显示效果如下图:



如果使用如下代码:

```
ffplay -f lavfi -i smptebars -vf transpose=2,vflip
```

显示效果如下图:



image.png

类似地，在接下来的两个命令中，可以用两个过滤器替换值3:

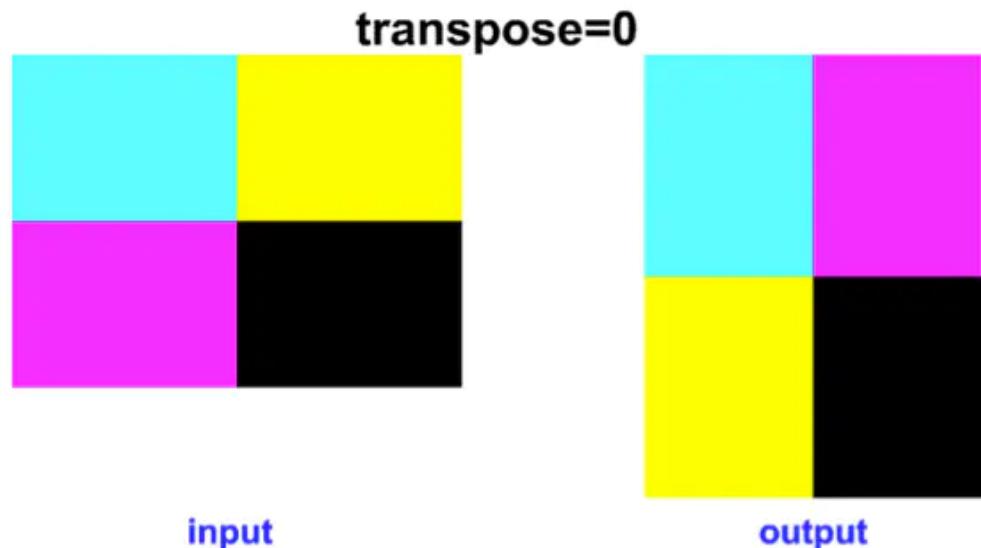
```
ffplay -f lavfi -i smptebars -vf transpose=3  
ffplay -f lavfi -i smptebars -vf transpose=1,vflip
```

在接下来的章节中，我们描述了带有插图的转置过滤器的每个值的用法。

## 逆时针旋转90度，垂直翻转

下一个命令以顺时针方向旋转90度的输入:

```
ffmpeg -i CMYK.avi -vf transpose=0 CMYK_transposed.avi
```



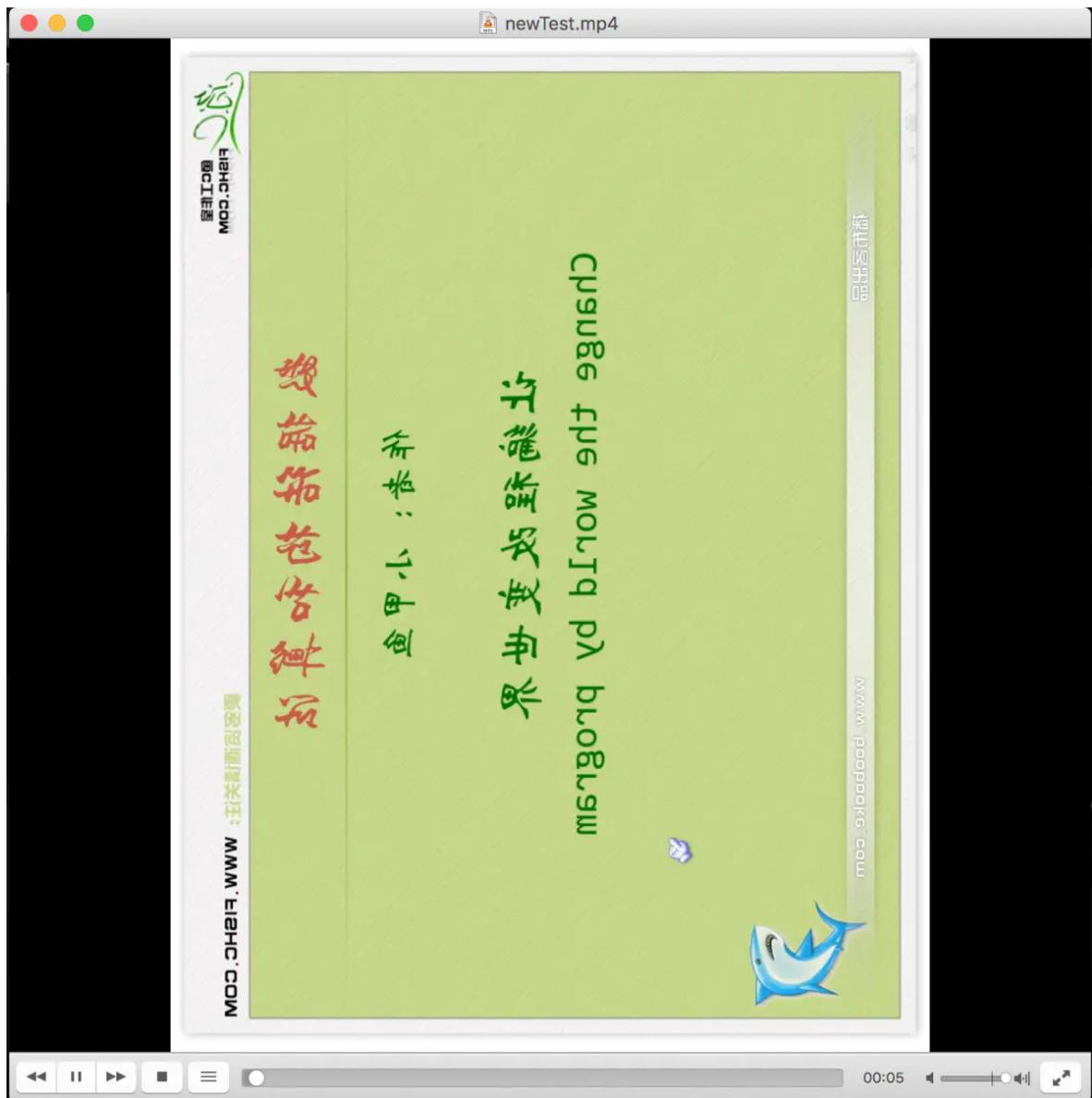
我自己的测试命令如下：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf transpose=0 /Users/zhangfangtao/Desktop/newTest.mp4
```

原来的视频：



原来的视频

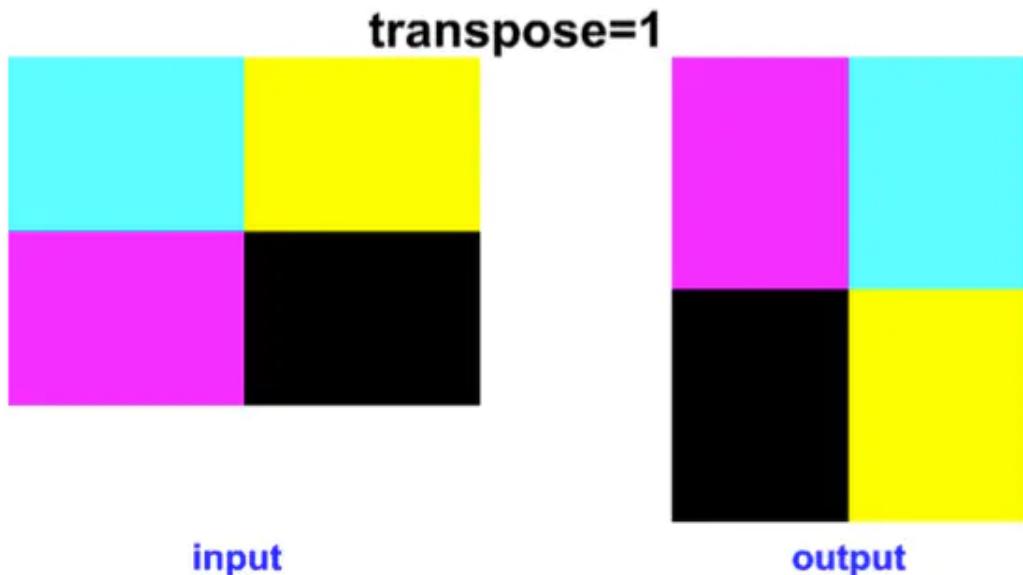


转换之后的视频

## 顺时针旋转90度

下一个命令输入90°旋转顺时针方向:

```
ffmpeg -i CMYK.avi -vf transpose=1 CMYK_transposed.avi
```



79

我的测试命令如下：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf transpose=1  
/Users/zhangfangtao/Desktop/newTest.mp4
```

显示效果如下：

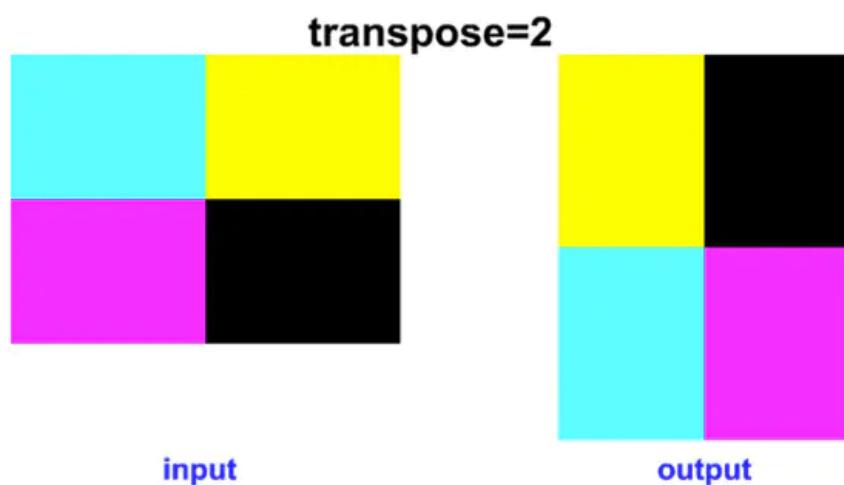


顺时针旋转90度

**逆时针旋转90度**

下一个命令输入90°逆时针旋转:

```
ffmpeg -i CMYK.avi -vf transpose=2 CMYK_transposed.avi
```



我的测试命令如下：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf transpose=2  
/Users/zhangfangtao/Desktop/newTest.mp4
```

显示效果：



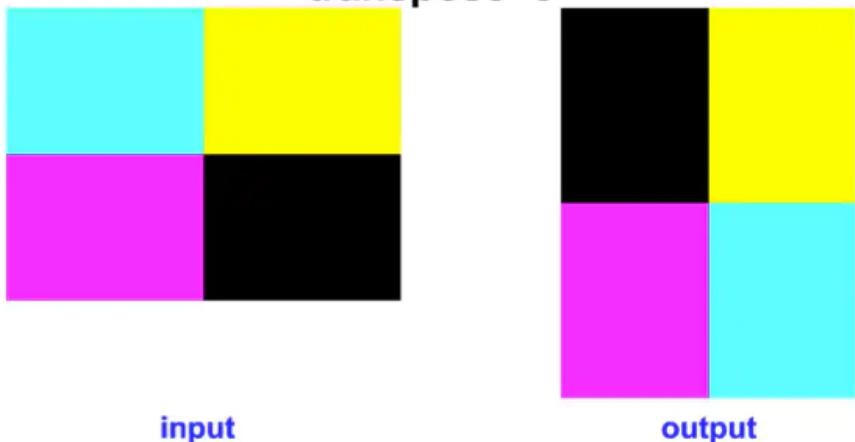
image.png

**顺时针旋转90度，垂直翻转**

下一个命令输入90°旋转顺时针方向和垂直翻转它：

```
ffmpeg -i CMYK.avi -vf transpose=3 CMYK_transposed.avi
```

**transpose=3**



我的测试命令如下：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf transpose=3  
/Users/zhangfangtao/Desktop/newTest.mp4
```

效果图如下：



## 08-模糊，锐化和其他去噪

包含各种噪声的视频输入可以使用去噪滤波器和选项来增强。在视频编码之前，去噪是视频预处理的一部分。

### 模糊视频效果

模糊效果用于提高图像（视频帧）中某些类型的噪声的质量，其中每个输出像素值是根据相邻像素值计算的。例如，模糊效果可以改善从印刷的半色调图片扫描的图像。为了模糊输入视频，我们可以使用表中描述的均值模糊过滤器：

**Video filter: boxblur**

描述	使用均值模糊算法在输入上创建一个模糊效果
语法	boxblur=luma_r:luma_p[:chroma_r:chroma_p[:alpha_r:alpha_p]] filter expects 2 or 4 or 6 parameters, r = 半径, p = 权重, 程度, 功率
**	参数
alpha_r	- 用于模糊相关输入平面(以像素为单位)的盒子的半径 - value是下面描述的变量的表达式 - 默认值来源于luma_radius和luma_power
alpha_p	- alpha功率, 确定过滤器被应用到相关平面的次数 - 默认值来源于luma_radius和luma_power

<b>描述</b>	<b>使用均值模糊算法在输入上创建一个模糊效果</b>
chroma_r	-用于模糊相关输入平面(以像素为单位)的box的色度半径 - value是下面描述的变量的表达式 -默认值来源于luma_radius和luma_power
chroma_p	-色度功率, 确定过滤器被应用到相关平面的次数 -默认值来源于luma_radius和luma_power
luma_r	-用于模糊相关输入平面(以像素为单位)的box的半径 - value是下面描述的变量的表达式
luma_p	- luma功率, 确定过滤器被应用到相关平面的次数
***	在表达式中, 对阿尔法, 色度和luma半径的变量
w,h	输入宽度和像素高度
cw, ch	输入色度图像的像素宽度和高度
hsub	水平色度子样本值, 为yuv422p像素格式为2
vsub	垂直色度子样本值, 为yuv422p像素格式为1
	半径是一个非负数, 并且不能大于luma和阿尔法平面的表达式 $\min(w,h)/2$ 的值, 以及对chroma平面的 $\min(cw,ch)/2$ 的值

例如, 在输入视频中, 当luma半径值为1.5,luma功率值为1时, 我们可以使用下一个命令:

```
ffmpeg -i input.mpg -vf boxblur=1.5:1 output.mp4
```

我的测试命令是:

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf boxblur=1.5:1
/Users/zhangfangtao/Desktop/newTest.mp4
```

之前的视频界面如下:



之前的视频

转换之后的视频如下：



转换之后的视频

另一个FFmpeg过滤器与模糊效果是一个smartblur过滤器在表中描述：

## Video filter: smartblur

描述	模糊输入而不影响轮廓
语法	smartblur=luma_r:luma_s:luma_t[:chroma_r:chroma_s:chroma_t] parameters in [] are optional, r = radius, p = power, t = threshold
***	参数的描述
chroma_r	色度(颜色)半径, 从0.1到5.0的浮点数, 它指定用于模糊图像的高斯滤波器的方差(如果更大)
chroma_s	色度强度, 在范围-1.0到1.0之间的浮点数, 配置模糊;从0.0到1.0的值将模糊图像, 从-1.0到0.0的值将增强图像
chroma_t	chrominance threshold, 一个从-30到30的整数, 它被用作一个系数来决定一个像素是否应该被模糊;0的值将过滤所有的图像, 0到30的值将过滤平坦区域, 从-30到0的值将过滤边缘
luma_r	亮度(亮度)半径, 从0.1到5.0的浮点数, 指定用于模糊图像的高斯滤波器的方差(如果更大的话, 会更慢)
luma_s	亮度强度, 从-1.0到1.0的浮动数值, 配置模糊;从0.0到1.0的值将模糊图像, 从-1.0到0.0的值将增强图像
luma_t	亮度threshold, 一个整数, 范围从-30到30, 作为一个系数来决定一个像素是否应该被模糊;0的值将过滤所有图像, 0到30的值将过滤平面区域, 从-30到0的值将过滤edges图像
	如果色度参数没有设置, 则使用luma参数来实现像素的色度

例如, 为了改进半色调图像, 我们将luma半径设为最大值5, 亮度强度为0.8, 亮度阈值为0, 因此整个图像是模糊的:

```
ffmpeg -i halftone.jpg -vf smartblur=5:0.8:0 blurred_halftone.png
```



我的测试命令如下:

```
ffmpeg -i /Users/zhangfangtao/Desktop/timg.jpeg -vf smartblur=5:0.8:0  
/Users/zhangfangtao/Desktop/newTimg.jpeg
```

原来的图像：



原来的图像

新的图像：



转换之后的图像

**锐化视频**

为了锐化或模糊视频帧，我们可以使用表中描述的不清晰的过滤器。

### Video filter: unsharp

描述	根据指定的参数增加或模糊输入视频
语法	<code>l_msize_x:l_msize_y:l_amount:c_msize_x:c_msize_y:c_amount</code> all parameters are optional, if not set, the default is 5:5:1.0:5:5:0.0
***	参数的描述
<code>l_msize_x,luma_msize_x</code>	luma矩阵水平尺寸，3和13之间的整数，默认值为5
<code>l_msize_y,luma_msize_y</code>	luma矩阵的垂直大小，整数在3和13之间，默认值是5
<code>l_amount,luma_amount</code>	luma效应强度，介于-2.0和5.0之间的浮点数，负值创建模糊效果，默认值为1
<code>c_msize_x,chroma_msize_x</code>	色度矩阵的水平大小，整数在3和13之间，默认值是5
<code>c_msize_y,chroma_msize_y</code>	色度矩阵的垂直大小，整数在3和13之间，默认值为5
<code>c_amount,chroma_amount</code>	chroma效果强度，-2.0和5.0之间的浮动值，负值创建模糊效果，默认值为0.0

锐化滤波器可以作为普通的不锐掩模和高斯模糊。例如，要使用默认值锐化输入，我们可以使用该命令。

```
ffmpeg -i input -vf unsharp output.mp4
```

我的测试命令是：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf unsharp /Users/zhangfangtao/Desktop/newTest.mp4
```

之前的视频界面如下图：

# 数据结构和算法

作者: 小甲鱼

让编程改变世界

Change the world by program



播布客出品

[www.bobooke.com](http://www.bobooke.com)

转换之后的视频界面如下: (不太明显, 不过还是有一些效果的, 锐化了)

# 数据结构和算法

作者: 小甲鱼

让编程改变世界

Change the world by program



播布客出品

[www.bobooke.com](http://www.bobooke.com)

输出将使用尺寸为 $5 \times 5$ 的亮度矩阵和亮度效果强度为1.0的锐化。为了产生高斯模糊效果, 例如, 我们可以使用负数来表示亮度和/或色度值

```
ffmpeg -i input -vf unsharp=6:6:-2 output.mp4
```

- 这里面我有一个问题，我用书上的测试命令，但是终端给出的错误提示是： Invalid even size for luma matrix size 6x6 (对于亮度矩阵大小为6x6的均匀尺寸无效) 所以我就测试成5\*5的矩阵模式  
我的测试命令如下：

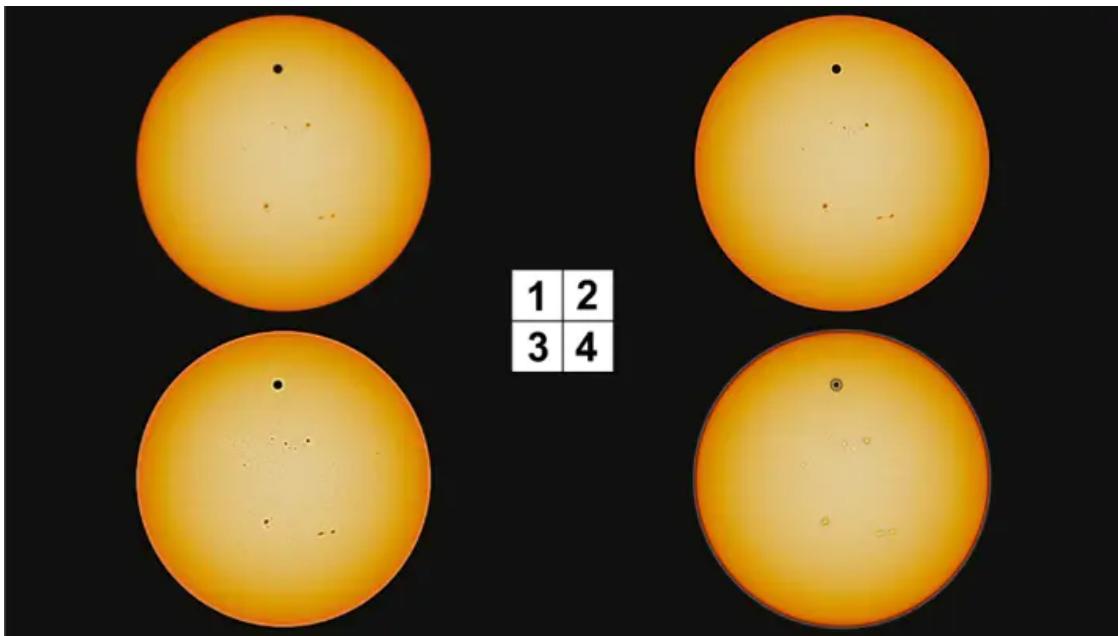
```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf unsharp=5:5:3  
/Users/zhangfangtao/Desktop/newTest.mp4
```

效果图：（感觉每一个图像都加了一个边框）



下一个复杂的图像说明了不锋利的过滤器的使用价值：

- 图片1:美国国家航空航天局(NASA)在2012年6月5日记录了金星凌日的情况，这段视频可以从NASA的网站上下载。
- 图2:-vf unsharp(没有参数，默认值被使用)黑点更明显，没有可见的工件。
- 图像3:-vf unsharp=6:6:3(相对较强的锐化效果)黑点更明显，但可见是轻微的扭曲。
- 图4:-vf unsharp=6:6:-2(相对较强的模糊效果)-2的luma量参数模糊了结果，在金星周围形成了一个虚拟的环。



## 降噪与denoise3d

视频过滤器denoise3d减少了噪音，它是mp过滤器的一部分(来自MPlayer项目)。

### Video filter: denoise3d (part of mp filter)

描述	生成质量更好的平滑视频帧，并尝试提高可压缩性。
语法	mp=denoise3d[=luma_spatial[:chroma_spatial[:luma_tmp[:chroma_tmp]]]] (所有参数都是可选的)
***	参数的描述
luma_spatial	空间luma强度，非负浮点数，默认值为4.0
chroma_spatial	空间色度强度，非负浮点数，默认值为3.0
luma_tmp	时间luma强度，非负浮点数，默认值为6.0
chroma_tmp	时间色度强度，非负浮点数，默认值为 $\text{luma\_tmp} * \text{chroma\_spatial} / \text{luma\_spatial}$

例如，要使用denoise3d过滤器的默认值来增强输入，我们可以使用该命令

```
ffmpeg -i input.mpg -vf mp=denoise3d output.webm
```

- 我这边测试失败，错误代码是：No such filter: 'mp' (没有'mp'这个过滤器)

这张图片展示了NASA阿波罗计划中使用denoise3d filter默认值的增强存档视频



## 降噪与hqdn3d

denoise3d过滤器的高级版本是hqdn3d过滤器，它已经在libavfilter库中，是一个本地的FFmpeg过滤器。过滤器的名称是高质量的denoise三维过滤器的缩写，它在表中描述：

### Video filter: hqdn3d

描述	生产高质量的平滑视频帧，并尝试提高压缩率，它是一个增强版的 <b>denoise3d</b> 过滤器
语法	hqdn3d=[luma_spatial[:chroma_spatial[:luma_tmp[:chroma_tmp]]]]
***	参数的描述
luma_spatial	空间luma强度，非负浮点数，默认值为4.0
chroma_spatial	空间色度强度，非负浮点数，默认值为 $3.0 * \text{luma\_spatial} / 4.0$
luma_tmp	时间luma强度，非负浮点数，默认值为 $6.0 * \text{luma\_spatial} / 4.0$
chroma_tmp	时间色度强度，非负浮点数，默认值为 $\text{luma\_tmp} * \text{chroma\_spatial} / \text{luma\_spatial}$

例如，为了减少视频输入中带有默认hqdn3d值的噪声，我们可以使用以下命令：

```
ffmpeg -i input.avi -vf hqdn3d output.mp4
```

我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf hqdn3d
/Users/zhangfangtao/Desktop/newTest.mp4
```

显示的效果图：

# 数据结构和算法

作者: 小甲鱼

让编程改变世界

Change the world by program

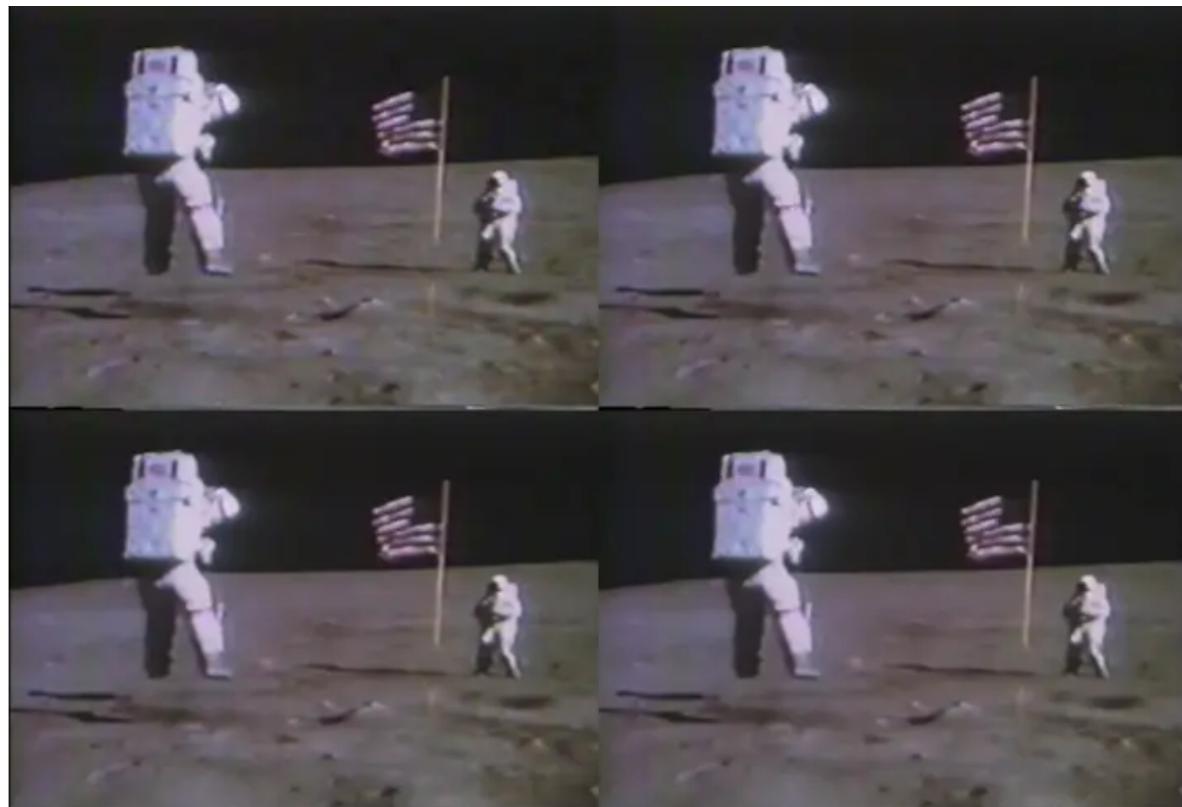


播布客出品

[www.boobooke.com](http://www.boobooke.com)

好像看不出来什么特殊的差别

下一个图像说明了hqdn3d过滤器的各种值的用法。



12 FFmpeg hqdn3d filter: 1 is input, 2 is hqdn3d default

34 window 3 uses hqdn3d=6 and window 4 uses hqdn3d=10:8

使用nr选项进行降噪

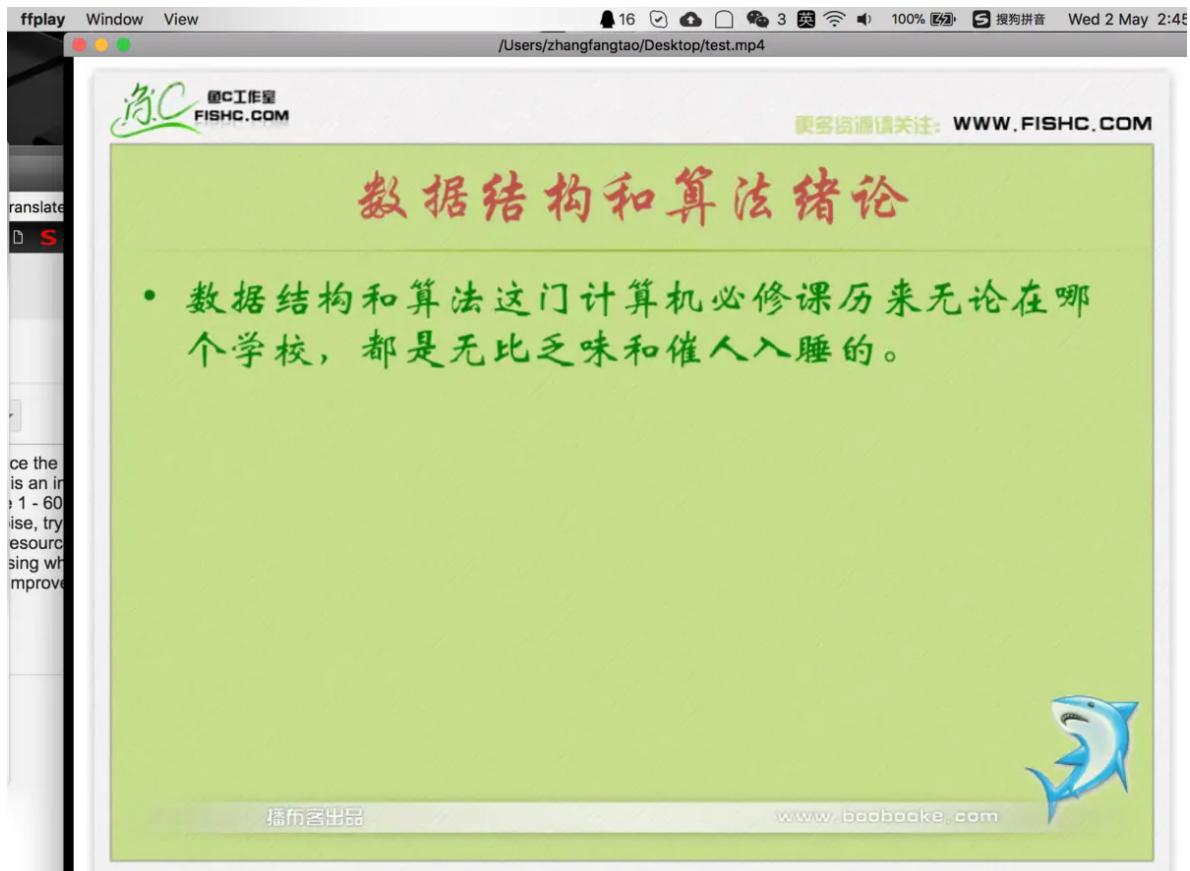
如何减少视频输入中的噪音的其他方法是-nr（降噪）选项。它的值是一个从0到100000的整数，其中0是默认值，范围1-600对公用内容有用。如果视频包含强烈的噪音，请尝试使用更高的值。由于此选项比denoise3d和hqdn3d过滤器使用的计算机资源少得多，因此当速度很重要时，它是消噪的首选方式。例如，在较旧的计算机上，我们可以使用以下命令改善观看稍微噪音的视频：

```
ffplay -i input.avi -nr 500
```

我的测试命令：

```
ffplay -i /Users/zhangfangtao/Desktop/test.mp4 -nr 500
```

效果图（确实是用ffplay播放出来的）



## 09-overlay-画中画

overlay视频技术经常被使用，常见的例子是放置在电视屏幕上的电视频道标志，通常位于右上角，以标识特定的频道。另一个例子是画中画功能，可以在主屏幕的其中一个角落显示小窗口。小窗口包含选定的电视频道或其他内容，同时在主屏幕上观看节目 - 这在等待特定内容，跳过广告等时很有用。

本章仅包含简单的overlay实例，更复杂的例子是在[颜色修正](#)，[高级技术](#)等章节中。

### 关于overlay的介绍

视频overlay是一种技术，它可以在(通常是较大的)背景视频或图像上显示前景视频或图像。我们可以使用在表格中描述的覆盖视频过滤器：

### Video filter: overlay

描述	在指定位置上覆盖第一个输入
语法	overlay[=:y[:rgb={0, 1}]] 参数x和y是可选的，其默认值为0 rgb参数是可选的，其值为0或1
***	参数的描述
x	从左上角的水平坐标，默认值为0
y	从左上角的垂直坐标，默认值为0
rgb	rgb = 0...输入的颜色空间不改变，默认值 rgb = 1...输入的颜色空间设置为RGB
***	变量，可以用在x和y的表达式中
main_w or W	主要输入宽度
main_h or H	主要输入高度
overlay_w or w	overlay输入宽度
overlay_h or h	overlay输入高度

## overlay命令结构

视频覆盖命令的结构如下，input1是视频背景，input2是前景：

```
ffmpeg -i input1 -i input2 -filter_complex overlay=x:y output
```

我的测试命令如下：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -i
/Users/zhangfangtao/Desktop/PDXlogoanimationHDh264.mp4 -filter_complex
overlay=100:100 /Users/zhangfangtao/Desktop/newTest.mp4
```

效果图：



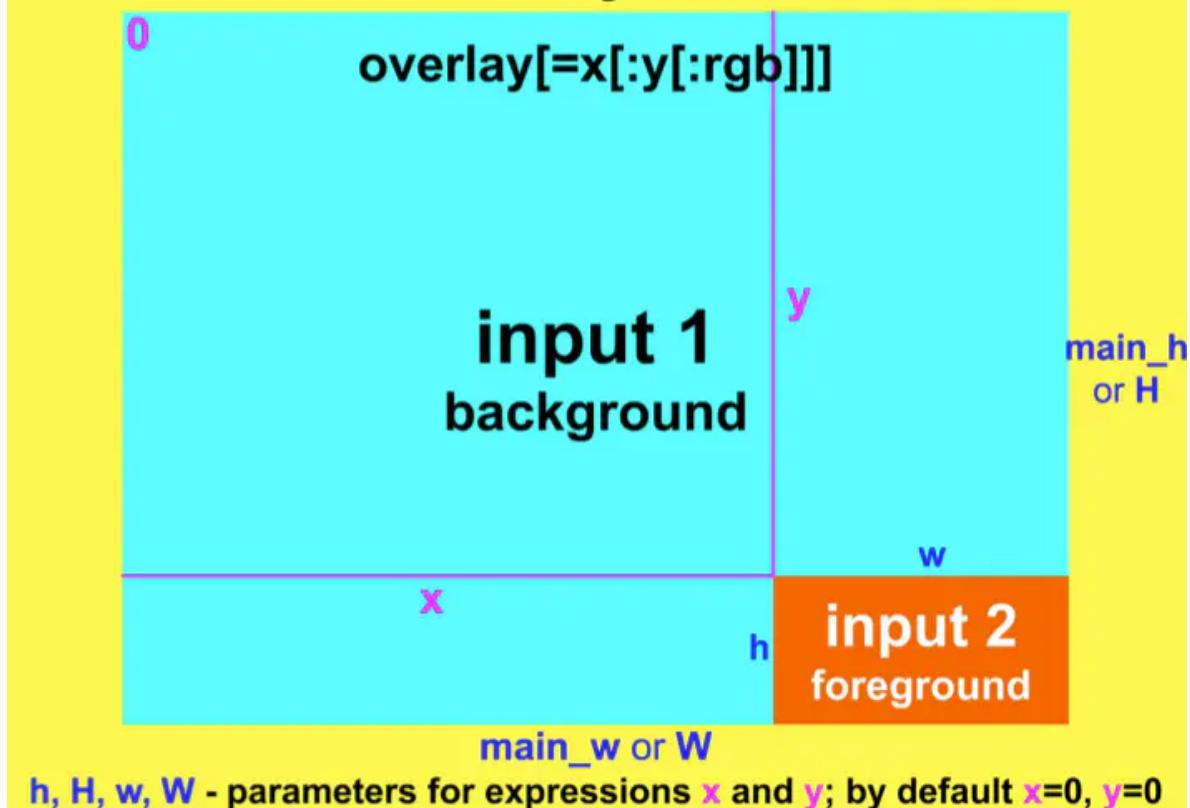
前置的那个有点大。。。

请注意，不是使用-vf选项，而是使用-filter\_complex选项，因为现在有两个输入源(通常是视频文件或图像)。但是使用带有链接标签的filtergraph，我们可以使用一个电影视频源，它将包含第二个输入，并且只使用-vf选项：

```
ffmpeg -i input1 -vf movie=input2[logo];[in][logo]overlay=x:y output
```

另一种方法是将一个输入拆分为几个输出，并使用pad过滤器创建更大的背景。这个背景在filterchain中作为覆盖过滤器的第一个输入，这个方法已经在[第一个章节](#)中的过滤器，过滤链和过滤器图部分中被描述了，。

# Overlay filter



## 一个角落的logo

为了让内容保持可见，logo经常被放置在屏幕的四个角落里。接下来的4个例子使用这一对。mp4视频作为第一个包含一对结婚对象的输入，第二个输入是包含文本M+P(例如，Mary和Peter)的红色心脏。视频分辨率为1280x720像素，logo大小为150x140像素，但我们不需要这个尺寸来计算logo的位置。logo的左上角(x和y坐标)的正确位置是由背景和前景的宽度和高度值决定的：

W H -宽度和背景高度(视频)

w h -宽度和前景高度(logo)

### Logo在左上角

```
ffmpeg -i pair.mp4 -i logo.png -filter_complex overlay pair1.mp4
```



我自己的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -i  
/Users/zhangfangtao/Desktop/001.jpg -filter_complex overlay=0:0  
/Users/zhangfangtao/Desktop/newTest.mp4
```

显示的效果图：



Logo在右上角

```
ffmpeg -i pair.mp4 -i logo.png -filter_complex overlay=W-w pair2.mp4
```



我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -i  
/Users/zhangfangtao/Desktop/001.jpg -filter_complex overlay=W-w:0  
/Users/zhangfangtao/Desktop/newTest.mp4
```

实现的效果如下图：



Logo在右下角

```
ffmpeg -i pair.mp4 -i logo.png -filter_complex overlay=W-w:H-h pair3.mp4
```



我的测试代码：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -i  
/Users/zhangfangtao/Desktop/001.jpg -filter_complex overlay=W-w:H-h  
/Users/zhangfangtao/Desktop/newTest.mp4
```

效果图：



Logo在左下角

```
ffmpeg -i pair.mp4 -i logo.png -filter_complex overlay=0:H-h pair4.mp4
```



我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -i  
/Users/zhangfangtao/Desktop/001.jpg -filter_complex overlay=0:H-h  
/Users/zhangfangtao/Desktop/newTest.mp4
```

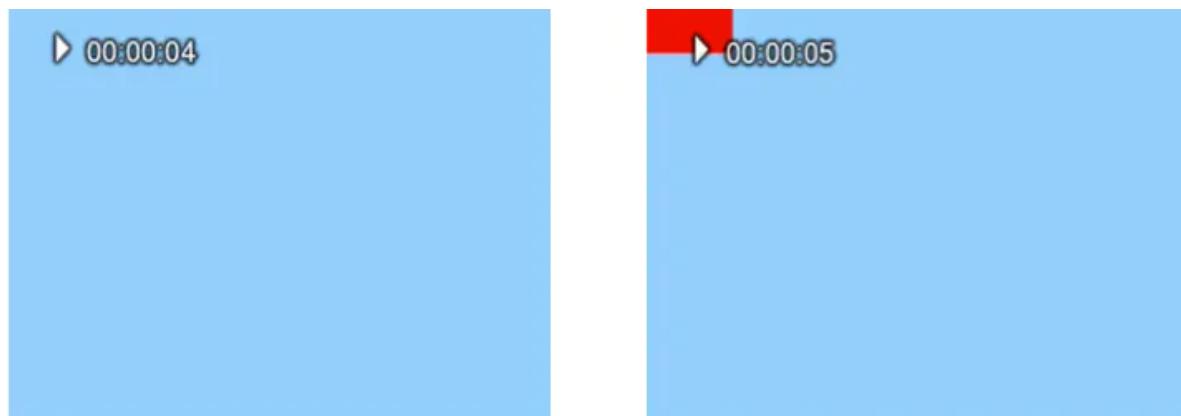
效果图：



## Logo显示在指定的时刻

在某些情况下，例如当视频包含一个特别的介绍时，可以在一个时间间隔后加上一个-itsoffset选项来添加标识(或其他源到覆盖)。例如，在开始的5秒后，在蓝色背景上添加一个红色标志，我们可以使用以下命令：

```
ffmpeg -i video_with_timer.mp4 -itsoffset 5 -i logo.png ^ -filter_complex  
overlay timer_with_logo.mp4
```



- 注意：这里我怀疑作者的命令是有问题的，因为那个'^'符号好像是不支持的

我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -itsoffset 5 -i  
/Users/zhangfangtao/Desktop/001.jpg -filter_complex overlay=100:100  
/Users/zhangfangtao/Desktop/newTest.mp4
```

前五秒钟的画面：



前五秒钟的画面

五秒钟之后的画面：



五秒钟之后的画面

在第二个输入之前直接输入`-itsoffset`选项很重要，否则叠加效果将从输出的开始处开始。`-itsoffset`选项的更多示例请参见[时间操作](#)一章。其他延迟徽标的方法是使用[高级技术](#)一章中介绍的电影过滤器。

## 视频计时器

这个例子使用了1973年的公共领域NASA视频，其中阿波罗17号从月球表面开始到它的轨道。视频持续时间为29.93秒，分辨率为512x384像素。我们使用2位数计时器，就像使用[裁剪视频](#)章节中的数字一样。

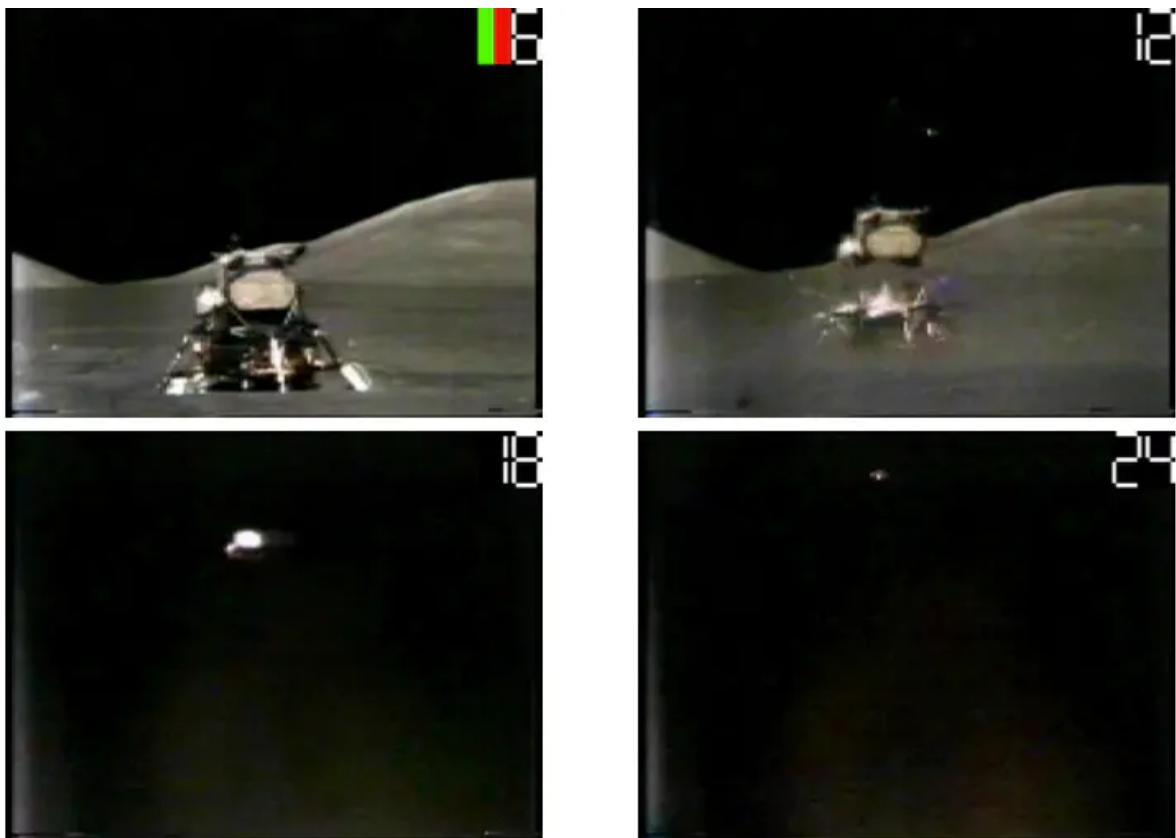
下面的指令可以生成timer.ogg视频文件：

```
ffmpeg -f lavfi -i testsrc -vf crop=61:52:224:94 -t 30 timer.ogg
```

现在我们有一个61x52像素大小的小视频，显示定时器从0到30秒。这段视频将会在阿波罗17号月球启动视频中被覆盖在右上角的命令：

```
ffmpeg -i start.mp4 -i timer.ogg -filter_complex overlay=451:startl.mp4
```

定时器的x坐标为 $512 - 61 = 451$ ,y坐标为0。



我的测试命令如下：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -i  
/Users/zhangfangtao/Desktop/timer.ogg -filter_complex overlay=451  
/Users/zhangfangtao/Desktop/newTest.mp4
```

效果图：



下一个命令将计时器调到1 / 2，并将其置于底部中心：

```
ffmpeg -i start.mp4 -vf movie=timer.ogv,scale=15:14[tm];^ [in]
[tm]overlay=248:371 overlay.mp4
```



现在计时器几乎看不到了。我们使用一个命名的标签[tm]作为缩放过滤器输出板，以便将改变大小的定时器作为覆盖过滤器的第二个输入，第一个输入是由默认[in]命名标签表示的文件start.mp4。

- 这个命令在我这儿测试不通过。。。错误代码：At least one output file must be specified（至少制定一个输出文件）

## 其他overlay的例子

其他的用到了overlay技术的例子：

- [FFmpeg基本介绍](#)章节下面的过滤器，过滤链和过滤图部分内容。
- [图像处理](#)下面的，切片，旋转和覆盖图像部分内容。
- [麦克风和网络摄像头](#)章节里面的使用两个摄像头部分内容。
- [颜色修正](#)章节：
- 

在两个window窗口中进行比较。

- 

在3个window窗口中比较。

- 

2和3window窗的亮度校正。

- 

4个window窗口的截面比较。

- [高级技术](#)章节里面的部分额外的媒体输入到filtergraph。

## 10-为视频添加文字

视频中包含的文本数据可以显着提高其信息质量。

### 在视频中添加文字的相关介绍

如何将一些文本添加到视频输出中的两种常用方法是使用[overlay-画中画](#)中的字幕或叠加技术(overlay)。具有许多可能性的最高级选项是使用表中描述的抽象滤镜：

**Video filter: drawtext**

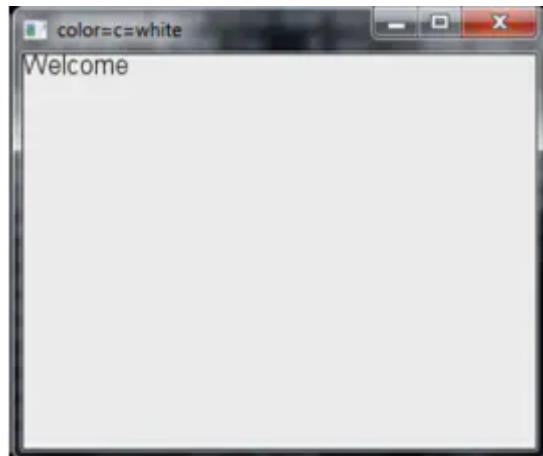
描述	从文本文件或字符串在视频中添加文本，并使用各种参数进行修改。文本从文本文件参数指定的文件中加载，或直接使用文本参数输入。其他必需参数是指定选定字体的字体文件。文本位置由x和y参数设置。
Syntax	drawtext=fontfile=font_f:text=text1[:p3=v3[:p4=v4[...]]] p3, p4 ...表示参数#3, 参数#4等
**	参数的描述
box	如果box=1，在文本周围绘制一个方框，颜色由boxcolor参数设置，默认值为0
boxcolor	颜色为box参数，颜色名称或0xRRGGBB[AA]格式(详见第1章的颜色名称)，默认值为白色
draw	表达式指定如果表达式求值为0时，是否应该绘制文本，则不绘制文本，默认为“1”。它用于指定只在特定条件下绘制文本。接受的变量和函数将在下一页和本章的内置数学函数中描述
fix_bounds	如果是true，文本坐标是固定的，以避免剪切
fontcolor	用于绘制字体、颜色名称或0xRRGGBB[AA]格式的颜色，默认为黑色
fontfile	字体文件用于绘制文本的正确路径，强制参数
fontsize	要绘制的文本字体大小，默认值为16
ft_load_flags	用于加载字体的标志，默认值是“render”;更多信息在FT_LOAD_* libfreetype标志的文档中
shadowcolor	在绘制的文本、颜色名称或0xRRGGBB[AA]格式后面绘制阴影的颜色，可能后面跟着一个alpha说明符，默认值是黑色
shadowx, shadowy	x和y抵消了文本阴影位置对文本位置的影响，它们可以是正的，也可以是负值，两者的默认值是“0”
tabsize	用于呈现选项卡的空间大小，默认值为4
timecode	hh:mm:ss[::]ff格式，可以使用或不使用文本参数，但必须指定timecode_rate参数
timecode_rate, rate, r	timecode帧率(仅限时间)
text	要绘制的文本字符串，必须是UTF-8编码的字符序列，如果没有指定textfile参数，该参数是必需的
textfile	文本文件与要绘制的文本，文本必须是一个UTF-8编码字符序列;如果不使用文本参数，则该参数是强制性的;如果指定了文本和文本文件参数，则显示一条错误消息

描述	从文本文件或字符串在视频中添加文本，并使用各种参数进行修改。文本从文本文件参数指定的文件中加载，或直接使用文本参数输入。其他必需参数是指定选定字体的字体文件。文本位置由x和y参数设置。
x, y	x和y值是表示文本将在视频帧中绘制的偏移量的表达式;它们相对于左上角，而x和y的默认值为“0”;下面描述了接受的变量和函数
***	接受变量和函数表达式中的x和y参数
dar	输入显示纵横比，与(w / h) * sar相同
hsub, vsub	水平和垂直的色度子样本值。例如，像素格式的“yuv422p”hsub是2，而vsub是1
line_h, lh	每个文本行的高度
main_h, h, H	输入的高度
main_w, w, W	输入的宽度
max_glyph_a, ascent	从基线到最高/上格坐标的最大距离，用于放置一个字形轮廓点，用于所有呈现的字形;一个正值，由于网格
max_glyph_d, descent	从基线到最低网格坐标的最大距离，用于放置一个字形轮廓点，用于所有呈现的字形;一个负值，由于网格
max_glyph_h	最大字形高度，即所呈现文本中所包含的所有字形的最大高度，相当于上升下降
max_glyph_w	最大的字形宽度，这是在呈现的文本中所包含的所有字形的最大宽度
n	输入框的数目，从0开始
rand(min, max)	返回最小值和最大值之间的随机数
sar	输入样本比例
t	时间戳以秒表示，如果输入时间戳未知
text_h or th	呈现文本的高度
text_w or tw	渲染文本的宽度
x, y	x和y坐标，在这里文本被绘制，这些参数允许x和y表达式相互引用，所以你可以指定y=x/dar

例如，要在白色背景上使用黑色字体的Arial字体绘制一个受欢迎的消息(默认位于左上角)，我们可以使用该命令(字符在一行上键入):

```
ffplay -f lavfi -i color=c=white ^ -vf
drawtext=fontfile=/Windows/Fonts/arial.ttf:text=Welcome
```

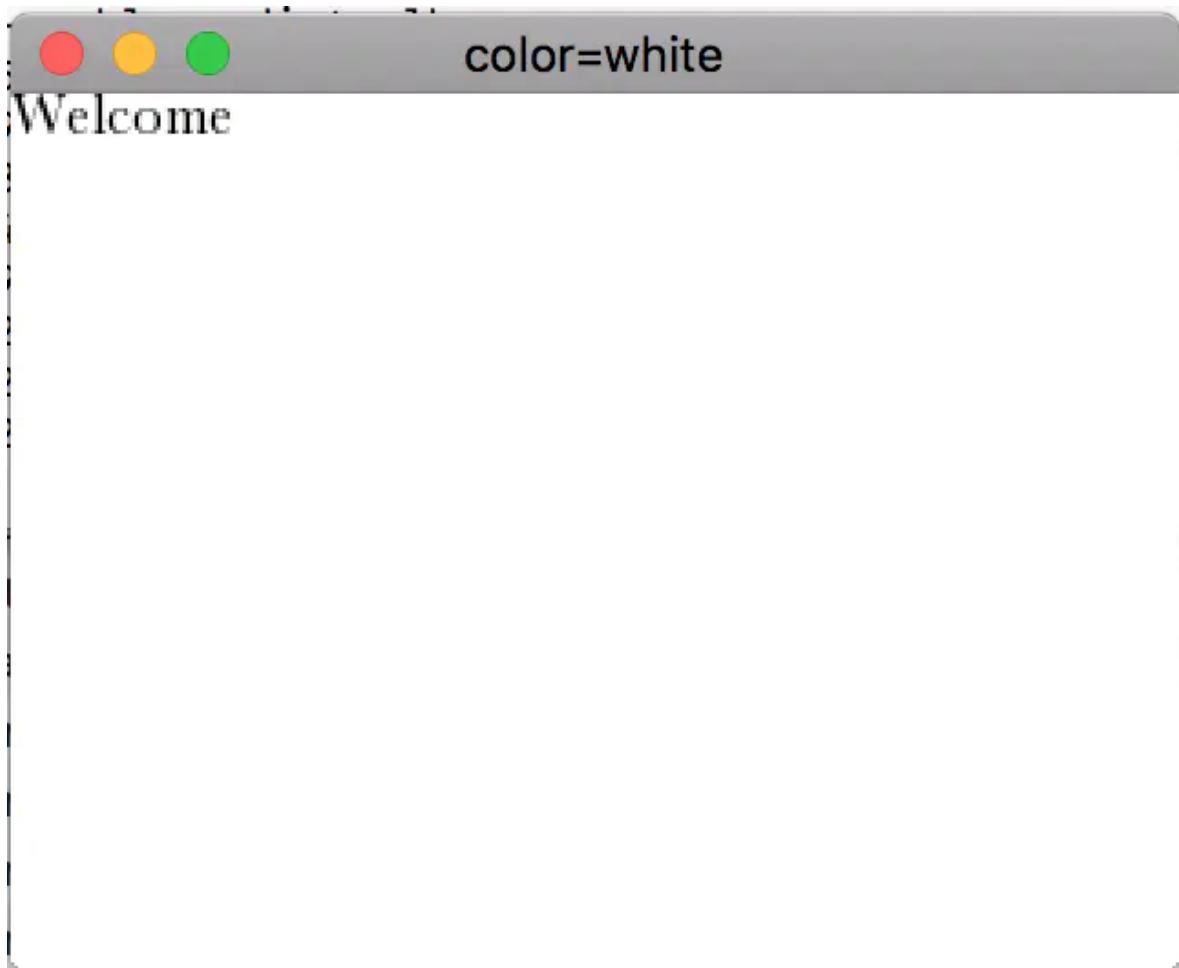
在Linux上，TTF字体位于文件夹“/usr/share/字体/TTF”中。结果如下图：



我的测试命令如下：

```
ffplay -f lavfi -i color=white -vf  
drawtext=fontfile=/Library/Fonts/Baskerville.ttc:text=Welcome
```

显示的效果如图：



如果难以指定字体文件的路径，则可以将字体文件(例如arial.ttf)复制到当前目录，以便将前面的命令简化为表单：

```
ffplay -f lavfi -i color=c=white -vf drawtext=fontfile=arial.ttf:text=Welcome
```

- 这个我就不测试了，太简单了。。。。。

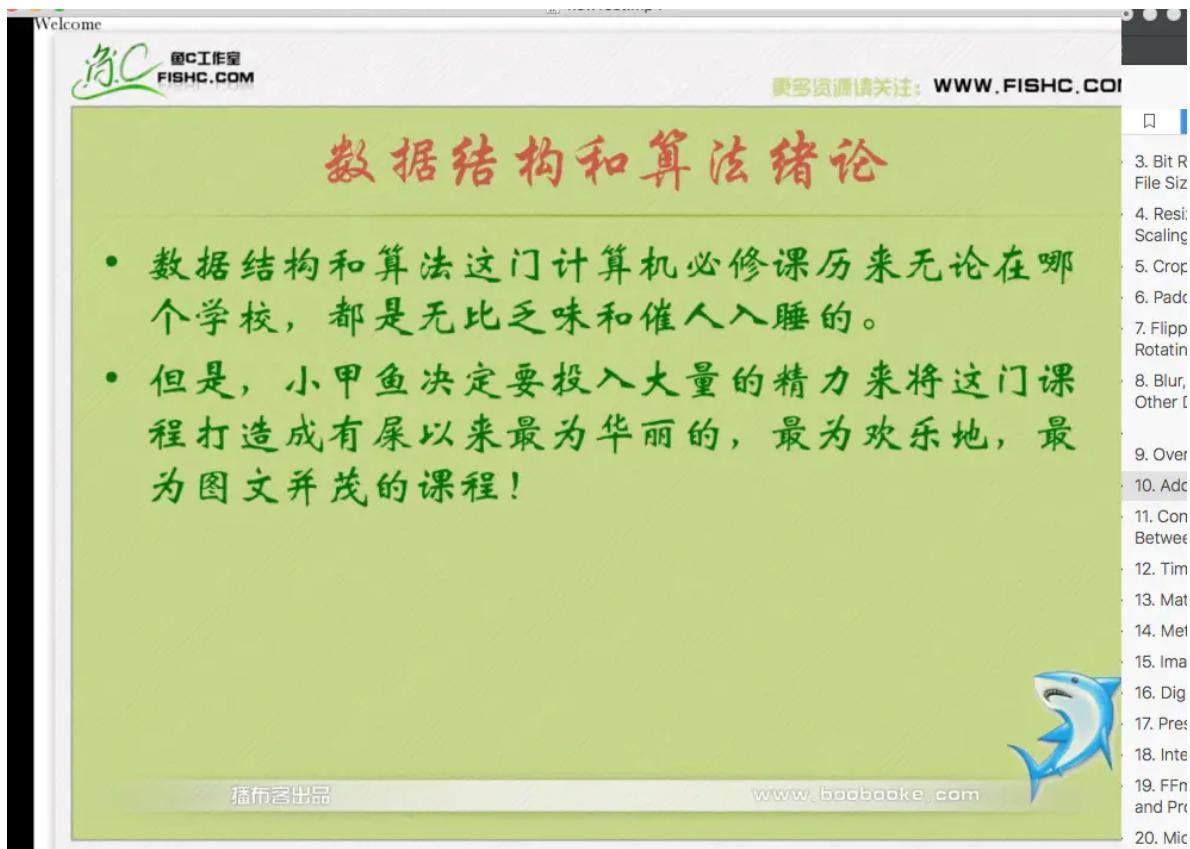
使用ffmpeg的例子，省略“-f lavfi -i颜色=c=白色”，并包含输入和输出文件：

```
ffmpeg -i input -vf drawtext=fontfile=arial.ttf:text=Welcome output
```

我的测试命令如下：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf  
drawtext=fontfile=/Library/Fonts/Baskerville.ttc:text=Welcome  
/Users/zhangfangtao/Desktop/newTest.mp4
```

效果图如下：（在右上角，看到了么，特别小）



右上角，看到了么

为了关注其他参数，下面的示例将指定位于当前目录中的字体文件，您可以根据实际的字体文件位置来包含路径。

## 文本定位

文本位置设置为指定为需要值的x和y参数，这是可以包含变量的数学表达式，以及在上一页中描述的特殊的rand()函数

### 水平位置设置

水平文本放置是通过将x坐标设置为需要值来管理的，例如，从左侧放置文本40像素，我们使用x=40。表达式x=(w-tw)/2将文本定位到中心，其中tw为文本宽度，w为帧宽度。为了将文本对齐到右边，我们使用x=w-tw的表达式。

## 垂直位置设置

y坐标的设置决定了水平文本位置，例如，我们用y=50来定位文本50像素。表达式 $y=(h-th)/2$ 将文本放置到中心，其中th为文本高度，h为帧高度。对于对齐到底部，我们使用表达式x=h

下一个示例文本在一个中心位置的视频帧，请注意，该文本包含空格或制表符必须被引号括起来，有时ffmpeg引擎也需要引用所有drawtext参数，所以所有参数的下一个命令使用双引号和单引号的文本参数（单引号和双引号可以交换，但不能混合）：

```
ffplay -f lavfi -i color=c=white -vf ^ drawtext="fontfile=arial.ttf:text='Good day':x=(w-tw)/2:y=(h-th)/2"
```



我自己的测试命令：

```
ffplay -f lavfi -i color=c=white -vf  
drawtext="fontfile=/Users/zhangfangtao/Desktop/Baskerville.ttc:text='Good day  
, How are you?':x=(w-tw)/2:y=(h-th)/2"
```

效果图：

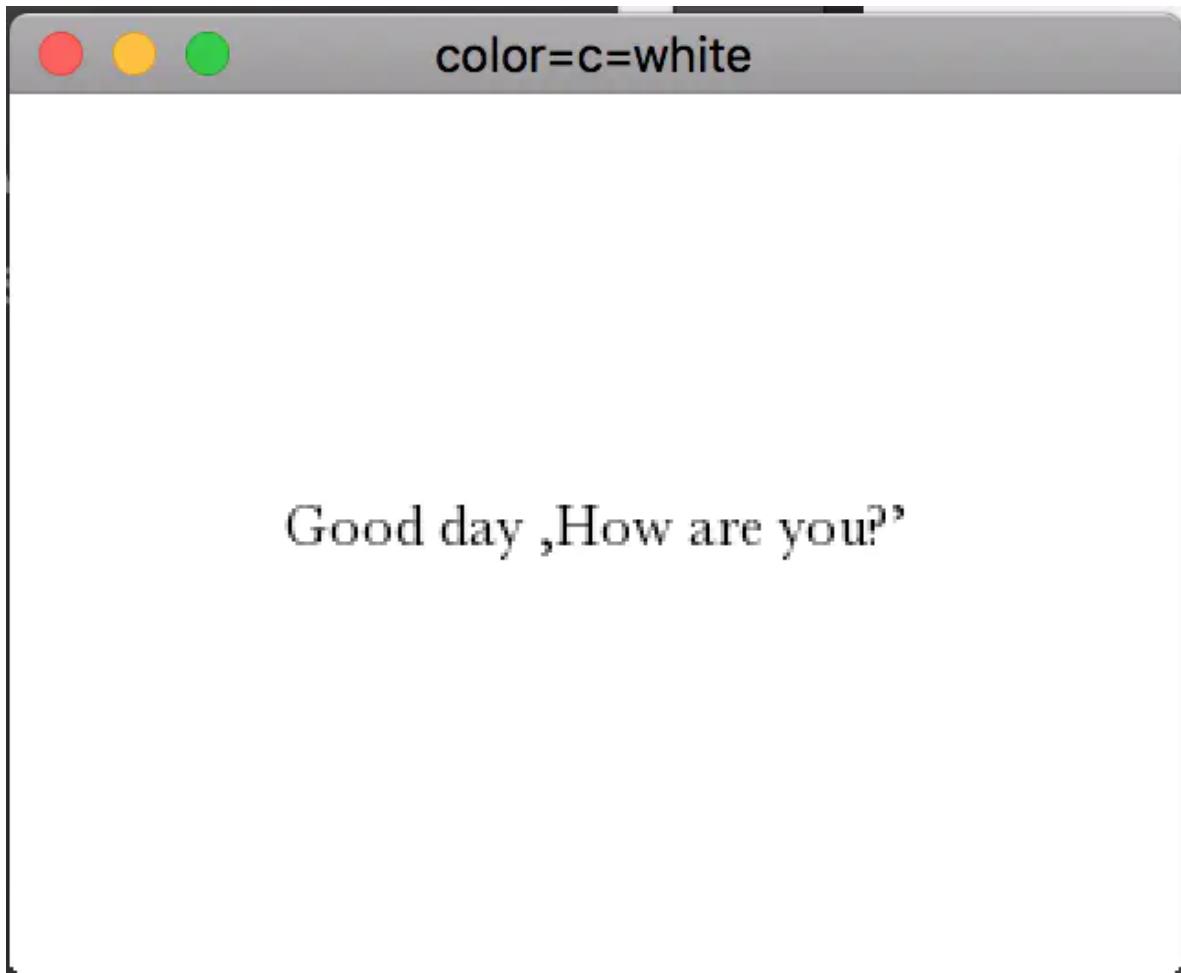
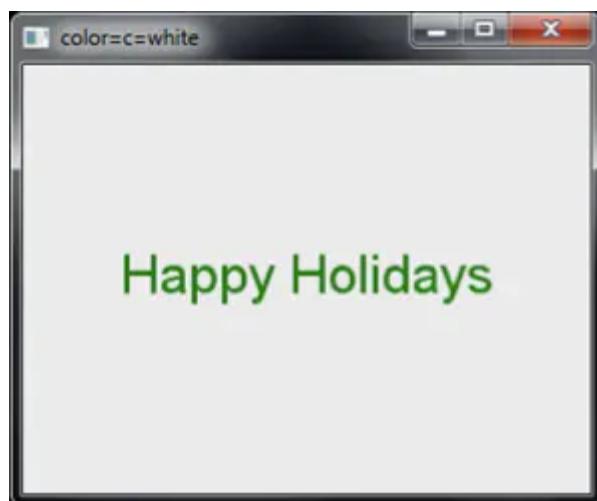


image.png

## 字体大小和颜色设置

为了使文本更醒目、更有趣，使用了比默认16像素更大字体的彩色文本。使用其他参数`fontcolor`和`fontsize`，我们可以修改前面的例子，以30像素字体大小为中心的绿色文本“Happy Holidays”：

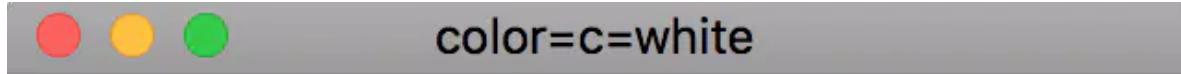
```
ffplay -f lavfi -i color=c=white -vf drawtext=^ "fontfile=arial.ttf:text='Happy Holidays':x=(w-tw)/2:y=(h-th)/2:^ fontcolor=green:fontsize=30"
```



我自己的测试命令：

```
ffplay -f lavfi -i color=c=white -vf  
drawtext="fontfile=/Users/zhangfangtao/Desktop/Baskerville.ttc:text='Happy  
Holidys':x=(w-tw)/2:y=(h-th)/2:fontcolor=green:fontsize=30"
```

显示的效果：



# Happy Holidys

---

为了改变背景色，从绿白到黄蓝，我们用蓝色的值替换白色，用黄色的值代替绿色，字体大小的值增加到40：

```
ffplay -f lavfi -i color=c=blue -vf drawtext=^ "fontfile=arial.ttf:text='Happy  
Holidays':x=(w-tw)/2:y=(h-th)/2:^ fontcolor=yellow:fontsize=40"
```

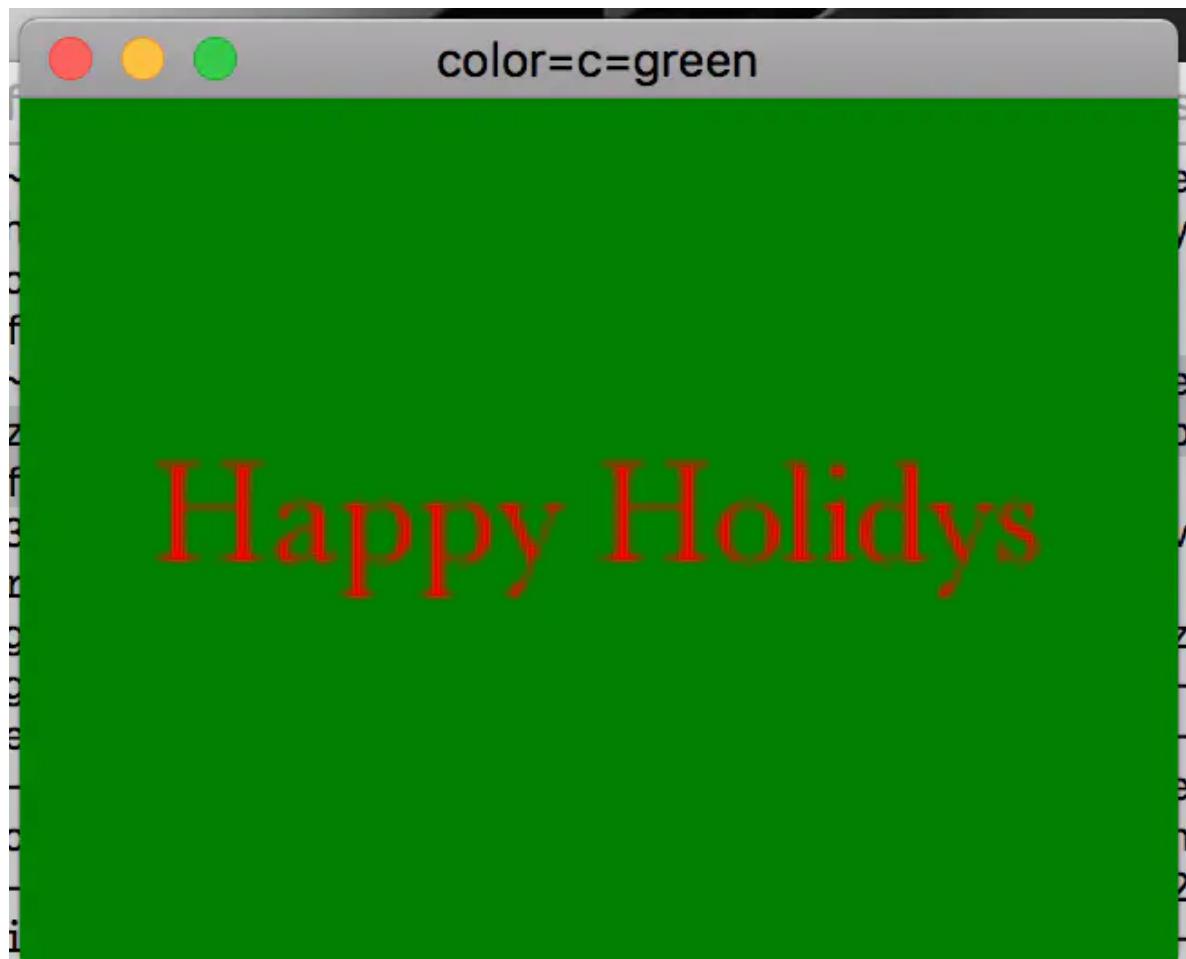
颜色也可以用HTML格式指定，例如红色=#ff0000，绿色=#00ff00，等等。



我的测试命令:

```
ffplay -f lavfi -i color=c=green -vf  
drawtext="fontfile=/Users/zhangfangtao/Desktop/Baskerville.ttc:text='Happy  
Holidys':x=(w-tw)/2:y=(h-th)/2:fontcolor=red:fontsize=40"
```

显示的效果:



## 动态文本

表示时间的t(时间)变量可以根据当前时间改变x和y值。

### 文字水平运动

要将文本横向移动到视频帧中，我们将t变量包括到x参数的表达式中，例如，在右向左方向以n个像素每一秒移动提供的文本，我们使用 $x=w-tn$ 的表达式。为了将运动改变为从左到右的方向，使用 $x=w+tn$ 的表达式。例如，要显示“动态RTL文本”字符串在顶部移动，我们使用命令。

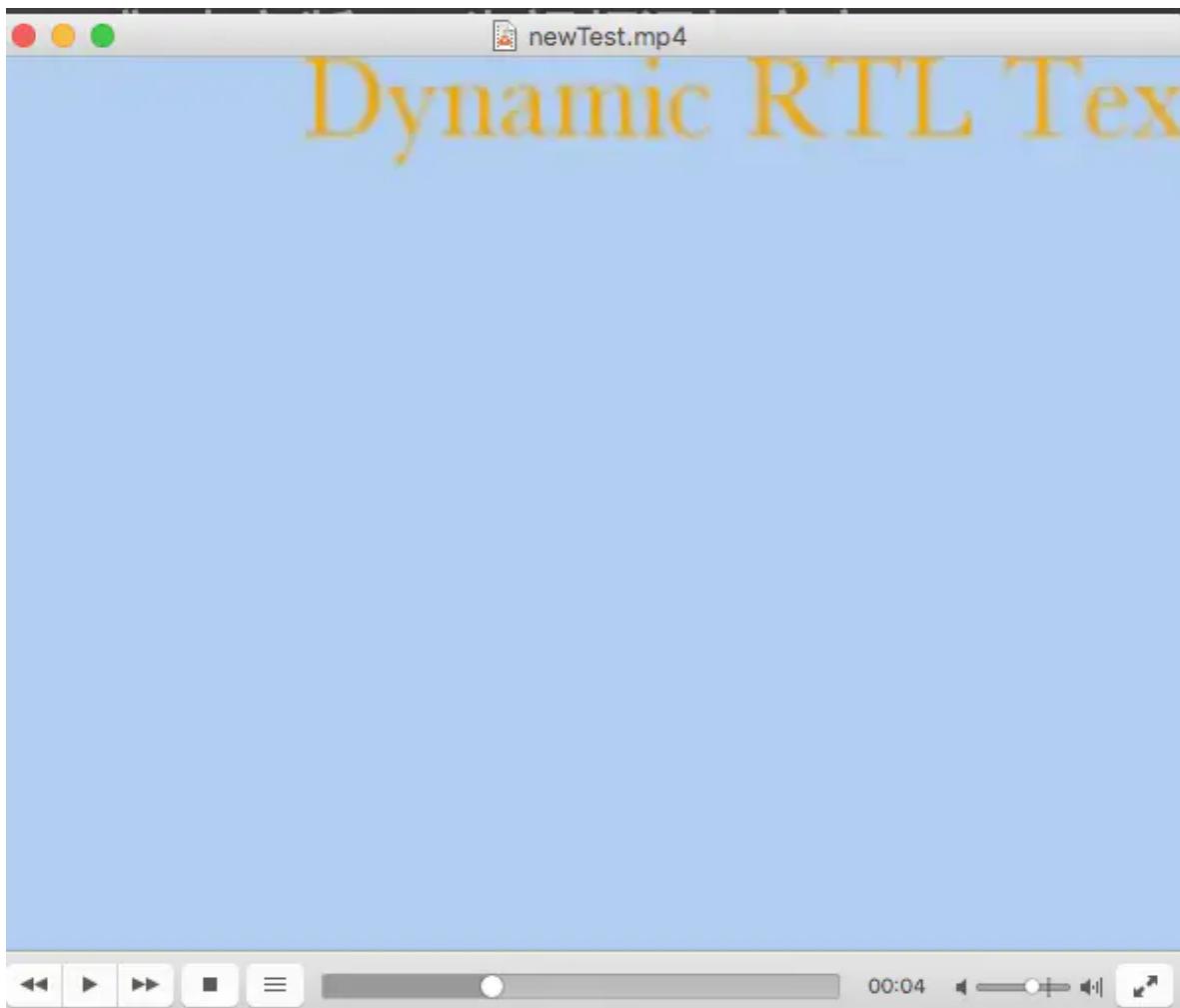
```
ffmpeg -f lavfi -i color=c=#abcdef -vf drawtext=^
"fontfile=arial.ttf:text='Dynamic RTL text':x=w-t*50:^
fontcolor=darkorange:fontsize=30" output
```



我的测试命令：

```
ffmpeg -f lavfi -i color=#abcdef -vf
drawtext="fontfile=/Users/zhangfangtao/Desktop/Baskerville.ttc:text='Dynamic RTL
Text':x=w-t*50:fontcolor=orange:fontsize=30" -t 15
/Users/zhangfangtao/Desktop/newTest.mp4
```

效果图如下：

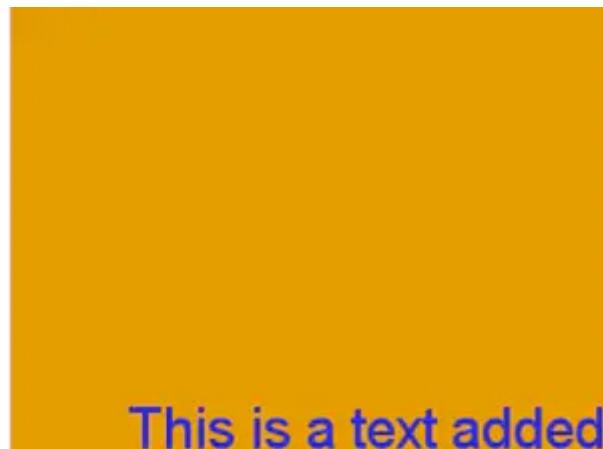


更常见的用法是在底部附近滚动一行文本，文本位于文本文件中，该文本文件只包含一条非常长的行。下一个示例使用带有值info.txt的textfile参数。

```
ffmpeg -f lavfi -i color=c=orange -vf drawtext="fontfile=arial.ttf:^  
textfile=info.txt:x=w-t*50:y=h-th:fontcolor=blue:fontsize=30" output
```

信息的内容。txt文件如下，下面是在t=5时滚动的图片。

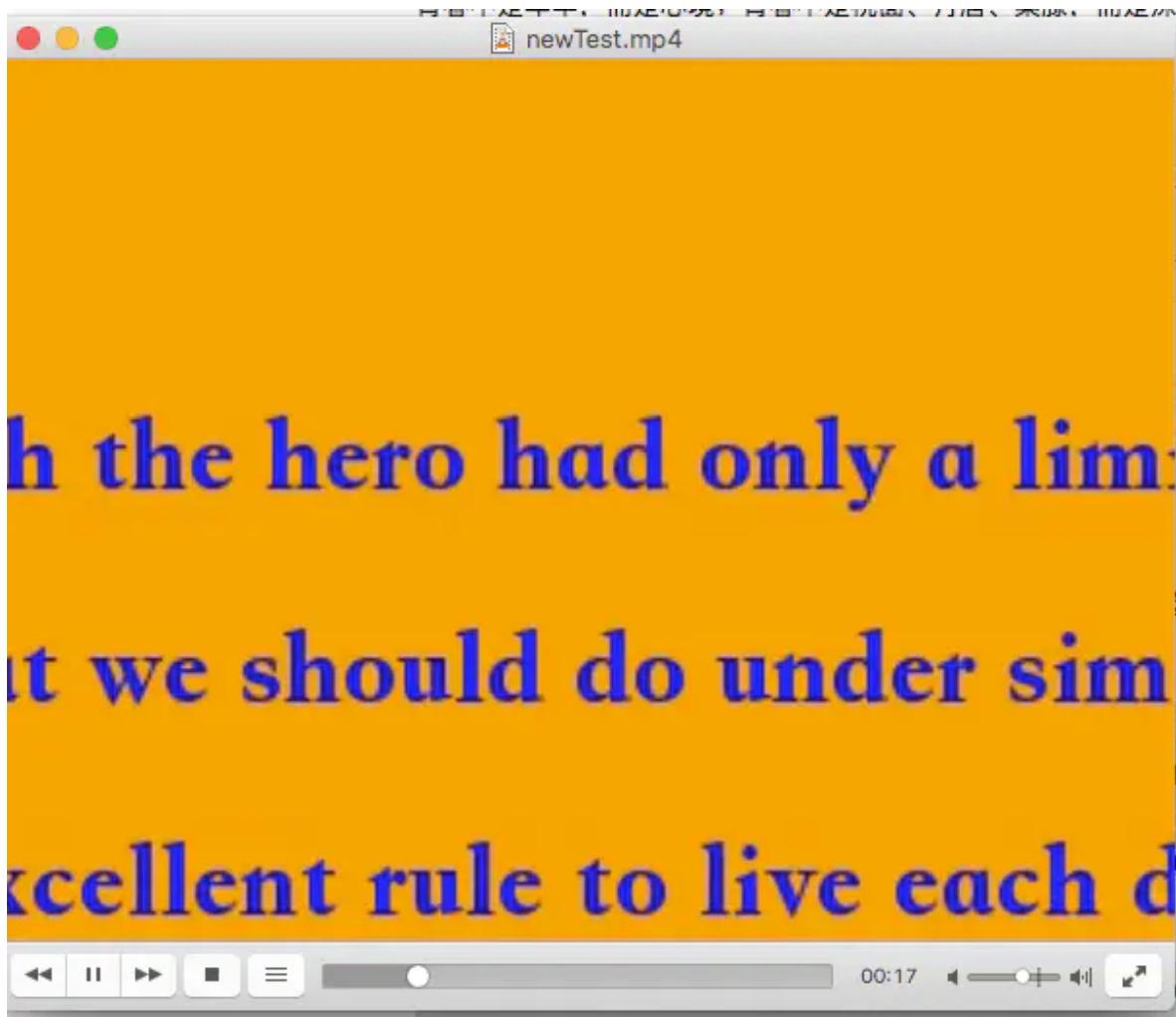
这是一个添加到橙色背景的文本，使用带有30像素字体大小的Arial字体的drawtext过滤器。



我的测试命令：

```
ffmpeg -f lavfi -i color=orange -vf  
drawtext="fontfile=/Users/zhangfangtao/Desktop/Songti.ttc:textfile=/Users/zhang  
angtao/Desktop/test.rtf:x=w-t*50:y=h-th:fontcolor=blue:fontsize=30" -t 50  
/Users/zhangfangtao/Desktop/newTest.mp4
```

显示的效果：



### 垂直文本运动

文本垂直滚动从底部到顶部是常用的视频显示生产商的名称,演员,日期,等。文本垂直移动,t变量包含y参数的表达式,例如移动提供的文本每一秒xn像素从上到下,我们使用一个表达式 $y = t * n$ 。从下到上滚动,使用 $y=h-t*n$ 。下一个命令显示信用文件的内容,从底部向上移动,速度为每秒100像素。信用档案的内容,包括开始的空格处是:

```
Production: FFmpeg User  
Date: December 2012  
Filter: drawtext
```

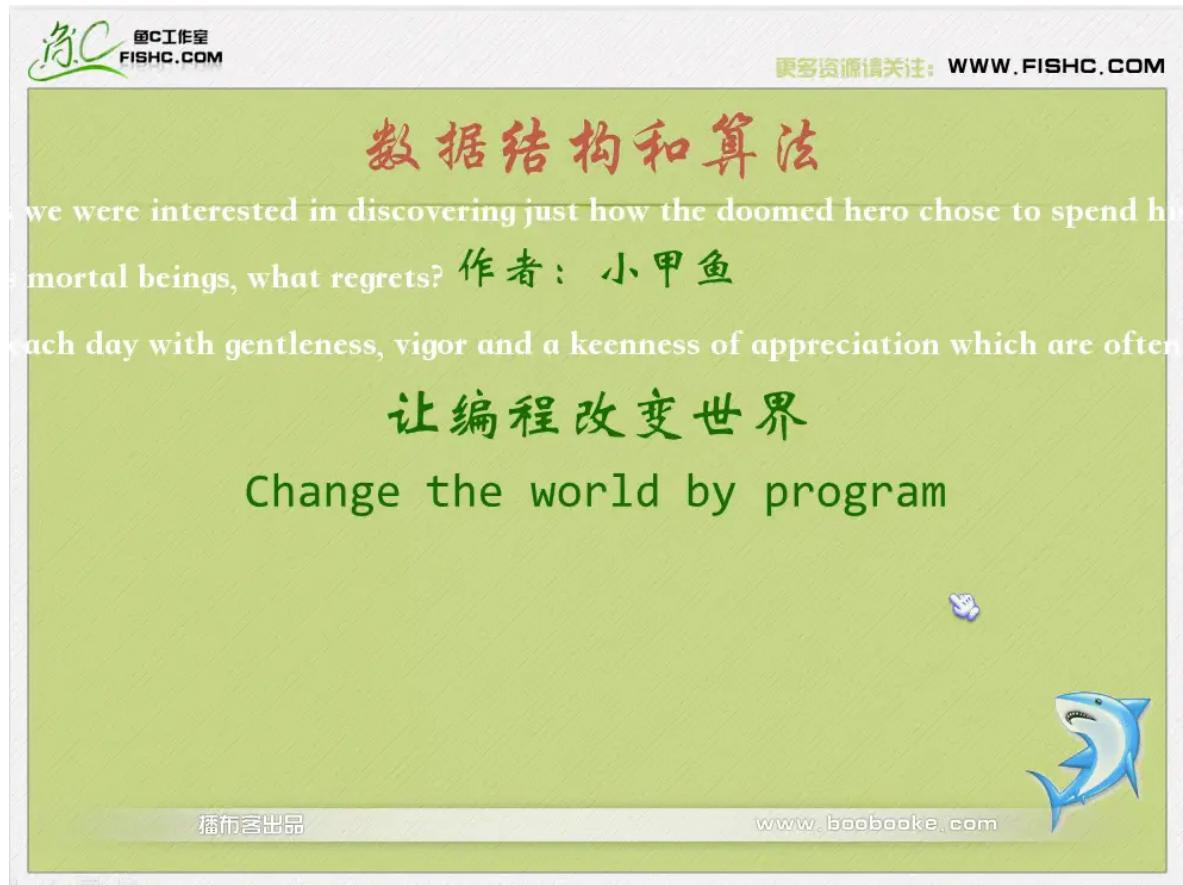
```
ffmpeg -i palms.avi -vf drawtext="fontfile=arial.ttf:textfile=Credits:^ x=(w-  
tw)/2:y=h-t*100:fontcolor=white:fontsize=30" clip.mp4
```



我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf  
drawtext="fontfile=/Users/zhangfangtao/Desktop/Songti.ttc:textfile=/Users/zhang  
angtao/Desktop/test.txt:x=(w-tw)/2:y=h-t*100:fontcolor=white:fontsize=30" -t 100  
/Users/zhangfangtao/Desktop/newTest.mp4
```

效果图：（从下往上滑动）



## 11-格式之间转换

ffmpeg工具的最常见用法是从一种音频或视频格式转换为另一种相关的格式。格式参数在输出文件之前由-f选项设置，或者在输入文件之前也有原始输入，具体的可用格式信息在[显示帮助和功能](#)一章中列出来了。

## 多媒体格式介绍

### 文件格式

媒体格式是能够存储音频或视频数据的特殊文件类型。其中一些能够存储更多类型的数据与多个流，这些被称为容器。[显示帮助和功能](#)列出了可用的媒体格式，并可以使用命令 `ffmpeg -formats` 进行显示。

视频文件格式通常可以同时包含视频和音频流，但是有一些特殊的格式，只能包含音频，详细信息在[数字音频](#)章节中有描述。

### 多媒体容器

媒体容器是特定类型的包装文件，用于存储多媒体流和相关元数据的特殊文件格式。由于音频和视频可以通过各种方法（算法）进行编码和解码，容器提供了将各种媒体流存储在一个文件中的简单方法。一些容器只能存储音频（AIFF, WAV, XMF等），一些只能存储图片（TIFF ...），但大多数容器存储音频，视频，字幕，元数据等。所有列出的视频容器也支持一些字幕格式，特别是SubRip和Advanced SubStation Alpha。

Characteristics of common media containers												
Container	Support for particular file format											
	Audio					Video						
	AAC	AC-3	MP3	PCM	WMA	MPEG1 MPEG2	MPEG4	H.264/ MPEG4 AVC	VC1 WMV	Theora		
AVI	Y	Y	Y	Y	Y	Y	Y	partially	Y	Y		
Matroska	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		
MP4	Y	Y	Y	N	Y	Y	Y	Y	Y	Y		N
MXF	Y	Y	Y	Y	N	Y	Y	Y	Y	Y		N
Ogg/OGM	N	N	Y	Y	N	Y	Y	Y	Y	Y		Y
QuickTime	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y

### 媒体容器

如果只更改容器并保留编解码器，我们可以使用 `-c copy` 或 `-c:a copy` 或 `-c:v copy` 选项：

```
ffmpeg -i input.avi -q 1 -c copy output.mov
```

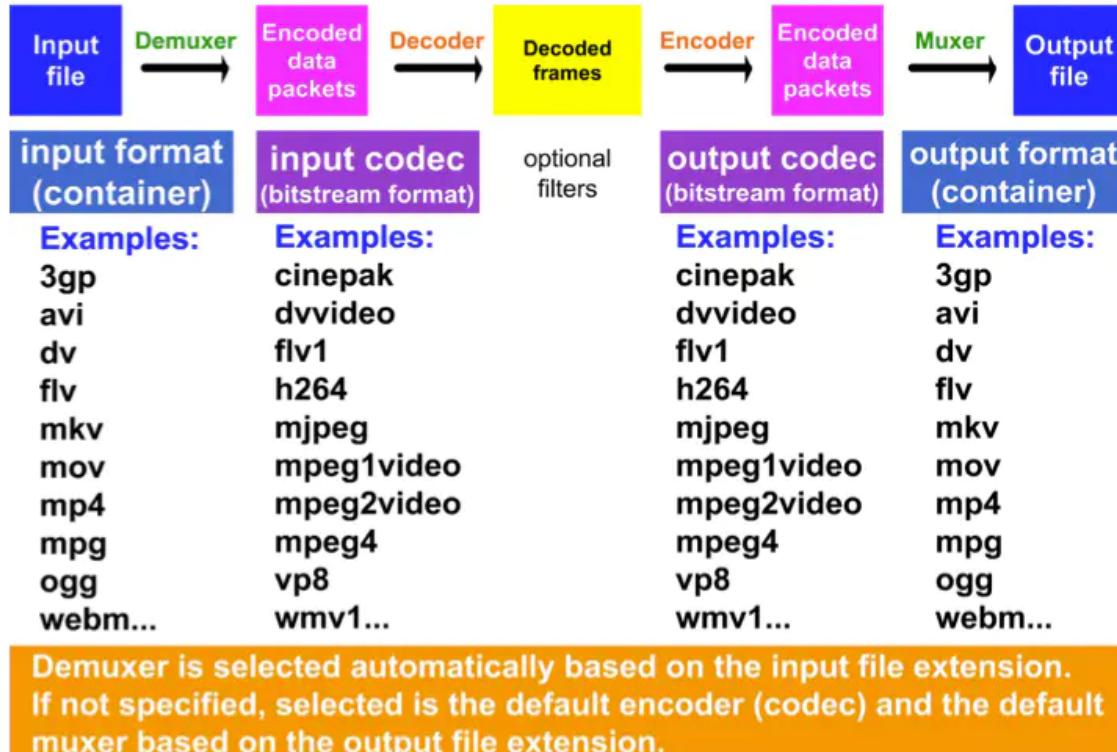
## 转码和转换

将输入文件使用ffmpeg处理成输出文件称为转换，它可以包括格式之间的转换或者仅修改某些数据，输出媒体格式保持不变的转码。数据包可以被编码压缩或解压缩，压缩包括使用特定的编解码器。转码过程可以分为几个部分：

\*解复用 (demultiplexing) - 基于文件扩展名 (.avi, .mpg等) 被选中来自libavformat库的最好的解复用 (解复用器)，从输入文件生成编码数据包

- 解码-数据包是由一个适当的解码器解码，产生未压缩的帧;如果使用 `-c copy` (或 `-codec copy`) 选项，则不会发生解码(也不进行过滤)。
- 可选的过滤器 - 解码的帧可以通过指定的过滤器进行修改
- 编码 - 未压缩的帧由选定的编码器编码为数据包
- 复用 (multiplexing) - 将数据包复用 (multiplexed) 为选定的媒体格式。

## Media formats and codecs in conversion



作者喜欢这种花花绿绿的图，，我也没办法

ffmpeg中转换的可用选项被划分为通用的和私有的。可以为任何容器、编解码器或设备设置通用选项，私有选项针对所选的编解码器、容器或设备。

## 编解码器介绍

codec的名字来源于单词编码解码器(或编码解码器)，它表示一个设备或软件工具，用于编码和解码一个被压缩的视频或音频流。FFmpeg编解码器定义是一种媒体比特流格式。下一个命令显示可用的编解码器：

- `ffmpeg -codecs` ...显示的都是解码器和编码器
- `ffmpeg -decoders` ...只显示解码器
- `ffmpeg -encoders` ...只显示编码器

命令行上的编解码器是由 `-c` 或 `-codec` 选项指定的，语法是：

```
-codec[:streamSpecifier] codecName
```

可以为输入和输出文件指定编解码器，如果输出包含多个流，则每个流可以使用不同的编解码器。如果我们在没有编解码器的情况下指定输出格式，则ffmpeg会选择默认编解码器，常见媒体格式的默认编解码器列表如下：

<b>Default codecs (encoders) for common video file extensions (file formats)</b>
--

格式	编解码器	其他数据
.avi	mpeg4	mpeg4 (Simple profile), yuv420p; audio: mp3
.flv	flv1	yuv420p; audio: mp3
.mkv	h264	h264 (High), yuvj420p; audio: vorbis codec, fltp sample format
.mov	h264	h264 (High), yuvj420p; audio: aac (mp4a)
.mp4	h264	h264 (High), yuvj420p; audio: aac (mp4a)
.mpg	mpeg1video	yuv420p; audio: mp2
.ogg	theora	yuv422p, bitrate very low; audio excluded during conversion
.ts	mpeg2video	yuv422p; audio: mp2
.webm	vp8	yuv420p; audio: vorbis codec, fltp sample format

### Default codecs (encoders) for common audio file extensions (file formats)

常用音频文件扩展(文件格式)的默认编解码器(编码器)

格式	编解码器	额外数据
.aac	aac	libvo_aacenc, bitrate 128 kb/s
.flac	flac	FLAC (Free Lossless Audio Codec), bitrate 128 kb/s
.m4a	aac	mp4a, bitrate 128 kb/s
.mp2	mp2	MPEG Audio Layer 2, bitrate 128 kb/s
.mp3	mp3	libmp3lame, bitrate 128 kb/s
.wav	pcm_s16le	PCM (Pulse Code Modulation), uncompressed
.wma	wmav2	Windows Media Audio

## 覆盖相同的命名输出文件

如果在ffmpeg命令中具有指定名称的文件已经存在，控制台将请求“y”(yes)或“n”(no)来覆盖旧文件。为了避免这个问题，可以使用-n选项来取消处理， -y选项用于设置覆盖而不需要请求。例如，在默认情况下，我们可以使用以下命令来覆盖旧的输出文件：

```
ffmpeg -y -i input.avi output.mp4
```

## 通用选项转换

通用选项可设置为任何编解码器、容器或设备。在表中描述了与编码器(codec)规范相关的转换中包含的最常见的通用选项，目标列包含5个字母代码，限制了特定选项的使用。某些字母的存在意味着该选项适用于编码(E)、解码(D)、视频(V)、音频(A)或字幕(S)。

选项	类型	标签	描述
-flags	flags	EDVAS	数值: ( <a href="#">ffmpeg帮助</a> 的详细信息):mv4、qpel、loop、gmc、mv0、gray、psnr、naq、ildct、low_delay、global_header、bitreal、aic、cbp、qprd、ilme、cgop
-me_method	int	E.V..	数值: (请参阅 <a href="#">ffmpeg帮助</a> 详细信息):0、full、epzs、esa、tesa、dia、log、phods、x1、hex、umh、iter
-g	int	E.V..	设置图片组大小
-qcomp	float	E.V..	视频量化压缩(VBR)。ratecontrol方程的常数。 默认rc_eq: 0-1.0的推荐范围
-qblur	float	E.V..	视频量化器尺度模糊(VBR)
-qmin	int	E.V..	最小视频量化器规模(VBR)
-qmax	int	E.V..	最大视频数字转换器规模(VBR)
-qdiff	int	E.V..	量化器的最大差异(VBR)
-bf	int	E.V..	使用“帧”B帧
-b_qfactor	float	E.V..	p和b之间的qp因子
-rc_strategy	int	E.V..	ratecontrol方法
-b_strategy	int	E.V..	在I/P/ b帧之间选择策略
-ps	int	E.V..	rtp负载大小以字节为单位
-lelim	int	E.V..	亮度的单系数消除阈值(负值也考虑直流系数)
-celim	int	E.V..	色度的单系数消除阈值(负值也考虑直流系数)
-strict	int	ED.VA.	如何严格遵循标准, 值为(请参阅 <a href="#">ffmpeg帮助</a> 细节):very, strict, normal, unofficial, experimental
-b_qoffset	float	E.V..	qp偏移在P和B帧之间
-err_detect	flags	.D.VA.	设置错误检测标志, 值为(请参阅 <a href="#">ffmpeg帮助</a> 细节):crccheck, bitstream, buffer, explode, careful, compliant, aggressive
-mpeg_quant	int	E.V..	使用MPEG量化器代替H.263
-qsquish	float	E.V..	如何在qmin - qmax (0=clip, 1=使用可微函数)之间保持量化器
-rc_qmod_amp	float	E.V..	实验量化器调制
-rc_qmod_freq	int	E.V..	实验量化器调制

选项	类型	标签	描述
-rc_eq	string	E.V..	设置速度控制方程。在计算表达式时，除了在 <a href="#">数学函数</a> 章节中定义的标准函数外，还有以下函数:bits2qp(bits), qp2bits(qp)。也可以使用以下常数:iTex pTex texmv fCode iCount mcVar iount isB avgQP qComp avgpiex avgPPTex avgTex
-i_qfactor	float	E.V..	P和I之间的qp因子
-i_qoffset	float	E.V..	qp偏移在P和I帧之间
-rc_init_cplx	float	E.V..	1-pass编码的初始复杂度
-dct	int	E.V..	DCT算法，值为(请参阅 <a href="#">ffmpeg帮助</a> 细节):auto, fastint, int, mmx, altivec, faan
-lumi_mask	float	E.V..	压缩明亮的区域比中等的区域强
-tcplx_mask	float	E.V..	时间复杂性掩盖
-scplx_mask	float	E.V..	空间复杂性掩盖
-p_mask	float	E.V..	inter屏蔽
-dark_mask	float	E.V..	压缩暗区比中型区域强
-idct	int	ED.V..	选择IDCT实现，值为(请参阅 <a href="#">ffmpeg帮助</a> 细节):auto, int, simple, simplemmx, libmpeg2mmx, mmi, arm, altivec, sh4, simplearm, simplearmv5te, simplearmv6, simpleneon, simplealpha, h264, vp3, ipp, xvidmmx, faani
-ec	flags	.D.V..	设置错误隐藏策略，值:guess_mvs(迭代运动矢量(MV)搜索(慢)), deblock (.D.V. 使用强大的deblock过滤器, 对损坏的MBs)
-pred	int	E.V..	预测方法，值是(更多的见 <a href="#">ffmpeg帮助</a> ):left, plane, median
-vismv	int	.D.V..	可视化运动矢量(MVs)，值是(更多的见 <a href="#">ffmpeg帮助</a> ):pf, bf, bb
-cmp	int	E.V..	完整的pel我比较函数，值(更多的见 <a href="#">ffmpeg帮助</a> 章节):sad, sse, satd, dct, psnr, bit, rd, 0, vsad, vsse, nsse, w53, w97, dctmax, chroma
-subcmp	int	E.V..	比较函数，值和-cmp选项(更多的见 <a href="#">ffmpeg帮助</a> 章节)
-mbcmp	int	E.V..	macroblock比较函数，值与-cmp选项(更多的是 <a href="#">ffmpeg帮助</a> 里面)
-ildctcmp	int	E.V..	交错dct比较函数，值与-cmp选项(更多在 <a href="#">ffmpeg帮助</a> 章节里面)

选项	类型	标签	描述
-dia_size	int	E.V..	运动估计的diamond 类型和尺寸
-last_pred	int	E.V..	从上一帧的运动预测器数量
-preme	int	E.V..	前运动估计
-precmp	int	E.V..	前运动估计比较函数, 值与-cmp选项(更多的见 <a href="#">ffmpeg帮助</a> )
-pre_dia_size	int	E.V..	运动预估的diamond类型和尺寸
-subq	int	E.V..	子波运动估计质量
-me_range	int	E.V..	极限运动矢量范围(DivX播放器的1023)
-ibias	int	E.V..	内部定量偏差
-pbias	int	E.V..	inter定量偏差
-coder	int	E.V..	值:vlc(可变长度/huffman编码器), ac(算术), raw(无编码), rle(运行长度), deflate (deflate-based)
-context	int	E.V..	上下文模型
-mbd	int	E.V..	macroblock决策算法(高质量模式), 值为(请参阅 <a href="#">ffmpeg帮助</a> 细节):simple, bits, rd
-sc_threshold	int	E.V..	场景变化阈值
-lmin	int	E.V..	最小拉格朗日因子(VBR)
-lmax	int	E.V..	最大拉格朗日因子(VBR)
-flags2	flags	ED.VA	值(更多的见 <a href="#">ffmpeg帮助</a> ):快速、sgop、noout、local_header、块、showall、skiprd
-threads	int	ED.V..	自动数值(检测大量线程)
-dc	int	E.V..	intra_dc_precision (不知道怎么翻译)
-nssew	int	E.V..	nsse weight
-skip_top	int	.D.V..	在顶部跳过的macroblock行数
-skip_bottom	int	.D.V..	在底部跳过的macroblock行数
-profile	int	E.VA.	值(更多的键 <a href="#">ffmpeg帮助</a> ):未知的, aac_main, aac_low, aac_ssr, aac_ltp, aac_he, aac_he_v2, aac_id, aac_ld, dts, dts_es, dts 96 24, dts_hd_hra, dts_hd_ma
-level	int	E.VA.	数值: 未知的
-lowres	int	.D.VA.	解码1=1/2,2=1/4,3=1/8
-skip_factor	int	E.V..	帧跳跃因素

选项	类型	标签	描述
-skip_exp	int	E.V..	帧跳跃指数
-skipcmp	int	E.V..	帧跳过比较函数，与-cmp选项相同的值(更多的信息见帮助)
-border_mask	float	E.V..	增加接近边界的宏块的量化器
-mblmin	int	E.V..	min macroblock拉格朗日因子(VBR)
-mblmax	int	E.V..	max macroblock拉格朗日因子(VBR)
-mepc	int	E.V..	运动估计比特率惩罚补偿(1.0 = 256)
-skip_loop_filter	int	.D.V..	值(更多的是 <a href="#">ffmpeg帮助</a> ):none、default、noref、bidir、nokey、all
-skip_idct	int	.D.V..	和-skip_loop里面参数的值是一样的(更多内容见 <a href="#">ffmpeg帮助</a> )
-skip_frame	int	.D.V..	和-skip_loop里面参数的值是一样的(更多内容见 <a href="#">ffmpeg帮助</a> )
-bidir_refine	int	E.V..	细化双向宏块中使用的两个运动矢量
-brd_scale	int	E.V..	用于动态b帧决策的下尺度框架
-keyint_min	int	E.V..	最小间隔IDR-frames
-refs	int	E.V..	考虑运动补偿的参考系
-chromaoffset	int	E.V..	色度qp从luma偏移
-trellis	int	E.VA.	率失真优化量化
-sc_factor	int	E.V..	每一帧乘以qscale，并添加到scene_change_score
-b_sensitivity	int	E.V..	调整b_frame_strategy 1的灵敏度
-colorspace	int	ED.V..	名字的颜色空间
-slices	int	E.V..	片数，用于并行编码
-thread_type	flags	ED.V..	选择多线程类型，值:slice, frame
-rc_init_occupancy	int	E.V..	在解码开始之前，要加载到rc缓冲区的位数
-me_threshold	int	E.V..	运动估计阈值
-mb_threshold	int	E.V..	macroblock阈值
-skip_threshold	int	E.V..	帧跳过阈值
-timecode_frame_start	int64	E.V..	GOP timecode框架启动号码，在非下降帧格式
-request_channels	int	.D..A.	设置所需的音频通道数
-channel_layout	int64	ED..A.	可用值:ffmpeg布局

选项	类型	标签	描述
-audio_service_type	int	E...A.	音频服务类型, 值(更多的见 <a href="#">ffmpeg帮助</a> ):ma, ef, vi, hi, di, co, em, vo, ka
-request_sample_fmt	s_fmt	.D..A.	示例格式音频解码器应使用(列表:ffmpeg -sample_fmts)

有关如何使用某些选项的示例, 请参阅“[预设编解码器](#)”一章。为了保持输出质量相同, 我们使用 `-q` 或 `-qscale [: stream_specifier]` 选项来设置固定的质量范围, 通常从1到31, 其中值1表示最高质量(某些编解码器使用其他比例)。

## 私有的的选择转选项

虽然可以为任何编解码器, 容器或设备设置通用选项, 但私有选项是可以仅为选定的编解码器, 容器或设备指定的附加选项。

### MPEG-1视频编码器

除了通用选项之外, `mpeg1video` 编码器可以使用表中描述的私有选项:

option	type	description
--------	------	-------------

选项	类型	描述
-gop_timecode	string	MPEG GOP Timecode在hh:mm:ss[::]ff格式
-intra_vlc	int	使用MPEG-2 intra VLC表
-drop_frame_timecode	int	Timecode采用了drop - frame格式
-scan_offset	int	预留空间用于SVCD扫描偏移用户数据
-mpv_flags	flags	所有基于mpegvideo的编码器通用的标志, 值是(更多的 <a href="#">ffmpeg帮助</a> ):skip_rd, strict_gop, qp_rd, cbp_rd
-luma_elim_threshold	int	亮度的单系数消除阈值(负值也考虑直流系数)
-chroma_elim_threshold	int	色度的单系数消除阈值(负值也考虑直流系数)
-quantizer_noise_shaping	int	没有描述

### MPEG-2 视频编码器

`mpeg2video` 编码器可以使用 `mpeg1` 视频编码器的所有选项和2个附加选项:

option	type	Description
--------	------	-------------

选项	类型	描述
-non_linear_quant	int	使用非线性量化器
-alternate_scan	int	启用备用表扫描

## MPEG-4 视频编码器

mpeg4编码器包括前面2个表中描述的下一个选项:

- data\_partitioning
- alternate\_scan
- mpv\_flags
- luma\_elim\_threshold
- chroma\_elim\_threshold
- quantizer\_noise\_shaping

## libvpx视频编码器

libvpx编码器以WEBM格式为例，包括下一个选项:

option	type	description
选项	类型	描述
-cpu-used	int	质量/速度比修饰符
-auto-alt-ref	int	启用备用参考框架(仅2-pass)
-lag-in-frames	int	帧的数量，以展望备用参考帧的选择
-arnr-maxframes	int	altref降噪最大帧数
-arnr-strength	int	altref降噪滤波强度
-arnr-type	int	altref降噪滤波类型，值为:向后、正向、中心
-deadline	int	花在编码上的时间，在微秒内，值是:最好的，好的，实时的
-error-resilient	flags	错误恢复配置:值(更多在 <a href="#">ffmpeg帮助</a> 下):默认值，分区
-max-intra-rate	int	最大i帧比特率(pct) 0=无限
-speed	int	没有描述
-quality	int	价值是:最好的，好的，实时的
-vp8flags	flags	值是(更多的 <a href="#">ffmpeg帮助</a> ):error_， altref
-arnr_max_frames	int	altref降噪最大帧数
-arnr_strength	int	altref降噪滤波强度
-arnr_type	int	altref降噪滤波器类型
-rc_lookahead	int	帧的数量，以展望备用参考帧的选择
-crf	int	选择质量不变的质量模式

## AC-3音频编码器

ac3音频编码器可以使用表中描述的其他选项:

Audio codec: AC-3		
选项	类型	描述
per_frame_metadata	integer	允许改变元数据每帧
center_mixlev	float	中心组合水平
surround_mixlev	float	围绕混合水平
mixing_level	integer	混合水平
room_type	integer	房间类型, 值是:不指示, 大, 小
copyright	integer	版权位
dialnorm	integer	对话水平(dB)
dsur_mode	integer	杜比环绕模式, 值:不显示, 打开, 关闭
original	integer	原始比特流
dmix_mode	integer	首选立体声调合模式, 值:不显示
ltrt_cmixlev	float	lt / rt中心混合水平
ltrt_surmixlev	float	lt / rt环绕混合水平
lro_cmixlev	float	lo / ro中心混合水平
lro_surmixlev	float	lo / ro环绕混合水平
dsurex_mode	integer	Dolby环绕EX模式, 值:不显示, on, off
dheadphone_mode	integer	杜比耳机模式, 值:不显示, 打开, 关闭
ad_conv_type	integer	A/D转换器类型, 值:标准(默认), hdcd
stereo_rematrixing	integer	立体重映射
channel_coupling	integer	通道耦合 值: 自动
cpl_start_band	integer	耦合开始band 值: 自动

## 简化的VCD、SVCD、DVD、DV、DV50编码

一个特殊的目标选项只允许使用一个选项, 而不是特定媒体类型所需要的大量选项(VCD=视频CD、SVCD=超级视频CD、DV=数字视频等), 可用的值为:

- vcd, pal-vcd, ntsc-vcd, film-vcd
- svcd, pal-svcd, ntsc-svcd, film-svcd
- dvd, pal-dvd, ntsc-dvd, film-dvd
- dv, pal-dv, ntsc-dv, film-dv

- dv50, pal-dv50, ntsc-dv50, film-dv50

所有需要的参数，如帧率、纵横比、比特率等都是根据特定媒体格式的规格设置的。例如，为DVD的视频编码我们可以使用命令：

```
ffmpeg -i input.avi -target dvd output.mpg
```

## 12-时间操作

多媒体处理包括改变输入持续时间，设置延迟，仅从输入中选择特定部分等。这些时间操作接受2种格式的时间规格：

- [-]HH:MM:SS[.m...]
- [-]S+[.m...]

HH是小时数，MM是分钟数，SS或S是秒数，m是毫秒数。

### 音频和视频的持续时间

#### 使用-t选项设置

要设置媒体文件的持续时间，我们可以使用-t选项，其值是以秒为单位的时间或格式为HH:MM:SS.milliseconds的时间。例如，要为music.mp3文件设置3分钟的持续时间，我们可以使用该命令

```
ffmpeg -i music.mp3 -t 180 music_3_minutes.mp3
```

- 上面这个命令我就不测试了，应该没啥问题

#### 通过帧数设置

在某些情况下，通过指定具有可用选项的帧数来设置录制的持续时间可能很有用：

- 音频: -aframes number 或者 -frames:a number
- 数据: -dframes number 或者 -frames:d number
- 视频: -vframes number 或者 -frames:v number

帧数等于以秒为单位的持续时间乘以帧速率。例如，要将25帧/秒的video.avi文件的持续时间设置为10分钟（600秒），我们可以使用以下命令：

```
ffmpeg -i video.avi -vframes 15000 video_10_minutes.avi
```

我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vframes 1500 /Users/zhangfangtao/Desktop/newtest.mp4
```

### 从开始设置延迟

要从指定时间开始记录输入，我们可以使用 `-ss`（从开始搜索）选项，其值是以秒或 `HH: MM: SS.milliseconds` 格式表示的时间。该选项既可以在输入文件和输出文件之前使用，也可以在输出文件之前使用，编码更精确。例如，要从第10秒开始转换，我们可以使用以下命令：

```
ffmpeg -i input.avi -ss 10 output.mp4
```

我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -ss 10  
/Users/zhangfangtao/Desktop/test2.mp4
```

## 从媒体文件中提取特定部分。

要从音频或视频文件中剪辑特定部分，我们同时使用 `-ss` 和 `-t` 选项，`ffplay` 在左下角显示当前时间，可以使用空格键或P键暂停/开启播放。例如，要从文件 `video.mpg` 保存第5分钟（ $4 \times 60 = 240$  秒），我们可以使用以下命令：

```
ffmpeg -i video.mpg -ss 240 -t 60 clip_5th_minute.mpg
```

- 太简单了，就不测试了

## 输入流之间的延迟

通常有两种情况，当其中一个输入流应该被延迟到输出时，我们都使用 `-itsoffset`（输入时间戳偏移）选项来创建延迟和 `-map` 选项来选择特定的流。请注意，像AVI、FLV、MOV、MP4等容器有不同的标题，在某些情况下，它的偏移选项不起作用，然后可以使用 `-ss` 选项将较慢的流保存到文件中，并且可以将这两个文件合并为如下例所示，其中音频延迟1秒：

```
ffmpeg -i input.avi -ss 1 audio.mp3  
ffmpeg -i input.avi -i audio.mp3 -map 0:v -map 1:a video.mp4
```

\*很麻烦的样子

第一个测试的命令行：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -ss 20  
/Users/zhangfangtao/Desktop/test2.mp3
```

- 结果是生成了这个视频里面的所有的音频数据，文件名是 `test2.mp3`，并且这个 `mp3` 文件里面的音频信息是从视频信息里面的第20秒开始的。

第二个测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -i  
/Users/zhangfangtao/Desktop/DYZDJ.mp3 -map 0:v -map 1:a  
/Users/zhangfangtao/Desktop/test2.mp4
```

- 结果是生成了一个新的视频，这个视频的音频是我传入的那个mp3的声音，画面还是视频的画面，因为视频只有四十秒，音频三分钟，所以新的视频三分钟，不过进度条只能拖动到四十秒的位置。剩下的时间音乐继续播放，只是视频画面不动了。

## 一个输入文件

输入文件包含不同步的音频和视频流，例如，如果音频提前1.5秒，我们可以通过命令延迟音频流:(这个命令我试了，报错。。。。。-itsoffset语法问题)

```
ffmpeg -i input.mov -map 0:v -map 0:a -itsoffset 1.5 -c:a copy -c:v copy  
output.mov
```

我的测试命令：

```
ffmpeg -itsoffset 5 -i /Users/zhangfangtao/Desktop/test.mp4  
/Users/zhangfangtao/Desktop/test2.mp4
```

\*我把那些杂七杂八的参数都删除了，也不知道干啥用的，反正加上就会报错，这行命令执行之后的效果是在视频的开头，会有5秒钟的空白声音，5秒钟之后才开始正常的声音播放

如果视频是提前的，例如5秒，我们可以用命令延迟它:(这种命令在我的电脑上也是编译不过的)

```
ffmpeg -i input.mov -map 0:v -itsoffset 5 -map 0:a -c:a copy -c:v copy ^  
output.mov
```

我自己的测试命令如下：

```
ffmpeg -itsoffset -5 -i /Users/zhangfangtao/Desktop/test.mp4  
/Users/zhangfangtao/Desktop/test3.mp4
```

- 这样岂不是更简单？？？？？

## 两个或多个输入文件。

输出是由两个文件创建的，通常音频应该比视频流晚些开始，然后我们改变映射参数，例如延迟音频3秒，我们可以使用命令：

```
ffmpeg -i v.mpg -itsoffset 3 -i a.mp3 -map 0:v:0 -map 1:a:0 output.mp4
```

我的测试命令如下：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -itsoffset 5 -i  
/Users/zhangfangtao/Desktop/DYZDJ.mp3 -t 35 -map 0:v:0 -map 1:a:0  
/Users/zhangfangtao/Desktop/test2.mp4
```

- 上述代码的执行效果是：从第五秒钟之后开始有音乐，因为音乐时间比较长，我截取了前35秒钟。

## 限制处理时间

有时候，限制ffmpeg命令运行和`-timelimit`选项可以在几秒钟内设置这个限制是有用的。例如，在10分钟(600秒)之后停止编码，我们可以使用下一个命令：

```
ffmpeg -i input.mpg -timelimit 600 output.mkv
```

我的测试指令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -timelimit 10  
/Users/zhangfangtao/Desktop/test2.mp4
```

\*这个时间也不是很严谨的，我设置的时长是10秒钟，结果生成的是20多秒。

## 最短的流决定编码时间

要将整体输出持续时间设置为最短输入流值，可以使用`-shortest`选项在最短流处理准备就绪时完成编码。例如，要加入（复用）video.avi文件和audio.mp3文件，其中音频文件持续时间少于视频，我们可以使用下一个命令（没有`-shortest`选项，剩下的音频流将被静音替代）：

```
ffmpeg -i video.avi -i audio.mp3 -shortest output.mp4
```

我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -i  
/Users/zhangfangtao/Desktop/DYZDJ.mp3 -shortest  
/Users/zhangfangtao/Desktop/test2.mp4
```

- 这样会按照音视频中最短的那个来生成。

## 时间戳和时间基

要在媒体容器中设置记录时间戳，我们可以使用`-timestamp`选项，该选项的值是一个以表单输入的时间：

- 现在(当前时间)
- 日期被指定为`YYYY-MM-DD`或`YYYYMMDD`，如果未指定，则使用当前日期。
- 时间被指定为`HH:MM:SS[.m...]`或`HHMMSS[m...]`，秒的小数部分是可选的。
- 在时间值可以是可选的字母T或T之前。
- 如果附加`Z`或`z`，则时间为`UTC`，否则为本地

时间戳的例子:2010-12-24t12 00:00, 20101224t120000z, 20101224120000。

一个视频流的FFmpeg处理的控制台输出包含了关于流时间基的信息，这些信息可以看起来像下一个例子:

```
Stream #0:0, 1, 1/25: Video: mpeg4 (Simple Profile) (FMP4 yuv420p, 320x240 [SAR  
1:1 DAR 4:3], 25 tbr, 25 tbn, 25 tbc
```

缩写 tbr, tbn 和 tbc 表示FFmpeg时间戳的3个不同时基:

基于FFmpeg的时间戳的时间基础	
Specification	控制台输出包含每个视频流3个时基值，其中打印值是实际值的倒数，这意味着打印值为1 / tbc, 1 / tbn和1 / tbr。
	时间基的描述
tbc	在AVCodecContext中用于给定流的编解码器的时间基，它用于所有的AVCodecContext和相关的时间戳
tbn	来自容器的AVStream时基，它用于所有AVStream时间戳
tbr	从视频流中猜测（计算）的时基并等于帧速率值，除非输入是交错的，那么它会加倍

## 编码器时基设置

为了为流复制指定编码器的时间基，我们可以使用 -copytb 选项，该选项的值模式有3个可能的整数值:

- 1 - 使用分路器时基

时基从相应的输入分路器复制到输出编码器，有时需要复制具有可变帧频（VBR）的视频流以避免非单调增加时间戳。

- 0 -解码器使用时基

时间戳从对应的输入解码器复制到输出编码器。

- -1 -自动选择最佳输出，默认值

例如，要为输出选择一个demuxer时间基数，我们可以使用以下命令:

```
ffmpeg -i input.mp4 -copytb 1 output.webm
```

## 音频和视频速度修改

### 视频速度变化

为了改变视频文件的速度，我们可以使用表中描述的 setpts (设置表示时间戳)过滤器:

**Video filter: setpts**

描述	更改输入帧的表示时间戳(PTS)
语法	setpts=expression
	可用变量的表达式
FRAME_RATE	帧速率, 只定义为一个有固定帧的视频
INTERLACED	判断当前帧是否交错
N	输入框的计数, 从0开始
NB_CONSUMED_SAMPLES	消耗的样本数量, 没有当前帧(只有音频)
NB_SAMPLES	当前帧中样本数量(仅为音频)
POS	文件中的初始帧位置, 或者如果未定义当前帧, 则未定义
PREV_INT	之前的输入时间以秒为单位
PREV_INPTS	之前输入的PTS
PREV_OUTPTS	之前输出的PTS
PREV_OUTT	之前的输出时间以秒为单位
PTS	展示正在输入的时间戳
SAMPLE_RATE	音频采样率
STARTPTS	第一帧的PTS
STARTT	第一个帧的秒数
T	在当前帧数秒内的时间
TB	时间基

每个视频帧都包含一个带有时间戳值的标头, 顺序中2帧之间的差值为 $1 / \text{fps}$ , 例如, 如果fps为25, 则差值为0.04秒。为了加速视频, 这个时间差必须更小, 速度更低一定要更大。例如, 要快3倍观看视频, 输入时间戳除以3, 命令为:

```
ffplay -i input.mpg -vf setpts=PTS/3
```

我的测试命令:

```
ffplay -i /Users/zhangfangtao/Desktop/test.mp4 -vf setpts=PTS/5
```

- 显示的结果: 音频正常播放, 视频的播放速度是原来的五倍

为了以 $3/4$ 的速度观看视频, 输入时间戳除以 $3/4$ , 我们可以使用命令:(我测试了一下, 这个括号属于语法错误)

```
ffplay -i input.mpg -vf setpts=PTS/(3/4)
```

我的测试命令如下：

```
ffplay -i /Users/zhangfangtao/Desktop/test.mp4 -vf setpts=PTS/3*4
```

## 音频变速

为了调整音频的速度，我们可以使用表中描述的特殊atempo过滤器。

### Audio filter: atempo

描述	改变音频速度-音频流的速度
语法	atempo[=tempo]
	参数的描述
tempo	从0.5 - 2.0范围的浮点数， 小于1.0的值降低， 超过1.0的值加快速度， 默认值为1.0

例如，要以2倍的速度听到输入音频，我们可以使用以下命令：

```
ffplay -i speech.mp3 -af atempo=2
```

我的测试命令如下：

```
ffplay -i /Users/zhangfangtao/Desktop/DYZDJ.mp3 -af atempo=2
```

- 如果输入的结果不在0.5~2这个范围，也能打开ffplay，不过没有声音，终端也会报错的：

```

Input #0, mp3, from '/Users/zhangfangtao/Desktop/DYZDJ.mp3':
Metadata:
  encoder      : Lavf56.4.101
  album        : 冬雨
  title         : 大约在冬季
  comment       : 163 key(Don't modify):L64FU3W4YxX3ZFTmbZ+8/cbGIWzMwTjyEbHi
  artist        : 齐秦
  track         : 8
Duration: 00:03:54.29, start: 0.025056, bitrate: 321 kb/s
  Stream #0:0: Audio: mp3, 44100 Hz, stereo, fltp, 320 kb/s
  Stream #0:1: Video: mjpeg, yuvj420p(pc, bt470bg/unknown/unknown), 640x640 [S
AR 72:72 DAR 1:1], 90k tbr, 90k tbn, 90k tbc
Metadata:
  comment       : Other
[atempo @ 0x7f9340561840] Value 4.000000 for parameter 'tempo' out of range [0.5
- 2]
  Last message repeated 1 times
[atempo @ 0x7f9340561840] Error setting option tempo to value 4.
[Parsed_atempo_0 @ 0x7f9340561740] Error applying options to the filter.
Error initializing filter 'atempo' with args '4'
[swscaler @ 0x7f9341042e00] deprecated pixel format used, make sure you did set
range correctly
  Last message repeated 1 times
  nan M-V:    nan fd=    0 aq=    0KB vq=    0KB sq=    0B f=0/0

```

四倍速度播放就这样，没有声音

我们可以使用atempo=0.5设置，如果速度变化不够，则可以使用更多的时间。

## 同步音频数据与时间截

为了使音频数据与时间截同步，我们可以使用表中描述的asyncts音频过滤器：

### Audio filter: asyncts

描述	将音频数据与时间截同步，通过压缩和删除样本或在需要时进行拉伸和添加静默
语法	asyncts=parameters
	描述可用的参数
compensate	启用拉伸/压缩数据以使其与时间截相匹配。默认情况下禁用。当残疾的时候，时间间隔是沉默的
min_delta	时间截和音频数据之间的最小差异(以秒为单位)触发添加/删除示例。默认值是0.1。如果您与此过滤器不完全同步，请尝试将此参数设置为0
max_comp	max。样品每秒钟的补偿，仅当补偿=1时，默认值为500
first_cts	假设第一个pts应该是这个值。这允许在流的开始处填充/修剪。默认情况下，没有对第一个帧的预期值做任何假设，所以没有填充或修剪。例如，如果音频流在视频流之后启动，则可以将其设置为0以填充开始时的静默

例如，与时间截同步文件音乐中的数据。mpg我们可以使用以下命令:(在我电脑上-asyncts命令会报错)

```
ffmpeg -i music.mpg -af asyncts=compensate=1 -f mpegs music.ts
```

我这边会报错：（我已经在github上面提问了，还没有人回复我）

```
[AVFilterGraph @ 0x7f88a542e5c0] No such filter: 'asyncs'  
Error reinitializing filters!  
Failed to inject frame into filter network: Invalid argument  
Error while processing the decoded data for stream #0:1  
Conversion failed!
```

## 13-数学函数

FFmpeg工具提供的一个巨大优势是内置的数学函数，可以对某些音频和视频过滤器、选项和源进行各种修改。

### 可以使用数学函数的表达式

许多FFmpeg选项都需要数值作为参数，其中一些可以是表达式形式，可以包含算术运算符、常量和各种数学函数。函数通常用于音频和视频过滤器和源，下一个表包含它们的列表，包括在哪里找到它们的描述。

FFmpeg中的算术表达式的评估提供了一个内部公式评估器，通过位于文件libavutil/eval.h中的接口实现。这个评估人员也接受国际系统编号前缀(在FFmpeg文档中被称为后缀，因为它们是在数字之后立即输入的)。如果我是在前缀后面加上，使用的是2的幂而不是10的幂。B(字节)前缀将值乘以8，并且可以在另一个前缀后附加或单独使用。这意味着，例如B, KB, MiB可以像前缀一样使用。可用的SI数字前缀的列表在FFmpeg基础上。C代码中的开发人员可以扩展一元函数和二进制函数的列表，并定义额外的常量，这些常量将在描述的表达式中可用。

名称	类型	具体描述的章节
aevalsrc	音频源	<a href="#">数字音频章节</a>
asettb	音频过滤器	<a href="#">高级技术章节</a>
aspect	option	<a href="#">词汇表章节</a>
astreamsync	音频过滤器	<a href="#">数字音频章节</a>
boxblur	视频过滤器	<a href="#">模糊、锐化和其他去噪章节</a>
crop	视频过滤器	<a href="#">裁剪视频章节</a>
drawtext	视频过滤器	<a href="#">为视频添加文字章节</a>
hue	视频过滤器	<a href="#">颜色修正章节</a>
lut, lutrgb, lutyuv	视频过滤器	<a href="#">颜色修正章节</a>
overlay	视频过滤器	<a href="#">overlay-画中画章节</a>
rc_eq	option	<a href="#">格式之间转换章节</a>
pad	视频过滤器	<a href="#">填充视频章节</a>
scale	视频过滤器	<a href="#">调整和伸缩视频章节</a>
select	视频过滤器	<a href="#">高级技术章节</a>
setdar, setsar	视频过滤器	<a href="#">高级技术章节</a>
setpts	视频过滤器	<a href="#">时间操作章节</a>
settbb	视频过滤器	<a href="#">高级技术章节</a>
volume	音频过滤器	<a href="#">数字音频章节</a>

## 内置算术运算符

FFmpeg工具的用户可以使用常用的单目和双目算术运算符，如下表所述。

操作符	类型	描述	例子
+	单目	将负值转换为正数	$+(-3)=3$
-	单目	将正数转换为负数	$-(2+3)=-5$
+	双目	提供加法操作	$4+5=9$
-	双目	提供减法运算	$10-6=4$
*	双目	提供乘法运算	$4*5=20$
/	双目	提供除法运算	$9/3=3$
^	双目	提供了一个指数函数	$10^2=10*10=100$

## 内置的常量

最近FFmpeg仅包含下表中描述的3个常量，但开发人员可以通过修改源代码来定义附加常量。

符号	数值	描述
PI	3.14159265358979323846	圆周率
E	2.7182818284590452354	自然对数的底数，欧拉数
PHI	1.61803398874989484820	黄金比例,(1 +sqrt(5))/ 2

PI常数通常用作三角函数的正弦，余弦，正切等参数。例如，要产生频率为523.251 Hz的C5音调（中音高C）的音调，我们可以使用命令(命令里面右括号，我这儿会报语法错误)

```
ffplay -f lavfi -i aevalsrc=sin(523.251*2*PI*t)
```

我的测试命令：

```
ffplay -f lavfi -i aevalsrc=sin\(523.251*2*PI*t\)
```

由于余弦函数具有相似的周期性，下一个命令给出了相同的结果：

```
ffplay -f lavfi -i aevalsrc=cos(523.251*2*PI*t)
```

## 内置数学函数表

如果我们使用2个不同的表达式，并希望将它们组合成另一个表达式，我们可以使用一个符号“expr1;expr2”，其中expr1和expr2依次被计算，新的表达式求出expr2的值。

当使用函数来评估表达式为“true”时，如果它们具有非零值，我们可以利用\*符号(星号)的工作方式类似于逻辑，并且+符号(+)的工作方式类似或。接下来的两页包含FFmpeg工具中可用函数的表。

可用的函数表达式

函数	描述
abs(x)	计算x的绝对值
acos(x)	x的反余弦计算
asin(x)	计算x的反正弦
atan(x)	计算x的反正切
ceil(expr)	将expr扩展到最近的整数, 例如ceil(4.5)=5.0
cos(x)	计算cos(x)
cosh(x)	计算双曲余弦x
eq(x, y)	如果x=y, 则返回1, 否则返回0
exp(x)	以e=2.71828182(欧拉数)计算x的指数
floor(expr)	85/5000
将expr扩展到最近的整数, 例如地板(4.5)=4, 地板(-4.5)=-5	
gauss(x)	计算x的高斯函数, 对应 $\exp(-xx/2) / \sqrt{2\pi}$
gcd(x, y)	计算x和y的最大公约数, 如果x=y=0, 或者如果x<0和y<0, 结果是未定义的
gt(x, y)	大于比较, 返回1如果x > y, 否则返回0
gte(x, y)	大于或等于比较, 返回1如果x ≥ y, 否则返回0
hypot(x, y)	计算斜边(直角三角形最长边), $\sqrt{xx + yy}$
if(expr1, expr2)	评估expr1, 如果结果为非零返回expr1的评估, 否则返回0
ifnot(exp1, exp2)	评估exp1, 如果结果为零, 则返回值的评估, 否则返回0
isinf(x)	如果x是 $+\infty$ , 则返回1.0, 否则返回0
isnan(x)	如果x是NaN(不是数字), 则返回0
ld(var)	使用var标识符返回一个由st(var, expr)函数设置的内部变量值
log(x)	用e=2.71828182(欧拉数)计算x的自然对数
lt(x, y)	小于比较, 返回1如果x < y, 否则返回0
lte(x, y)	小于或等于比较, 返回1如果x≤y, 否则返回0
max(x, y)	计算x和y的最大值
min(x, y)	计算x和y的最小值
mod(x, y)	计算模, 除法x/y的余数
not(expr)	如果expr为0, 则返回1, 否则返回1

函数	描述
pow(x, y)	计算x的值提高到y的力量,结果是相当于 $(x)^y$
random(x)	从0.0 - 1.0返回一个伪随机数, x是用于保存种子/状态的内部变量的索引
root(expr, max)	在区间0中找到 $f(x)=0$ 的地方。函数f()必须是连续的, 否则结果是未定义的
sin(x)	计算 $\sin(x)$
sinh(x)	计算双曲正弦函数
sqrt(expr)	计算 $\sqrt{expr}$ ,结果是相当于 $(expr)^{0.5}$
squish(x)	计算表达式 $1/(1 + \exp(4*x))$
st(var, expr)	将表达式expr的值存储到带有数字var(值0到9)的内部变量中, 变量目前不在表达式之间共享
tan(x)	计算 $\tan x$
tanh(x)	计算双曲正切x
taylor(expr, x) taylor(expr, x, id)	-求x的泰勒级数, expr表示 $f(x)$ 在0处的导数 -如果id没有指定, 则假定为0 -如果你有y的导数而不是0, 可以用taloy(expr, x-y) -如果级数不收敛, 结果是未定义的
trunc(expr)	向0到最近的整数, 例如地板(-4.5)=-4
while(cond, expr)	计算表达式expr时, 表达式cond为非零, 并返回最后一个expr评价的值, 或者NAN if cond总是false
	-rc_eq选项的特殊函数
bits2qp(bits) qp2bits(qp)	附加功能, 可与其他函数一起使用, 以定义由所选编解码器的-rc_eq选项指定的速率控制方程

## 使用函数的例子

函数的大量应用程序提供了drawtext过滤器。例如, 可以使用lt(x, y)和gt(x, y)函数来设置文本出现或从视频帧中消失的时间, 下一个命令在开始时将文本延迟5秒:

```
ffplay -f lavfi -i color=c=orange -vf ^
drawtext=fontfile=/Windows/Fonts/arial.ttf:fontcolor=white:fontsize=20:^ text="5
seconds delayed text":x=(w-tw)/2:y=(h-th)/2:draw=gt(t\,5)
```

在书中使用函数的其他例子:

- [在视频上添加文字](#)
- [数字音频](#)章节里面的声音合成部分
- [批处理文件](#)章节

# 14-元数据和字幕

媒体文件中的元数据包含艺术家，作者，日期，流派，发布者，标题等附加信息，并且不会显示在视频帧中。字幕是文本数据，通常包含在单独的文件中，并显示在视频帧底部附近，尽管一些容器文件格式（如VOB）支持包含字幕文件。

## 元数据介绍

元数据通常用于MP3文件，媒体播放器通常在其中显示诸如歌曲标题，艺术家，专辑等的项目。例如，要显示位于Windows 7的Sample Music文件夹中的文件Kalimba.mp3的元数据（具有其他操作系统的用户可以选择具有始终存在于官方发布的音乐和视频中的元数据的其他媒体文件），我们可以使用该命令

```
ffplay -i "/Users/Public/Music/Sample Music/Kalimba.mp3"
```

控制台输出包括表单中的元数据：

```
Input #0, mp3, from 'Kalimba.mp3':
Metadata:
  publisher      : Ninja Tune
  track         : 1
  album          : Ninja Tuna
  artist         : Mr. Scruff
  album_artist   : Mr. Scruff
  title          : Kalimba
  genre          : Electronic
  composer        : A. Carthy and A. Kingslow
  date           : 2008
Duration: 00:05:50.60, start: 0.000000, bitrate: 191 kb/s
  Stream #0:0, 194, 1/14112000: Audio: mp3, 44100 Hz, stereo, s16, 192 kb/s
  Stream #0:1, 1, 1/90000: Video: mjpeg, yuvj420p, 512x512, 90k tbr, 90k
tbn, 90k tbc
Metadata:
  title          : thumbnail
  comment        : Cover (front)
```

我的测试命令：

```
ffplay -i /Users/zhangfangtao/Desktop/DYZDJ.mp3
```

显示的结果如下图：

```

Input #0, mp3, from '/Users/zhangfangtao/Desktop/DYZDJ.mp3':
Metadata:
  encoder      : Lavf56.4.101
  album        : 冬雨
  title         : 大约在冬季
  comment       : 163 key(Don't modify):L64FU3W4YxX3ZFTmbZ+8/cbGIWzMwTjyEbHi
  0icBF9w0MTJM+XMh2P3fo9c0SgDg3Gj/0b8nkrbtJNw5104vFhbVAKq4TcBZsuRfFgEjN+diydtiyTnk
  9IuEo8LgrYf/Mpp6+rqfTGGxDgjxX0/aAxoqJ12DErs6pRL7VKLJTRVfK6dikKJh/NO+YJ38EUrYaKNX
  4lM38GZ9GkrmvY1lVzovhWOkM9BuGa0HK4D14
  artist        : 齐秦
  track         : 8
Duration: 00:03:54.29, start: 0.025056, bitrate: 321 kb/s
  Stream #0:0: Audio: mp3, 44100 Hz, stereo, fltp, 320 kb/s
  Stream #0:1: Video: mjpeg, yuvj420p(pc, bt470bg/unknown/unknown), 640x640 [S
AR 72:72 DAR 1:1], 90k tbr, 90k tbn, 90k tbc
Metadata:
  comment       : Other

```

测试结果

## 创建元数据

元数据被包含在带有-元数据选项的媒体文件中，后跟一个键=值对，其中的键或值必须是双引号，如果包含空格。当需要输入更多的密钥时，可以使用几个元数据选项，例如：

```
ffmpeg -i input -metadata artist=FFmpeg -metadata title="Test 1" output
```

我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/DYZDJ.mp3 -metadata artist=FFmpeg -
  metadata title="zhangfangtao" -metadata age="22" -metadata sex="man"
  /Users/zhangfangtao/Desktop/DYZDJ2.mp3
```

显示的效果如下图：

```

Input #0, mp3, from '/Users/zhangfangtao/Desktop/DYZDJ2.mp3':=0/0
Metadata:
  sex          : man
  album        : 冬雨
  artist       : FFmpeg
  comment      : 163 key(Don't modify):L64FU3W4YxX3ZFTmbZ+8/cbGIWzMwTjyEbHi
  0icBF9w0MTJM+XMh2P3fo9c0SgDg3Gj/0b8nkrbtJNw5104vFhbVAKq4TcBZsuRfFgEjN+diydtiyTnk
  9IuEo8LgrYf/Mpp6+rqfTGGxDgjxX0/aAxoqJ12DErs6pRL7VKLJTRVfK6dikKJh/NO+YJ38EUrYaKNX
  4lM38GZ9GkrmvY1lVzovhWOkM9BuGa0HK4D14
  track        : 8
  title        : zhangfangtao
  age          : 22
  encoder      : Lavf58.13.100
Duration: 00:03:54.32, start: 0.025056, bitrate: 152 kb/s
  Stream #0:0: Audio: mp3, 44100 Hz, stereo, fltp, 128 kb/s
  Metadata:
    encoder      : Lavc58.19
  Stream #0:1: Video: png, rgb24(pc), 640x640 [SAR 1:1 DAR 1:1], 90k tbr, 90k
  tbn, 90k tbc
  Metadata:
    comment     : Other

```

自己定义了很多字段

ASF、FLV、Matroska、WMA和WMV文件格式都支持任何元数据键，而其他格式只支持某些键，细节在下面的表中(源:MultimediaWiki、wiki.multimedia.cx的FFmpeg元数据文章)。

Metadata key support in various media formats (Y = yes, in yellow - any keys)							
Key	AVI	ASF/WMV/ WMA	FLV	Matroska	MP3	MPEG TS transport stream	Quicktime/ MOV/MP4
album	Y	Y	Y	Y	Y		Y
album_artist		Y	Y	Y			Y
artist	Y	Y	Y	Y	Y		N
author		Y	Y	Y	Y		Y
comment	Y	Y	Y	Y	Y		Y
composer		Y	Y	Y			Y
copyright	Y	Y	Y	Y			Y
date	Y	Y	Y	Y			
description		Y	Y	Y			Y
encoded_by	Y	Y	Y	Y			
episode_id		Y	Y	Y			Y
genre	Y	Y	Y	Y	Y		Y
grouping		Y	Y	Y			Y
language	Y	Y	Y	Y		Y	
lyrics		Y	Y	Y			Y
network		Y	Y	Y			Y
rating		Y	Y	Y			
show		Y	Y	Y			Y
title	Y	Y	Y	Y	Y	Y	Y
track	Y	Y	Y	Y	Y		Y
year		Y	Y	Y	Y		
user-defined		Y	Y	Y			

用户定义的元数据可以包含表中未列出的键，例如添加信息

```

location          : London, United Kingdom
camera type      : SONY DSC
camera mode       : movie
weather           : sunny

```

我们可以使用这个命令：

```
ffmpeg -i video.avi -metadata location="London, United Kingdom" ^ -metadata
"camera type"="SONY DSC" -metadata "camera mode"=movie ^ -metadata
weather="sunny" video.wmv
```

我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/DYZDJ.mp3 -metadata  
location="shandongsheng" -metadata "camera type"="SONY DSC" -metadata "camera  
mode"=movie -metadata weather="sunny" /Users/zhangfangtao/Desktop/DYZDJ2.mp3
```

显示的结果如下图：

```
Input #0, mp3, from '/Users/zhangfangtao/Desktop/DYZDJ2.mp3':=0/0  
Metadata:  
    weather      : sunny  
    album        : 冬雨  
    title        : 大约在冬季  
    comment      : 163 key(Don't modify):L64FU3W4YxX3ZFTmbZ+8/cbGIWzMwTjyEbHi  
0icBF9w0MTJM+XMh2P3fo9c0SgDg3Gj/0b8nkrbtJNw5104vFhbVAKq4TcBZsuRfFgEjN+diydtiyTnk  
9IuEo8LgrYf/Mpp6+rqftGGxDgjxX0/aAxoqJ12DErs6pRL7VKLJTRVfK6dikKJh/NO+YJ38EUrYaKNX  
4lM38GZ9GkrmvY11VzovhWOkM9BuGa0HK4D14  
    artist       : 齐秦  
    track        : 8  
    location     : shandongsheng  
    camera type  : SONY DSC  
    camera mode   : movie  
    encoder      : Lavf58.13.100  
Duration: 00:03:54.32, start: 0.025056, bitrate: 152 kb/s  
Stream #0:0: Audio: mp3, 44100 Hz, stereo, fltp, 128 kb/s  
Metadata:  
    encoder      : Lavc58.19  
Stream #0:1: Video: png, rgb24(pc), 640x640 [SAR 1:1 DAR 1:1], 90k tbr, 90k  
tbn, 90k tbc
```

## 保存和加载文件的元数据

为了保存媒体文件中包含的元数据，我们可以使用-f选项指定的ffmetadata格式，在该文本文件的名称之前存储元数据。例如，从视频中保存元数据。在前面的示例中创建的wmv文件，我们可以使用该命令。

```
ffmpeg -i video.wmv -f ffmetadata data.txt
```

输出文件data.txt包含以下几行(最后一行将包含当前的编码器版本):

```
;FFMETADATA1  
weather=sunny  
location=London, United Kingdom  
camera type=SONY DSC  
camera mode=movie  
encoder=Lavf54.33.100
```

测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/DYZDJ2.mp3 -f ffmetadata  
/Users/zhangfangtao/Desktop/data.txt
```

txt文本里面显示的信息有：

```

;FFMETADATA1
weather=sunny
album=冬雨
title=大约在冬季
comment=163 key(Don't modify):L64FU3W4YxX3ZFTmbZ+8/
cbGIWzMwTjyEbHi0icBF9w0MTJM+XMh2P3fo9c0SgDg3Gj/
0b8nkrbtJNw5104vFhbVAKq4TcBZsuRffgEjN+diydtiyTnk9IuEo8LgrYf/Mpp6+rqfTGGxDgjxX0/
aAxoqJ12DErs6pRL7VKLJTRVfK6dikKJh/
NO+YJ38EURYaKNX4lM38GZ9GkrmvY1lVzovhW0kM9BuGa0HK4D14mSJyTm5SQuS12sNCsZCBaUJBTvp1k0P9GOIgpg
tkRiKEaVnsM0PMp89wRN0LwN5riBxMCXU2Bxdf3MNbjqKA06Kr7y3kJk0PffUk6B/
rpUfNM0umrMf3ZVUoymCL00HmwjUcGS+N8Fe152nD2w4d9RYQZ4MDCThezPxzKcxoKFWsxFR8bJW3N1w/
mkVeQrLB6H/uoAH9yMlqXfiHtVteK4H0/1lCzb7mK9Dpx7uEpIrFA\=\=
artist=齐秦
track=8
location=shandongsheng
camera type=SONY DSC
camera mode=movie
encoder=Lavf58.13.100

```

从data.txt加载元数据到其他相关的媒体文件，我们可以简单地把它作为第一个输入文件，在媒体文件之前，例如：

```
ffmpeg -i data.txt -i video1.avi video1.wmv
```

我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/data.txt -i
/Users/zhangfangtao/Desktop/test.mp4 /Users/zhangfangtao/Desktop/test2.wmv
```

生成新的视频信息：

```

Input #0, asf, from '/Users/zhangfangtao/Desktop/test2.wmv':
Metadata:
  album          : 冬雨
  comment        : 163 key(Don't modify):L64FU3W4YxX3ZFTmbZ+8/cbGIWzMwTjyEbHi
0icBF9w0MTJM+XMh2P3fo9c0SgDg3Gj/0b8nkrbtJNw5104vFhbVAKq4TcBZsuRffgEjN+diydtiyTnk9IuEo8LgrYf/Mpp6+rqfTGGxDgjxX0/
aAxoqJ12DErs6pRL7VKLJTRVfK6dikKJh/NO+YJ38EURYaKNX4lM38GZ9GkrmvY1lVzovhW0kM9BuGa0HK4D14
  weather        : sunny
  title          : 大约在冬季
  artist         : 齐秦
  track          : 8
  location       : shandongsheng
  camera type    : SONY DSC
  camera mode    : movie
  encoder         : Lavf58.13.100
Duration: 00:00:40.09, start: 0.000000, bitrate: 432 kb/s
  Stream #0:0(eng): Video: msmpeg4v3 (MP43 / 0x3334504D), yuv420p, 1024x768, 2
00 kb/s, SAR 1:1 DAR 4:3, 15 fps, 15 tbr, 1k tbn, 1k tbc
  Stream #0:1(eng): Audio: wma2 (a[1][0][0] / 0x0161), 22050 Hz, 2 channels,
fltp, 128 kb/s
    1.50 A-V: -0.009 fd= 0 aq= 32KB vq= 135KB sq= 0B f=0/0
zhangfangtaodeMacBook-Pro:~ zhangfangtao$ 
```

现在文件video1.wmv包含与data.txt文件传输的文件video.wmv相同的元数据。Loaded不仅可以是由ffmpeg保存的元数据文件，还可以创建具有特殊格式的全新文件。在这些文件中，第一行是包含文本的标题; FFMETADATA1，下一行是包含所需内容的key = value对，如上例所示。

## 删除元数据

要删除不是实际的元数据，我们可以使用设置为负值的 `-map_metadata` 选项，例如从文件 `input.avi` 中删除所有元数据，我们可以使用以下命令：

```
ffmpeg -i input.avi -map_metadata -1 output.mp4
```

我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test2.wmv -map_metadata -1  
/Users/zhangfangtao/Desktop/newtest2.wmv
```

生成的视频信息：

```
Input #0, asf, from '/Users/zhangfangtao/Desktop/newtest2.wmv':  
  Metadata:  
    encoder : Lavf58.13.100  
  Duration: 00:00:40.16, start: 0.000000, bitrate: 436 kb/s  
    Stream #0:0: Video: msmpeg4v3 (MP43 / 0x3334504D), yuv420p, 1024x768, SAR 1:  
1 DAR 4:3, 15 fps, 15 tbr, 1k tbn, 1k tbc  
    Stream #0:1: Audio: wmaV2 (a[1][0][0] / 0x0161), 22050 Hz, 2 channels, fltp,  
128 kb/s  
    8.53 A-V: -0.051 fd= 1 aq= 25KB vq= 57KB sq= 0B f=0/0
```

啥信息都没了

## 关于字幕的介绍

字幕是包含在视频帧底部附近的文本数据，用于提供附加信息，如将口语外语翻译为本地语言，提高识别率的相同语言字幕等。字幕可以分为两种主要类型：

- 外部媒体播放器在播放期间包含在独立文件中并且包含在视频帧中的优点是在没有视频的情况下进行编辑和分发
- 内部的，包含在具有视频和音频流的媒体文件容器中

其他部分包括在实况视频广播期间同时创建的准备好的字幕和实况字幕。其他排序将字幕分为打开和关闭 - 打开或关闭字幕和字幕等关闭字幕时，不能关闭打开的字幕。

支持的字幕编解码器和文件格式列表位于表格中，支持列D表示此格式可以解码，E表示编码的可用性（dvb\_teletext和eia\_608尚未指定）。例如，要将SRT格式的字幕转换为ASS格式，我们可以使用以下命令：

```
ffmpeg -i subtitles.srt subtitles.ass
```

可用解码器字幕

编解码器	支持	描述
dvb_subtitle	DE	DVB字幕(解码器:dvbsub)(编码器:dvbsub)
dvb_teletext		DVB电子文本
dvd_subtitle	DE	DVD字幕(译码器:dvdsub)(编码器:dvdsub)
eia_608		eia - 608关闭字幕
hdmv_pgs_subtitle	D	HDMV表示图形流字幕(解码器:pgssub)
jacosub	D	JACOsub字幕
microdvd	D	MicroDVD字幕
mov_text	DE	MOV文本
realtext	D	RealText字幕
sami	D	SAMI 字幕
srt	DE	带有嵌入式计时的SubRip字幕
ssa	DE	SSA (SubStation Alpha) / ASS(高级SSA)字幕(解码器:ASS)(编码器:ASS)
subrip	DE	SubRip字幕
subviewer	D	子视图字幕
text	D	生utf - 8的文本
webvtt	D	WebVTT字幕
xsub	DE	XSUB
		可用文件格式(支持列:D=demuxing yes, E=muxing yes)
文件格式	支持	描述
ass	DE	SSA (SubStation Alpha)字幕
jacosub	DE	JACOsub字幕格式
microdvd	DE	MicroDVD字幕格式
realtext	D	RealText字幕格式
sami	D	SAMI字幕格式
srt	DE	SubRip字幕
subviewer	D	子视图字幕格式
vobsub	D	VobSub字幕格式
webvtt	D	WebVTT字幕

## 直接编码到视频的字幕

例如，如果我们想要将一个字幕视频包含到网页中，我们需要将字幕编码到视频流中，2个过滤器可以做到:ass(只编码ass格式)和在表中描述的字幕过滤器:

视频过滤器:字幕

描述	包括使用libass库的输入视频的字幕
语法	subtitles=filename[:original_size]
	描述的选项
f, filename	包含字幕的文件的名称
original_size	原始视频的大小，当输入被调整时需要

为了防止Windows上的错误消息，需要指定可以从<http://ffmpeg.tv/fonts.conf>下载的fontconfig配置文件的位置。

请保存字体。conf文件到相同的目录，其中是文件ffmpeg.exe(或f.exe)，通过点击环境变量模态对话框中系统变量部分的按钮，添加3个新的环境变量(如何显示它，在第一章，分段路径设置):

新的系统变量模态对话框

变量名	变量值(ffmpeg_dir是ffmpeg.exe的位置)
FC_CONFIG_DIR	C:\ffmpeg_dir
FONTCONFIG_FILE	fonts.conf
FONTCONFIG_PATH	C:\ffmpeg_dir

请注意，并非所有的字幕格式都由所有的容器支持，大多数容器(AVI, Matroska, MP4, MPG, 等等)支持ASS和SRT。例如，从文件标题将字幕编码到视频流。srt到文件视频。mp4，我们可以使用这个命令(其他例子在图片上显示):

```
ffmpeg -i video.avi -vf subtitles=titles.srt video.mp4
```



## 15-图像处理

虽然FFmpeg工具的主要用途与音频和视频有关，但ffmpeg可以对各种图像格式进行解码和编码，并且许多图像相关任务可以快速完成。在网络服务器上使用ffmpeg可创建Web图像编辑器，支持FFmpeg的网络主机相信信息位于[Web视频](#)一章。

## 支持的图像格式

表格中列出了FFmpeg支持的图像格式及其特征后缀。除JPEG（无损JPEG）以外的所有这些文件类型都可以解码，除EXR，PIC和PTX之外都可以编码。

FFmpeg支持的图像格式

扩展名	编码	解码	描述
.Y.U.V	X	X	每个组件的一行文件
BMP	X	X	微软BMP图像
DPX	X	X	数码照片交换
EXR		X	OpenEXR
GIF	X	X	动画gif是未压缩的
JPG	X	X	不支持渐进式JPEG
JP2	X	X	JPEG 2000
JLS	X	X	JPEG-LS
LJPG		X	无损的JPEG
PAM	X	X	PAM是带有alpha支持的PNM扩展
PBM	X	X	便携式位图图像
PCX	X	X	PC画笔
PGM	X	X	便携式GrayMap形象
PGMYUV	X	X	PGM与U和V的分量在YUV 4:2:0
PIC		X	Pictor/PC Paint
PNG	X	X	便携式网络图形
PPM	X	X	便携式PixelMap形象
PTX		X	V.Flash PTX格式
SGI	X	X	SGI RGB图像格式
RAS	X	X	Sun Rasterfile图像格式
TIFF	X	X	YUV, JPEG和一些扩展还没有被支持
TGA	X	X	Truevision Targa 图像格式
XBM	X	X	X位图图像格式
XFace	X	X	XFace图像格式
XWD	X	X	X窗口转储图像格式

## 创建图像

### 从视频截图

为了将一个视频帧从指定的时刻保存到图像中，使用-ss(从start开始)选项来指定从开始的延迟。在时间t中截图的语法

```
ffmpeg -i input -ss t image.type
```

我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -ss 20  
/Users/zhangfangtao/Desktop/image.jpg
```

结果生成了第二十秒的一张图片：



背后的视频正好处在20秒钟的位置

-ss选项也可以在输入文件之前使用，但结果不太准确。例如，从文件视频中截取1小时23分45秒的截图。.avi，我们可以使用命令：

```
ffmpeg -i videoclip.avi -ss 01:23:45 image.jpg
```

## 从视频动画gif

视频文件是从可以保存到GIF动画帧的帧中创建的，这是一种经常在web上以条幅和动画形式使用的图像类型。因为帧是没有压缩的，所以只有在较短的视频中才有用，否则GIF动画文件的文件大小就会非常大。例如，要将一个短的SWF文件转换为GIF动画，以便为没有Flash插件的用户创建一个替代文件，我们可以使用该命令(必须将像素格式设置为rgb24)：

```
ffmpeg -i promotion.swf -pix_fmt rgb24 promotion.gif
```

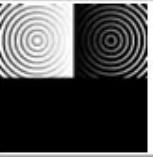
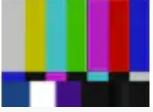
我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -pix_fmt rgb24  
/Users/zhangfangtao/Desktop/test.gif
```

- 结果就是生成了一个gif图

## 来自FFmpeg视频源的图像

另一个创建图像的选项是使用内置的视频源，在表中描述：

Video sources for creating images		
名称	描述	图片
color	提供以其名称或十六进制格式指定的任何颜色，例如颜色 =c=#87cefa	
mptestsrc	不同的测试模式，详细的描述都在章节的调试和测试中	
rgbtstsrc	红绿蓝颜色模式	
smp테bars	彩色条纹图案来自于电影和电视工程师协会，工程指南(1-1990)	
testsrc	带有滚动渐变和时间戳的视频测试模式	

mptestsrc视频源的默认分辨率为512x512像素，其他列出的源分辨率为320x240像素。最通用的是能够生成任何颜色和任何大小的图像的彩色图像源，例如为一个标题大小的728x90像素创建一个teal背景，我们可以使用这个命令。

```
ffmpeg -f lavfi -i color=c=#008080:s=728x90 leaderboard.jpg
```

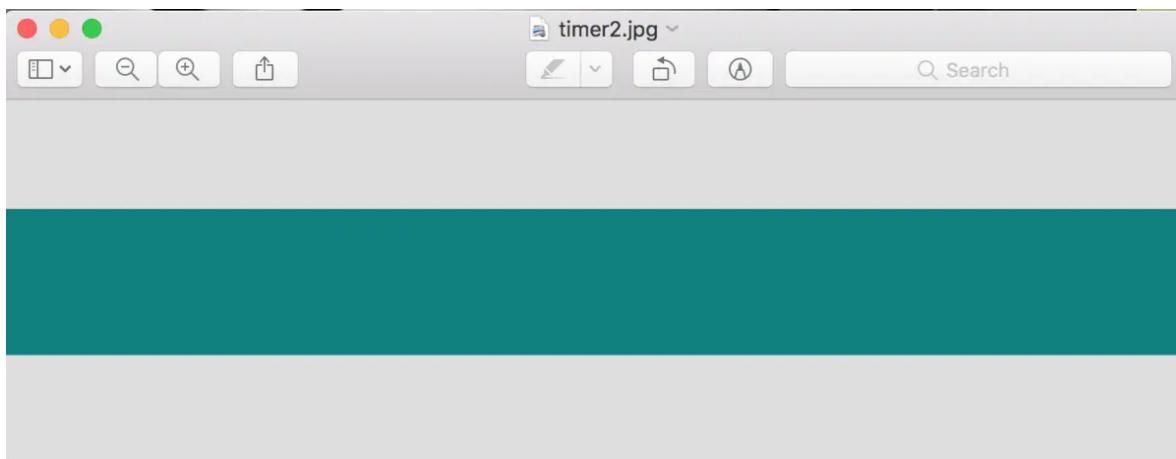


Video source: color

我的测试命令：

```
ffmpeg -f lavfi -i color=c=#008080:s=728x90  
/Users/zhangfangtao/Desktop/timer2.jpg
```

\*生成的图片：



描述	提供一种颜色为320x240的矩形，并带有指定的颜色
语法	color[=c=clr[:d=time[:r=fps[:sar=value[:s=resolution]]]]] 所有参数都是可选的，斜体中的项目将被替换为实际值
	参数描述
color, c	源的颜色、颜色的名称(不区分大小写的匹配)或0xRRGGBB[AA]序列，可能后跟一个alpha说明符，默认值为黑色
duration, d	源视频的持续时间，被接受的句法是:[-]HH[:]或[-]S+[m...], 如果未指定，或表示持续时间为负，则视频将永远生成
rate, r	源视频的帧速率，每秒生成的帧数，它可以是format frame_rate_factor / frame_rate_denominator，一个整数或浮点数，或者一个有效的视频帧率缩写，默认值是25
sar	样本纵横比的源视频
size, s	源视频的大小、窗体宽度的字符串或相应的缩写，默认值为320x240

## 视频转换为图像

视频文件是由可以通过一个命令保存到图像文件的帧组成的，结果图像的数量是视频帧速率的产物，它的持续时间为秒。例如，如果剪辑。avi文件的持续时间为1分钟，帧率为25 fps，下面的命令将产生 $60 \times 25 = 1500$ 张图片，每秒25张：

```
ffmpeg -i clip.avi frame%4d.jpg
```

输出目录将包含1500个文件，名为frame1.jpg、frame2.jpg等。为了保持所有文件名的长度相同，我们在%符号后指定附加数字的数目：

```
ffmpeg -i clip.avi frame%4d.jpg
```

现在该目录包含名为frame0001.jpg、frame0002.jpg的文件...,frame1500.jpg。

我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4  
/Users/zhangfangtao/Desktop/Images/frame%03d.jpg
```

- 效果：

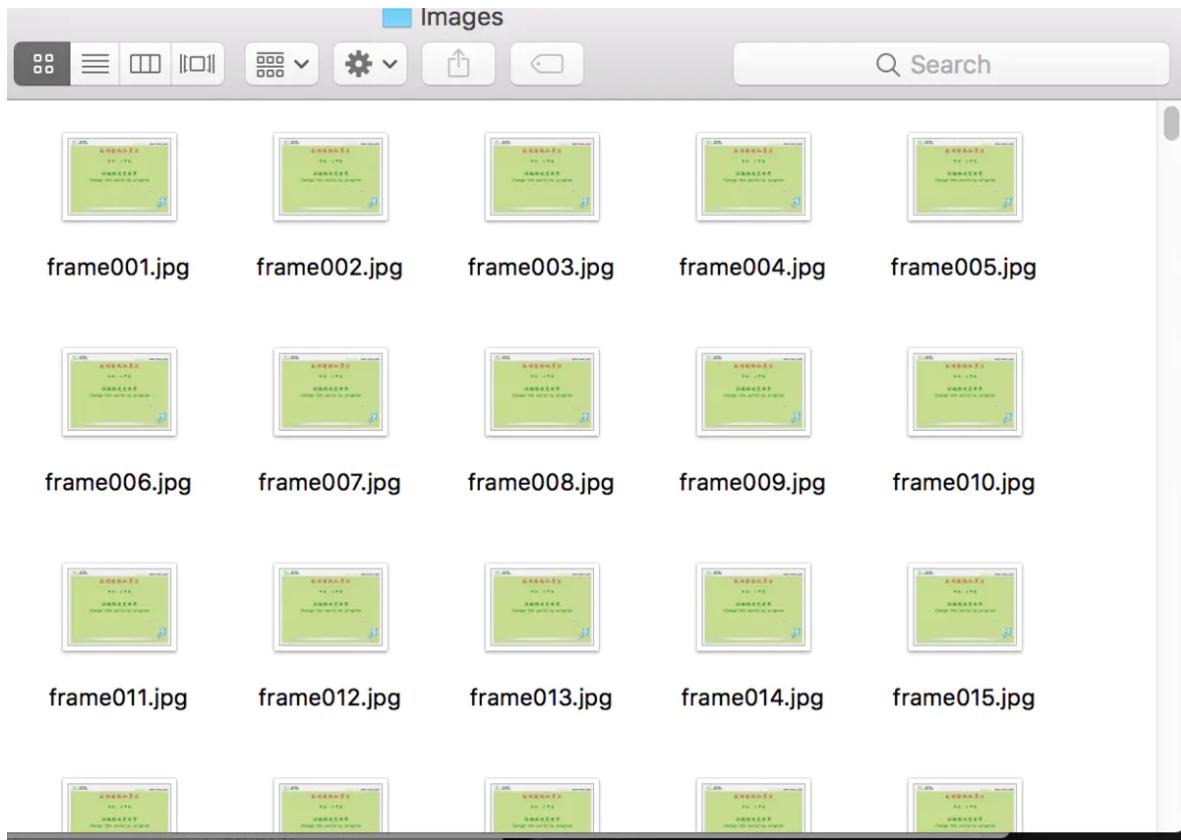


image.png

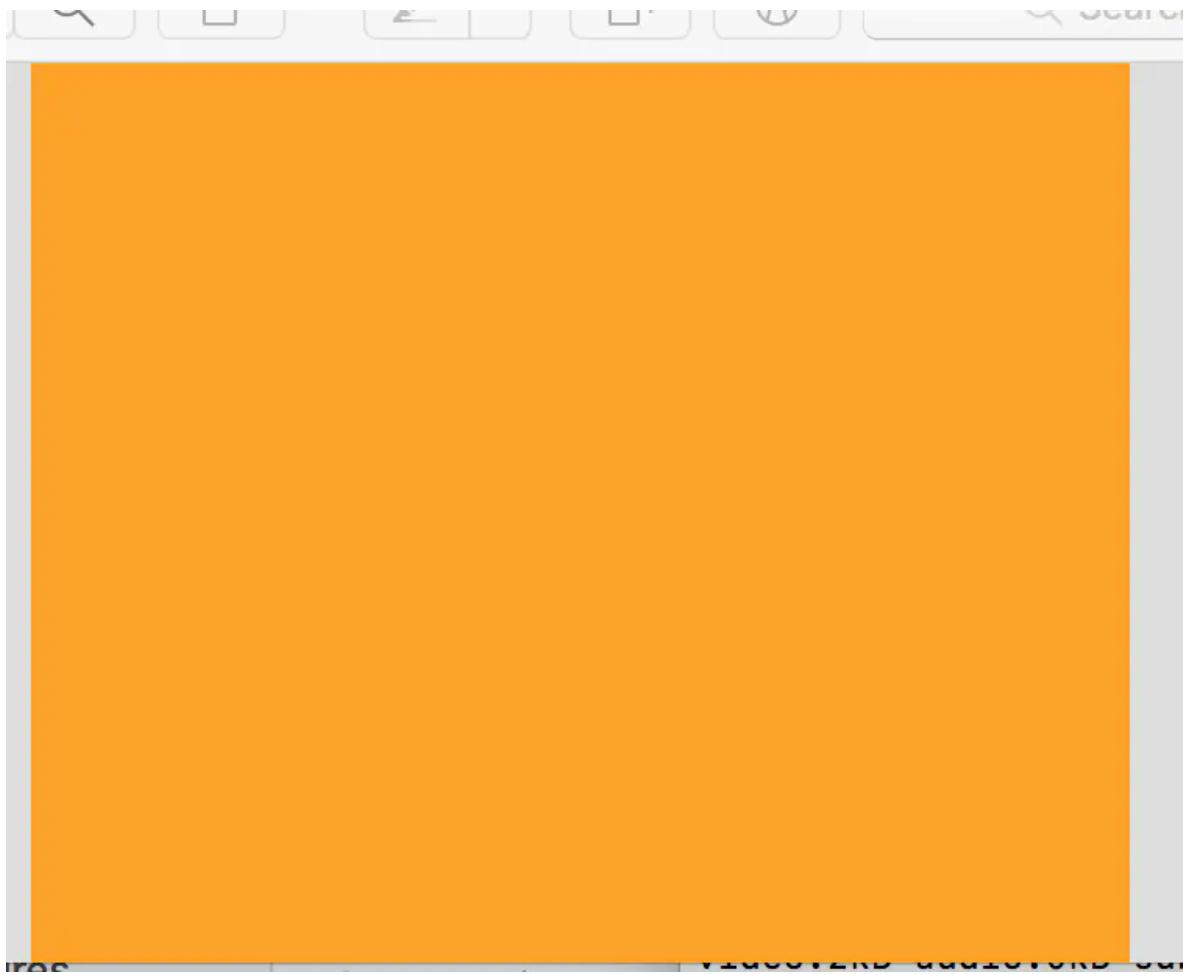
## 调整大小、裁剪和填充图像

图像可以以类似视频的方式调整大小，例如，彩色视频源的输出分辨率为320x240像素，可以通过两种方式放大到VGA分辨率：

- 使用彩色视频源的s或大小参数。
- 使用 -s 选项来输出

例如，接下来的两个命令具有相同的结果，CIF（352x288）大小的橙色矩形：

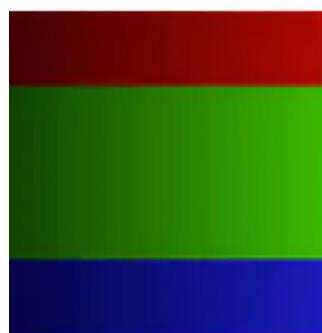
```
ffmpeg -f lavfi -i color=c=orange:s=cif orange_rect1.png  
ffmpeg -f lavfi -i color=c=orange -s cif orange_rect2.png
```



当过滤链内的输入应该具有特定的分辨率时，带参数的大小规格对过滤链很有用，因此大小不能被指定为选项。一个常见的例子是使用颜色源作为叠加层的输入之一。

裁剪图像与使用作物过滤器的视频是一样的，下一个示例的结果是rgbttestsrc视频源中心的150x150像素的正方形：

```
ffmpeg -f lavfi -i rgbttestsrc -vf crop=150:150 crop_rgb.png
```



图像可以像使用pad过滤器的视频一样进行padd，例如下一个命令为smptebars视频源创建一个橙色框架：

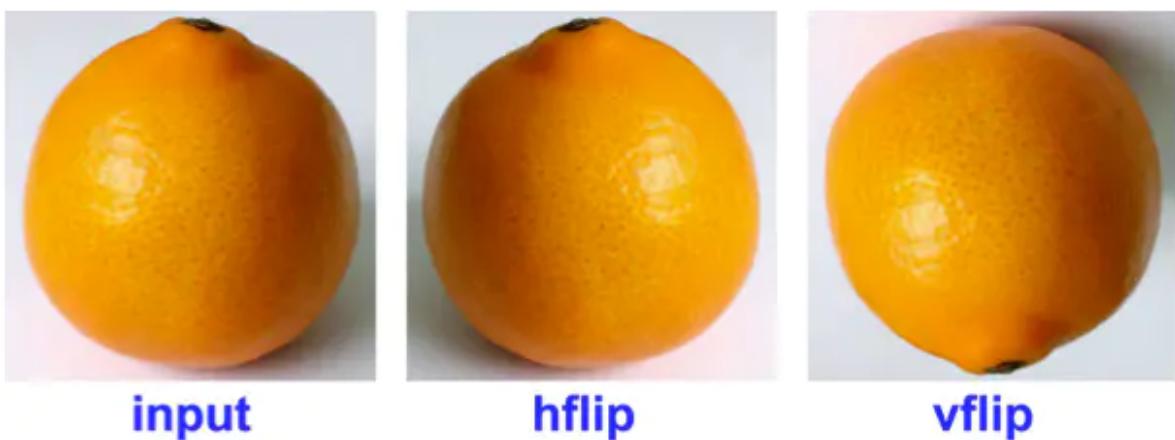
```
ffmpeg -f lavfi -i smptebars -vf pad=360:280:20:20:orange pad_smpte.jpg
```



## 翻转，旋转和叠加图像

翻转图像的镜像版本与由`hflip`和`vflip`过滤器提供的翻转视频类似，例如，接下来的两个命令会翻转输入图像，第一个是水平的，第二个是垂直的：

```
ffmpeg -i orange.jpg -vf hflip orange_hflip.jpg  
ffmpeg -i orange.jpg -vf vflip orange_vflip.jpg
```



我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/001.jpg -vf hflip  
/Users/zhangfangtao/Desktop/002.jpg  
ffmpeg -i /Users/zhangfangtao/Desktop/001.jpg -vf vflip  
/Users/zhangfangtao/Desktop/003.jpg
```



旋转的图像也类似于旋转的视频，使用的转置滤波器有四个可能的值：

- 值0逆时针旋转90度，垂直翻转。
- 数值1按顺时针方向旋转90度。
- 数值2逆时针旋转90度。
- 值3顺时针旋转90度，然后垂直翻转。

例如，把图像顺时针旋转90°我们可以使用命令：

```
ffmpeg -i image.png -vf transpose=1 image_rotated.png
```

我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/001.jpg -vf transpose=1  
/Users/zhangfangtao/Desktop/004.jpg
```

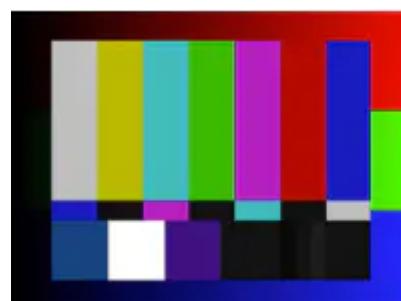
- 生成的图像：



004旋转90度之后的照片.jpg

类似于视频叠加，图像可以通过叠加过滤器放置在另一个图像上。例如，要将smptebars包含到rgbttestsrc视频源中，我们可以使用以下命令：

```
ffmpeg -f lavfi -i rgbttestsrc -s 400x300 rgb.png  
ffmpeg -f lavfi -i smptebars smpte.png  
ffmpeg -i rgb.png -i smpte.png -filter_complex overlay=(W-w)/2:(H-h)/2 ^  
rgb_smpte.png
```



我的测试命令:

```
ffmpeg -f lavfi -i rgbtests -s 400x300 /Users/zhangfangtao/Desktop/005.png  
ffmpeg -f lavfi -i smptebars /Users/zhangfangtao/Desktop/006.png  
ffmpeg -i /Users/zhangfangtao/Desktop/005.png -i  
/Users/zhangfangtao/Desktop/006.png -filter_complex overlay=(W-w)/2:(H-h)/2  
/Users/zhangfangtao/Desktop/007.png
```

- 生成的效果图(从左到右依次是005, 006, 007):



## 图像类型之间的转换

几乎所有支持的图像类型都可以转换为另一个，异常是EXR、LJPEG、PIC和PTX文件类型，只能进行解码。转换的语法是:

```
ffmpeg -i image.type1 image.type2
```

例如，要将PNG图像转换为JPG图像格式，我们可以使用以下命令:

```
ffmpeg -i illustration.png illustration.jpg
```

我的测试命令:

```
ffmpeg -i /Users/zhangfangtao/Desktop/001.jpg  
/Users/zhangfangtao/Desktop/001.png
```

## 创建视频图像

### 视频来自一个图像

将静态图像转换为视频很容易，可以用来创建幻灯片，从图像(添加文本)的短视频连接到一起，连接视频在第[23章](#)中有描述。例如，在photo.jpg文件中创建一个10秒的视频，我们在命令中包含一个值true或1的-loop boolean选项:

```
ffmpeg -loop 1 -i photo.jpg -t 10 photo.mp4
```

我的测试命令:

```
ffmpeg -i /Users/zhangfangtao/Desktop/001.jpg  
/Users/zhangfangtao/Desktop/001.png
```

- 显示的效果：



## 视频来自多个图像

要从多个图像创建视频，它们的文件名必须以数字结束，这些数字与图像编码到视频文件的顺序一致。在这种情况下，媒体格式是在输入之前指定的，它是一个image2格式。例如，来自img1.jpg、img2.jpg的100幅图片。, img100.jpg可以使用以下命令创建一个4秒视频，帧率为25 fps。

```
ffmpeg -f image2 -i img%d.jpg -r 25 video.mp4
```

如果图像编号以0开头，例如img001.jpg、img002.jpg等，以提供相同的文件名长度，则命令为：

```
ffmpeg -f image2 -i img%3d.jpg -r 25 video.mp4
```

%符号后的数字必须与图像文件名中的位数相同。

我的测试命令：

```
ffmpeg -f image2 -i /Users/zhangfangtao/Desktop/Images/frame%3d.jpg  
/Users/zhangfangtao/Desktop/001.mp4
```

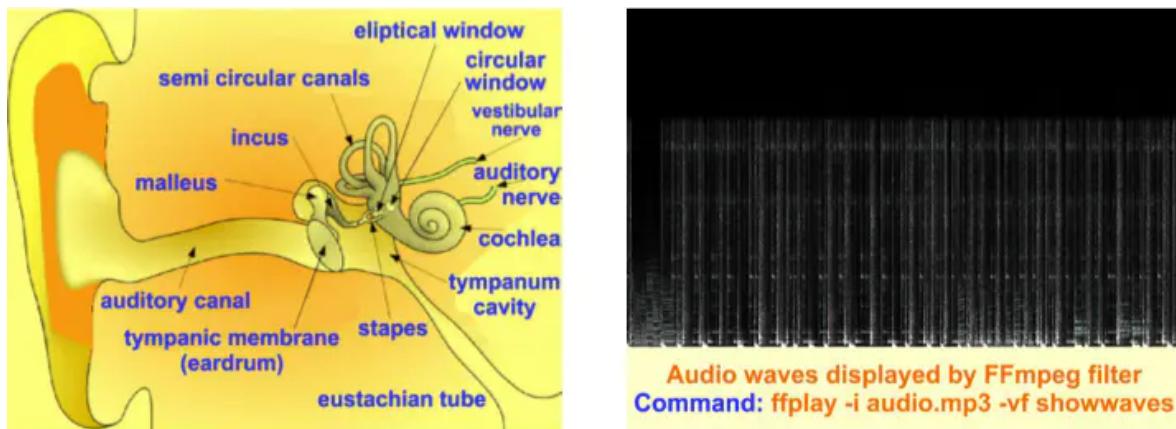
- 结果就是生成了一个没有声音的视频，视频里面的图片就是我刚才从test.mp4里面解析出来的图片

## 16-数字音频

“数字音频”一词与“数字视频”一词相比，它是一种处理和显示移动图像的技术，而音频则与声音有关。数字音频是一种技术，用于捕获、记录、编辑、编码和复制声音，这些声音通常由脉冲编码调制(PCM)进行编码。FFmpeg支持许多音频格式，包括AAC、MP3、Vorbis、WAV、WMA等。FFmpeg中所有的音频格式都在[第二章](#)中列出。

### 关于数字音频的介绍

由耳朵感知的声音可分为音调和噪音，音调是由不规则振动产生的常规物质振动和噪音产生的。机械振动以鼓膜感知的压力波的形式传递到听觉系统，并转换为神经信号。



### 音频量化和采样

由于人类听觉系统的生理限制，压力波的连续值可以用有限的一系列值代替，这些值可以作为数字存储在计算机文件中。计算机使用二进制数字，所以常见的音频位深度（音频分辨率）是两个幂：

Common audio bit depths (quantization levels)

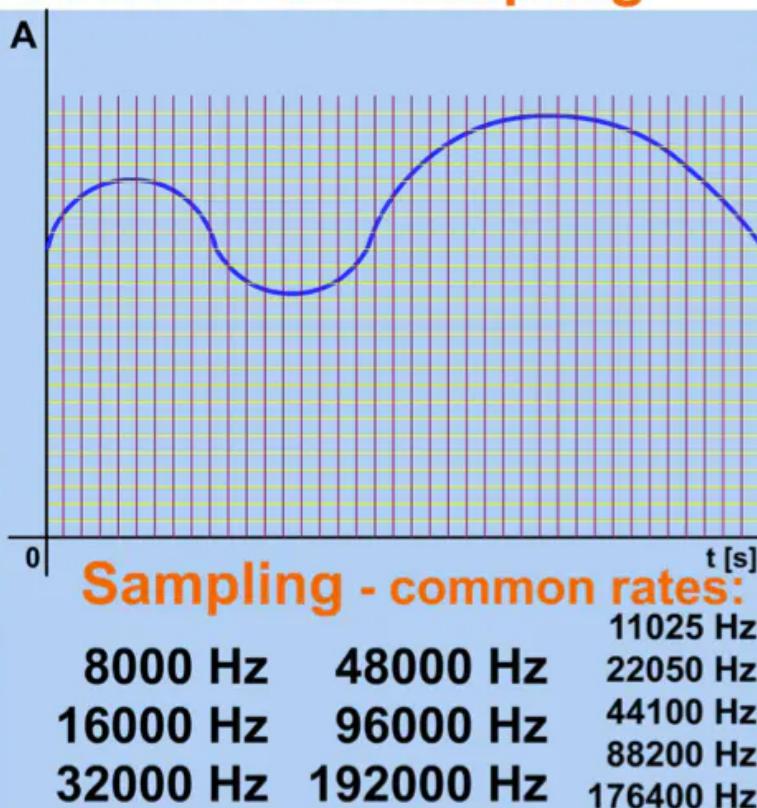
位深度	值计算	描述
8 bit	$2^8 = 256$	用于电话，旧设备
12 bit	$2^{12} = 4,096$	DV(数字视频)的标准，用于数码相机等
14 bit	$2^{14} = 16,384$	用于NICAM压缩，电视立体声，等等
16 bit	$2^{16} = 65,536$	标准音频CD和DAT(数字音频磁带)，是当今最常见的
20 bit	$2^{20} = 1,048,576$	附加标准的超级音频CD和DVD音频
24 bit	$2^{24} = 16,777,216$	标准的超级音频CD和DVD音频
32 bit	$2^{32} = 4,294,967,296$	专业设备,蓝光技术

# Audio quantization and sampling

Quantization bits	values
8	256
12	4096
16	65536
24	16777216

Quantization substitutes continuous analog signal with the discrete values from regular intervals.

Sampling frequency or sampling rate defines the number of samples per second.



模拟音频信号（大组数值）通过在每个时间单位创建一组较小的采样数字化，常见采样频率（采样率）在表中描述：

Common audio sample rates (frequencies)

8000 Hz	用于电话、无线网络和麦克风等
11025 Hz	用于低质量PCM和MPEG音频等
16000 Hz	电话宽带(2倍8000 Hz), 用于VOIP设备等
22050 Hz	用于低质量PCM和MPEG音频等
32000 Hz	用于DAT, NICAM, 迷你DV相机, 无线麦克风等
44100 Hz	音频CD标准, 用于MPEG-1, PAL电视等
48000 Hz	专业使用标准费率, 为消费者提供DV、DVD、数字电视等
96000 Hz	标准的DVD-音频, 蓝光光盘, HD DVD等
192000 Hz	用于DVD-音频, 蓝光光盘, HD DVD, 专业设备
352800 Hz	数字极端定义, 用于超级音频光盘

## 音频文件格式

量化和采样音频被保存在不同的媒体文件格式，下一个表描述特定的文件格式，仅用于音频(MP3格式支持也包括图像)：

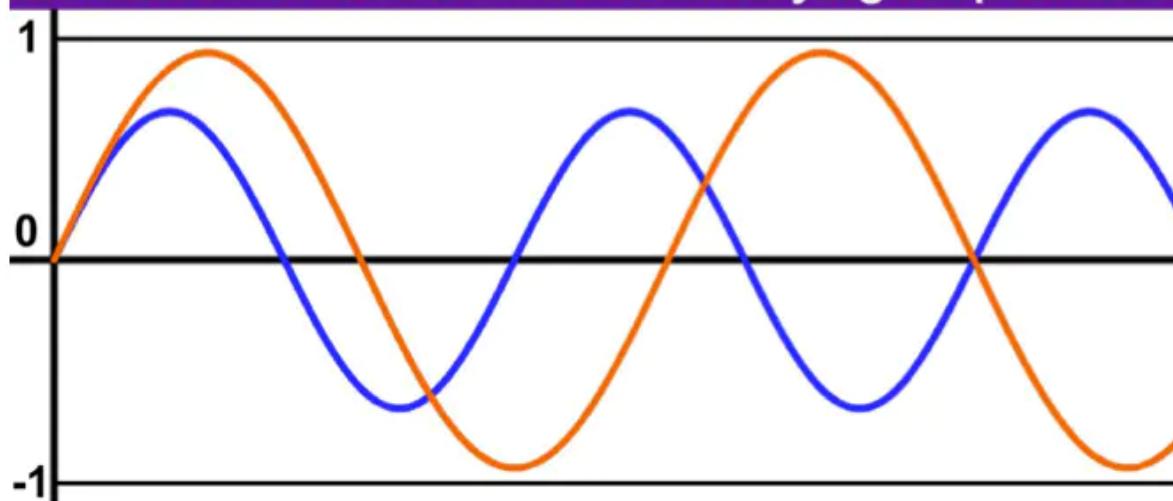
## Common audio formats

未压缩的	无损压缩	有损压缩
ALAC	AIFF (PCM)	AAC
AU	ALS	AC-3
BWF	ATRAC	AMR
PCM (raw, without header)	FLAC	MP2, MP3
WAV (PCM)	WavPack	Musepack
	WMA	Speex
		Vorbis (OGG)

## 声音合成

声音是由在固定位置振动的物体的振动而产生的，规则的振动被称为音调，可以用不同振幅和频率的正弦和余弦波来表示。

**Each sound can be substituted with a combination of various sines and cosines of varying frequencies.**



**2 sine waves of different amplitude and frequency.**

用一个表达式 $\sin(\text{tone\_height} \cdot 2\pi \cdot t)$ 来创建一定高度的连续色调，在Hz中，tone\_height为给定频率， $\pi$ 是一个数学常量，t是在秒内指定时间的变量。为了用数学表达式创建声音，我们可以使用音频源aevalsrc，它的输出可以作为音频文件保存。输出声音可以包含多个通道，每个通道由一个带有三个可能变量的表达式指定，详细信息在表中：

**Audio source: aevalsrc**

描述	创建一个由一个(mono)、两个(立体声)或更多表达式指定的音频信号
语法	aevalsrc=exprs[:options] exprs:是一个冒号分隔的表达式列表，每个新表达式都指定了新通道 options:键=值对的冒号分隔列表
	表达式exprs中可用变量的描述
n	评估样本的数量，从0开始
t	以秒为单位的时间，从0开始
s	采样率
	可用选项的描述
c or channel_layout	通道布局，通道数量必须等于表达式的数量
d or duration	max。持续时间，如果没有指定，或者是负数，音频将生成直到程序停止
n or nb_samples	每个通道每个输出帧的样本数量，默认为1024个样本
s or sample_rate	采样率，默认值为44100 Hz

根据音符A4的音调标准，下一个表格包含从C1到B8的音调频率，频率为440 Hz。人声范围从E2（男低音）到C6（女高音）。

Frequencies of notes based on A4 = 440 Hz								
Note/Octave	1	2	3	4	5	6	7	8
C	32.703	65.406	130.81	261.63	523.25	1046.5	2093.0	4186.0
C#/Db	34.648	69.296	138.59	277.18	554.37	1108.7	2217.5	4434.9
D	36.708	73.416	146.83	293.66	587.33	1174.7	2349.3	4698.6
Eb/D#	38.891	77.782	155.56	311.13	622.25	1244.5	2489.0	4978.0
E	41.203	82.407	164.81	329.63	659.26	1318.5	2637.0	5274.0
F	43.654	87.307	174.61	349.23	698.46	1396.9	2793.8	5587.7
F#/Gb	46.249	92.499	185.00	369.99	739.99	1480.0	2960.0	5919.9
G	48.999	97.999	196.00	392.00	783.99	1568.0	3136.0	6271.9
Ab/G#	51.913	103.83	207.65	415.30	830.61	1661.2	3322.4	6644.9
A	55.000	110.00	220.00	440.00	880.00	1760.0	3520.0	7040.0
Bb/A#	58.271	116.54	233.08	466.16	932.33	1864.7	3729.3	7458.6
B	61.735	123.47	246.94	493.88	987.77	1975.5	3951.1	7902.1

为了产生音符A4，音高的调优标准，我们可以将 tone\_height 设置为440 Hz:

```
ffmpeg -f lavfi -i aevalsrc=sin(440*2*PI*t) -t 10 noteA4.mp3
```

我的测试命令是：

```
ffmpeg -f lavfi -i aevalsrc=sin\(440*2*PI*t\) -t 10
/Users/zhangfangtao/Desktop/001.mp3
```

\*显示结果生成一个10秒钟的音频文件。

## 立体声和更复杂的声音

要使用 `aevalsrc` 音频源创建多声道声音，我们为每个声道指定一个定义表达式，声道由冒号分隔，然后在双冒号后指定其定位。表格中描述了可以使用 `-layout` 选项显示的可用通道布局：

Abbreviations of available channel layouts							
FL	front left	FLC	front-left-of-center	TFL	top front left	DL	downmix left
FR	front right	FRC	front-right-of-center	TFC	top front center	DR	downmix right
FC	front center	BC	back-center	TFR	top front right	WL	wide left
LFE	low frequency	SL	side-left	TBL	top back left	WR	wide right
BL	back left	SR	side-right	TBC	top back center	SDL	surround direct left
BR	back right	TC	top center	TBR	top back right	SDR	surround direct right
Complex channel layouts (FC is mono and FL+FR is stereo)							
2.1	FL+FR+LFE	5.0 side	FL+FR+FC+SL+SR	6.1	FL+FR+FC+LFE+BL+BR+BC		
3.0	FL+FR+FC	4.1	FL+FR+FC+LFE+BC	6.1 front	FL+FR+LFE+FLC+FRC+SL+SR		
3.0 back	FL+FR+BC	5.1	FL+FR+FC+LFE+BL+BR	7.0	FL+FR+FC+BL+BR+SL+SR		
4.0	FL+FR+FC+BC	5.1 side	FL+FR+FC+LFE+SL+SR	7.0 front	FL+FR+FC+FLC+FRC+SL+SR		
quad	FL+FR+BL+BR	6.0	FL+FR+FC+BC+SL+SR	7.1	FL+FR+FC+LFE+BL+BR+SL+SR		
quad side	FL+FR+SL+SR	6.0 front	FL+FR+FLC+FRC+SL+SR	7.1 front	FL+FR+FC+LFE+FLC+FRC+SL+SR		
3.1	FL+FR+FC+LFE	hexagonal	FL+FR+FC+BL+BR+BC	octagonal	FL+FR+FC+BL+BR+BC+SL+SR		
5.0	FL+FR+FC+BL+BR	6.1	FL+FR+FC+LFE+BC+SL+SR	downmix	DL+DR		

例如，在左边的通道中创建C4音调，在右边的C5音调中，我们可以使用命令：

```
ffplay -f lavfi -i aevalsrc=sin(261.63*2*pi*t):cos(523.25*2*pi*t)::c=FL+FR
```

我的电脑上上面的命令会报错，我使用的测试命令如下；

```
ffplay -f lavfi -i aevalsrc=sin\(261.63*2*pi*t\)|cos\(523.25*2*pi*t\)::c=FL+FR
```

## 减轻压力的双耳音调

立体声的特殊类型是双耳音（节拍） - 两个频率差约30Hz或更小的音，两个音的频率必须低于1000Hz。使用立体声耳机收听双耳音可以为听众带来积极的影响，如减轻压力，提高学习能力和对大脑功能产生的其他积极影响，但结果因使用的频率基准和频率差异而有所不同。要在基频为500 Hz时以10 Hz的差异创建双耳节拍，我们会指定具有略微不同声道的立体声：

```
ffplay -f lavfi -i aevalsrc=sin(495*2*pi*t):sin(505*2*pi*t)::c=FL+FR
```

我的测试命令：

```
ffplay -f lavfi -i aevalsrc=sin\((495*2*PI*t\)|sin\((505*2*PI*t\)::c=FL+FR
```

## 音量设置

声音音量应该仔细调整，以保护我们的耳朵和ffmpeg提供2种方法。第一个使用-vol选项，它接受从0到256的整数值，其中256是最大值，例如：

```
ffmpeg -i sound.wav -vol 180 sound_middle_loud.wav
```

我的测试命令

```
ffmpeg -i /Users/zhangfangtao/Desktop/DYZDJ.mp3 -vol 128  
/Users/zhangfangtao/Desktop/DYZDJ3.mp3
```

另一种方法是使用表中描述的卷过滤器：

### Audio filter: volume

描述	将输入音频卷更改为指定的值
语法	volume=vol
	vol的描述参数
vol	参数vol是一个表达式，其值可以通过以下两种方式来指定：1.作为一个小数，那么 output_volume = vol * input_volume 2.作为一个十进制数与dB后缀,然后output_volume = 10^(卷/20)* input_volume

例如，将音量降低到三分之二，我们可以使用以下命令：

```
ffmpeg -i music.wav -af volume=2/3 quiet_music.wav
```

我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/DYZDJ.mp3 -af volume=1/3  
/Users/zhangfangtao/Desktop/DYZDJ4.mp3
```

为了增加10分贝的音量，我们可以使用以下命令：

```
ffmpeg -i sound.aac -af volume=10dB louder_sound.aac
```

我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/DYZDJ.mp3 -af volume=20dB  
/Users/zhangfangtao/Desktop/DYZDJ5.mp3
```

## 多个声音混合到一个输出

为了混合不同长度的声音并指定一个过渡段，我们可以使用amix过滤器。

### Audio filter: amix

描述	音频混频器，从多个音频输入中创建一个指定持续时间的输出，输入中断持续时间——可以指定输入之间的转换时间
语法	amix=inputs=ins[:duration=dur[:dropout_transition=dt]]
	参数描述
inputs	输入的数量，默认值是2。
duration	指定如何确定流的结束，可用选项为(下面对应着序列号是0~2): 1.最长持续时间最长的输入， 默认值 2.最短时间的最短输入 3.第一次输入的持续时长
dropout_transition	当输入流结束时，用于进行卷重正化的转换时， 默认值为2

例如，下一个命令将4个输入音频文件混合到一个，它的持续时间与最长输入的持续时间相同，而特定声音输入之间的转换是5秒：

```
ffmpeg -i sound1.wav -i sound2.wav -i sound3.wav -i sound4.wav ^ -filter_complex  
amix=inputs=4:dropout_transition=5 sounds.wav
```

我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/DYZDJ.mp3 -i  
/Users/zhangfangtao/Desktop/NWDSL.mp3 -filter_complex  
amix=inputs=2:dropout_transition=5 /Users/zhangfangtao/Desktop/NWDSL2.mp3
```

- 结果是生成了一个新的音乐文件，时长是这两个里面最长的那个时间，而且，里面的声音也是时长比较长的那个音乐的声音，音乐信息确实前者的。

\*下面的是添加了一些添加了结束时间的参数，具体的命令如下：

```
ffmpeg -i /Users/zhangfangtao/Desktop/DYZDJ.mp3 -i
/Users/zhangfangtao/Desktop/NWDSL.mp3 -filter_complex
amix=inputs=2:duration=0:dropout_transition=5
/Users/zhangfangtao/Desktop/NWDSL2.mp3
```

- 结果表明：duration可选范围是0, 1, 2, 不过只有当我把参数结果设置成0的时候才会生成一个有最长时长的音频文件，其他的参数类型情况下生成的音乐长度都是0.

## 将立体声调至单声道，环绕立体声

要将立体声声音缩混为单声道声音，我们可以使用表格中所述的平移滤波器：

### Audio filter: pan

描述	根据输入的通道布局混合具有特定增益级别的通道，接着是一组通道定义。典型用途是将立体声改为单声道，将5 + 1声道改为立体声等。声像滤波器还可以重新映射音频流的声音。
语法	pan=layout:channel_def[:channel_def[:channel_def...]]
	主要参数
layout	输出通道布局或通道数
channel_def	通道定义的形式：ch_name = [gain *] in_name [+ [gain *] in_name ...]
	通道定义参数
ch_name	通道定义，通道名称(FL, FR, 等等)或通道数(c0, c1, 等等)
gain	通道的乘法系数，值1保持体积不变
in_name	输入通道使用，指定与ch_name相同的方式，不要混合命名和编号的输入通道

下面的几个例子将立体音响与单声道的声音联系在一起：

- 左右通道和相同的音量混合在一起。

```
ffmpeg -i stereo.wav -af pan=1:c0=0.5*c0+0.5*c1 mono.wav
```

我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/NWDSL.mp3 -af pan=1|c0=0.5*c0+0.5*c1
/Users/zhangfangtao/Desktop/NWDSL2.mp3
```

- 结果是，生成了一个0kb的音乐文件，原因是这个歌曲是单声道的，没有设置的意义

或者使用下面简单的方式：

```
ffmpeg -i stereo.wav -af pan=mono mono.wav
ffmpeg -i stereo.wav -af pan=1 mono.wav
```

\*左通道与更大的音量混合，而不是正确的通道

```
ffmpeg -i stereo.wav -af pan=1:c0=0.6*c0+0.4*c1 mono.wav
```

- 右通道与比左通道更大的音量混合

```
ffmpeg -i stereo.wav -af pan=1:c0=0.7*c0+0.3*c1 mono.wav
```

一个简单的方法，没有过滤器，如何将多通道音频与超过2个通道混合使用，使用-ac[:stream\_specifier]选项，该选项包含一个整数参数，该参数指定输出通道的数量：

```
ffmpeg -i 5_1_surround_sound.wav -ac 2 stereo.wav
```

为了指定其他的参数，比如一个特定通道的增益，我们使用pan过滤器。下一个例子自动减少到立体声，3,4,5或7频道的多通道音频：

```
ffmpeg -i surround.wav -af pan=stereo:^  
FL<FL+0.5*FC+0.6*BL+0.6*SL:FR<FR+0.5*FC+0.6*BR+0.6*SR stereo.wav
```

## 简单的音频分析仪

每个输入音频帧的详细信息由一个ashowinfo过滤器提供，该过滤器输出10个不同的参数，每一个音频帧，并在表中描述。

### Audio filter: ashowinfo

描述	显示每个输入音频帧的一行，其中包含组织到键=值对的参数信息
语法	-af ashinfo
	描述生成的参数
n	帧的序号，从0开始
pts	表示时间戳的输入框，表示为若干时间单位
pts_time	输入框的表示时间戳，表示为若干秒
pos	输入流中的帧位置，值-1表示该参数不可用或没有意义(例如合成音频)
fmt	样本格式名称
chlayout	通道布局(mono, 立体声等)
nb_samples	当前帧中每个通道的采样数
rate	音频帧采样率
checksum	输入帧中所有飞机的Adler-32校验和的十六进制值
plane_checksum	每个输入帧平面的Adler-32校验和的十六进制值，在形式中表示[c0 c1 c2 c3 c5 c6 c7]

因为 `ashinfo` 输出可以很长，所以应该用 `-report` 选项保存到文件中。

```
ffmpeg -report -i audio.wav -af ashinfo -f null /dev/null
```

在44100 Hz中编码10秒的立体声音频的结果，s16在图片中说明：

```
C:\windows\system32\cmd.exe
:s16 chlayout:stereo nb_samples:1152 rate:44100 checksum:4786A0B7 plane_checksum:[4786A0B7]
[Parse_ashinfo_0 @ 01f2fd20] n:379 pts:436608 pts_time:9.90041 pos:158623 fmt:s16 chlayout:stereo nb_samples:1152 rate:44100 checksum:D447B7BD plane_checksum:[D447B7BD]
[Parse_ashinfo_0 @ 01f2fd20] n:380 pts:437760 pts_time:9.92653 pos:159041 fmt:s16 chlayout:stereo nb_samples:1152 rate:44100 checksum:6DE73DD2 plane_checksum:[6DE73DD2]
[Parse_ashinfo_0 @ 01f2fd20] n:381 pts:438912 pts_time:9.95265 pos:159459 fmt:s16 chlayout:stereo nb_samples:1152 rate:44100 checksum:5C175E86 plane_checksum:[5C175E86]
[Parse_ashinfo_0 @ 01f2fd20] n:382 pts:440064 pts_time:9.97878 pos:159877 fmt:s16 chlayout:stereo nb_samples:1152 rate:44100 checksum:0E7EF48F plane_checksum:[0E7EF48F]
[Parse_ashinfo_0 @ 01f2fd20] n:383 pts:441216 pts_time:10.0049 pos:160295 fmt:s16 chlayout:stereo nb_samples:1152 rate:44100 checksum:6C6E9C21 plane_checksum:[6C6E9C21]
[Parse_ashinfo_0 @ 01f2fd20] n:384 pts:442368 pts_time:10.031 pos:160713 fmt:s16 chlayout:stereo nb_samples:1152 rate:44100 checksum:CD9B2E0D plane_checksum:[CD9B2E0D]
size=      0kB time=00:00:10.05 bitrate= 0.0kbytes/s
video:0kB audio:1728kB subtitle:0 global headers:0kB muxing overhead -100.000000%
%
c:\media>
```

我的测试命令：

```
ffmpeg -report -i /Users/zhangfangtao/Desktop/NWDSL.mp3 -af ashowinfo -f null  
/Users/zhangfangtao/Desktop/null
```

- 效果图:

```
:9938D07C plane_checksums: [ 3B3AEB61 93ACE50C ]  
[Parsed_ashowinfo_0 @ 0x7ff017d0f900] n:10533 pts:12132911 pts_time:275.123 pos:  
11036480 fmt:fltp channels:2 chlayout:stereo rate:44100 nb_samples:1152 checksum  
:4907F3E4 plane_checksums: [ F89977AC BA1A7C38 ]  
[Parsed_ashowinfo_0 @ 0x7ff017d0f900] n:10534 pts:12134063 pts_time:275.149 pos:  
11037525 fmt:fltp channels:2 chlayout:stereo rate:44100 nb_samples:1152 checksum  
:823C83AA plane_checksums: [ 4CC0BF17 CDDDC484 ]  
[Parsed_ashowinfo_0 @ 0x7ff017d0f900] n:10535 pts:12135215 pts_time:275.175 pos:  
11038570 fmt:fltp channels:2 chlayout:stereo rate:44100 nb_samples:1152 checksum  
:1313B91E plane_checksums: [ EEA7F24C CCB8C6C3 ]  
[Parsed_ashowinfo_0 @ 0x7ff017d0f900] n:10536 pts:12136367 pts_time:275.201 pos:  
11039614 fmt:fltp channels:2 chlayout:stereo rate:44100 nb_samples:1152 checksum  
:E7BCD19B plane_checksums: [ 0DE6E98B 1D82E801 ]  
[Parsed_ashowinfo_0 @ 0x7ff017d0f900] n:10537 pts:12137519 pts_time:275.227 pos:  
11040659 fmt:fltp channels:2 chlayout:stereo rate:44100 nb_samples:1152 checksum  
:569FAC0A plane_checksums: [ 1ED0CBDC E8B6E01F ]  
[Parsed_ashowinfo_0 @ 0x7ff017d0f900] n:10538 pts:12138671 pts_time:275.253 pos:  
11041704 fmt:fltp channels:2 chlayout:stereo rate:44100 nb_samples:1152 checksum  
:06ADD1F2 plane_checksums: [ 68CFF0D9 5DBFE10A ]  
[Parsed_ashowinfo_0 @ 0x7ff017d0f900] n:10539 pts:12139823 pts_time:275.279 pos:  
11042749 fmt:fltp channels:2 chlayout:stereo rate:44100 nb_samples:1152 checksum  
:1CDA9D66 plane_checksums: [ 260C4E56 98214F10 ]  
[Parsed_ashowinfo_0 @ 0x7ff017d0f900] n:10540 pts:12140975 pts_time:275.306 pos:  
11043794 fmt:fltp channels:2 chlayout:stereo rate:44100 nb_samples:1152 checksum
```

## 调整耳机听音

为了增加输入音频文件的立体声效果，我们可以使用表格中描述的耳垢过滤器：

### Audio filter: earwax

描述	将立体图像的位置从内部(耳机的标准)改变到外部和听众的前方(比如扬声器)。它使用CD音频(44,1 kHz频率)，它插入特殊的提示
语法	-af earwax

例如，在.mp3文件音乐中扩大音频的立体声效果，我们可以使用以下命令：

```
ffmpeg -i music.mp3 -af earwax -q 1 music_headphones.mp3
```

我的测试命令如下：

```
ffmpeg -i /Users/zhangfangtao/Desktop/NWDSL.mp3 -af earwax -q 1  
/Users/zhangfangtao/Desktop/NWDSL2.mp3
```

- 结果就是生成了一个立体声效果更加显著的音乐。。。听起来确实很明显

## 使用-map\_channel选项进行音频修改

`-map_channel` 选项可以更改各种音频参数，其语法为：

```
-map_channel [in_file_id.stream_spec.channel_id|-1][:out_file_id.stream_spec]
```

- 如果 `out_file_id.stream_spec` 参数没有设置，音频通道被映射到所有音频流上
- 如果使用“-1”而不是 `in_file_id.stream_spec.channel_id`，映射是一个静音通道
- `-map_channel` 选项的顺序决定了输出流中通道的顺序，输出通道的布局是从映射的通道数计算出来的(如果使用了一个 `-map_channel` 选项，如果使用了两个 `-map_channel` 选项，等等)
- 如果输入和输出通道布局不匹配(例如两个 “`-map_channel`” 选项和 “`-ac 6`”)，与 `-map_channel` 组合使用的 `-ac` 选项将使通道增益级别得到更新

## 在立体声输入中切换音频通道

若要在立体声音频文件中以正确的通道交换左通道，我们可以使用以下命令：

```
ffmpeg -i stereo.mp3 -map_channel 0.0.1 -map_channel 0.0.0 ch_switch.mp3
```

我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/NWDSL.mp3 -map 0:0 -map_channel 0.0.0:0.0  
-map_channel 0.0.1:0.1 /Users/zhangfangtao/Desktop/NWDSL2.mp3
```

- 结果：生成了一个新的mp3音乐，左右声道换了，听不出来

## 将立体声声音分割成两个不同的流

将立体声输入的2个通道分成2个不同的流，编码为1个输出文件，我们使用命令(MP3只能包含1个音频流，因此输出格式必须是AAC、OGG、WAV等)：

```
ffmpeg -i stereo.mp3 -map 0:0 -map 0:0 -map_channel 0.0.0:0.0 ^ -map_channel  
0.0.1:0.1 output.aac
```

我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/NWDSL.mp3 -map 0:0 -map 0:0 -map_channel  
0.0.0:0.0 -map_channel 0.0.1:0.1 /Users/zhangfangtao/Desktop/NWDSL2.aac
```

- 结果生成了一个新的aac文件，不过新的文件播放起来，像是一种慢放。。。那种声音。。。销魂。。。

## 从立体声输入中调出一个频道

要从输入中关闭特定的通道，我们可以使用 `-map_channel` 选项的-1值。例如，为了使第一个通道从立体声声音中静音，我们可以使用以下命令：

```
ffmpeg -i stereo12.mp3 -map_channel -1 -map_channel 0.0.1 mono2.mp3
```

我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/NWDSL.mp3 -map_channel -1 -map_channel 0.0.1 /Users/zhangfangtao/Desktop/NWDSL2.mp3
```

- 显示的结果是：生成了一个只有右声道的音乐文件。

## 将两个音频流合并到一个多通道流

要将2个音频流连接到1个多通道流，我们可以使用一个有1个可选参数输入的 `amerge` 过滤器，该参数值设置输入文件的数量，默认值为2。所有输入文件都必须使用相同的采样率和文件格式进行编码。例如，将2个文件中的2个 `mono` 声音合并到一个单一的立体声流中，我们可以使用这个命令(总持续时间等于较短输入的持续时间)：

```
ffmpeg -i mono1.mp3 -af amovie=mono2.mp3[2];[in][2]amerge stereo.mp3
```

## 音频流转发与缓冲buffet order控制。

两个音频输入的流同步可以由 `astreamsync` 过滤器控制，它有一个参数，可以由一个可选的表达式和多个变量来设置值。表达式的默认值是`t2-t1`(如下所示)，这意味着总是以较小的时间戳转发流。

### Audio filter: `astreamsync`

描述	转发两个音频流并控制转发缓冲区的顺序
语法	<code>astreamsync[=expr]</code> 如果 <code>expr &lt; 0</code> ，则将第一个流转发，否则将转发第二个流
	在表达式中可以使用的变量
<code>b1, b2</code>	到现在流1和流2的缓冲区的数量
<code>s1, s2</code>	到目前为止，在第1条和第2条上转发的样品数量
<code>t1, t2</code>	stream1和stream2的当前时间戳

## 17-预设编解码器

为了简化在某些编解码器中使用的大量选项，我们可以使用预设置文件，其中的选项更好地格式化并保存以便将来使用。

### 关于预设文件的介绍

预置文件是用于各种选项的文本文件，包括特定的编解码器。它们包含键=值对，每个选项和注释都包含在以#符号开始的行中。

指定预设置文件的选项

选项	编码类型	描述
-apre	audio	对于音频，在Windows上最好使用-fpre选项
-spre	subtitle	对于字幕，在Windows上最好使用-fpre选项
-vpre	video	对于视频，在Windows上最好使用-fpre选项
-fpre	any codec	对于任何编解码器类型，该值都是包含选项的文件名

一个简单的预设置文件mpeg2。ffpreset只能包含1个选项，例如：

```
vcodec=mpeg2video
```

要用mpeg2video编解码器编码一些输入，我们可以使用以下命令：

```
ffmpeg -i input -fpre mpeg2.ffpreset -q 1 MPEG2_video.mpg
```

下一个命令用flv (Flash视频)编解码器编码在网络上的使用：

```
ffmpeg -i input.avi -vcodec flv -f flv -r 29.97 -vf scale=320:240 ^ -aspect 4:3
-b:v 300k -g 160 -cmp dct -subcmp dct -mbd 2 -flags ^ +aic+mv0+mv4 -trellis 1 -
ac 1 -ar 22050 -b:a 56k output.flv
```

该命令很长，并且在命令行上对各种更改进行编辑不容易，因此我们将其修改为名为flv的预设置文件。ffpreset将包含与flv编解码器相关的选项(括号中的注释不是文件的一部分)：

vcodec=flv	(视频编解码器)
b:v=300k	(视频比特率)
g=160	(图片组大小)
mbd=2	(macroblock决策算法)
flags=+aic+mv0+mv4	(aic - h263高级内部编码;总是尝试使用mv=<0,0>;mv4 -使用macroblock的4运动矢量)
trellis=1	(rate失真优化量化)
ac=1	(声道数)
ar=22050	(音频采样率)
b:a=56k	(音频比特率)

现在，具有相同结果的命令将是：

```
ffmpeg -i input.avi -f flv -r 29.97 -vf scale=320:240 -aspect 4:3 ^ -cmp dct -  
subcmp dct -fpre flv.ffpreset output.flv
```

## 预置文件的例子

FFmpeg文档提供了几种常见的预置，并将其描述如下：

libx264-ipod320.ffpreset	libx264-ipod640.ffpreset
vcodec=libx264 vprofile=baseline level=13 maxrate=768000 bufsize=3000000	vcodec=libx264 vprofile=baseline level=30 maxrate=10000000 bufsize=10000000

### libvpx - 1080 p.ffpreset预置文件

```
vcodec=libvpx  
g=120  
lag-in-frames=16  
deadline=good  
cpu-used=0  
vprofile=1  
qmax=51  
qmin=11  
slices=4  
b=2M  
#ignored unless using -pass 2  
maxrate=24M  
minrate=100k  
auto-alt-ref=1  
arnr-maxframes=7  
arnr-strength=5  
arnr-type=centered
```

### libvpx - 1080 p50\_60.ffpreset预置文件

```
vcodec=libvpx  
g=120  
lag-in-frames=25  
deadline=good  
cpu-used=0  
vprofile=1  
qmax=51  
qmin=11  
slices=4  
b=2M  
#ignored unless using -pass 2  
maxrate=24M  
minrate=100k  
auto-alt-ref=1  
arnr-maxframes=7  
arnr-strength=5  
arnr-type=centered
```

### libvpx - 360 p.ffpreset预置文件

```
vcodec=libvpx
g=120
lag-in-frames=16
deadline=good
cpu-used=0
vprofile=0
qmax=63
qmin=0
b=768k
#ignored unless using -pass 2
maxrate=1.5M
minrate=40k
auto-alt-ref=1
arnr-maxframes=7
arnr-strength=5
arnr-type=centered
```

### **libvpx - 720 p.ffpreset预置文件**

```
vcodec=libvpx
g=120
lag-in-frames=16
deadline=good
cpu-used=0
vprofile=0
qmax=51
qmin=11
slices=4
b=2M
#ignored unless using -pass 2
maxrate=24M
minrate=100k
auto-alt-ref=1
arnr-maxframes=7
arnr-strength=5
arnr-type=centered
```

### **libvpx - 720 p50\_60.ffpreset预置文件**

```
vcodec=libvpx
g=120
lag-in-frames=25
deadline=good
cpu-used=0
vprofile=0
qmax=51
qmin=11
slices=4
b=2M
#ignored unless using -pass 2
maxrate=24M
minrate=100k
auto-alt-ref=1
```

```
arnr-maxframes=7  
arnr-strength=5  
arnr-type=centered
```

## 18-隔行视频

隔行扫描是在单色模拟电视开发过程中发明的技术，可以消除旧CRT显示器的闪烁。视频帧被水平划分为规则线，然后划分为2个场，其中第一个场包含奇数行，第二个场包含偶数行。

### NTSC, PAL和SECAM电视标准

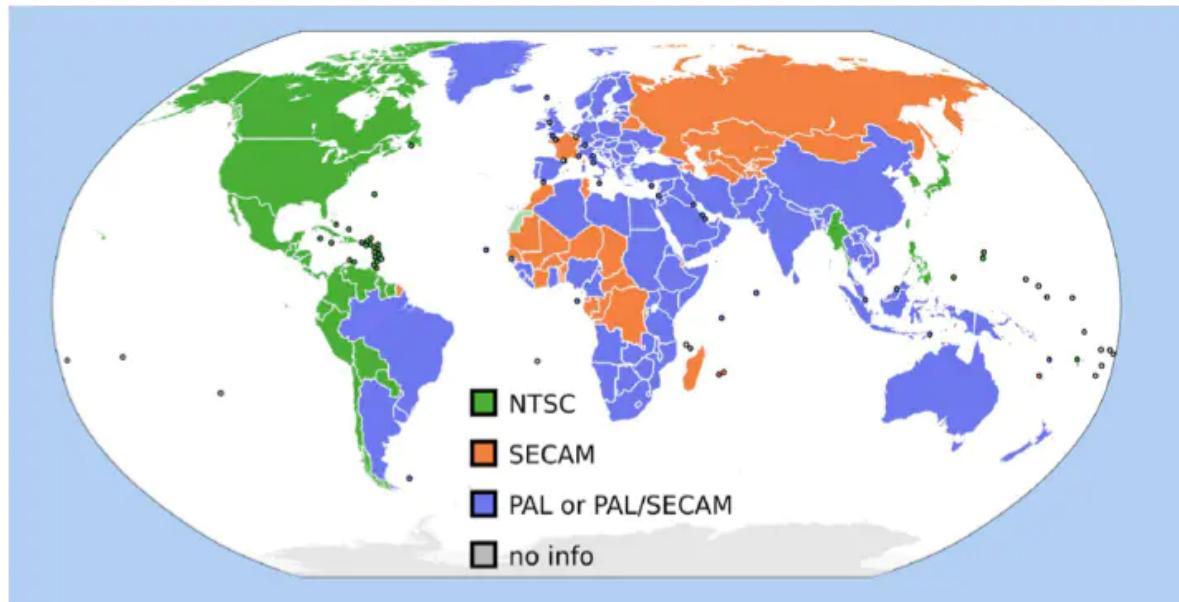
在NTSC标准中，帧有525行，其中483是可见的，其他帧用于同步，垂直回扫等。帧频30fps意味着每秒60场，这对应于美国交流电60Hz频率，并且防止互调，屏幕上滚动条的可能来源。由于120个国家的50赫兹电力频率使用PAL或SECAM标准（非洲，阿根廷，亚洲，澳大利亚，巴西，欧洲等）。这个标准使用25帧/秒的帧率，50场和更高的分辨率，625条扫描线。表中描述了NTSC和PAL / SECAM标准的比较：

- 在电视标准中隔行扫描视频帧

功能	NTSC	PAL, SECAM
扫描的行数	525	625
可见扫描行	483	576
每秒帧数	30	25
每秒场数	60	50

FFmpeg包含多个过滤器和选项，可以更改帧类型和字段顺序，将视频从隔行转换为渐进等。

下图说明了NTSC, PAL和SECAM标准的全球使用情况，但近年来它们已被数字电视标准取代，详情请参阅本章的最后一节。



### 隔行帧类型设置

当使用本章描述的 `fieldorder` 和 `yadif` 过滤器时，在使用多个过滤器进行复杂转码时，使用 `setfield` 过滤器设置输出帧的字段类型可能很有用：

## Video filter: setfield

描述	在输出帧中标记隔行场的类型，帧的内容不变，只更新其属性。对于下一个使用 <b>fieldorder</b> 和 <b>yadif</b> 等过滤器进行处理的过滤链非常有用。
语法	<code>setfield=type</code>
	类型的数值
auto	不要标记任何东西， 默认值
bff	帧首先是底部的场
tff	帧首先是顶部的场
prog	帧是渐进的

例如，要首先将字段类型设置为顶部字段，我们可以使用以下命令：

```
ffmpeg -i input.vob -vf setfield=tff output.mov
```

我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf setfield=tff  
/Users/zhangfangtao/Desktop/test2.mp4
```

- 看不出来有啥不一样。。。。。

## 交错视频的字段顺序更改

以PAL DV格式编码的视频先与底部字段交错，而字段顺序滤镜可以在从其他隔行格式转码或转码时改变它。

## Video filter: fieldorder

描述	将隔行扫描视频输入的字段顺序从底部字段首先改变为顶部字段，反之亦然。变换将帧内容向上或向下移动1行，并用适当的帧内容填充剩余的行。该方法符合大多数广播场序转换器。如果输入不是交错的，或者其字段顺序与命令中设置的相同，则输入不会被更改。
语法	fieldorder[=order_type]
	order_type参数值
0 或者 bff	底部区域优先
1 或者 tff	顶部区域优先，默认值

例如，要将隔行视频从DVD（VOB格式）转换为DV（数字视频）格式，我们可以使用以下命令：

```
ffmpeg -i dvd.vob -vf fieldorder=0 output.dv
```

我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf fieldorder=0
/Users/zhangfangtao/Desktop/test2.mp4
```

## 去隔行

隔行扫描视频是为模拟CRT显示器开发的，不能在像LCD，等离子显示器等渐进式数字显示器上重现。必须使用某些硬件或软件实用工具对其进行去隔行扫描，这意味着将相应的字段合并到完整的视频帧中，到输出视频流。

### yadif过滤器

FFmpeg包含一个名为yadif的特殊滤波器（又一个去隔行滤波器），提供对输入的隔行扫描，但是由于隔行扫描源无法完全恢复，所以导致视频质量低于原始扫描。

Video filter: yadif

描述	<b>yadif = Yet Another Deinterlacing Filter</b>
语法	yadif[=mode[:parity[:auto]]]
	参数
mode	隔行扫描模式下，有4个整数值可用： 0 - 每帧输出1帧， 默认值 1 - 为每个字段输出1帧 2 - 类似于0，但空间隔行扫描检查被跳过 3 - 像1，但是空间隔行检查被跳过
parity	输入隔行视频的图像场奇偶校验，3个整数值可用： 0 - 顶部场优先，如果交错未知，则为默认值 1 - 底部区域优先 -1 - 启用自动检测， 默认值
auto	设置哪些帧是去隔行的，一个布尔值： 0 - 所有帧， 默认值 1 - 仅标记为隔行的帧

例如，要对movie.avi文件进行去隔行处理，我们可以使用以下命令：

```
ffmpeg -i movie.avi -vf yadif movie-progressive.mov
```

我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf yadif
/Users/zhangfangtao/Desktop/test2.mp4
```

## 反隔行选项

此选项没有参数并提供视频帧的解除隔行扫描，但由于质量较差，建议使用 `yadif` 或其他反隔行扫描滤镜。

### 从MPlayer项目中去隔行扫描

MPlayer项目包含多个设计用于解隔行的滤波器，包括 `detc`, `divtc`, `ivtc`, `mcdeint`, `pullup`, `softpulldown`, `softskip` 等。此滤波器使用 `mp` 滤波器的实验包装，质量并非总是最佳。例如，要使用 `ivtc` 过滤器对输入进行隔行扫描，我们可以使用以下命令：

```
ffmpeg -i input.mpg -vf mp=ivtc output.mp4
```

- 我编译的环境不支持 `mp` 这个参数

```
[AVFilterGraph @ 0x7f9f8d521d40] No such filter: 'mp'
Error reinitializing filters!
Failed to inject frame into filter network: Invalid argument
Error while processing the decoded data for stream #0:0
[aac @ 0x7f9f8e001e00] Qavg: 2568.623
[aac @ 0x7f9f8e001e00] 2 frames left in the queue on closing
Conversion failed!
zhangfangtaodeMacBook-Pro:~ zhangfangtao$ ffmpeg -i /Users/zhangfangtao/Desktop/
```

## Pullup过滤器

MPlayer项目的pullup过滤器的设计要比detc或ivtc过滤器更健壮，因为它利用了未来的上下文来进行决策。与ivtc一样，pullup是无状态的，因为它没有锁定一个模式来遵循，但是它期待着下面的字段来识别匹配和重建进程框架。

### Video filter: pullup

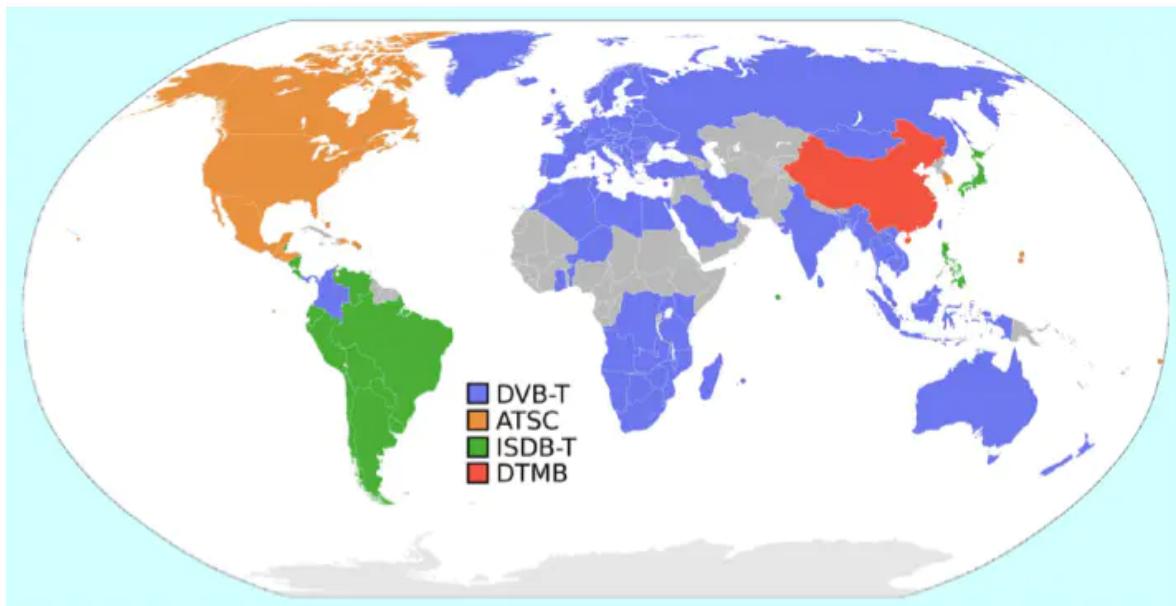
描述	第三代拉低反转(反转电视)过滤器，能够处理混合的硬电视， <b>24000/1001 fps</b> 的进步，和 <b>30000/1001 fps</b> 的进步内容。在编码时，需要使用softskip过滤器，以确保pullup能够看到每个帧。如果不这样做，将导致错误的输出，并且通常会崩溃，这是由于编解码器/过滤层的设计限制
语法	mp=pullup[=jl:jr:jt:jb:sb:mp]
	参数的描述
jt jl jr jb	这些选项设置“垃圾”的数量分别在图像的左边、右边、顶部和底部。左/右为8个像素单元，顶部/底部为2行单元。默认值是每边8个像素
sb	Strick break选项，将其设置为1将减少产生偶尔不匹配的帧的机会，但是它也可能导致在高动作序列中被丢弃的帧数过多。相反，将其设置为1将更容易地使pullup匹配字段。这可能有助于处理在字段之间有轻微模糊的视频，但也可能导致输出中有交错的帧
mp	Metric平面选项，它可以被设置为1或2，以使用色度平面而不是luma面来做拉升的计算。这可能提高非常清洁的源材料的准确性，但更有可能降低准确性，特别是如果有色度噪声(彩虹效应)或任何灰度视频。将mp设置为chroma平面的主要目的是减少CPU负载，并在慢速机器上实时地进行拉升

输入的高度必须能被4整除，建议使用setpts过滤器来更改表示时间戳。例如，在电影中，把一个电视电影的视频从film.vob文件删除，我们可以使用命令：

```
ffmpeg -i film.vob -qscale 2 -vf ^
mp=pullup=4:4:20:20:-1:0,mp=softskip,setpts=N/(24000/1001*TB) ^ -r 24001/1001
film.avi
```

## 交错的视频和数字电视

近年来，使用交错视频格式的模拟电视广播被采用一种渐进格式的数字电视标准取代。数字电视提供更高的质量，更多的频道以相同的带宽传输。虽然主要传输格式是MPEG传输流(MPEG-2第1部分中指定的容器)，但在ATSC和DVB标准中仍然支持交错视频。下一幅图展示了2012年全球数码电视的使用情况：



Comparison of digital TV standards for terrestrial broadcast (colored support interlaced video)				
	ATSC	DTMB	DVB-T	ISDB-T
<b>Complete name</b>	Advanced Television Systems Committee	Digital Terrestrial Multimedia Broadcast	Digital Video Broadcasting - Terrestrial	Integrated Services Digital Broadcasting -Terrestrial
<b>Frequency range</b>	various	various	470–862 MHz 174–230 MHz	470 MHz-770 MHz
<b>Video coding</b>	MPEG-2, MPEG-4	MPEG-2, AVS	MPEG-2, MPEG-4	MPEG-2, MPEG-4
<b>Modulation</b>	8VSB	TDS-OFDM	CP-OFDM	COFDM
<b>Bandwidth per channel</b>	6 MHz	8 MHz	6, 7, 8 MHz	6, 7, 8 MHz
<b>Bitrate</b>	max. 19.39 Mbit/s	from 4.813 Mbit/s to 32.486 Mbit/s	5 - 32 Mbit/s	3.65 - 30.98 Mbit/s
<b>Countries</b>	Canada, South Korea, USA, etc.	Cambodia, China, Hong Kong, Laos, Macau, etc.	Africa (north and south), Asia, Australia, Europe	Japan, Philippines, Thailand, South America

MPEG传输流(MPEG-ts)有一个.ts文件扩展名，它的格式(muxer)是mpegts，所以对于这种格式的多路输出，我们可以使用以下命令：

```
ffmpeg -i input.avi -f mpegts output.ts
```

## 19-FFmpeg组件和项目

FFmpeg项目由4个命令行工具和9个软件库组成，可供许多公司和软件项目使用。 ffmpeg工具的语法和用法在[第一章](#)有相关介绍。

### FFplay介绍

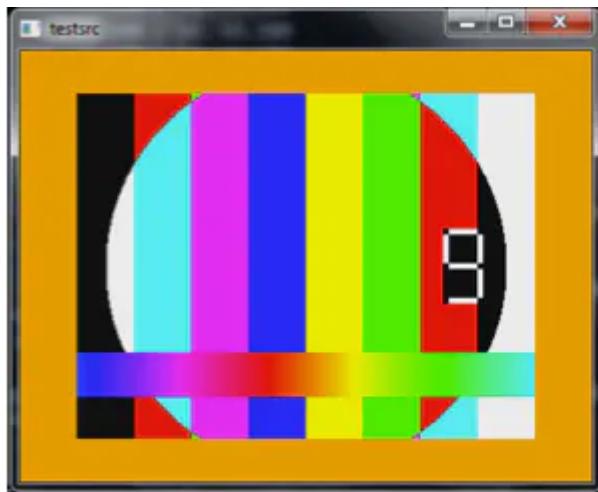
FFplay是一个简单的媒体播放器，能够播放ffmpeg工具可以解码的所有媒体格式，请参阅[第二章](#)显示可用的格式和其他列表。

## FFmpeg组件:ffplay

描述	简单的媒体播放器，使用 <b>FFmpeg</b> 和 <b>SDL库</b> ，它主要用于测试和开发
语法	ffplay [options] [input_file]
	参数描述
options	几乎所有可用于ffmpeg工具的选项都可以与ffplay一起使用
input_file	输入可以是常规文件，管道，网络流，抓取设备等

在显示相同的输出之前，FFplay在编码到文件之前非常有用，请参阅[第一章](#)的显示输出预览部分了解详细信息。例如，要使用ffplay在lightorange背景上显示各种testsrt视频源，我们可以使用以下命令：

```
ffplay -f lavfi -i testsrc -vf pad=400:300:(ow-iw)/2:(oh-ih)/2:orange
```



我的测试命令：

```
ffplay -f lavfi -i testsrc -vf pad=400:300:(ow-iw)/2:(oh-ih)/2:orange
```

- 显示的效果：



如果我们想从文件文档中观看视频。从文件评论中收听mp3格式的音频，我们可以使用以下命令：

```
ffplay -i lavfi "movie=document.avi[out0];amovie=comments.mp3[out1]
```

- 声明一下，书上写错了，至少在我电脑上这样执行是不行的，下面是我的测试命令：

```
ffplay -f lavfi  
"movie=/Users/zhangfangtao/Desktop/test.mp4[out0];amovie=/Users/zhangfangtao  
/Desktop/DYZDJ.mp3[out1]"
```

- 执行结果就是播放视频里面的画面，然而声音是我指定的mp3的声音。

如果FFmpeg被编译为 `-enable-libiec61883` 选项，连接到计算机的FireWire DV/HDV设备的输入可以通过命令显示：

```
ffplay -f iec61883 -i auto
```

- 我电脑上测试报错。也不知道哪儿可以用得着。。。

## 重放期间的键和鼠标控制

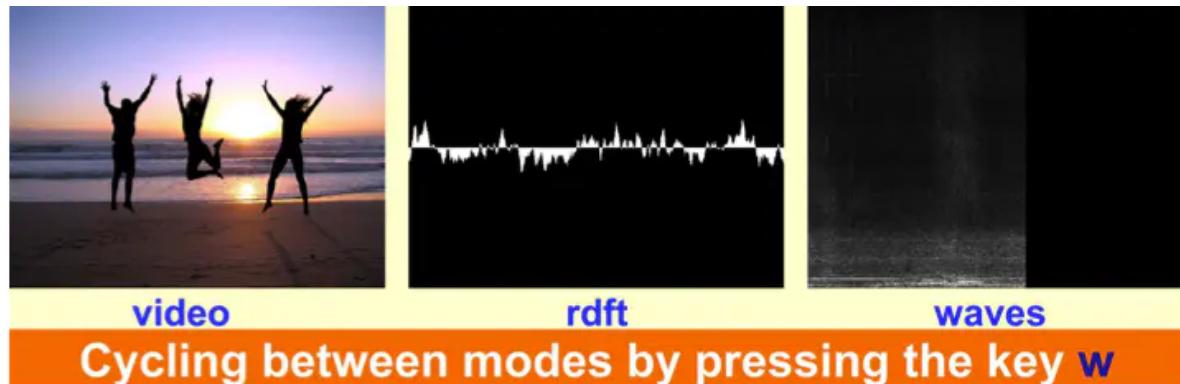
在播放过程中，ffplay可以用键和鼠标控制，细节在下面这个表格里面：

key	描述
q, ESC	退出
f	切换全屏
p, Spacebar	切换暂停
a	音频通道
v	视频通道
t	可用的字幕
w	在可用的显示模式选项中循环:视频, rdft, 音频
向左的箭头 / 向右的箭头	向后/向前拖动10秒钟
向下翻页/向上翻页	向后/向前拖动10分钟
点击鼠标	查找与宽度部分对应的文件中的百分比

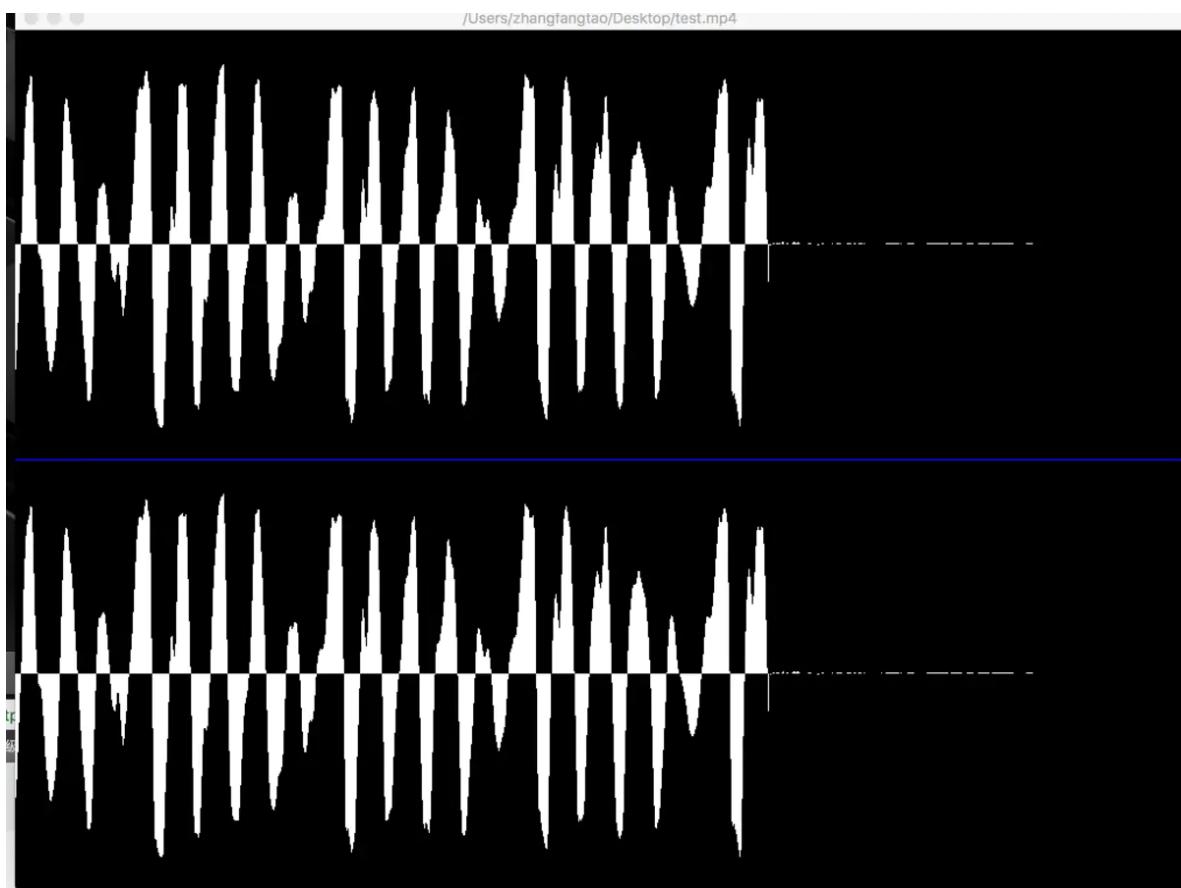
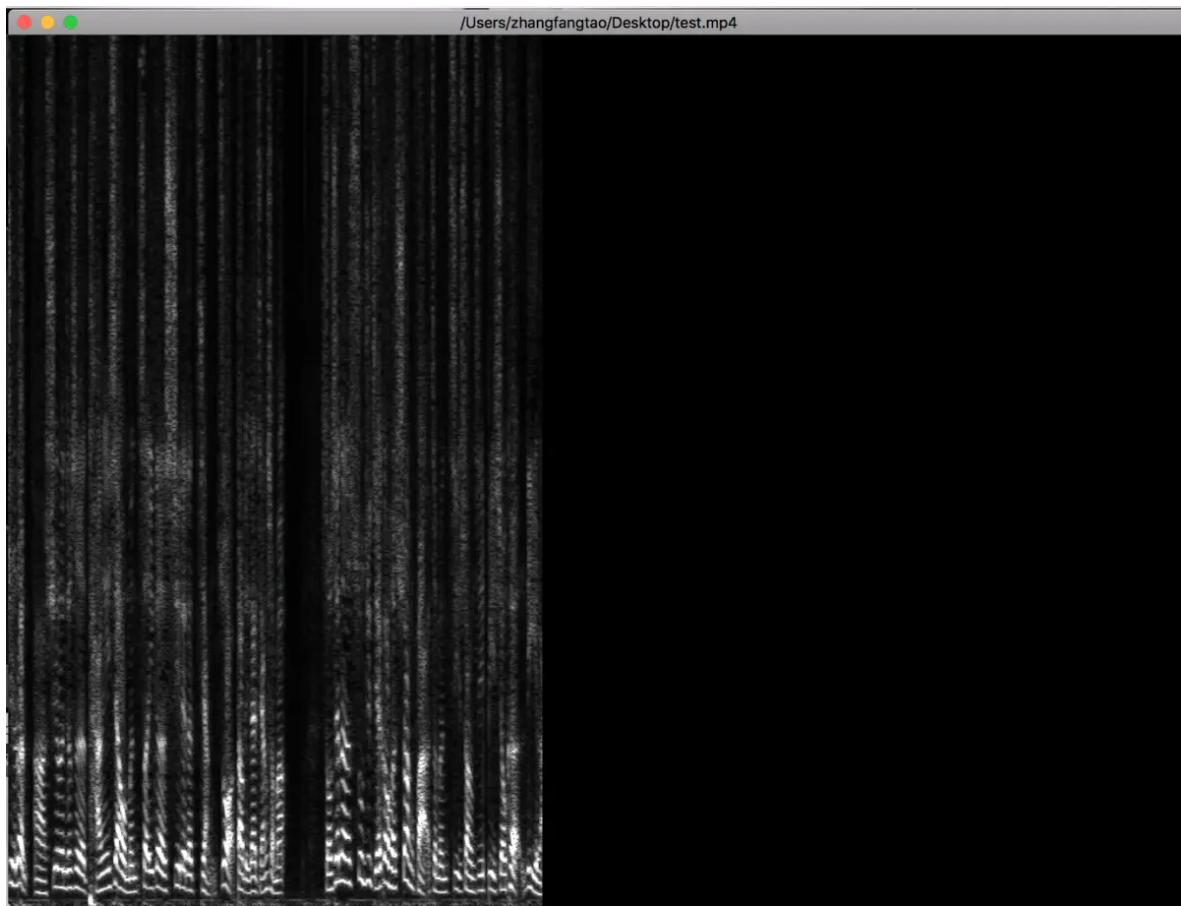
就像MPlayer一样，ffplay也会退出ESC键，并用空格键切换暂停。f键切换全屏模式，但有时它会中止Windows计算机，至少是在旧版本会这样。如果媒体文件包含多个视频流，则可以使用v键循环播放，音频流可以使用t键循环播放一个键和字幕流。按下右箭头键将视频快进10秒，PageUp键视频快进10分钟；箭头左键视频快退10秒，PageDown键退10分钟。鼠标点击事件提供了功能更灵活的时间搜索，我们可以通过点击相应的播放器窗口部分移动到任何部分，例如点击中心将移动到媒体文件的中间。

## FFplay显示模式

当播放视频文件时，ffplay显示视频，它是其 `-showmode` 选项的默认值，其他值是 `rdft`（逆实数离散傅立叶变换）和 `waves`（来自滤波器显示波的音频波）。按w键可在播放过程中切换这些模式：



我的测试结果：



## FFprobe介绍

ffprobe是一种从多媒体流中收集信息并以人机和机器可读方式打印的实用程序。它可用于检查多媒体流使用的容器的格式以及其中每个媒体流的格式和类型。选项用于列出ffprobe支持的一些格式或设置显示哪些信息，并设置ffprobe如何显示它。其输出易于通过文本过滤器进行分析，并且由 `-of`（或 `-print_format`）选项指定的选定writer定义的表单的一个或多个部分组成。ffprobe使用的例子在[调试](#)

和测试章节中。

FFmpeg组件:ffprobe

描述	命令行工具，用于检测多媒体流中的各种数据以进行分析。它可以单独使用或与文本过滤器一起使用，以获得复杂的处理
语法	ffprobe [options] [input_file]
	参数描述
options	几乎所有的ffmpeg工具可用的选项都可以使用ffprobe
input_file	输入可以是常规文件、管道、网络流、抓取设备等
	附加的ffprobe选项
-bitexact	force bit精确输出，用于产生不依赖于特定构建的输出
-count_frames	每个流的帧数，并在相应的流部分报告
-count_packets	每个流的数据包数，并以相应的流部分方式报告设置打印格式， w_name是写入器名称，w_options是写入器选项
-of w_name[=w_options]	设置打印格式，w_name是写入器名称，w_options是写入器选项
-select_streams str_spec	只选择str_spec指定的流，可以是下一个字母:a=audio, d=data, s=subtitle, t=attachment, v=video
-show_data	显示有效负载数据，如hex和ASCII转储，加上- show_数据包，它转储 数据包的数据，再加上-show_streams，它转储codec extradata
-show_error	在探测输入时显示有关发现错误的信息
-show_format	显示有关输入媒体流的容器格式的信息
-show_format_entry name	像-show_format，但只打印由容器格式信息指定的条目，而不是全部
-show_frames	显示输入媒体流中包含的每个框架的信息
- show_library_version	显示与库版本相关的信息
-show_packets	输入媒体流中包含的每个数据包的信息如何
-show_private_data - private	显示数据依赖于特定显示元素的格式，选项是默认启用的，但是可以设 置为0，例如在创建符合xsd的XML输出时
-show_streams	显示输入媒体流中包含的每个媒体流的信息
-show_versions	显示与程序和库版本相关的信息，这相当于设置了- show_program_version和show_library_version选项

## FFserver介绍

ffserver是一个在Linux上运行的多媒体流媒体服务器，官方的Windows二进制文件还不能使用  
FFmpeg组件:ffserver

描述	为音频和视频提供流媒体服务器的实用程序。它支持多个实时供稿，从文件流式传输并在实时供稿上进行时间转换。如果在 <b>ffserver.conf</b> 配置文件中指定了足够的存储源存储，则可以在每个实时供稿中寻找过去的位置。 <b>ffserver</b> 默认在守护进程模式下在 <b>Linux</b> 上运行，这意味着它将自身置于后台并从其控制台分离，除非它以调试模式启动或在配置文件中指定了 <b>NoDaemon</b> 选项。
语法	<code>ffserver [options]</code>
	参数的描述
options	几乎所有的 <b>ffmpeg</b> 工具可用的选项都可以与 <b>ffserver</b> 一起使用
	额外的 <b>ffserver</b> 选项
<code>-d</code>	启用调试模式，这会增加日志的冗余性，将日志消息定向到 <b>stdout</b> ，并导致 <b>ffserver</b> 在前台运行，而不是作为一个守护进程
<code>-f configfile</code>	使用 <b>configfile</b> 而不是 <b>/etc/ffserver.conf</b>
<code>-n</code>	启用无启动模式，这将禁用各种部分的所有启动指令，因为 <b>ffserver</b> 将不会启动任何 <b>ffmpeg</b> 实例，您将不得不手动启动它们

## FFmpeg软件库

### libavcodec

**libavcodec**是一个用于解码和编码多媒体的编解码器库，它非常受欢迎，而**MPlayer**和**VLC**等多平台媒体播放器则用它来播放许多音频和视频格式。它能够解码，并且在某些情况下还可以编码一些专有格式，包括没有官方规范的格式。标准**libavcodec**框架中的这些编解码器提供了优于使用原始编解码器的优势，主要是增加了可移植性，有时还具有更好的性能，因为**libavcodec**包含一个标准库，用于像DCT和色彩空间转换这样的常见构建块的精确优化实现。

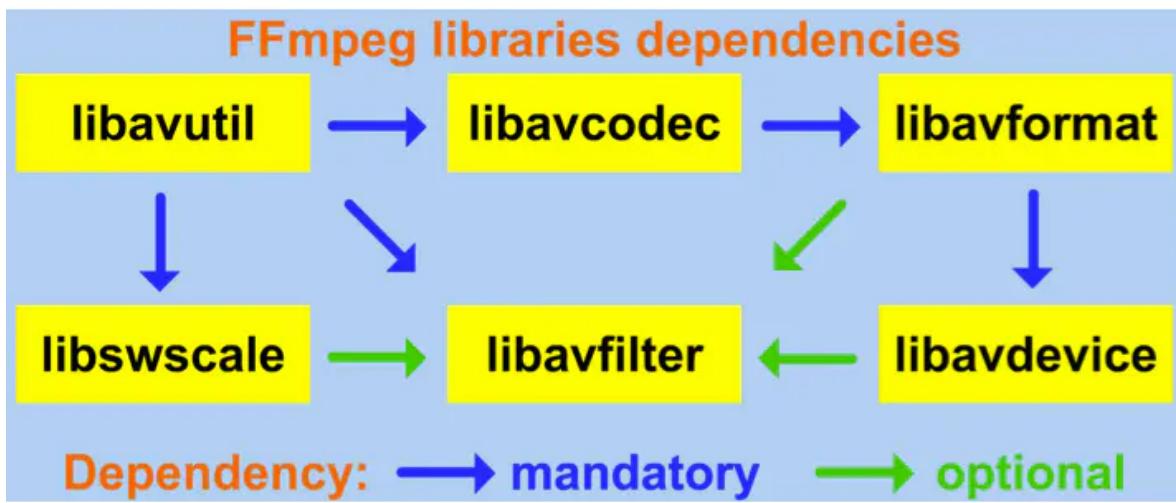
**libavcodec**中已实现的编解码器列表位于“[显示帮助和功能](#)”一章中。

### libavdevice

**libavdevice**是一个特殊的设备复用/解复用库，是**libavformat**库的补充。它提供了各种平台特定的复用器和解复用器，例如用于抓取设备，音频采集和播放。因此**libavdevice**中的(解复用)复用是**AVFMT\_NOFILE**类型（它们使用自己的I/O函数）。传递给**avformat\_open\_input()**的文件名通常不是指实际存在的文件，而是具有特殊的设备特定含义，例如对于**x11grab**设备而言，它是显示名称。可用设备列在[第二章“可用媒体格式”部分](#)。

### libavfilter

**libavfilter**是一个过滤器库，它为FFmpeg和客户端库或应用程序提供媒体过滤层。它简化了FFmpeg工具的设计并增强了它们的灵活性。



libavfilter包含格式协商的高级机制，并最小化像素/格式转换。过滤器处理缓冲区，其中缓冲区可以包含视频帧或音频源。每个缓冲区的属性 - 帧类型，时间戳，文件位置等可以在处理过程中访问和处理。可用过滤器的列表在[第二章](#)中。

## libavformat

libavformat是一个包含音频/视频容器格式的解复用和复用的库。在实现的复用器中有 `crc`, `framecrc`, `ico`, `md5`, `MOV / MP4 / ISMV`, `mpegs`, `matroska` 等。在[第二章](#)中列出可以使用的媒体格式。

## libavutil

libavutil是包含用于FFmpeg的不同部分的例程的辅助库，例如：

- `av_get_token` 函数在 `libavutil/avstring.h` 文件里面，可以用来解析或者转义
- `libavutil/eval.h` 文件包含用于计算算术表达式的接口
- `libavutil/samplefmt.h` 文件包含可用音频样本格式的定义
- `libavutil/audioconvert.h` 文件包含音频通道布局的规范

## libpostproc

libpostproc是一个包含视频后处理例程的库。

## libswresample

libswresample库能够处理不同的采样格式，采样率和不同数量的通道以及不同的通道布局。它支持直接转换样品格式和一次打包/平面。

## libswscale

libswscale是一个包含视频图像缩放例程的库，并提供快速模块化缩放界面。

## 项目使用FFmpeg组件

使用各种FFmpeg工具和库的项目数量很大，其中许多项目列在<http://ffmpeg.org/projects.html>上

## 谷歌Chrome浏览器中支持HTML5

可能是使用FFmpeg库最常用的应用程序是Google Chrome网络浏览器，它是最流行的网络浏览器之一。2009年，FFmpeg库被纳入Chrome，以支持HTML5音频和视频元素。其他使用FFmpeg的浏览器还包括Chromium和Orygin网络浏览器。

## 在YouTube和Facebook上播放视频

最大的视频分享网站YouTube和最大的社交网络Facebook是全球最大的使用ffmpeg的项目，在处理视频方面，每周有几百万个视频。

## 多媒体框架利用FFmpeg

在表中描述了使用FFmpeg库的多媒体框架：

名称	网站	描述
ffdshow	ffdshow-tryout.sourceforge.net	媒体编码器和解码器，实现为DirectShow和VFW过滤器，仅Windows
GStreamer	gstreamer.freedesktop.org	用于构建媒体处理组件图的库。它支持的应用范围从简单的Ogg/Vorbis回放、音频/视频流到复杂的音频(混合)和视频(非线性编辑)处理
MLT	<a href="http://www.mltframework.org">www.mltframework.org</a>	MLT是一个开源的多媒体框架，为电视广播设计和开发。它为广播公司、视频编辑、媒体播放器、转编码器、网络流媒体和其他类型的应用提供了一个工具箱
OpenMAX	<a href="http://www.khronos.org/openmax">www.khronos.org/openmax</a>	OpenMAX是一个免版权的、跨平台的API，它提供了全面的流媒体编码和应用程序的可移植性，它可以使加速的多媒体组件在多个操作系统和硅平台上被开发、集成和编程

### 视频编辑器

- Avidemux
- Blender (3D)
- Cinelerra
- Kdenlive
- Kino

### 音频编辑器

- Audacity
- Sox

## 媒体播放器使用FFmpeg

使用FFmpeg库的媒体框架在表中描述：

名称	网站	描述
Audacious	audacious-media-player.org	Audacious是一个开源的音频播放器。拖放文件夹和个人歌曲文件，在您的整个音乐库中搜索艺术家和专辑，或者创建和编辑您自己的自定义播放列表。从网上听CD或流音乐。用图形化的均衡器或实验来调整声音。享受现代的GTK主题的界面，或者用Winamp经典皮肤改变一些东西。使用包含大胆的插件来为你的音乐取歌词，在早上设置一个闹钟，等等
Gnash	<a href="http://www.gnashdev.org">www.gnashdev.org</a>	Gnash是GNU SWF电影播放器，它可以在桌面或嵌入式设备上独立运行，也可以作为多个浏览器的插件
KMPlayer	<a href="http://kmplayer.kde.org">http://kmplayer.kde.org</a>	KDE的视频播放器插件和基本的MPlayer / Xine / ffmpeg / ffserver / VDR前端。KMPlayer KPart插件用于Konqueror mimics QuickTime, MS Media Player和RealPlayer插件浏览器插件
MPlayer	<a href="http://www.mplayerhq.hu">www.mplayerhq.hu</a>	在许多系统上运行的电影播放器。它播放 MPEG/VOB、AVI、Ogg/OGM、VIVO、ASF/WMA/WMV、QT/MOV/MP4、RealMedia、Matroska、NUT、NuppelVideo、FLI、YUV4MPEG、FILM、RoQ、PVA文件，由许多本机、XAnim和Win32 DLL编解码器支持。你可以观看VideoCD, SVCD, DVD, 3ivx, DivX 3/4/5, WMV, 甚至H.264电影
Rockbox	<a href="http://www.rockbox.org">www.rockbox.org</a>	Rockbox是为数字音乐播放器提供的免费更换固件。它在很多player身上运行
VLC	<a href="http://www.videolan.org/vlc">www.videolan.org/vlc</a>	VLC是一个免费的开源跨平台多媒体播放器和框架，它播放大多数多媒体文件以及DVD、音频CD、VCD和各种流媒体协议
V-Player	vchannel.sourceforge.net/player.html	基于ffmpeg库的跨平台媒体播放器。现在V播放器支持Windows, Linux和OS X平台。用户界面是用c++编写的，所有平台上都有Qt 4
Xine	<a href="http://www.xine-project.org">www.xine-project.org</a>	免费的多媒体播放器。它播放cd、dvd和vcd。它还可以解码本地磁盘驱动器上的AVI、MOV、WMV和MP3等多媒体文件，并在Internet上显示多媒体流

## 20-麦克风和摄像头

麦克风和网络摄像头(网络摄像头)是计算机设备的常用部分，而FFmpeg包含了它们的使用元素。

### 输入设备介绍

FFmpeg可以识别麦克风和网络摄像头等输入设备，这些输入设备被定义为可以从附加的多媒体设备访问数据的元素。在 windows 上，麦克风和网络摄像头可通过 dshow 输入设备进行访问，如表中所述：

#### Input device: dshow

描述	在Windows操作系统上的输入设备，支持的是音频和视频设备
语法	options type=media_type[:type=media_type] [] 中的参数是可选的
	类型参数的描述
type	值可以是视频或音频
	选项参数的可用值
audio_buffer_size	音频设备缓冲区大小（以毫秒为单位）（可直接影响延迟，取决于设备），默认使用设备的默认缓冲区大小（通常为500ms的倍数）。将此值设置得过低可能会降低性能
audio_device_number	具有相同名称的设备的音频设备号(从0开始，默认为0)

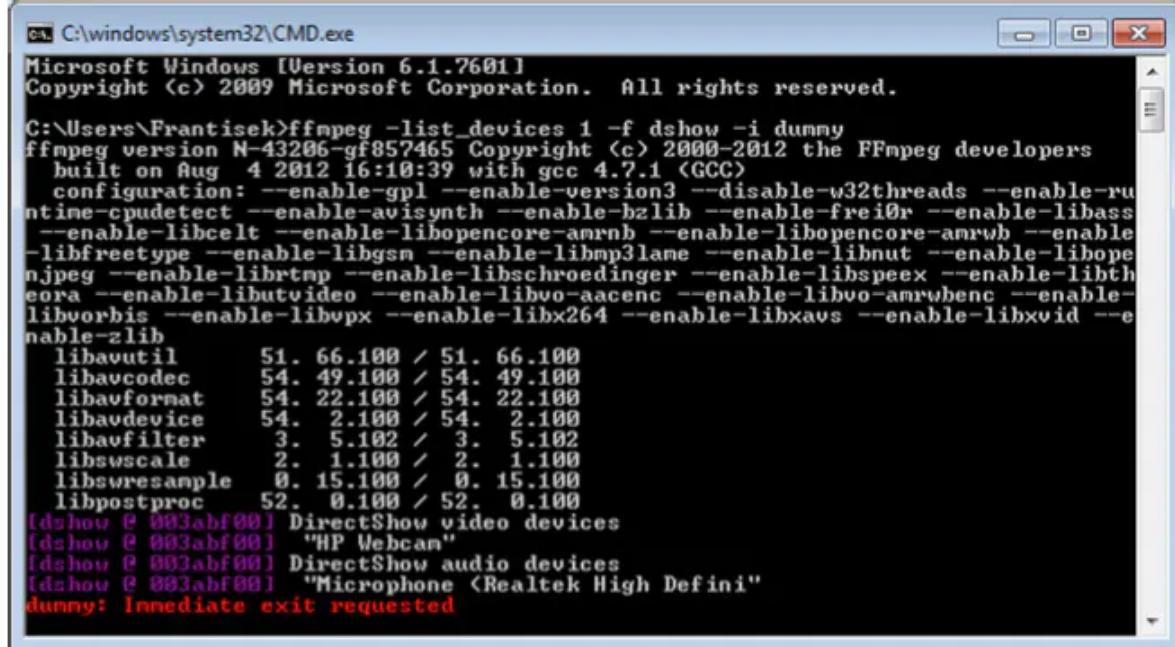
描述	在Windows操作系统上的输入设备，支持的是音频和视频设备
channels	捕获音频中的通道数
framerate	在拍摄的视频中帧频
list_devices	如果设置为1，则打印设备列表并退出
list_options	如果设置为1，则打印选定设备的选项列表并退出
pixel_format	像素格式的使用，只有在视频编解码器没有设置或设置为rawvideo时才能设置
sample_rate	捕获音频的采样率(在Hz中)
sample_size	捕获音频的样本大小(二进制)
video_device_number	同名设备的视频设备编号(从0开始，默认为0)
video_size	视频大小在捕获的视频

## 可用的相机和麦克风的列表

便携式电脑有一个内置的摄像头，或者我们可以通过USB接口连接到电脑。麦克风也经常在电脑里制造，或者我们可以把一个放在电脑的麦克风插孔上，通常是粉红色的，耳机的插孔是绿色的。要在Windows上显示所有可用的输入设备，我们使用dshow设备的list\_devices选项，比如在命令中：

```
ffmpeg -list_devices 1 -f dshow -i dummy
```

输出取决于所使用的计算机，示例输出说明下一个图像：



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Frantisek>ffmpeg -list_devices 1 -f dshow -i dummy
ffmpeg version N-43206-gf857465 Copyright (c) 2009-2012 the FFmpeg developers
  built on Aug 4 2012 16:10:39 with gcc 4.7.1 (GCC)
configuration: --enable-gpl --enable-version3 --disable-w32threads --enable-rustime-cpudetect --enable-avisynth --enable-bzlib --enable-frei0r --enable-libass --enable-libcelt --enable-libopencore-amrnb --enable-libopencore-amrwb --enable-libfreetype --enable-libgsm --enable-libmp3lame --enable-libnut --enable-libopengl --enable-librtmp --enable-libschrödinger --enable-libspeex --enable-libtheora --enable-libutvideo --enable-libvo-aacenc --enable-libvo-amrwbenc --enable-libvorbis --enable-libvpx --enable-libx264 --enable-libxavs --enable-libxvid --enable-zlib
  libavutil      51. 66.100 / 51. 66.100
  libavcodec     54. 49.100 / 54. 49.100
  libavformat    54. 22.100 / 54. 22.100
  libavdevice    54.  2.100 / 54.  2.100
  libavfilter     3.  5.102 /  3.  5.102
  libswscale      2.  1.100 /  2.  1.100
  libswresample   0. 15.100 /  0. 15.100
  libpostproc    52.  0.100 / 52.  0.100
[dshow @ 003abf00] DirectShow video devices
[dshow @ 003abf00] "HP Webcam"
[dshow @ 003abf00] DirectShow audio devices
[dshow @ 003abf00] "Microphone <Realtek High Defini"
dummy: Immediate exit requested
```

输出显示有一个名为“HP webcam”的网络摄像头和一个名为“麦克风(Realtek High Defini)”的麦克风。麦克风的全称是“麦克风(Realtek高清)”，但显示的只有31个字符。

- 因为上面的命令是针对Windows的，所以我的MAC平台下面需要下面的命令：

```
ffmpeg -f avfoundation -list_devices true -i ""
```

- 显示效果如下：

```
ay --enable-libfreetype --enable-libmp3lame --enable-libopencore-amrnb --enable-libopencore-amrwb --enable-libopenjpeg --enable-libopus --enable-libshine --enable-libsnappy --enable-libsoxr --enable-libtheora --enable-libtwolame --enable-libvpx --enable-libwavpack --enable-libwebp --enable-libx264 --enable-libx265 --enable-libxml2 --enable-libzimg --enable-lzma --enable-zlib --enable-gmp --enable-libvidstab --enable-libvorbis --enable-libvo-amrwbenc --enable-libmysofa --enable-libspeex --enable-libxvid --enable-libaom --enable-appkit --enable-avfoundation --enable-coreimage --enable-audiotoolbox  
libavutil      56. 15.100 / 56. 15.100  
libavcodec     58. 19.100 / 58. 19.100  
libavformat    58. 13.100 / 58. 13.100  
libavdevice    58.  4.100 / 58.  4.100  
libavfilter     7. 19.100 /  7. 19.100  
libswscale       5.  2.100 /   5.  2.100  
libswresample    3.  2.100 /   3.  2.100  
libpostproc     55.  2.100 /  55.  2.100  
[AVFoundation input device @ 0x7fcda2d22d80] AVFoundation video devices:  
[AVFoundation input device @ 0x7fcda2d22d80] [0] FaceTime HD Camera  
[AVFoundation input device @ 0x7fcda2d22d80] [1] Capture screen 0  
[AVFoundation input device @ 0x7fcda2d22d80] [2] Capture screen 1  
[AVFoundation input device @ 0x7fcda2d22d80] AVFoundation audio devices:  
[AVFoundation input device @ 0x7fcda2d22d80] [0] Built-in Microphone  
: Input/output error
```

## 可用选项的摄像头

Webcam通常有几种使用-list\_options参数显示的工作模式。要显示来自以前输出的“HP webcam”网络摄像头的选项，我们可以使用以下命令：

```
ffmpeg -list_options true -f dshow -i video="HP Webcam"
```

我的测试命令：

```
ffmpeg -f avfoundation -r 30 -i "1:0" -t 20  
/Users/zhangfangtao/Desktop/test2.mp4
```

- 声明一下，我的facetime没登录，也打不开，所以就录屏了，录了二十秒钟的屏幕

效果图如下：



输出取决于摄像机类型，通常显示可用的分辨率(帧大小)和帧速率：

```
[dshow @ 021bd040] DirectShow video device options
[dshow @ 021bd040] Pin "Capture"
[dshow @ 021bd040] min s=640x480 fps=15 max s=640x480 fps=30
[dshow @ 021bd040] min s=640x480 fps=15 max s=640x480 fps=30
[dshow @ 021bd040] min s=160x120 fps=15 max s=160x120 fps=30
[dshow @ 021bd040] min s=160x120 fps=15 max s=160x120 fps=30
[dshow @ 021bd040] min s=176x144 fps=15 max s=176x144 fps=30
[dshow @ 021bd040] min s=176x144 fps=15 max s=176x144 fps=30
[dshow @ 021bd040] min s=320x240 fps=15 max s=320x240 fps=30
[dshow @ 021bd040] min s=320x240 fps=15 max s=320x240 fps=30
[dshow @ 021bd040] min s=352x288 fps=15 max s=352x288 fps=30
[dshow @ 021bd040] min s=352x288 fps=15 max s=352x288 fps=30
video=HP Webcam: Immediate exit requested
```

## 显示和记录网络摄像头的输入

当我们知道网络摄像头的名称时，我们可以在屏幕上显示它的输入，或者将它记录到文件中。下一个命令显示带有默认设置的webcam输入(通常是最大小和最大帧速率)，第一个是ffplay媒体播放器，第二个是SDL输出设备：

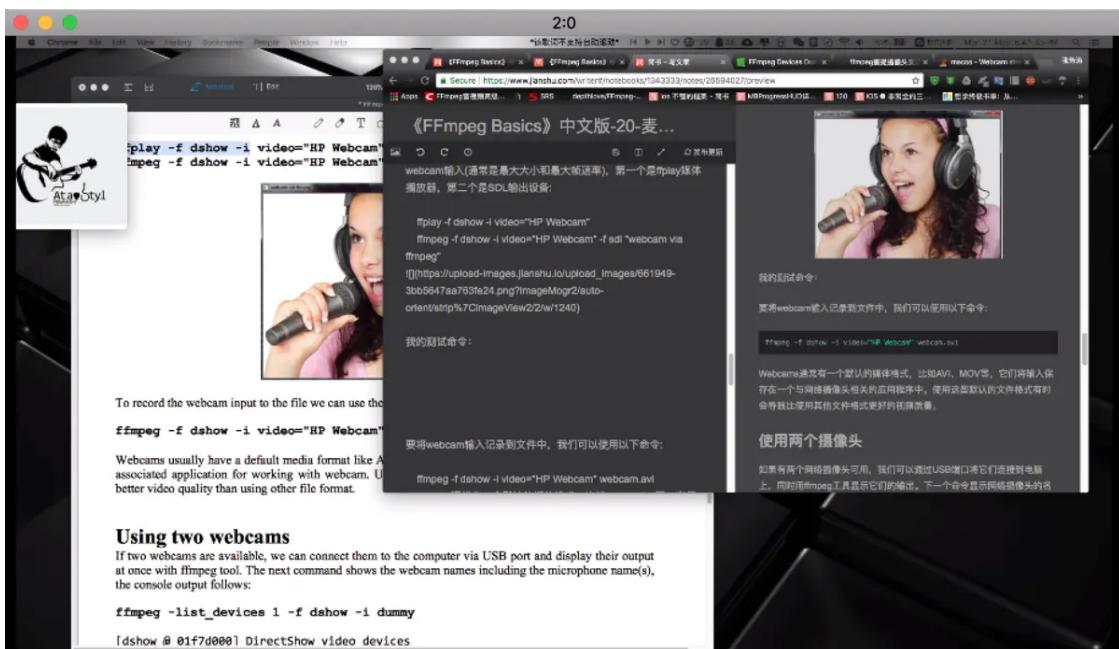
```
ffplay -f dshow -i video="HP Webcam"
ffmpeg -f dshow -i video="HP Webcam" -f sdl "webcam via ffmpeg"
```



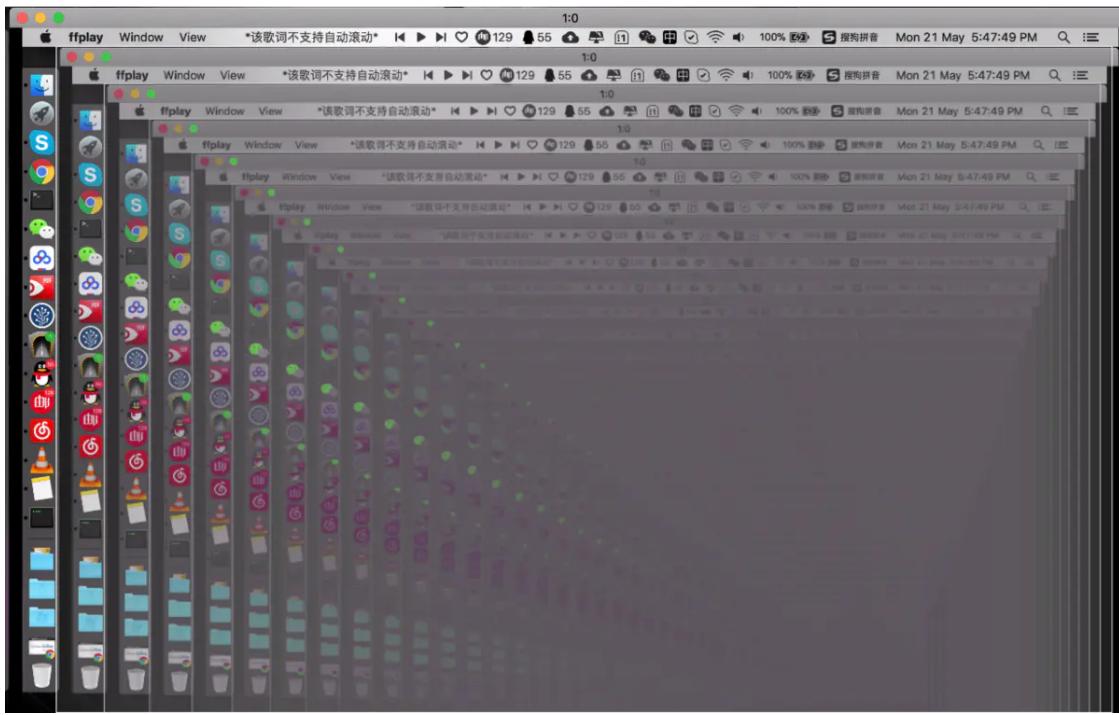
我的测试命令：

```
ffplay -f avfoundation -i "2:0"
```

- 录制的副屏的屏幕信息



- 解释一下，录制的是我的副屏屏幕信息。如果录制主屏幕信息，就是另外一个样子了。。。。：



要将webcam输入记录到文件中，我们可以使用以下命令：

```
ffmpeg -f dshow -i video="HP Webcam" webcam.avi
```

我的测试命令：

```
ffmpeg -f avfoundation -r 30 -i "0:0" -t 30  
/Users/zhangfangtao/Desktop/test2.mp4
```

- 结果就是生成了一个30秒钟的视频文件，视频内容就是FaceTime里面的内容。
- 这里说明一下，我一开始是想生成一个10MB的视频文件，结果都录了二十多分钟了，还没有停下来的意思，我就强制终止了，结果那时候视频已经将近五十兆了。。。大小设置真的很不准、。。。

Webcams通常有一个默认的媒体格式，比如AVI、MOV等，它们将输入保存在一个与网络摄像头相关的应用程序中。使用这些默认的文件格式有时会导致比使用其他文件格式更好的视频质量。

## 使用两个摄像头

如果有两个网络摄像头可用，我们可以通过USB端口将它们连接到电脑上，同时用ffmpeg工具显示它们的输出。下一个命令显示网络摄像头的名称，包括麦克风名，控制台输出如下：

```
ffmpeg -list_devices 1 -f dshow -i dummy
```

```
[dshow @ 01f7d000] DirectShow video devices
[dshow @ 01f7d000] "Sirius USB2.0 Camera"
[dshow @ 01f7d000] "HP Webcam"
[dshow @ 01f7d000] DirectShow audio devices
[dshow @ 01f7d000] "Microphone (Realtek High Defini"
dummy: Immediate exit requested
```

惠普Webcam的选项和如何显示它们的命令在Webcam的可用选项部分中，下面是第二个Webcam命名为Sirius USB2.0相机的选项：

```
[dshow @ 003fd080] DirectShow video device options
[dshow @ 003fd080] Pin "Capture"
[dshow @ 003fd080] pixel_format=yuyv422 min s=640x480 fps=15 max s=640x480 fps=30
[dshow @ 003fd080] pixel_format=yuyv422 min s=640x480 fps=15 max s=640x480 fps=30
[dshow @ 003fd080] pixel_format=yuyv422 min s=352x288 fps=15 max s=352x288 fps=30
[dshow @ 003fd080] pixel_format=yuyv422 min s=352x288 fps=15 max s=352x288 fps=30
[dshow @ 003fd080] pixel_format=yuyv422 min s=320x240 fps=15 max s=320x240 fps=30
[dshow @ 003fd080] pixel_format=yuyv422 min s=320x240 fps=15 max s=320x240 fps=30
[dshow @ 003fd080] pixel_format=yuyv422 min s=176x144 fps=15 max s=176x144 fps=30
[dshow @ 003fd080] pixel_format=yuyv422 min s=176x144 fps=15 max s=176x144 fps=30
[dshow @ 003fd080] pixel_format=yuyv422 min s=160x120 fps=15 max s=160x120 fps=30
[dshow @ 003fd080] pixel_format=yuyv422 min s=160x120 fps=15 max s=160x120 fps=30
[dshow @ 003fd080] pixel_format=yuyv422 min s=800x600 fps=10 max s=800x600 fps=20
[dshow @ 003fd080] pixel_format=yuyv422 min s=800x600 fps=10 max s=800x600 fps=20
[dshow @ 003fd080] pixel_format=yuyv422 min s=1280x960 fps=5 max s=1280x960 fps=7.5
[dshow @ 003fd080] pixel_format=yuyv422 min s=1280x960 fps=5 max s=1280x960 fps=7.5
[dshow @ 003fd080] pixel_format=yuyv422 min s=1280x1024 fps=5 max s=1280x1024 fps=7.5
[dshow @ 003fd080] pixel_format=yuyv422 min s=1280x1024 fps=5 max s=1280x1024 fps=7.5
[dshow @ 003fd080] pixel_format=yuyv422 min s=1600x1200 fps=2.5 max s=1600x1200 fps=5
[dshow @ 003fd080] pixel_format=yuyv422 min s=1600x1200 fps=2.5 max s=1600x1200 fps=5
[dshow @ 003fd080] vcodec=mjpeg min s=640x480 fps=7.5 max s=640x480 fps=15
[dshow @ 003fd080] vcodec=mjpeg min s=640x480 fps=7.5 max s=640x480 fps=15
[dshow @ 003fd080] vcodec=mjpeg min s=352x288 fps=7.5 max s=352x288 fps=15
[dshow @ 003fd080] vcodec=mjpeg min s=352x288 fps=7.5 max s=352x288 fps=15
[dshow @ 003fd080] vcodec=mjpeg min s=320x240 fps=7.5 max s=320x240 fps=15
[dshow @ 003fd080] vcodec=mjpeg min s=320x240 fps=7.5 max s=320x240 fps=15
[dshow @ 003fd080] vcodec=mjpeg min s=176x144 fps=7.5 max s=176x144 fps=15
[dshow @ 003fd080] vcodec=mjpeg min s=176x144 fps=7.5 max s=176x144 fps=15
[dshow @ 003fd080] vcodec=mjpeg min s=160x120 fps=7.5 max s=160x120 fps=15
[dshow @ 003fd080] vcodec=mjpeg min s=160x120 fps=7.5 max s=160x120 fps=15
[dshow @ 003fd080] vcodec=mjpeg min s=800x600 fps=7.5 max s=800x600 fps=15
[dshow @ 003fd080] vcodec=mjpeg min s=800x600 fps=7.5 max s=800x600 fps=15
[dshow @ 003fd080] vcodec=mjpeg min s=1280x960 fps=7.5 max s=1280x960 fps=15
[dshow @ 003fd080] vcodec=mjpeg min s=1280x960 fps=7.5 max s=1280x960 fps=15
[dshow @ 003fd080] vcodec=mjpeg min s=1280x1024 fps=7.5 max s=1280x1024 fps=15
[dshow @ 003fd080] vcodec=mjpeg min s=1280x1024 fps=7.5 max s=1280x1024 fps=15
[dshow @ 003fd080] vcodec=mjpeg min s=1600x1200 fps=7.5 max s=1600x1200 fps=15
[dshow @ 003fd080] vcodec=mjpeg min s=1600x1200 fps=7.5 max s=1600x1200 fps=15
video=Sirius USB2.0 Camera: Immediate exit requested
```

显示输入网络摄像头我们可以使用覆盖过滤器和因为默认视频大小的摄像头是640 x480像素,我们第二个输入的大小设置为320 x240(qvga)-video\_size选项,该命令(单引号返回错误,只能使用双引号):

```
ffmpeg -f dshow -i "video=Sirius USB2.0 Camera" -f dshow -video_size qvga ^ -i
"video=HP Webcam" -filter_complex overlay -f sdl "2 webcams"
```

前面的命令将第二个webcam输入定位到左上角，将其放置到右下角，我们将宽度和高度参数添加到叠加过滤器中:叠加=W/2:H/2。



## 录音并发送到扬声器

与网络摄像头类似，麦克风也有几个工作模式，这些工作模式显示为-list\_options参数设置为true或1。对于输入参数使用音频类型而不是视频，在前几节中使用-list\_devices选项列出的麦克风的命令是：

```
ffmpeg -list_options 1 -f dshow -i "audio=Microphone (Realtek High Defini"
```

```
[dshow @ 0030d0c0] DirectShow audio device options
[dshow @ 0030d0c0] Pin "Capture"
[dshow @ 0030d0c0] min ch=1 bits=8 rate= 11025 max ch=2 bits=16 rate= 44100
audio=Microphone (Realtek High Defini: Immediate exit requested
```

要将声音从麦克风传到扬声器，我们可以使用以下命令：

```
ffplay -f dshow -i audio="Microphone (Realtek High Defini"
```

将声音记录到音频文件的命令如下：

```
ffmpeg -f dshow -i audio="Microphone (Realtek High Defini" -t 60 mic.mp3
```

从麦克风和网络摄像头录制音频和视频，我们可以使用以下命令：

```
ffmpeg -f dshow -i audio="Microphone (Realtek High Defini":^ video="HP Webcam"
webcam_with_sound.avi
```

## 21-批处理文件

### 批处理文件的优点

FFmpeg工具通常用于各种任务，不容易记住不同的编解码器的所有参数，过滤器等等。将各种命令组合保存到批处理文件中，优化了工作，并将开发提升到下一个级别。批处理文件是带有.bat扩展名的文本文件，在Windows操作系统上主要用于管理任务。它们包含命令，这些命令是按顺序处理的，可以打印各种消息，请求输入等等。

```
@echo off  
ffmpeg -i %1
```

将该文本保存到文件测试中。bat并由下一个命令调用，其中filename是媒体文件，我们希望看到它的属性(。命令中的bat扩展是可选的):

```
test.bat filename or test filename
```

### 批处理文件的命令

可以使用命令帮助或帮助|来显示可用的Windows控制台命令。其中一些是特定于批处理文件的。这些和其他命令在表中描述:

基本批处理文件命令

@	在行开始时使用，然后命令没有响应。例子:@echo off
%n(n是自然数)	在命令行上输入空格分隔的参数的占位符，在批处理文件的名称之后，例如：greeting.bat用两行代码： @echo off Good %1, %2 如果用右边这个命令调用:greeting day friends 显示的结果就是：Good day, friends
:label	GOTO命令的起始点，更改处理顺序，批处理文件示例: line 1 ... line 10 (第1 - 10行包含各种命令) :NewItem line 12 ... line 16 (第12 - 16行包含各种命令) GOTO NewItem 当处理流到达第17行时，GOTO命令将它发送回NewItem标签，并在第12行继续运行
CALL	语法：调用[drive] [path] filename [batch parameters] 调用另一个批处理文件，并且在它的所有命令都准备好之后，继续处理调用文件的下一行。如果被调用的文件不存在，则会显示一条错误消息
CHOICE	停止处理，让用户选择一个选择，通常是或否 语法： CHOICE [/C[:]choices] [/N] [/S] [/T[:c,nn]] [text] /C[:]choices 指定允许的key。默认是YN /N 在提示字符串的末尾不显示choices和? /S 把选择键当作区分大小写 /T[:c,nn] 在秒之后会默认选择到c 文本提示字符串显示
CLS	清除屏幕，控制台输出从顶部继续
ECHO	语法： ECHO [ON   OFF] or ECHO [message] 命令“echo off”在处理过程中停止打印命令，“echo on”，即默认情况下，再次打开它。命令“echo some_text”将在处理过程中打印some_text
FOR	在一组文件中为每个文件运行指定的命令 FOR %%变量 IN(set) DO 命令[命令行设置] %变量 指定一个可替换的参数，该参数值由该命令使用。(set)指定一组或多个文件，可以使用通配符,e.g.(.doc) command(命令行) 指定为每个文件执行的命令。 command-parameters(命令行参数) 指定指定命令的参数或开关。示例命令显示所有TXT文件(在批量中使用%%f, 在cmd命令行以%f形式): FOR %%F IN (.txt) DO type %%F
GOTO label	将处理重定向到指定的标签，请参见:标签示例:上面的命令的
IF	IF [NOT] ERRORLEVEL 数字命令 IF [NOT] string1==string2命令 IF [NOT] EXIST 文件名命令 NOT 指定只有在条件为假时，DOS才应该执行命令 ERRORLEVEL 如果最后一个程序运行返回的退出码等于或大于指定的数字，则number指定一个真实的条件 command 指定在满足条件时执行的命令 string1==string2 如果指定的文本字符串匹配，则指定一个真实的条件 EXIST 如果指定的文件名存在，则filename指定一个真实的条件 IF 命令指定条件处理
PAUSE	停止处理并显示消息：“按任意键继续...”
REM	语法： REM [命令] 用于添加描述和其他信息，在处理过程中不使用
SHIFT	语法： SHIFT [n] (n是一个自然数) 用于通过命令行移动已编号参数的位置，并在批处理文件中使用%1、%2等。批文件名称shift.bat: @ECHO OFF ECHO %1 SHIFT ECHO %1 现在，当我们开始这个文件的时候，它的结果是: First First
START	START ["title"] [/Dpath] [/I] [/MIN] [/MAX] [/SEPARATE   /SHARED] [/LOW   /NORMAL   /HIGH   /REALTIME   /ABOVENORMAL   /BELOWNORMAL] [/WAIT] [/B] [command/program] [参数] 为指定的命令启动一个新窗口。对于所有选项的描述，请输入帮助启动

## 批处理文件的典型用法

- 便携式设备的视频转换
- 音频转换从各种格式到MP3文件的MP3播放器
- 减少在web上使用的帧大小和比特率

批处理文件用于音频和视频处理任务，这些任务经常重复。例如，我们可以将一个快捷方式放在桌面上的yt2mp3文件。bat位于目录C:\media，其中保存从YouTube下载的视频转换成MP3格式。yt2mp3.bat文件包含下一行：

```
@echo off
set /p i=Please enter the name of input file:
set /p o=Please enter the name of output file without MP3 extension:
ffmpeg -i %i% -b:a 128k -ar 44100 %o%.mp3
ffplay %o%.mp3
```

成功转换后，ffplay开始播放创建的MP3文件。

## 音频发生器

ToneGenerator.bat是生成指定音调和持续时间音调的批处理文件。为了区分同一音高的两个音调，每个音都加入了silence.mp3，持续时间是0.2秒，如果我们计划加快速度，它可以是0.3秒或更多秒，命令是：

```
ffmpeg -f lavfi -i aevalsrc=0 -t 0.2 silence.mp3
```

文件ToneGenerator.bat有以下内容(添加行号以方便解释，并没有出现在ToneGenerator中)。蝙蝠计算机文件)：

```
1  @echo off
2  set /p n=Please enter the note name:
3  set /p f=Please enter the frequency:
4  set /p d=Please enter the duration in seconds:
5  ffmpeg -f lavfi -i aevalsrc=sin(%f%*2*PI*t) -t %d% tone%n%_%d%.mp3
6  copy /b tone%n%_%d%.mp3+silence.mp3 tone%n%_%d%.mp3
```

对特定命令行的解释(如果音调单独使用，可以跳过加入silence这个步骤)：

- 1：在批处理文件处理过程中，命令echo off停止显示命令的内容，@符号也不包含此命令的显示。
- 2：命令：set /p variable\_name=text创建一个变量，在作业处理期间，显示一行文本并等待输入键的输入结束。这个字符串"Please enter the note name: "显示并在按下Enter之后，将创建一个新的变量n并包含所输入的值。
- 3：与第2行类似，创建的是带有输入频率的变量f。
- 4：类似于第2行，创建的变量d包含在秒内的音调的持续时间。
- 5：ffmpeg使用lavfi (libavfilter虚拟输入设备)和aevalsrc音频输入设备，通过变量d设定的持续时间来生成由变量f指定的频率的声音。MP3格式的输出文件的名称与注释名称和持续时间相结合。
- 6：在生成的文件中添加了一个带有0.2秒长度的短MP3文件，用于通过/b选项的复制命令来区分音调，它指定了二进制模式。最终文件的名称为“tone+ notename +\_duration+.mp3”，以A4音为例，有1秒的时间，它是noteA4\_1.mp3。

## 创建Jingle Bells (歌曲名)

Jingle Bells 是一款非常流行的冬季恋歌，它的副歌只有5个音符，虽然有些持续的时间很长，这张图片显示了这段文字的顺序：



现在我们生成9种不同的音调与前一节中的ToneGenerator.bat不同：

- E4, 频率329.63 Hz, 持续时间1秒-文件E\_1.mp3。
- E4, 频率329.63 Hz, 持续时间2秒-文件E\_2.mp3。
- E4, 频率329.63 Hz, 持续时间4秒-文件E\_4.mp3。
- G4, 频率392.00 Hz, 持续时间1秒-文件G\_1.mp3。
- G4, 频率392.00 Hz, 持续时间2秒-文件G\_2.mp3。
- C4, 频率261.63 Hz, 持续时间1秒-文件C\_1.mp3。
- D4, 频率293.66 Hz, 持续时间1秒-文件D\_1.mp3。
- D4, 频率293.66 Hz, 持续时间2秒-文件D\_2.mp3。
- F4, 频率349.23 Hz, 持续时间1秒-文件F\_1.mp3。

从文件名中跳过了octave 4。第一行和第二行的音调可以与接下来的两个命令连接在一起，第三个命令连接这两行：

```
copy /b E_1.mp3+E_1.mp3+E_2.mp3+E_1.mp3+E_1.mp3+E_2.mp3+E_1.mp3+^
G_1.mp3+C_1.mp3+D_1.mp3+E_4.mp3 line1.mp3

copy /b F_1.mp3+F_1.mp3+F_1.mp3+F_1.mp3+F_1.mp3+E_1.mp3+E_2.mp3+^
E_1.mp3+D_1.mp3+D_1.mp3+E_1.mp3+D_2.mp3+G_2.mp3 line2.mp3

copy /b line1.mp3+line2.mp3 refrain.mp3
```

另一种方法是修改ToneGenerator.bat文件用来生成编号的文件名，然后立即加入一个批处理文件。下面是修正版的ToneGenerator.bat文件，我们可以用它来生成24个MP3文件，文件名是tone01.mp3,tone02.mp3,...,tone24.mp3：

```
@echo off
echo If the number of notes is over 9, start numbering with 0.
set /p n=Please enter the note number:
set /p f=Please enter the frequency:
set /p d=Please enter the duration in seconds:
ffmpeg -f lavfi -i aevalsrc=sin(%f%*2*PI*t) -t %d% %n%.mp3
copy /b %n%.mp3+silence.mp3 tone%n%.mp3
```

现在我们可以通过FileJoiner连接所有的24个MP3文件。bat批文件与内容：

```
@echo off  
copy /y nul output >nul  
set /p t=Please enter the file type:  
for %%f in (*.%t%) do copy /b output+%%f output  
ren output output.%t%
```

- 第2行创建一个空文件，作为初始文件，从目录复制第一个文件。
- 第3行请求文件扩展名并将其存储在t变量中，将加入这个扩展的文件。
- 第4行使用for循环将文件按顺序复制到文件输出。
- 第5行增加了文件输出的扩展名。

两种方法产生的折射率都比原曲慢，atempo滤波器可以用来调节速度：

```
ffmpeg -i output.mp3 -af atempo=2 refrain.mp3
```

atempo过滤器可以应用更多的时间，细节在章节的时间操作中。

## 简化转换

为Codecs编写的章节介绍了如何使用预设置文件简化转换。如果您经常使用各种预置转换媒体，那么要求特定预置的批处理文件可能有用，一个名为Conversion.bat的简单批处理文件示例如下

```
1 @echo off  
2 echo Please enter 0 as the filename if no preset should be used.  
3 set /p i=Please enter the name of input file:  
4 set /p e=Please enter the output file extension:  
5 set /p o=Please enter the name of output file:  
6 set /p p=Please enter the name of the preset file:  
7 set /p a=Please enter additional parameters:  
8 if %p% == 0 goto NOPRESET  
9 ffmpeg -i %i% -fpre %p%.ffpreset %a% %o%.%e%  
10 exit  
11 :NOPRESET  
12 ffmpeg -i %i% %a% %o%.%e%
```

请参阅前面的章节来描述第1 - 7行

- 第8行使用IF构造来选择带有或没有预设置文件的转换，如果变量p为0，则在标签NOPRESET由于GOTO命令指示后继续处理。
- 如果指定了预置文件，第9行将转换输入。
- 第10行终止处理，因此跳过第12行中的转换。
- 第12行使用可选的附加参数将输入转换为没有预设置的文件。

文件转换。bat可以在许多方面进行修改，例如，包括过滤、更多的输入等等。

## 22- 颜色修正

色彩校正通常表示图像版本，如调整亮度，色彩平衡（红色，绿色和蓝色通道），伽马，色调，饱和度等。FFmpeg中的这些修改是通过为各种滤镜指定适当的参数来提供的，因此包括理论介绍。

## 使用查找表进行视频修改

FFmpeg包含3个视频滤镜，可以生成查找表（LUT），将每个像素分量输入值绑定到输出值。新值将应用于输入视频帧并编码到输出。

### Video filters: lut, lutrgb, lutyuv

这个表格不太好弄，我把它拆了

描述	<b>lut</b> 过滤器创建一个查找表，用于将每个像素分量输入值绑定到输出值并将其应用于输入视频。该滤镜在输入时需要YUV或RGB像素格式。与每个选项相关的确切组件取决于输入中的格式 <b>lutrgb</b> 过滤器与 <b>lut</b> 过滤器相同，但在输入中需要RGB像素格式 <b>lutyuv</b> 滤镜与 <b>lut</b> 滤镜相同，但输入中需要YUV像素格式
语法	<code>lut=[c0=expr[:c1=expr[:c2=expr[:c3=expr]]]] lutrgb=[r=expr[:g=expr[:b=expr[:a=expr]]]] lutyuv=[y=expr[:u=expr[:v=expr[:a=expr]]]]</code>

下面的表格是关于参数的解释

<b>lut</b> 过滤器	<b>lutrgb</b> 过滤器	<b>lutyuv</b> 过滤器
c0 第一个像素的组件	r 红色部分	y Y或亮度组件
c1 第二像素组件	g 绿色部分	u U或Cb组件
c2 第三像素组件	b 蓝色部分	v V或Cr组件
c3 第四,与a相同	a 透明度部分	a 透明度部分

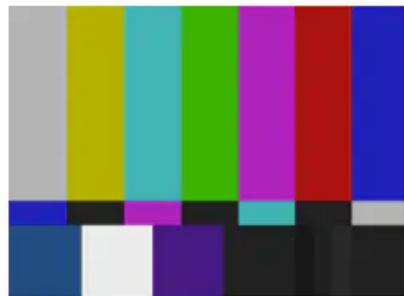
表达式expr中可用的变量和函数。

w, h	输入的宽度和高度
val	像素元素的输入值
clipval	输入值在minval-maxval范围内被剪切
maxval	像素组件的最大值
minval	像素组件的最小值
negval	在minval - maxval范围内对像素组件值进行了否定值; negval = maxval - clipval + minval
clip(val)	在minval - maxval范围内的val中计算值
gammaval(y)	在minval - maxval范围内的计算的伽玛校正值
注意:	所有表达式的默认值为val(像素输入值)，因此默认情况下输出没有变化

## 转换为单色（黑白）图像

将彩色输入更改为仅包含黑白色彩的单色输出，对于单色监视器上的播放非常有用。要在B&W中显示SMPTE条形图，我们可以使用以下命令之一：

```
ffplay -f lavfi -i smptebars -vf lut=c1=128:c2=128  
ffplay -f lavfi -i smptebars -vf lutyuv=u=128:v=128
```



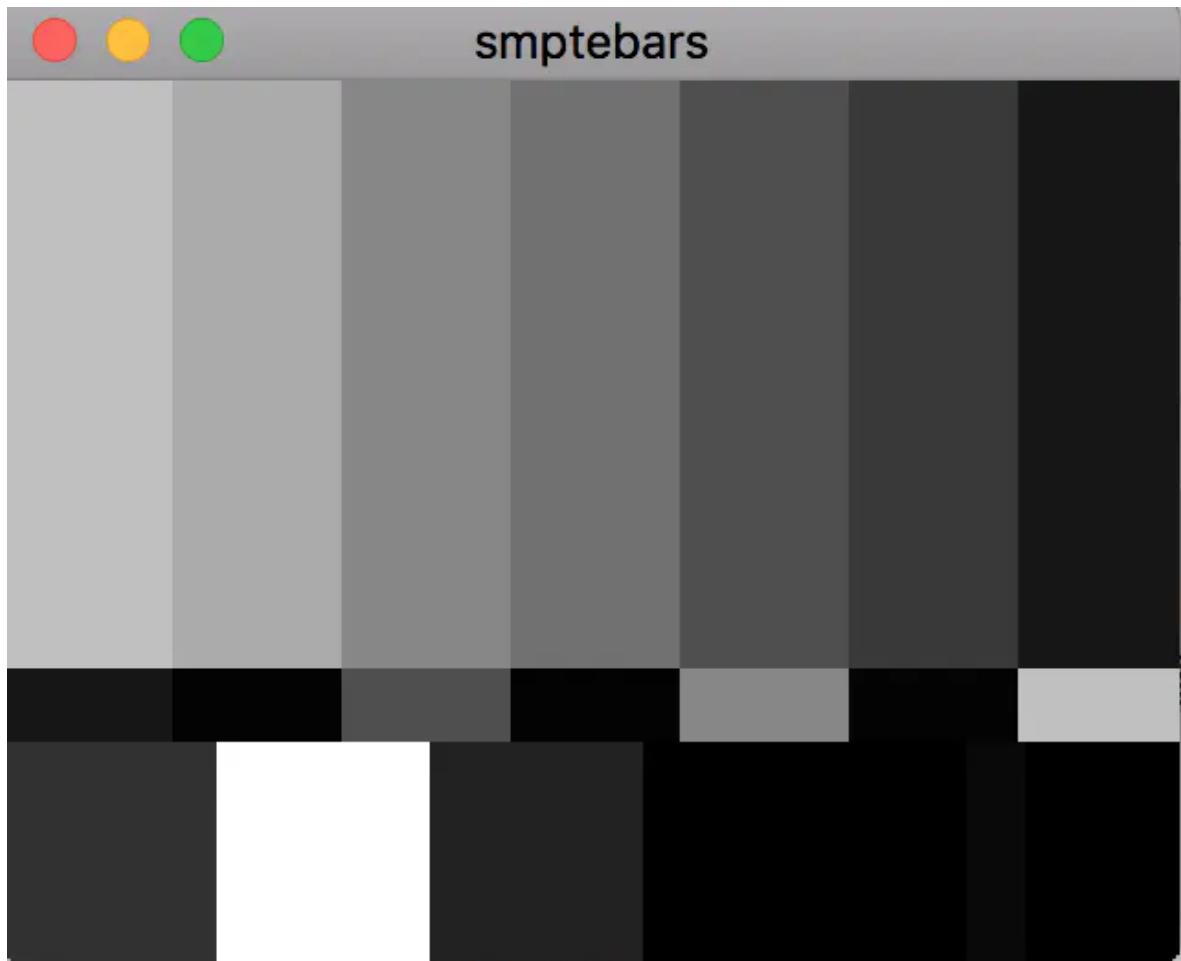
SMPTE bars

Monochrome version

我的测试命令：

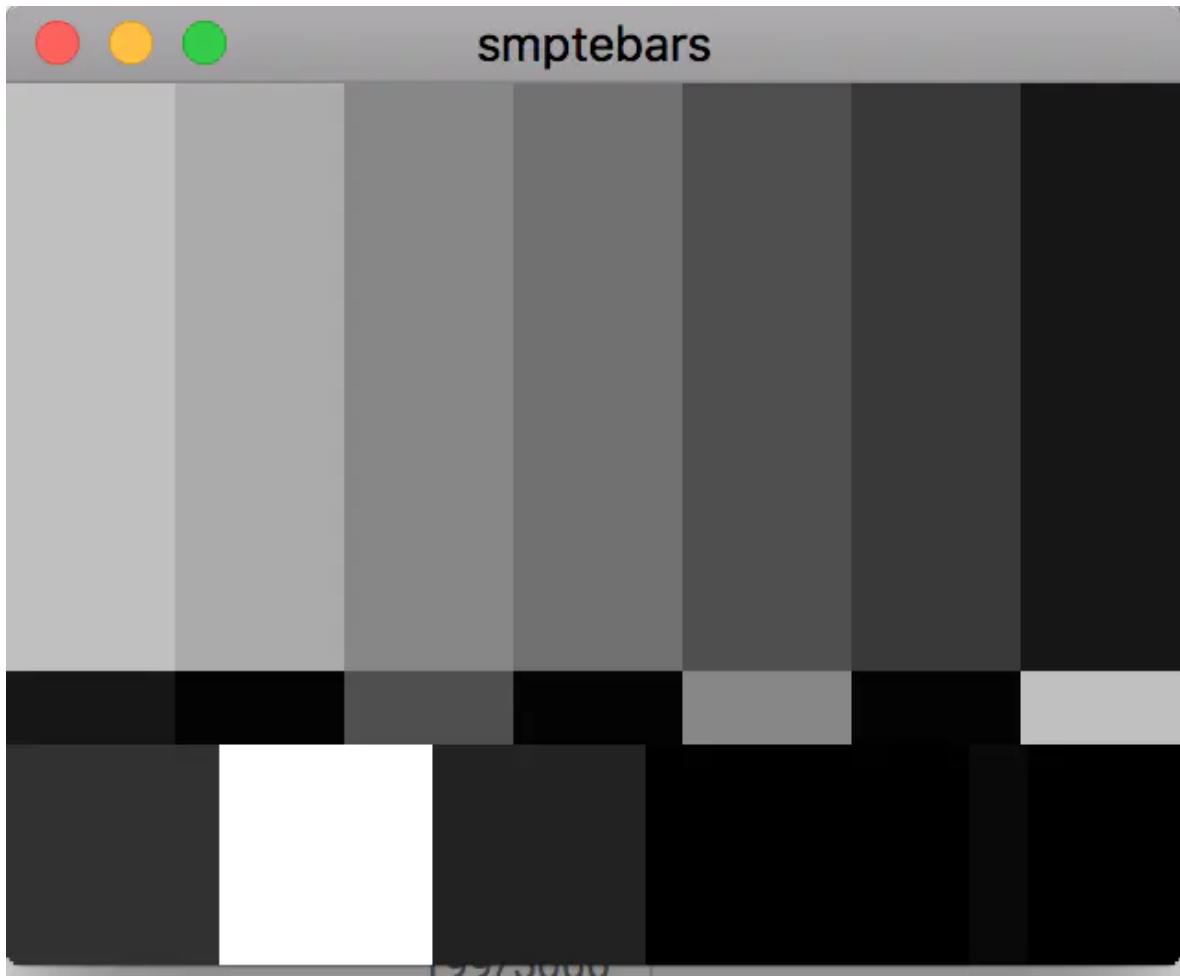
```
ffplay -f lavfi -i smptebars -vf lut=c1=128:c2=128
```

- 显示结果：



```
ffplay -f lavfi -i smptebars -vf lutyuv=u=128:v=128
```

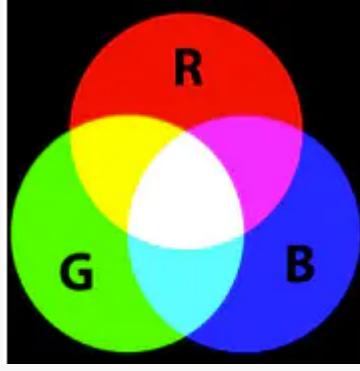
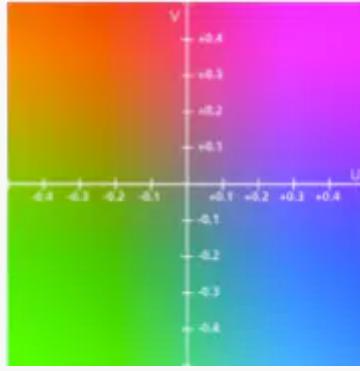
- 显示结果：



## 色彩空间简介

要正确使用 `lutrgb` 和 `lutyuv` 滤镜，请在表格中比较 RGB 和 YUV 色彩空间：

Comparison of RGB and YUV color spaces

	RGB	YUV (Y'CbCr)
描述	添加红色、绿色和蓝色光的加色空间	图像被划分为1个亮度和2个色度分量
组件	R = 红色通道	$Y'$ = luma (亮度)
组件	G = 绿色通道	$U = Y' - B = Cb$ (luma- 蓝色)
组件	B = 蓝色通道	$V = Y' - R = Cr$ (luma - 红色)
用途	电脑、数码相机等	电视、视频等
插图		

所有颜色都可以通过3种基本颜色的组合来创建：红色，绿色和蓝色。为了将这一事实适应数字视频，开发了色彩模型和色彩空间，以指定如何以数字形式呈现色彩的标准。基本颜色空间是RGB（红 - 绿 - 蓝），其中任何颜色表示为混合这三种颜色的各种强度的结果，通常以0到255(256 = 216)或十六进制从`x00`到`fff`。

## YUV色彩空间及其衍生物

当彩电被发明出来的时候，彩色电视就必须在黑白电视机上播放。基于人眼对绿色的敏感程度，对红色的敏感度降低，对蓝色的敏感度更低，开发出了一种新的颜色空间YUV和后来的Y'CbCr。

- Y'是伽马校正的绿色的亮度
- Cr是红色减去亮度的色度分量
- Cb是蓝色减去亮度的色度分量

### Luma(亮度)和chroma(色度)

亮度和亮度分别表示图像的亮度（无色部分），亮度用于视频工程和色彩理论（CIE, ICC等）中的亮度，详情见下表：

Luma and luminance comparison		
	亮度	色度
定义	伽马校正的R'G'B'视频分量的加权总和	线性RGB视频分量的加权总和
符号	Y'（主要符号表示伽马校正）	Y
CCIR 601的公式	$Y' = 0.299 R' + 0.587 G' + 0.114 B'$	$Y = 0.299 R + 0.587 G + 0.114 B$
Rec. BT 709的公式	$Y' = 0.2126 R' + 0.7152 G' + 0.0722 B'$	$Y = 0.2126 R + 0.7152 G + 0.0722 B$
***	R, G和B的系数来自许多人的颜色敏感度测试的平均值	***

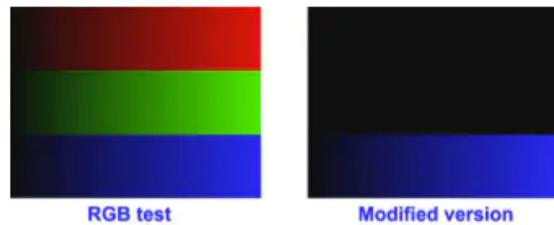
色度和色度表示图像的颜色部分，术语色度主要用于色彩理论，术语色度用于视频工程中，尤其是色度二次采样。色度通常分为两个分量（'表示伽马校正）：

- U = B' - Y'或U = C B（蓝色 - 亮度）
- V = R' - Y'或V = C R（红色 - 亮度）

## 像素格式

色彩空间理论在像素格式的计算机上实现（在[第2章](#)中列出）。常见像素格式包括：rgb8, rgb24, rgba（不透明度为alpha），yuv420p, yuv422p等。例如，要仅显示rgbtests为蓝色，我们将红色和绿色分量设置为零：

```
ffplay -f lavfi -i rgbtests -vf lutrgb=r=0:g=0
```



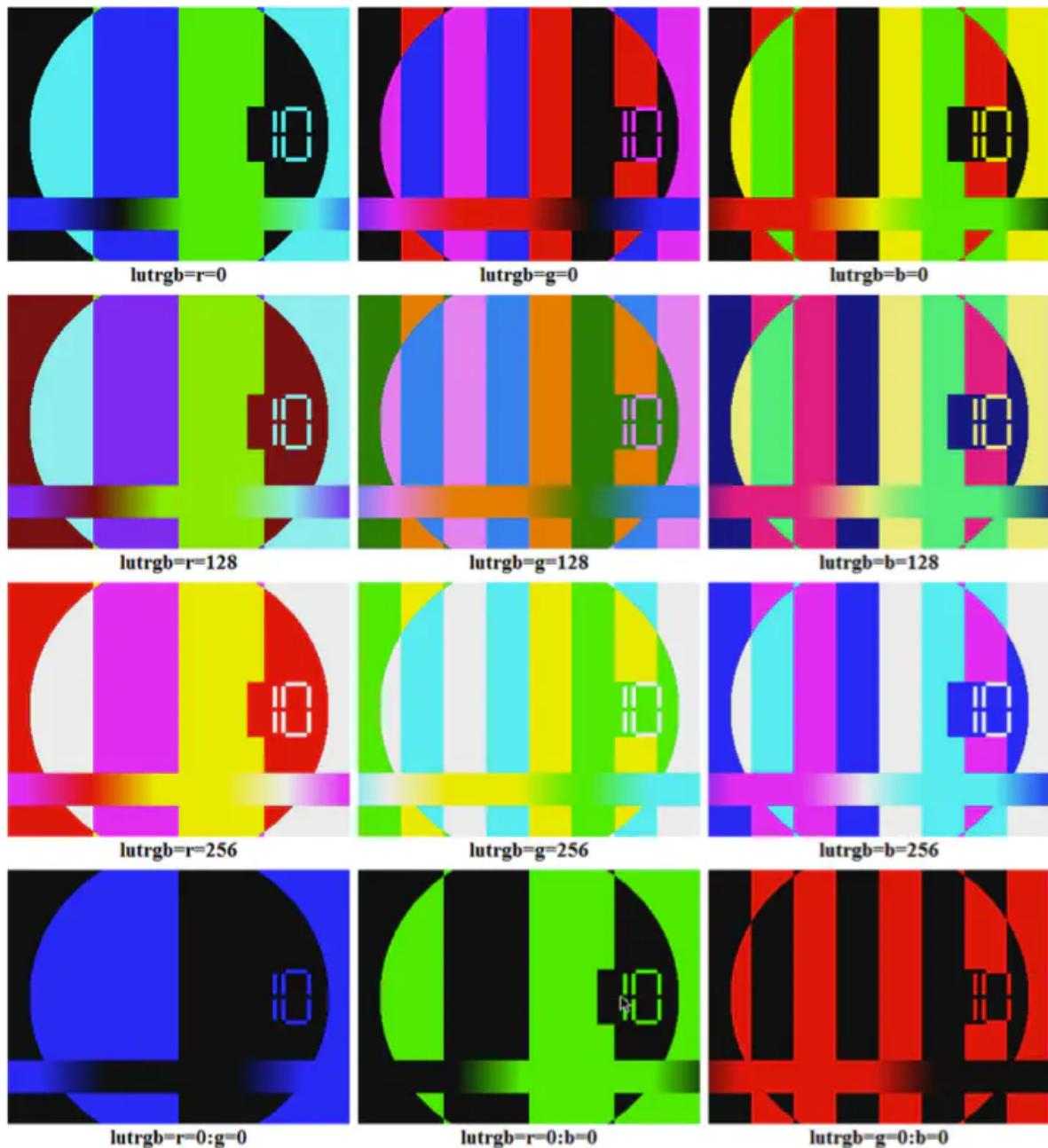
RGB test

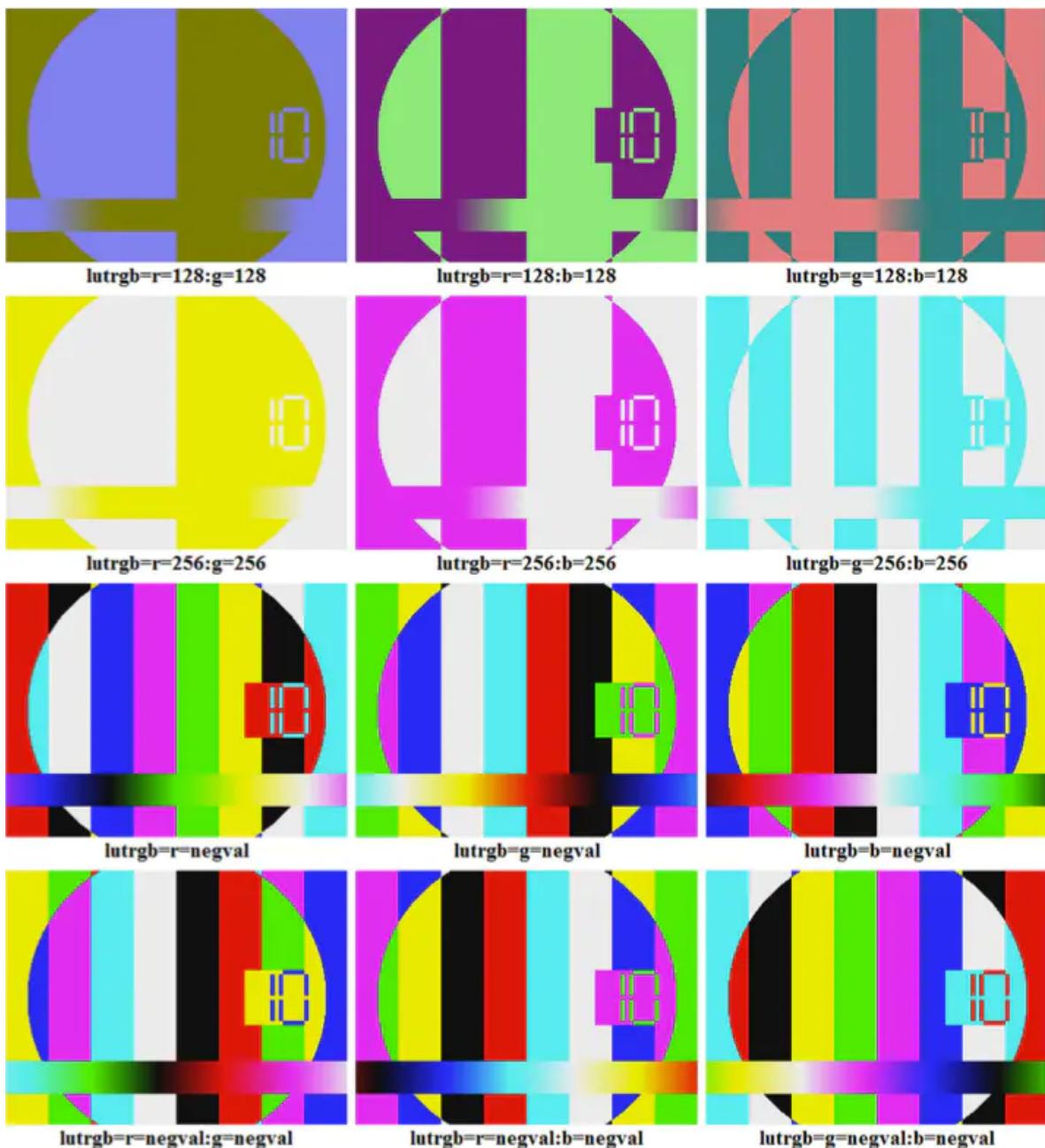
Modified version

- 这个我自己测试了没毛病。

## RGB像素格式修改

要改变RGB输入格式的特定通道，我们使用lutrgb滤波器。它通过将r, g和b参数的值设置为0到255（255以上的任何值被认为是255）来调整色彩平衡，常用组合的使用说明了接下来的两幅图像。



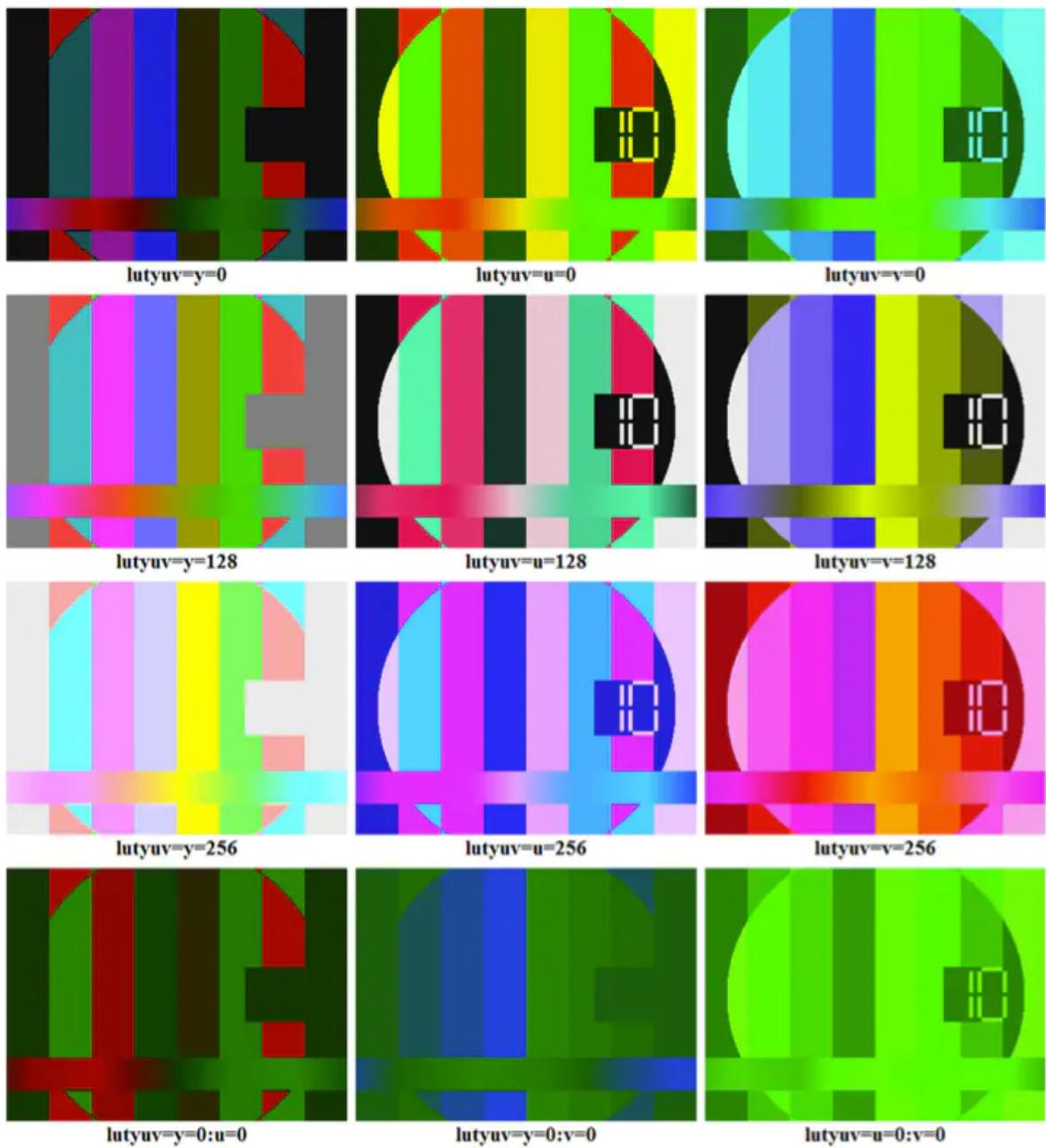


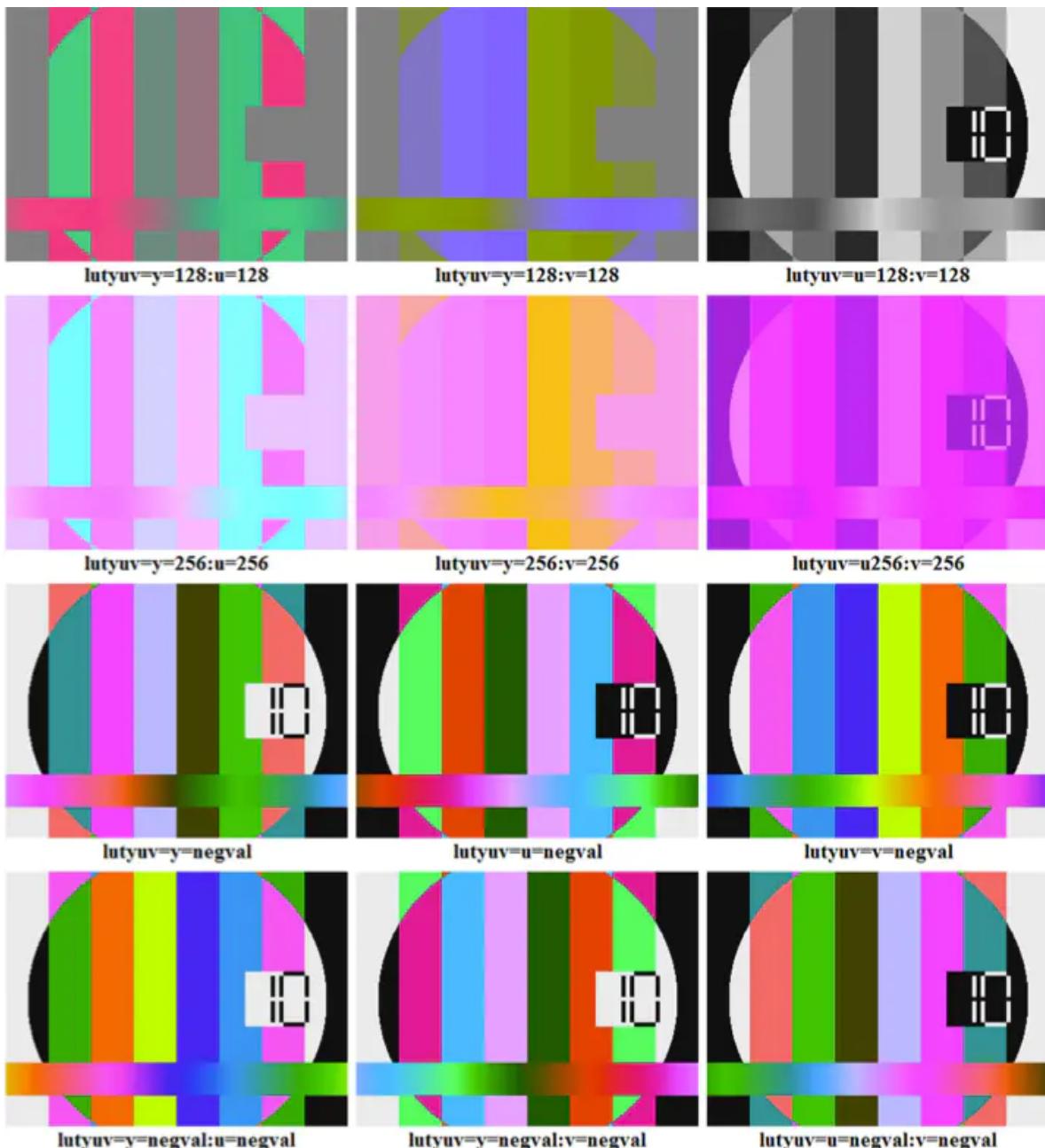
## 色彩均衡

要调整红色，绿色或蓝色通道的亮度，我们设置一个从0到255的数字，并将其输入为lutrgb滤镜的r, g或b参数。我们还可以对输入值进行分（减）或乘（增），例如将蓝色强度加倍，我们可以使用表达式 `lutrgb = b = val * 2`。

## YUV像素格式的修改

要修改YUV格式的组件，我们使用lutyuv过滤器。y参数调整亮度（亮度），u参数调整蓝色平衡，v参数调整红色平衡。这些参数的常见组合说明接下来的两个图像。





## 亮度校正

在RGB颜色模型中，亮度由三种颜色组合而成，在YUV (Y'CbCr)模型中直接与Y (luma)参数设置。例如，要将亮度调整到90%的输入，我们可以使用 `lutyuv=y=val*0.9` 的表达式。

## 色调和饱和度设置

表示RGB色彩空间的另一种方法是HSB (HSV)，色相饱和度 - 亮度 (色调 - 饱和度值) 色彩空间。它使用圆柱坐标系统代替线性立方体，其中色相是围绕中心垂直轴的角度，饱和度是与该轴的距离。对于色调和饱和度调整，FFmpeg提供了一个表格中描述的色调过滤器：

描述	调整输入帧的色调和饱和度
语法	<code>hue[=h=expr[:s=expr]]</code>
	参数的描述
h, H	色度角度, 默认值为0.0
s	浮点数范围从-10到10, 默认值为1.0
	表达式expr中可用的变量
n	输入帧的帧数, 数字从0开始
pts	输入帧的呈现时间戳, 以时基为单位表示
r	输入视频的帧速率, NaN如果未知
t	时间戳用秒表示, NaN如果未知
tb	输入视频的时间基准
	其他语法是 <code>hue = hue: saturation</code> , 其中色调和饱和度是数字, 而不是表达式。

## Hue angles and corresponding colors



色调是从0到360度范围内的角度，并且由CIE定义为“刺激可以被描述为与被描述为红色，绿色，蓝色和黄色的刺激类似或不同的程度”。例如，要将输入的色调调整为60度，我们可以使用以下命令：

```
ffplay -i coconut.jpg -vf hue=60
```



我的测试命令：

```
ffplay -i /Users/zhangfangtao/Desktop/fruit.jpg -vf hue=60
ffplay -i /Users/zhangfangtao/Desktop/fruit.jpg -vf hue=120
ffplay -i /Users/zhangfangtao/Desktop/fruit.jpg -vf hue=180
ffplay -i /Users/zhangfangtao/Desktop/fruit.jpg -vf hue=240
```

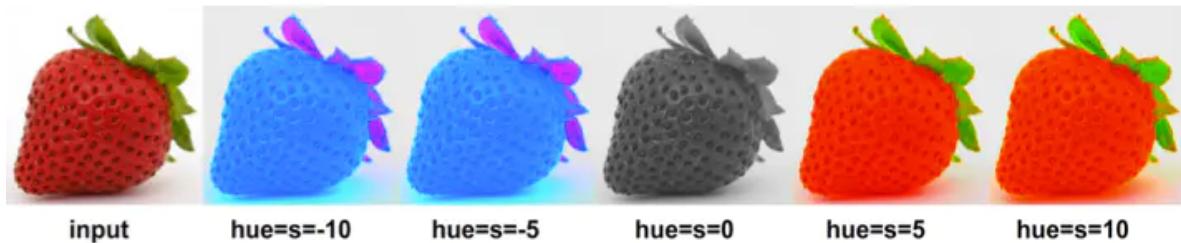
- 显示的效果(从左往右依次是原图, hue=60,120,180,240):



要调整图像饱和度，我们将 `s` 参数设置为足够的值，例如，将饱和度增加到值5，我们可以使用以下命令：

```
ffplay -i strawberry.jpg -vf hue=s=5
```

下一张图片说明了值-10, -5, 0, 5和10的用法。请注意值0导致单色（黑白）图像。



我的测试命令：

```
ffplay -i /Users/zhangfangtao/Desktop/ornage.jpeg -vf hue=s=-10  
ffplay -i /Users/zhangfangtao/Desktop/ornage.jpeg -vf hue=s=-5  
ffplay -i /Users/zhangfangtao/Desktop/ornage.jpeg -vf hue=s=0  
ffplay -i /Users/zhangfangtao/Desktop/ornage.jpeg -vf hue=s=5  
ffplay -i /Users/zhangfangtao/Desktop/ornage.jpeg -vf hue=s=10
```

- 显示的效果(从左向右依次是原图, `hue=s=-10,-5,0,5,10`):



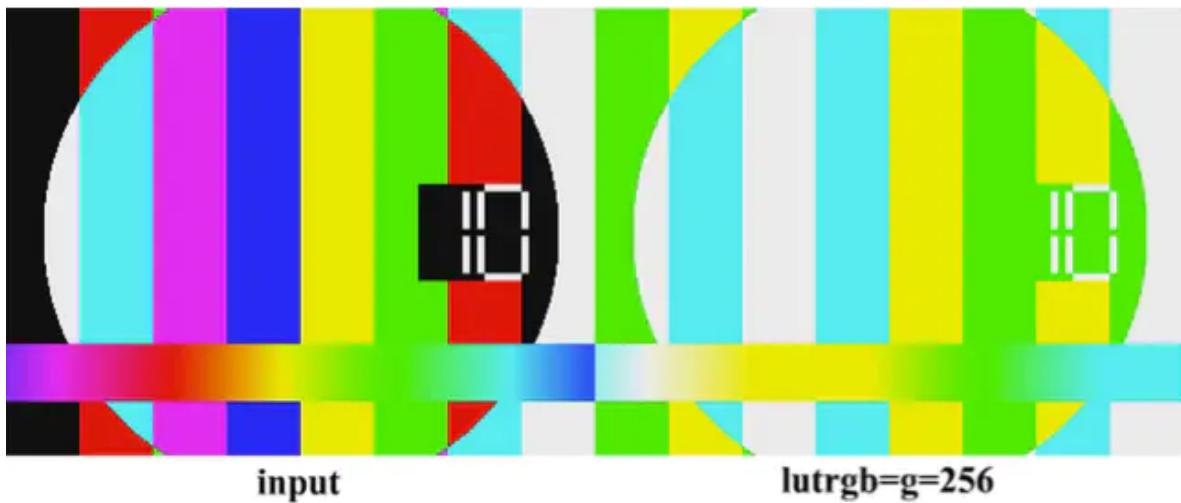
## 在2个窗口中比较

许多图像和视频编辑器在各种设置中提供了第二个窗口来比较输入如何改变。为了提供与FFmpeg类似的比较，我们可以在过滤器图片中使用4个过滤链的过滤器和叠加过滤器。

### 2个窗口水平比较

这种类型的比较已经在[第1章](#)中的过滤器，过滤链和过滤器图单元里面有介绍。第一个过滤链将输入分成两个输出，分别标记为[1]和[2]，第二个过滤链为两个标记为[A]的窗口创建一个填充，第三个将过滤器应用于输出[2]，结果标记为[B]。第四个过滤链将修改后的输入 ([B]) 叠加到新的打击垫上 ([A])。下一个示例使用lutrgb过滤器来说明此方法：

```
ffplay -f lavfi -i testsrc -vf ^ split[1][2];[1]pad=iw*2[A];[2]lutrgb=g=256[B];  
[A][B]overlay=w
```



我的测试命令:

```
ffplay -f lavfi -i testsrc -vf "split[1][2];[1]pad=iw*2[A];[2]lutrgb=g=256[B];
[A][B]overlay=w"
```

- 显示结果

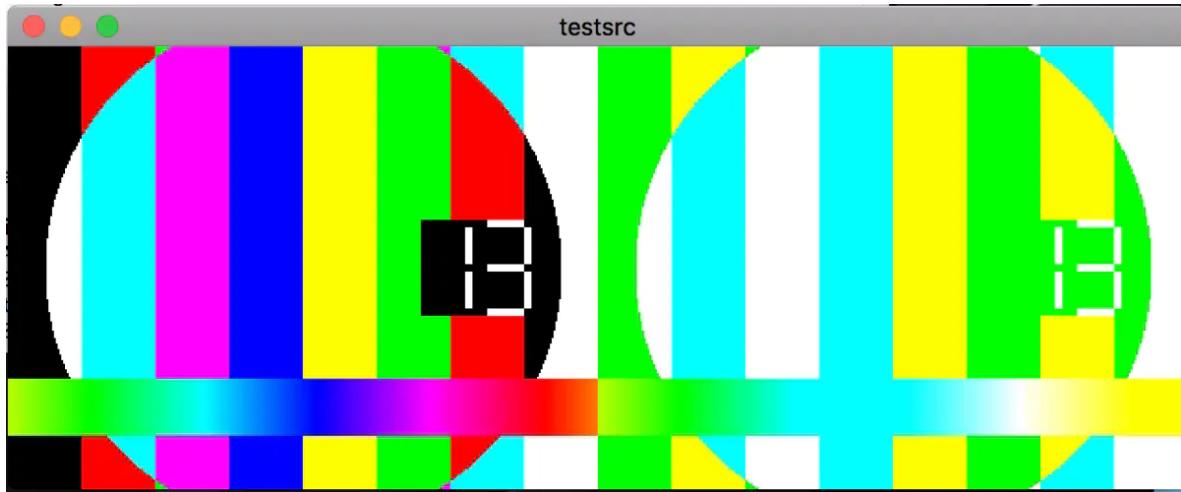
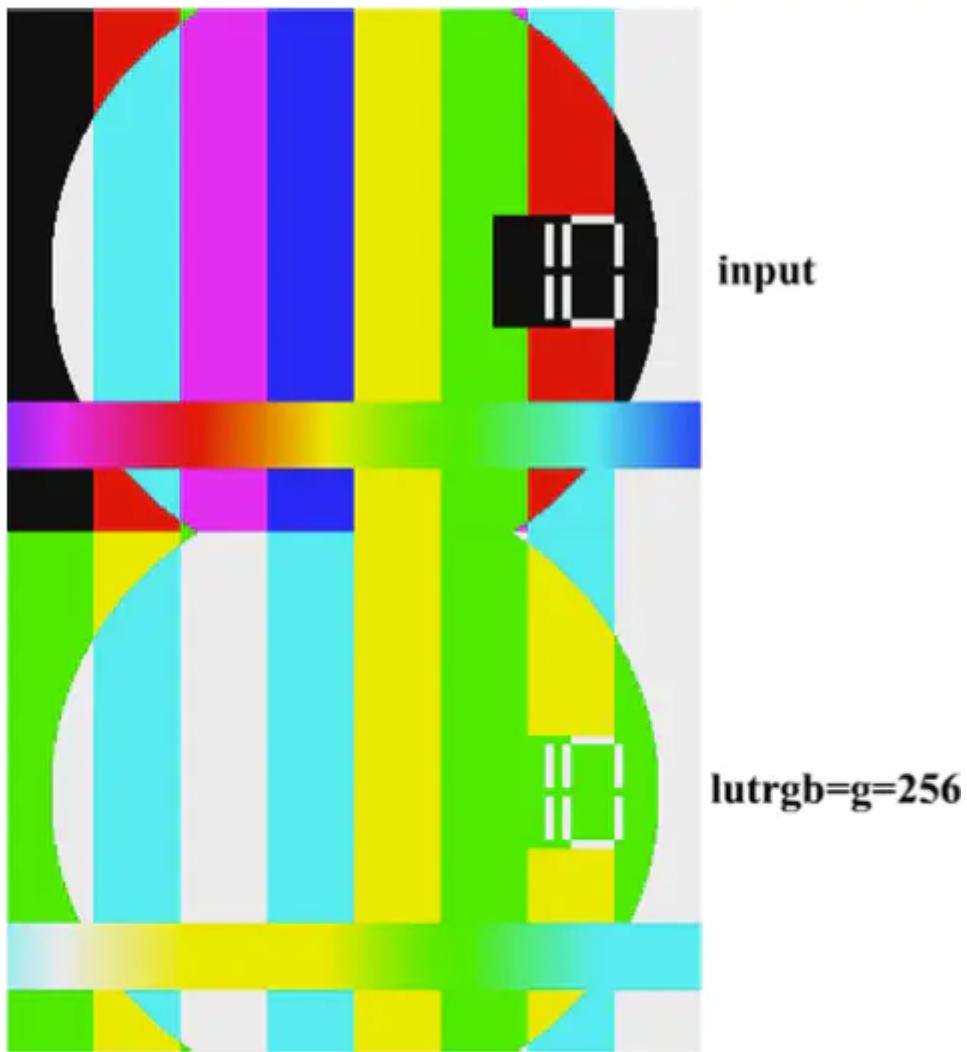


image.png

## 2个窗口垂直比较

为了在垂直的窗口中提供比较，只有第二次和第四个filtergraph被改变，其他参数在水平比较中保持不变。在第二个filtergraph中，我们将ih(输入高度)乘以2作为y参数，在第四个filtergraph中，我们为y参数指定x参数和h(输入高度)为零。为了演示此方法，下一个示例将与前面的相同图像进行比较，只是修改了windows的位置(该更改下划线):

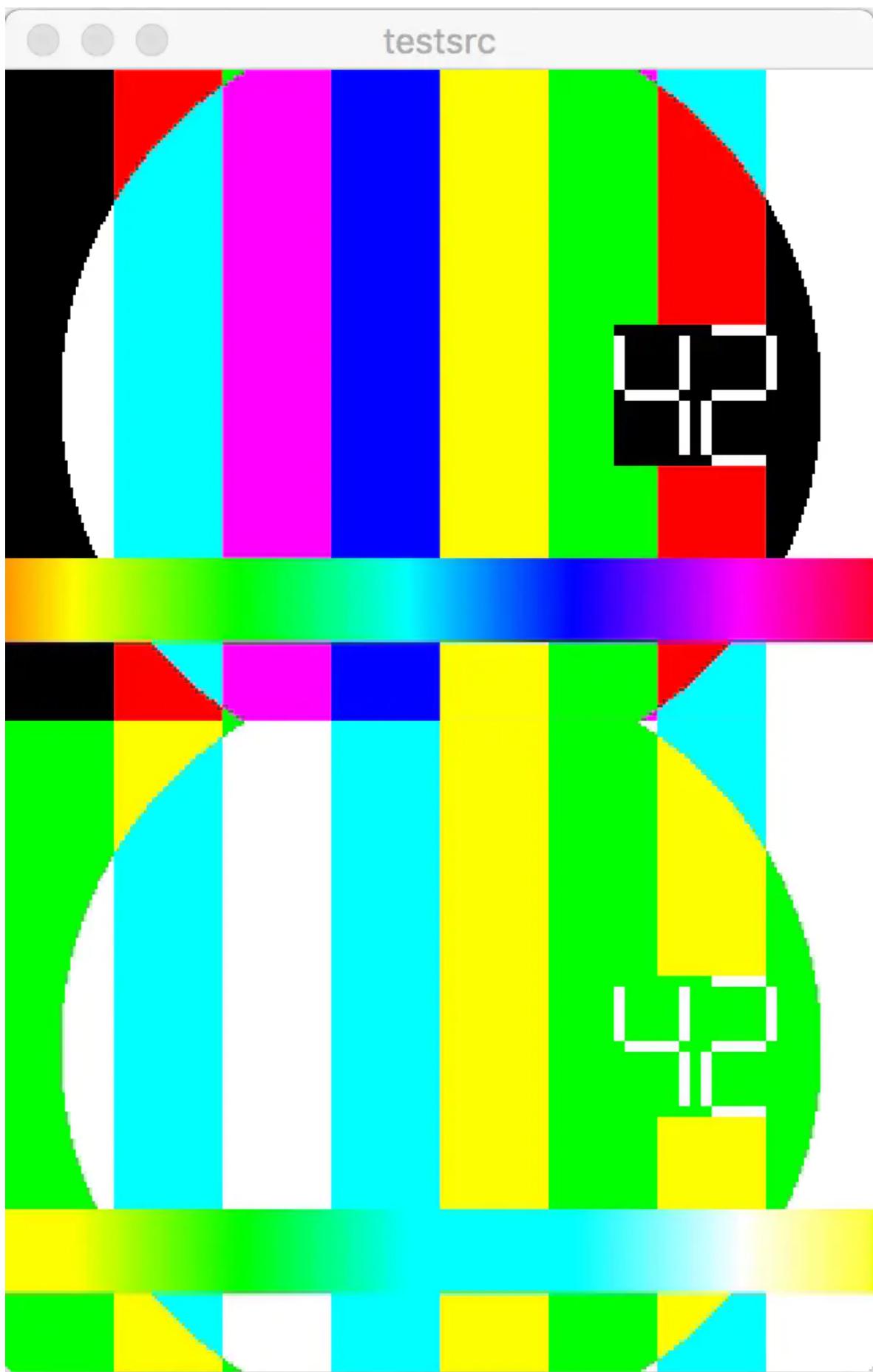
```
ffplay -f lavfi -i testsrc -vf ^ split[1][2];[1]pad=iw:ih*2[A];
[2]lutrgb=g=256[B];[A][B]overlay=0:h
```



我的测试命令:

```
ffplay -f lavfi -i testsrc -vf "split[1][2];[1]pad=iw:ih*2[A];  
[2]lutrgb=g=256[B];[A][B]overlay=0:h"
```

- 显示效果:



## 窗口之间的空间

如果我们需要两个窗口之间的空间，例如10个像素，我们指定它：

- 在第二个filterchain，如 $\text{pad}=\text{iw}:\text{ih}*2+10$ 。

- 在第四个filterchain, 如叠加=0:h+10为50像素空间的水平比较我们指定。
- 在第二个filterchain: pad=iw\*2+10。
- 在第四个filterchain: overlay=w+10。

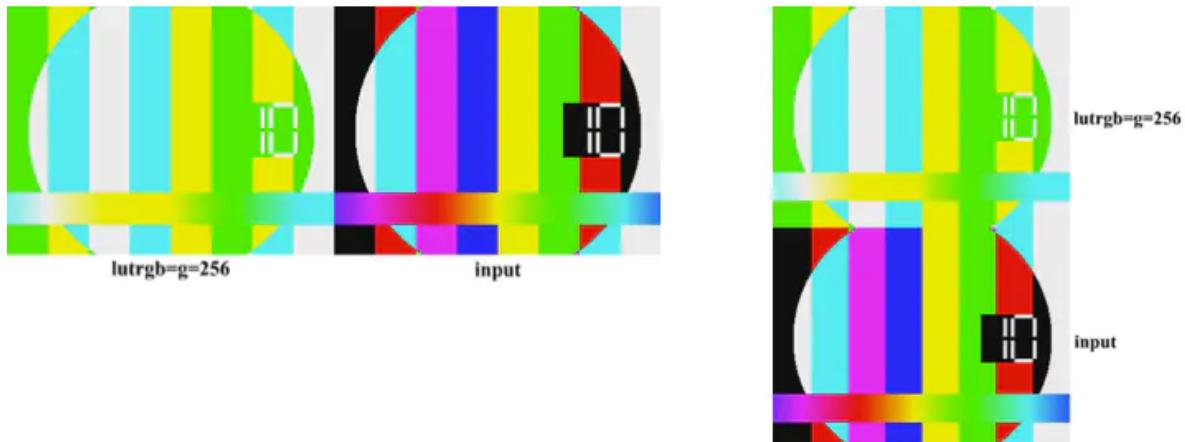
## 第一次修改版本

若要将修改后的版本放在水平位置，我们可以使用以下命令：

```
ffplay -f lavfi -i testsrc -vf ^ split[1][2];[1]pad=iw*2:ih:iw[A];
[2]lutrgb=g=256[B];[A][B]overlay
```

与顶部的修改输入的垂直比较是用命令创建的：

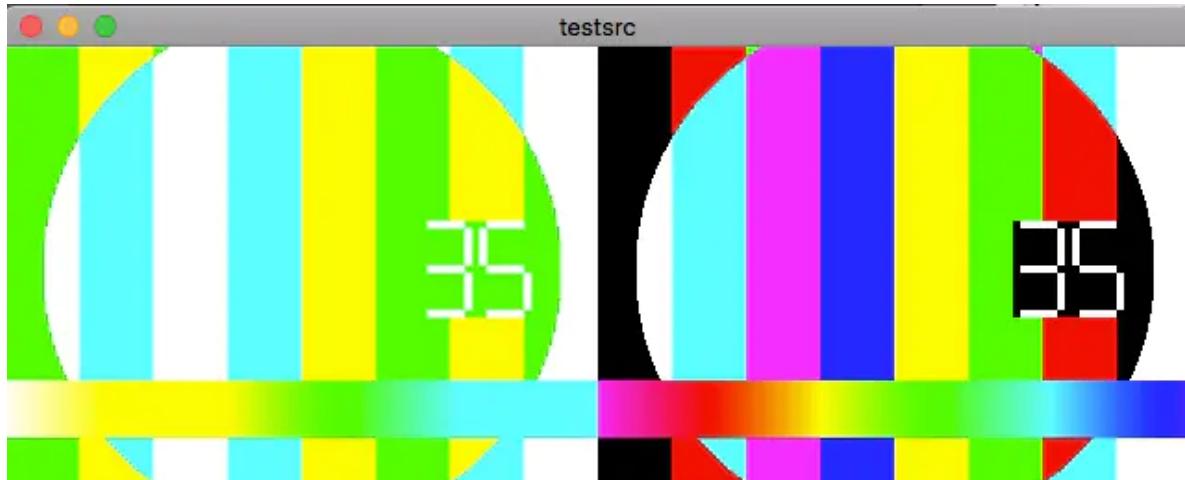
```
ffplay -f lavfi -i testsrc -vf split[1][2];[1]pad=iw:ih*2:0:ih[A];
[2]lutrgb=g=256[B];[A][B]overlay
```



我的测试命令：

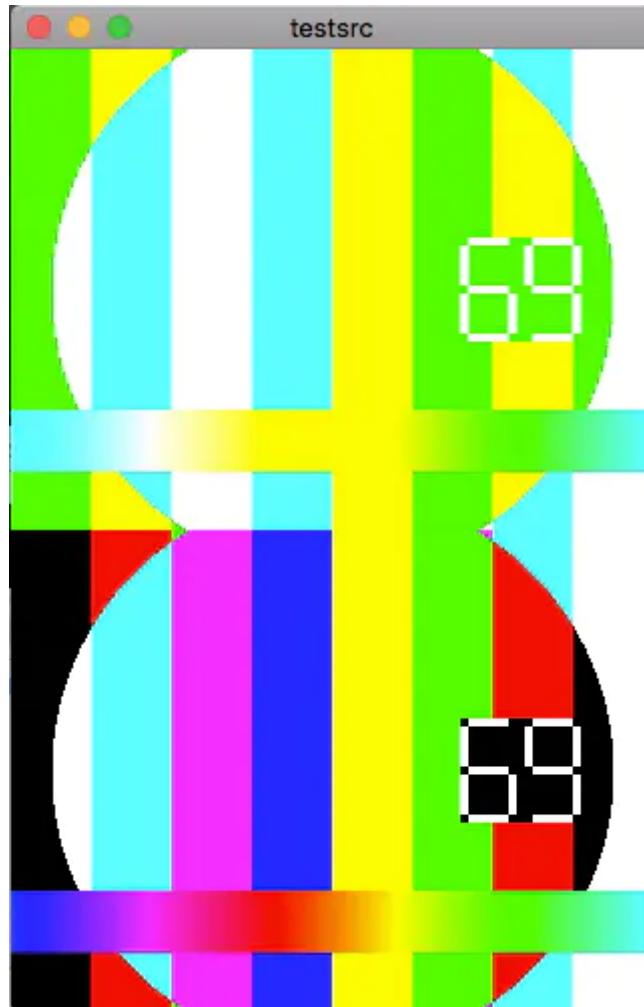
```
ffplay -f lavfi -i testsrc -vf "split[1][2];[1]pad=iw*2:ih:iw[A];
[2]lutrgb=g=256[B];[A][B]overlay"
```

- 显示效果：



```
ffplay -f lavfi -i testsrc -vf "split[1][2];[1]pad=iw:ih*2:0:ih[A];  
[2]lutrgb=g=256[B];[A][B]overlay"
```

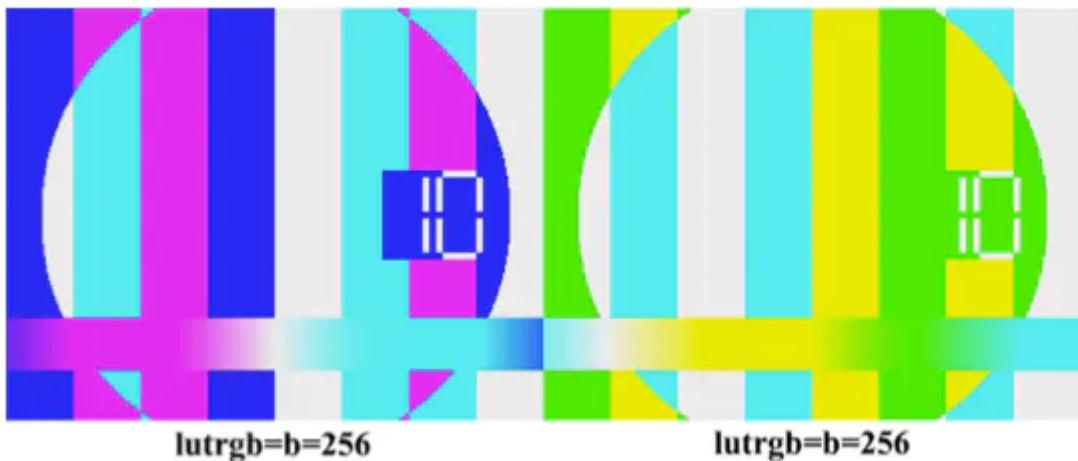
- 效果图：



## 没有输入的修改版本

下一个命令显示两个修改后的输入，没有输入本身，第二个过滤器包含第二个过滤器，它修改了第一个窗口的内容：

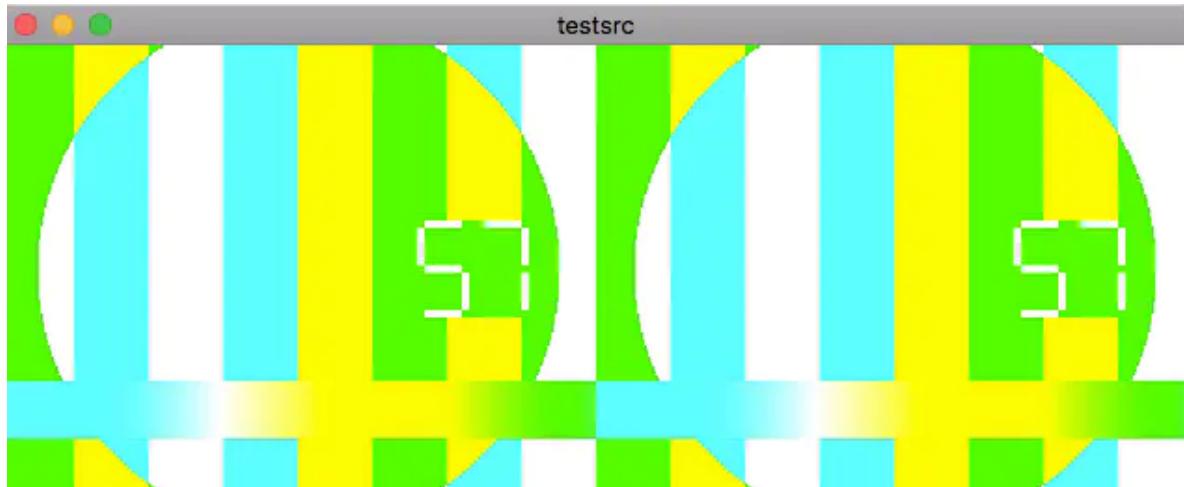
```
ffplay -f lavfi -i testsrc -vf split[1][2]; ^ [1]pad=iw*2,lutrgb=b=256[A];  
[2]lutrgb=g=256[B];[A][B]overlay=w
```



我的测试命令:

```
ffplay -f lavfi -i testsrc -vf "split[1][2];[1]pad=iw*2, lutrgb=g=256[A];
[2]lutrgb=g=256[B];[A][B]overlay=w"
```

- 显示效果



## 比较三个窗口

在一个复杂的视频编辑中，可以将输入与2个修改同时进行比较。为了在3个窗口中显示比较，我们使用了带有6个过滤链的 `filtergraph`。

### 三个窗口水平比较

为了创建一个水平3的窗口比较，可以指定6个 `filterchains`:

- 1.filterchain将输入拆分为3个相同的输出，标记为[1], [2], [3]
- 2.filterchain从[1]输入一个宽度为3倍的pad，输出为[a]
- 3.filterchain通过一些过滤器(s)修改[2]输入，输出被标记为[B]
- 4.filterchain通过一些过滤器(s)修改[3]输入，输出被标记为[C]
- 5.filterchain覆盖[B]输入[A]，其中x坐标为w，输出为[D]
- 6.filterchain覆盖[C]输入[D]，其中x坐标为w\*2(2倍输入宽度)

这里我需要澄清一下，上面的这部分是有背景颜色的，如下:

- 1. filterchain splits the input with the split filter to 3 identical outputs labeled [1], [2], [3]
- 2. filterchain creates from [1] input a pad with 3 times bigger width, output is labeled [A]
- 3. filterchain modifies [2] input with some filter(s), output is labeled [B]
- 4. filterchain modifies [3] input with some filter(s), output is labeled [C]
- 5. filterchain overlays [B] input on [A] input where x coordinate is w, output is labeled [D]
- 6. filterchain overlays [C] input on [D] input where x coordinate is w\*2 (2 times input width)

例如，下一个命令将tests src模式与修改的u组件(2)进行比较。与修改的v组件(3)。窗口):

```
ffplay -f lavfi -i testsrc -vf ^ split=3[1][2][3];[1]pad=iw*3[A];
[2]lutyuv=u=val*1.5[B];^ [3]lutyuv=v=val*1.5[C];[A][B]overlay=w[D];[D]
[C]overlay=w*2
```

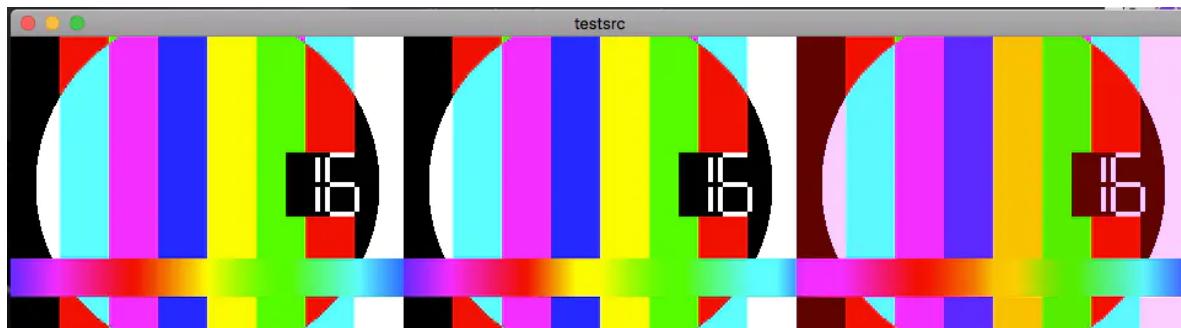
```
ffplay -f lavfi -i testsrc -vf ^
split=3[1][2][3];[1]pad=iw*3[A];[2]lutyuv=u=val*1.5[B];^
[3]lutyuv=v=val*1.5[C];[A][B]overlay=w[D];[D][C]overlay=w*2
```



我的测试命令如下:

```
ffplay -f lavfi -i testsrc -vf "split=3[1][2][3];[1]pad=iw*3[A];
[2]lutrgb=u=val*1.5[B];[3]lutyuv=v=val*1.5[C];[A][B]overlay=w[D];[D]
[C]overlay=w*2"
```

- 显示效果:



### 三个窗口垂直比较

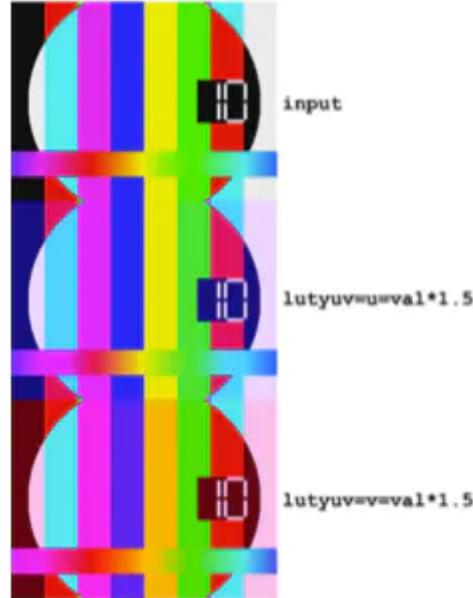
对于纵向比较，我们改变下一个过滤链:

- 2.filterchain -在衬垫过滤宽度参数为输入宽度，高度参数为输入高度乘以3:pad=iw:ih\*3。
- 5.filterchain -在叠加滤波器x参数为零时，y参数为输入高度:叠加=0:h。

- 6.filterchain -在叠加滤波器x参数为零时，y参数为输入高度乘以2:叠加=0。

```
ffplay -f lavfi -i testsrc -vf ^ split=3[1][2][3];[1]pad=iw:ih*3[A];^
[2]lutyuv=u=val*1.5[B];[3]lutyuv=v=val*1.5[C];^ [A][B]overlay=0:h[D];[D]
[C]overlay=0:h*2
```

下一个示例将与前面的相同的图像进行比较，但是垂直的(更改是下划线的):



我的测试命令:

```
ffplay -f lavfi -i testsrc -vf "split=3[1][2][3];[1]pad=iw:ih*3[A];
[2]lutyuv=u=val*1.5[B];[3]lutyuv=v=val*1.5[C];[A][B]overlay=0:h[D];[D]
[C]overlay=0:h*2"
```

- 效果图:



## 在中间窗口输入

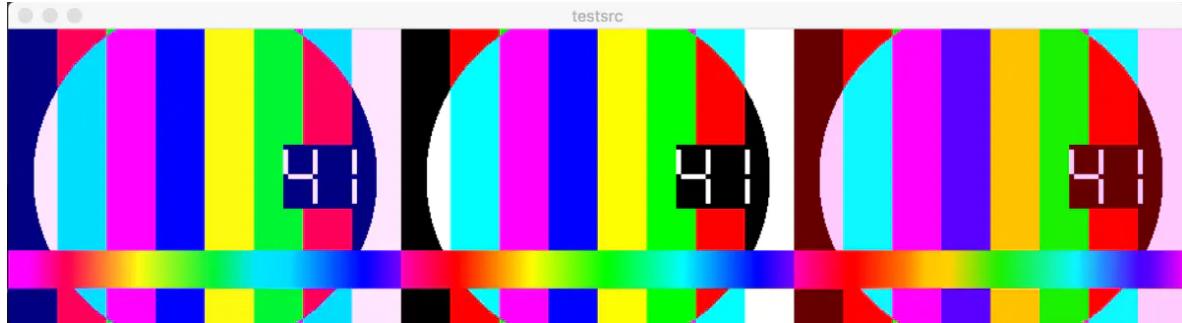
下一个命令将输入放置到中间，对filterchains的修改与前面的示例类似(将不变输入的x坐标设置为输入宽度: `iw`):

```
ffplay -f lavfi -i testsrc -vf ^ split=3[1][2][3];[1]pad=iw*3:ih:iw[A];
[2]lutyuv=u=val*1.5[B];^ [3]lutyuv=v=val*1.5[C];[A][B]overlay[D];[D]
[C]overlay=w*2
```

我的测试命令:

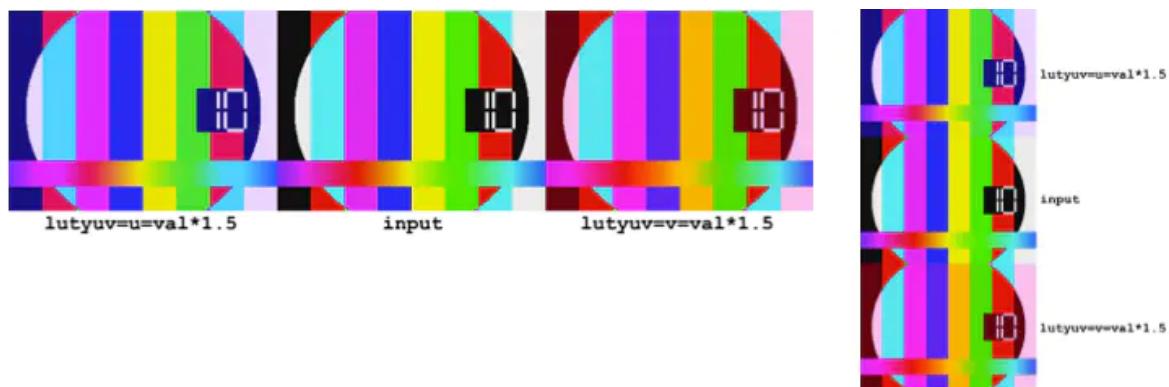
```
ffplay -f lavfi -i testsrc -vf "split=3[1][2][3];[1]pad=iw*3:ih:iw[A];
[2]lutyuv=u=val*1.5[B];[3]lutyuv=v=val*1.5[C];[A][B]overlay[D];[D]
[C]overlay=w*2"
```

- 效果图：



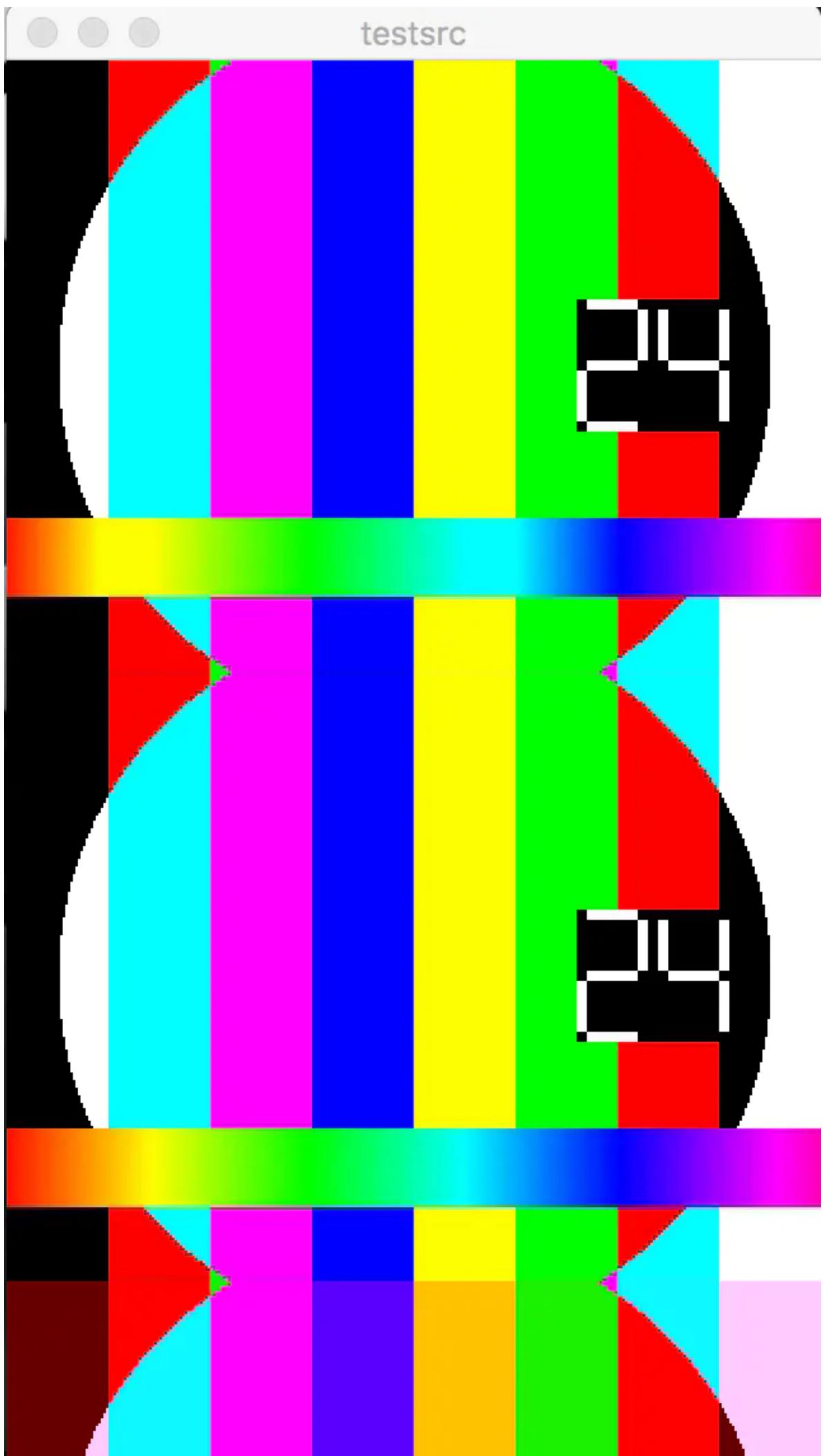
要将输入垂直放置到中间，我们可以使用以下命令：

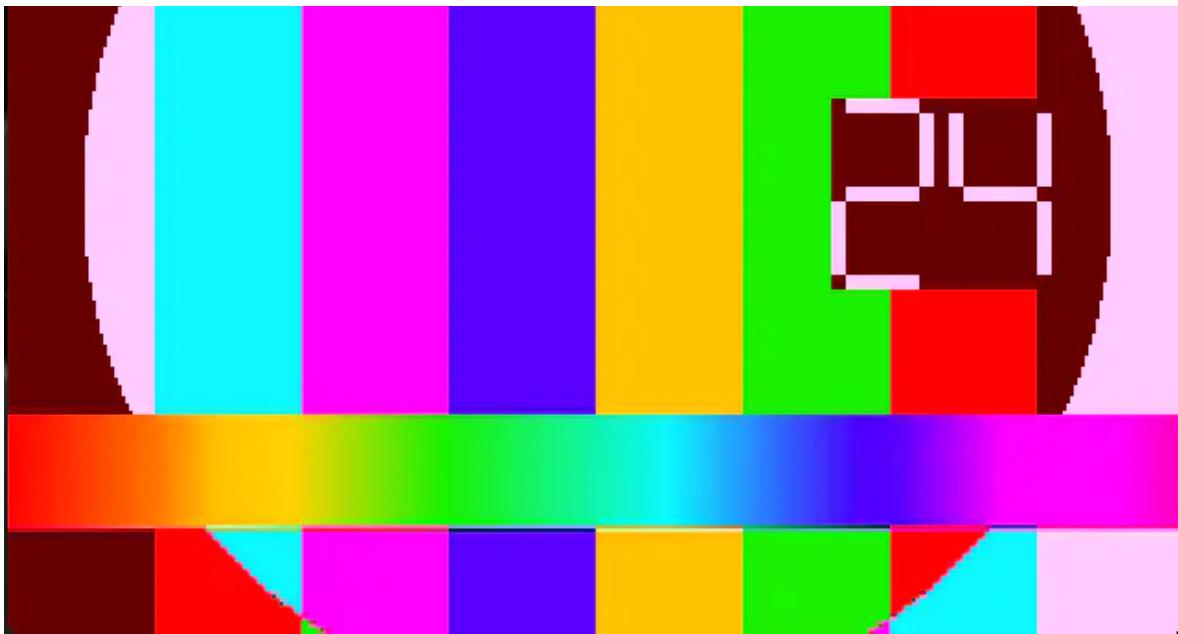
```
ffplay -f lavfi -i testsrc -vf ^ split=3[1][2][3];[1]pad=iw:ih*3:0:ih[A];
[2]lutyuv=u=val*1.5[B];^ [3]lutyuv=v=val*1.5[C];[A][B]overlay[D];[D]
[C]overlay=0:h*2
```



我的测试命令：

```
ffplay -f lavfi -i testsrc -vf "split=3[1][2][3];[1]pad=iw:ih*3:0:ih[A];
[2]lutrgb=u=val*1.5[B];[3]lutyuv=v=val*1.5[C];[A][B]overlay[D];[D]
[C]overlay=0:h*2"
```

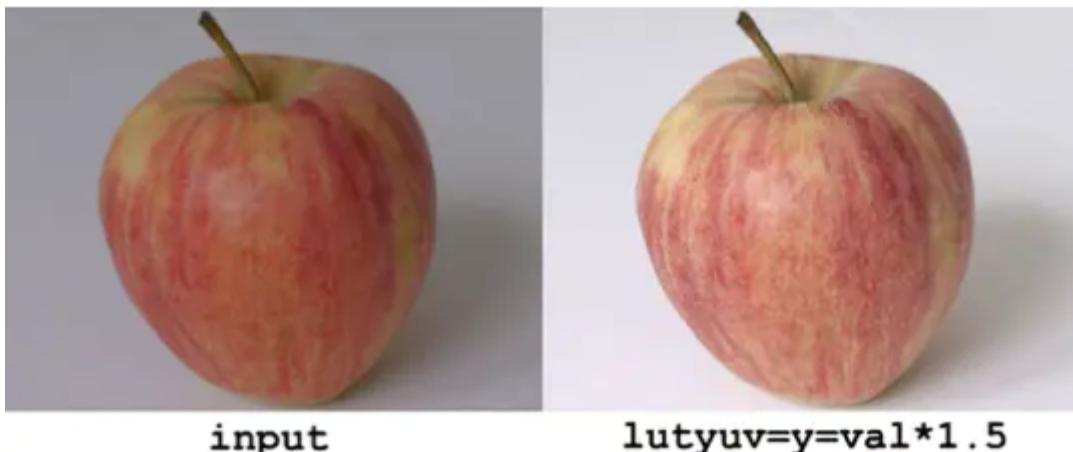




## 在2个和3个窗口中进行亮度矫正

下面的例子演示了2和3窗口预览的亮度调整。要在亮度是输入值1.5倍的版本旁边显示图像，我们可以使用以下命令：

```
ffplay -i apple.avi -vf ^ split[1][2];[1]pad=iw*2[A];[2]lutyuv=y=val*1.5[B];[A][B]overlay=w
```



我的测试命令：

```
ffplay -i /Users/zhangfangtao/Desktop/ornage.jpeg -vf "split[1][2];[1]pad=iw*2[A];[2]lutyuv=y=val*1.5[B];[A][B]overlay=w"
```



下一个例子是之前的3-windows版本，中间添加了一个修改后的版本，其中输入亮度的倍数为1.2：

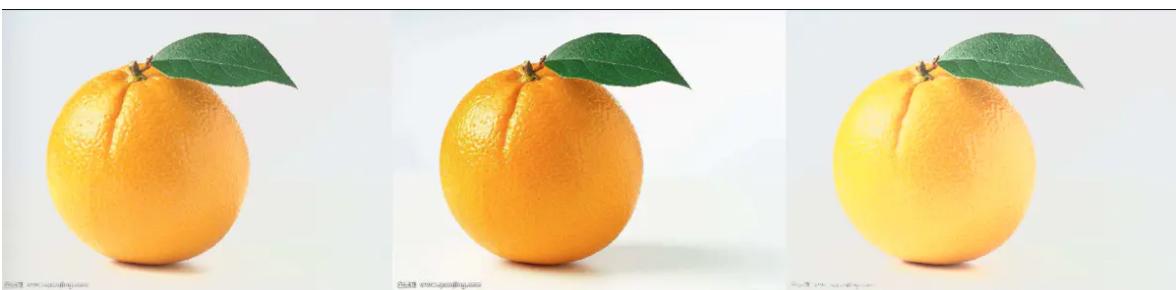
```
ffplay -i apple.avi -vf ^ split=3[1][2][3];[1]pad=iw*3[A];
[2]lutyuv=y=val*1.2[B];^ [3]lutyuv=y=val*1.5[C];[A][B]overlay=w[D];[D]
[C]overlay=w*2
```



我的测试命令：

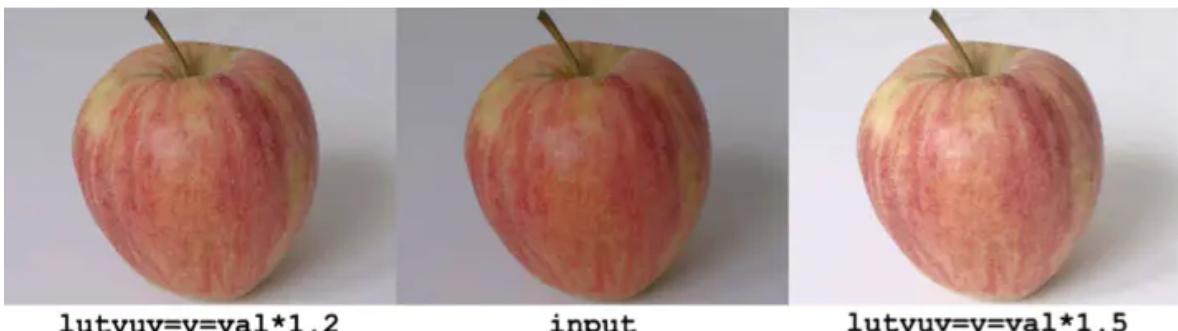
```
ffplay -i /Users/zhangfangtao/Desktop/ornage.jpeg -vf "split=3[1][2][3];
[1]pad=iw*3:ih:iw[A];[2]lutyuv=y=val*1.2[B];[3]lutyuv=y=val*1.5[C];[A]
[B]overlay[D];[D][C]overlay=w*2"
```

- 效果图：



要将输入定位到中央窗口，我们可以使用以下命令：

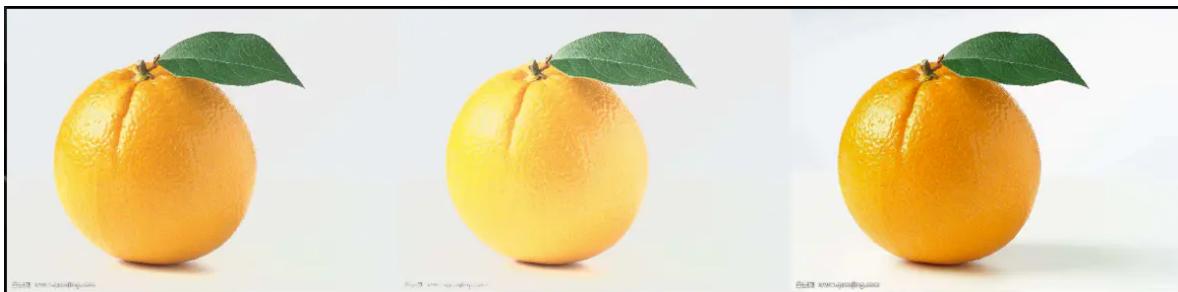
```
ffplay -i apple.avi -vf ^ split=3[1][2][3];[1]pad=iw*3:ih:iw[A];
[2]lutyuv=y=val*1.2[B];^ [3]lutyuv=y=val*1.5[C];[A][B]overlay[D];[D]
[C]overlay=w*2
```



我的测试命令:

```
ffplay -i /Users/zhangfangtao/Desktop/ornage.jpeg -vf "split=3[1][2][3];
[1]pad=iw*3:ih:iw*2[A];[2]lutyuv=y=val*1.2[B];[3]lutyuv=y=val*1.5[C];[A]
[B]overlay[D];[D][C]overlay=w"
```

- 效果图



如果我们想要3的输入。窗口，只有两个过滤链从上一个示例修改:

- 2.filterchain: pad过滤器的x参数值设为iw\*2。
- 6.filterchain: 覆盖过滤器的x参数设置为w(输入宽度)

下一个命令显示第三个窗口中的输入:

```
ffplay -i apple.avi -vf ^ split=3[1][2][3];[1]pad=iw*3:ih:iw*2[A];
[2]lutyuv=y=val*1.2[B];^ [3]lutyuv=y=val*1.5[C];[A][B]overlay[D];[D][C]overlay=w
```

## 在4个窗口中进行比较

为了得到更好的结果和各种实验，我们可以同时对输入进行3个修改。为了在4个窗口中显示比较，filtergraph包含8个filterchains:

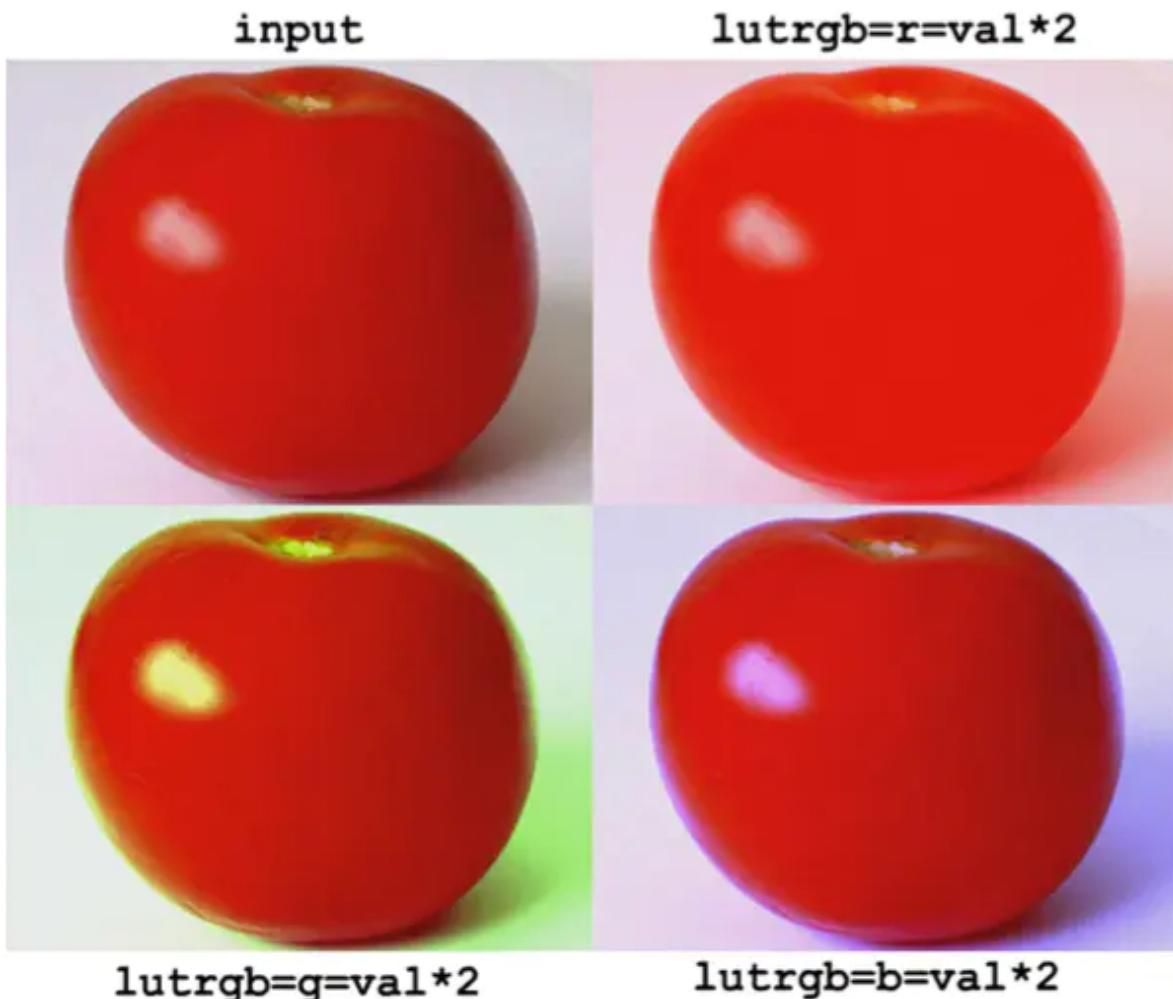
- 1.filterchain将输入分割为4个相同的输出，标记为[1], [2], [3], [4]
- 2.filterchain从[1]输入一个具有双倍宽度和双倍高度的衬垫，输出被标记为[a]
- 3.filterchain通过一些过滤器(s)修改[2]输入，输出被标记为[B]
- 4.filterchain通过一些过滤器(s)修改[3]输入，输出被标记为[C]
- 5.filterchain修改[4]输入带有一些过滤器，输出被标记为[D]
- 6.filterchain覆盖[B]输入[A]输入，其中x坐标为输入宽度，输出标签为[E]
- 7.filterchain覆盖[C]输入[E]输入，其中x为0,y为输入高度，输出标签为[F]
- 8.filterchain覆盖[D]输入[F]输入，其中x坐标为输入宽度，y为输入高度。

- 1. **filterchain** splits the input with the split filter to 4 identical outputs labeled [1], [2], [3], [4]
- 2. **filterchain** creates from [1] input a pad with a double width and double height, output is labeled [A]
- 3. **filterchain** modifies [2] input with some filter(s), output is labeled [B]
- 4. **filterchain** modifies [3] input with some filter(s), output is labeled [C]
- 5. **filterchain** modifies [4] input with some filter(s), output is labeled [D]
- 6. **filterchain** overlays [B] input on [A] input where x coordinate is input width, output label is [E]
- 7. **filterchain** overlays [C] input on [E] input where x is zero and y is input height, output label is [F]
- 8. **filterchain** overlays [D] input on [F] input where x coordinate is input width and y is input height.

例如，下一个命令将番茄与特定颜色通道值加倍的版本进行比较。右上方的窗口加强了红色通道，在底部左侧的绿色通道和右下方的蓝色通道：

```
ffplay -i tomato.mpg -vf split=4[1][2][3][4];[1]pad=iw*2:ih*2[A];^
[2]lutrgb=r=val*2[B];[3]lutrgb=g=val*2[C];[4]lutrgb=b=val*2[D];^
[A][B]overlay=w[E];[E][C]overlay=0:h[F];[F][D]overlay=w:h
```

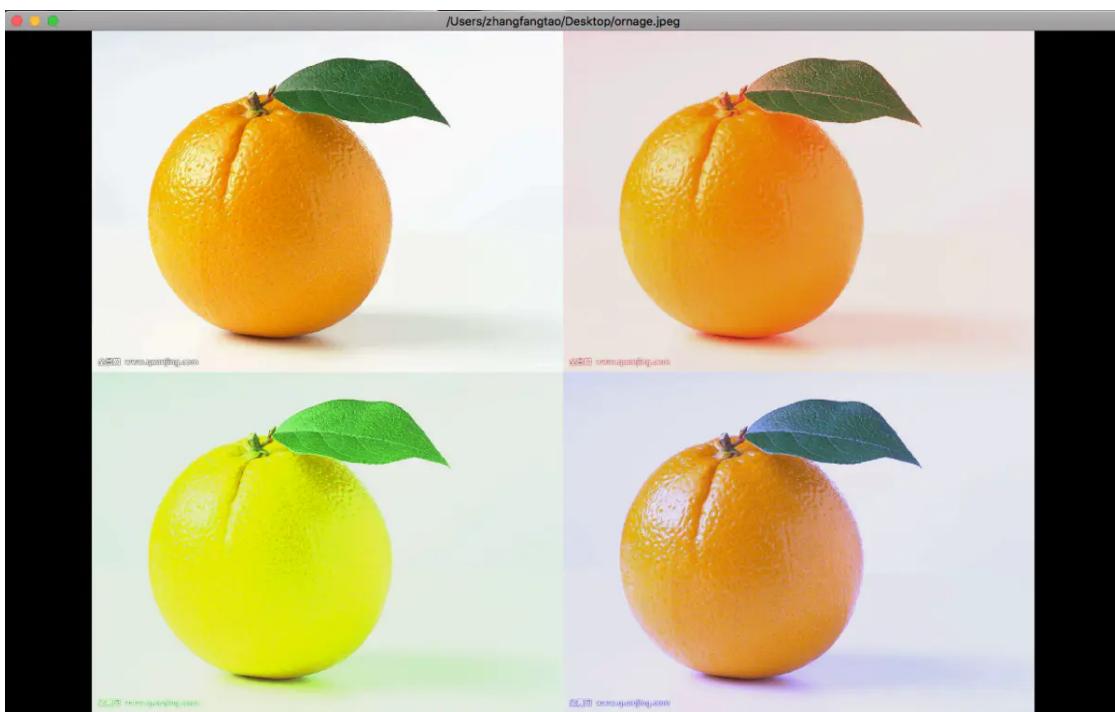
```
ffplay -i tomato.mpg -vf split=4[1][2][3][4];[1]pad=iw*2:ih*2[A];^
[2]lutrgb=r=val*2[B];[3]lutrgb=g=val*2[C];[4]lutrgb=b=val*2[D];^
[A][B]overlay=w[E];[E][C]overlay=0:h[F];[F][D]overlay=w:h
```



我的测试命令：

```
ffplay -i /Users/zhangfangtao/Desktop/ornage.jpeg -vf "split=4[1][2][3][4];
[1]pad=iw*2:ih*2[A]; [2]lutrgb=r=val*2[B];[3]lutrgb=g=val*2[C];
[4]lutrgb=b=val*2[D]; [A][B]overlay=w[E];[E][C]overlay=0:h[F];[F][D]overlay=w:h"
```

- 效果图：



## 23- 先进的技术点

### 加入音频和视频文件

有几种加入媒体文件，它们在表格中描述：

**Joining audio and video files**

类型	描述	针对音频	针对视频
级联	编码文件一个接一个；第一个结束，第二个开始	Yes	Yes
合并	将所有音频流编码为一个，例如2个单声道到1个立体声	Yes	No
混合	将2个或更多音频通道编码为1，音量可以调节	Yes	No
多路复用 (mux)	将2个或更多文件编码为1，例如1个音频和1个视频文件，如果存在更多相同类型的流，则选择是在用户	Yes	Yes
覆盖/画中画 (PiP)	2个或更多视频一次显示在另一个旁边或一个在另一个之上	No	Yes

### 连接与shell命令

媒体文件连接的先决条件

特殊文件格式	连接只能是某些文件格式：音频- MP3(第2个文件的头将消失)，未压缩的像WAV, PCM，等视频- MPEG-1, MPEG-2 TS, DV
格式的一致性	所有连接的文件都是相同的格式，这意味着可以加入2个MP3文件，但1个带有1个WAV的MP3不能
流的一致性	所有连接文件： - 包含相同数量的每种类型的流。 - 音频流使用相同的编解码器，采样率和通道布局 - 视频流使用相同的分辨率
	为符合此要求，通常需要转换输入文件，使用-q 1或类似选项来保持初始质量，详细信息请参阅 <a href="#">格式间转换</a>

在Windows上，我们可以使用带有/B标志的复制命令来指示二进制模式，在文件之间必须是一个加号。连接N文件的复制命令的一般形式是：

```
copy /B file1+file2+...+fileN-1+fileN outputFile
```

例如，连接文件videoclip1。mpg和videoclip2。mpg到文件视频。mpg，我们可以使用以下命令：

```
copy /B videoclip1.mpg+videoclip2.mpg video.mpg
```

在Linux、Unix和OS X上，我们可以在表单中使用cat命令:cat file1 files2 > file3，因此我们可以修改前面的例子：

```
cat videoclip1.mpg videoclip2.mpg > video.mpg
```

## 链接concat协议

另一种选择是使用concat协议，先决条件类似于复制命令。例如，要使用该协议修改前面的示例，我们可以使用以下命令：

```
ffmpeg -i concat:"videoclip1.mpg|videoclip2.mpg" -c copy video.mpg
```

我的测试命令：

```
ffmpeg -i
concat:"/Users/zhangfangtao/Desktop/test.mp4|/Users/zhangfangtao/Desktop/test2.mp4"
-c copy /Users/zhangfangtao/Desktop/test3.mp4
```

- 显示效果：

生成了一个新的视频，不过我感觉这个视频和test1没有什么差别。

## 连接 concat 过滤器

用于音频和视频拼接的特殊过滤器是在表中描述的 concat 过滤器:

### Multimedia filter: concat

描述	连接音频和视频文件一个接一个。该过滤器适用于同步视频和音频流的片段(文件), 其中所有片段必须具有相同数量的每种类型的流, 例如1个音频和1个视频, 或2个音频和1个视频, 等等
语法	concat=a=a_streams:v=v_streams:n=segments[:unsafe] 所有的参数都是可选的
	参数的描述
a	输出音频流的数量, 默认值为0
n	段数, 默认值为2
unsafe	安全模式激活, 如果设置, 连接将不会以不同格式的片段失败
v	输出视频流的数量, 默认值为1

适当筛选结果的先决条件:

- 所有段必须从时间戳0开始。
- 相应的流必须在所有段中使用相同的参数, 特别是视频大小。
- 建议是相同的帧速率, 否则输出将使用可变帧速率。

Concat filter 可以加入各种格式, 有些例子是:

```
ffmpeg -i input1.avi -i input2.avi -filter_complex concat output.avi
ffmpeg -i input1.avi -i input2.avi -filter_complex concat output.mp4
ffmpeg -i input1.avi -i input2.mp4 -filter_complex concat output.webm
ffmpeg -i input1.avi -i input2.mp4 -i input3.mkv -filter_complex ^ concat=n=3
output.flv
ffmpeg -i input1.avi -i input2.avi -i input3.avi -i input4.avi ^ -
filter_complex concat=n=4 output.mp4
f -i 1.avi -vf movie=2.avi[a];[in][a]concat a.mp4
```

我的测试命令如下:

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -i
/Users/zhangfangtao/Desktop/test3.mp4 -filter_complex concat
/Users/zhangfangtao/Desktop/test4.mp4
```

- 显示效果:

生成了一个视频, 里面包含前两段的是视频内容, 不过, 有效时长只是第一段视频的长度。后面有种无效的感觉。。。

## 其他类型的拼接技术

- 音频合并 (多个流到1个多声道流) - 在[数字音频](#)一章中介绍
- 将几个音频文件混合到1 - 在[数字音频](#)一章中有描述
- 多路复用 - 在[FFmpeg基本介绍](#)一章中介绍了媒体流的选择

- overlay-在[overlay-画中画](#)章节里面有具体的描述。

## 移除掉logo

一些视频包含公司标志，通常位于左上角，常见示例是录制电视节目。 FFmpeg包含2个特殊滤镜以去除徽标，而最终效果并不总是完美，不过在许多情况下这种移除logo的技术还是可以接受的。

### delego过滤器

#### Video filter: delego

描述	通过对周围像素的简单插值来隐藏一个电视台的标志。用户设置一个覆盖该徽标的矩形，它通常会消失(但在某些情况下，标识更明显)。过滤器接受参数作为表单“x:y:w:h:band”的字符串，或作为键=值对的列表，由“:”分隔
语法	delego=x=0:y=0:w=width:h=height[:t=band:show={0,1}] []中的参数是可选的，显示为0或1
	参数的描述
x, y	标志的左上角的坐标
w, h	标志的宽度和高度
band or t	该标志矩形的模糊边缘厚度，默认值为4
show	定位的参数，默认值为0，如果设置为1，屏幕上的绿色矩形显示为帮助查找正确的x、y、w和h参数

例如，我们首先从下图所示的800x600像素大小视频的右上角移除一个标志，我们通过显示一个绿色矩形的显示选项来估计logo的位置：

```
ffmpeg -i eagles.mpg -vf delego=x=700:y=0:w=100:h=50:t=3:show=1 nologo.mpg
```

现在我们可以精确地指定位置和标识的存在几乎是不可见的：

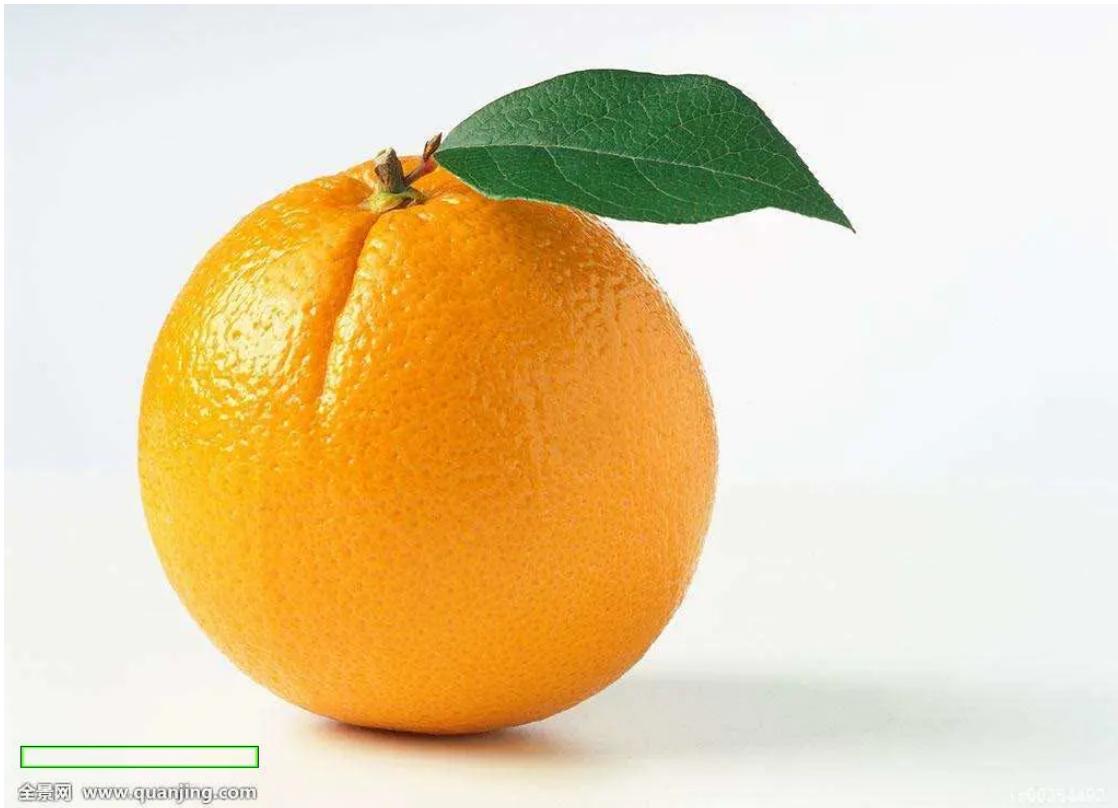
```
ffmpeg -i eagles.mpg -vf delego=x=730:y=0:w=70:h=46:t=1 nologo.mpg
```



我的测试命令如下（没有 `t` 参数，我这边不识别这个，报错）：

```
ffmpeg -i /Users/zhangfangtao/Desktop/ornage.jpeg -vf  
"delogo=x=15:y=678:w=218:h=20:show=1" /Users/zhangfangtao/Desktop/ornage2.jpeg
```

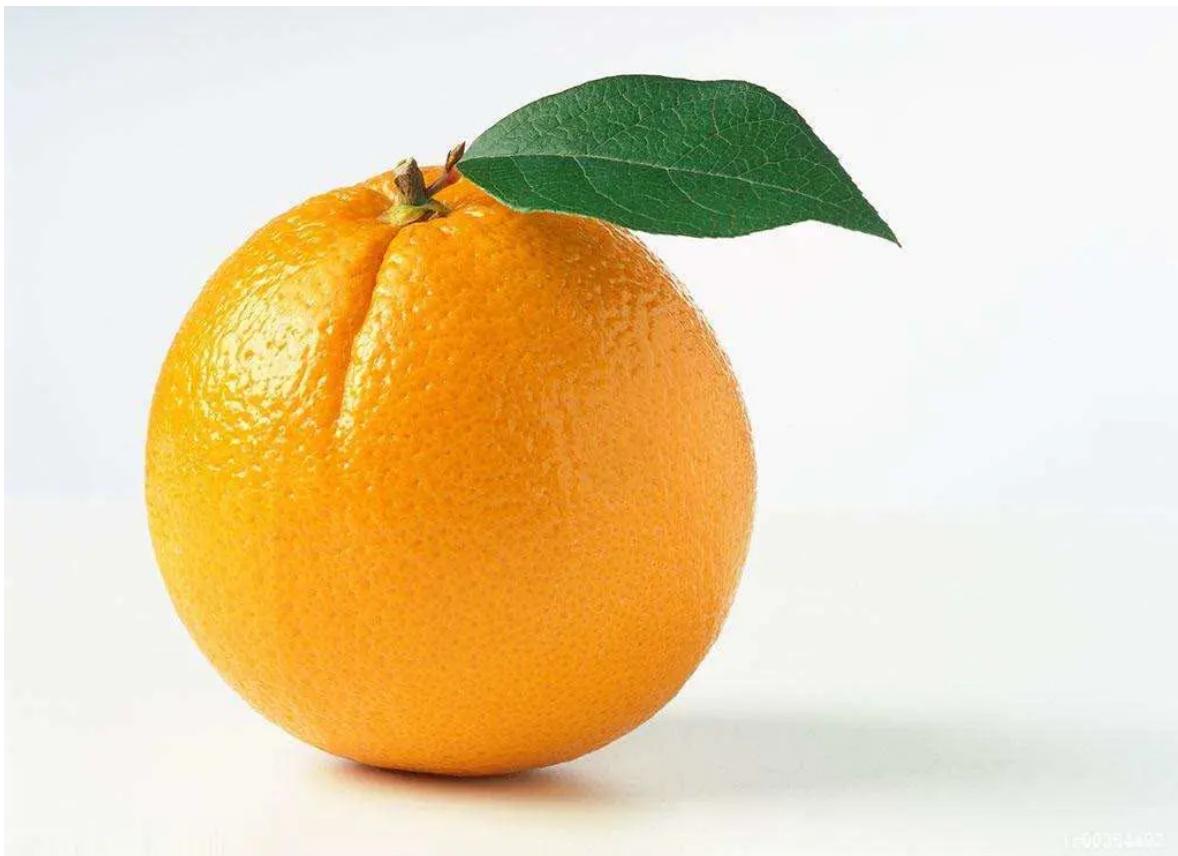
- 显示效果(有没有看到一个绿色的框)：



有没有看到一个绿色的框

```
ffmpeg -i /Users/zhangfangtao/Desktop/ornage.jpeg -vf  
"delogo=x=10:y=700:w=300:h=35" /Users/zhangfangtao/Desktop/ornage3.jpeg
```

- 显示效果(水印没了)：



水印没了.jpeg

## 抖动视频部分的固定

没有三脚架或车辆拍摄的视频的一些部分通常包括抖动 - 水平和垂直移动的小变化，在某些情况下可以使用去抖滤波器进行修正：

### Video filter: deshake

描述	修正水平和垂直位移的小变化，当视频没有三脚架或移动车辆时有用
语法	deshake=x:y:w:h:rx:ry:edge:blocksize:contrast:search:filename 所有的参数是可选的
	参数的描述
x, y, w, h	矩形区域的坐标和大小，搜索运动向量，x和y是左上角的坐标，w是宽度，h是高度。这些参数与drawbox过滤器具有相同的含义，可用于可视化边界框的位置。当物体在框架内同时运动时，运动矢量搜索可能会混淆摄像机的运动，这是很有用的。如果x, y, w和h都被设为-1那么整个框架就被使用了。这允许在不指定运动向量搜索的边界框的情况下设置后续选项。默认-搜索整个框架。
rx, ry	在0 - 64像素范围内指定x和y方向的最大运动范围，默认值为16
edge	指定如何生成像素来填充框架边缘的空白，值为从0到3的整数：0 - 在空白位置填充零 1 - 原始图像在空白位置 2 - 在空白位置的挤压边值 3 - 镜像边缘在空白位置，默认值
blocksize	指定用于运动搜索的块大小，其值为4 - 128像素， 默认值为8
contrast	指定块的对比度阈值。只有超过指定对比度的块(最黑和最轻的像素之间的区别)才会被考虑。该值来自范围1 - 255， 默认值为125

search 描述	指定搜索策略: 0 = 彻底搜索, 默认值 1 = 不彻底搜索 修正水平和垂直位移的小变化, 当视频没有三脚架或移动车辆时有用
filename	如果包含, 则将动作搜索的详细日志写入指定的文件

参数可以按顺序进入默认顺序或以任何顺序指定名称:

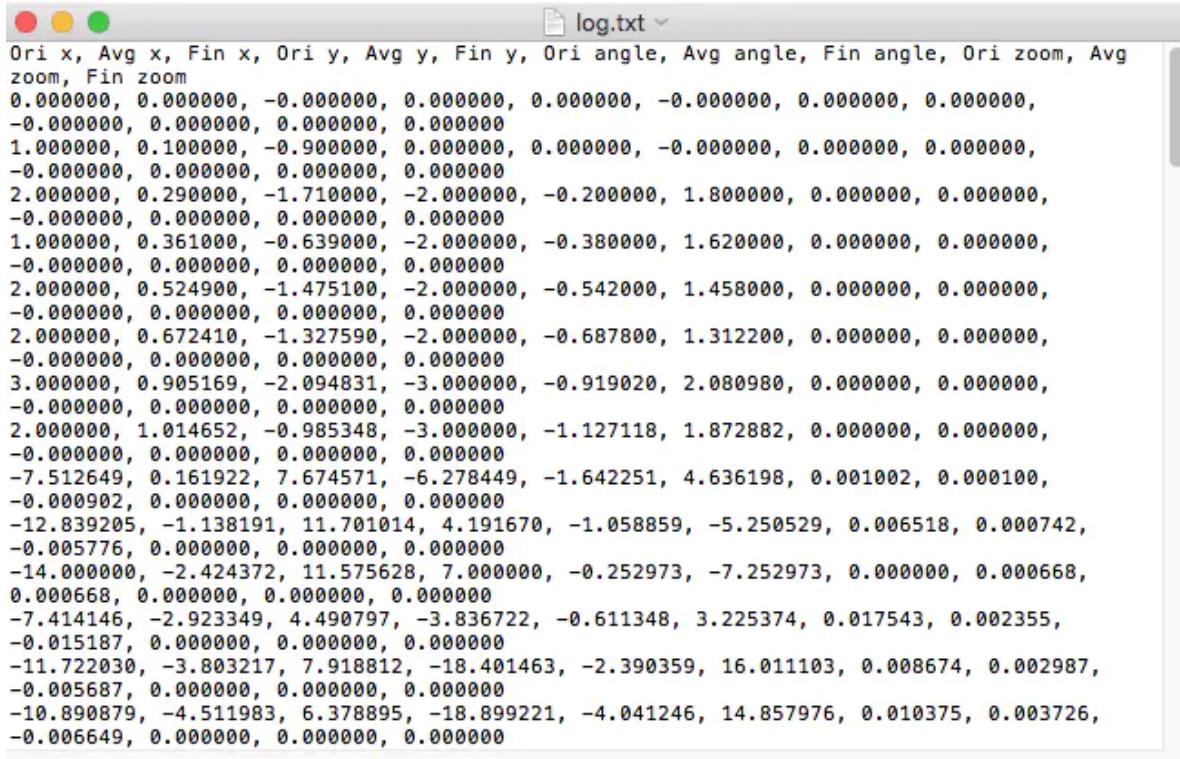
```
ffmpeg -i travel.avi -vf deshake fixed_travel.avi
ffmpeg -i travel.avi -vf deshake=contrast=160 fixed.avi
ffmpeg -i travel.avi -vf deshake=blocksize=4:filename=log.txt fixed.avi
```

我的测试命令:

```
ffmpeg -i /Users/zhangfangtao/Desktop/1527143197.mp4 -vf deshake
/Users/zhangfangtao/Desktop/1527143198.mp4
ffmpeg -i /Users/zhangfangtao/Desktop/1527143197.mp4 -vf deshake=contrast=160
/Users/zhangfangtao/Desktop/1527143199.mp4
ffmpeg -i /Users/zhangfangtao/Desktop/1527143197.mp4 -vf
deshake=blocksize=4:filename=/Users/zhangfangtao/Desktop/log.txt
/Users/zhangfangtao/Desktop/1527143200.mp4
```

- 显示效果: 其实, 我并没有看出来什么效果。。。可能我用手机拍的抖得太厉害了。。。

第三条指令打印出来的txt信息截图如下:



```
Ori x, Avg x, Fin x, Ori y, Avg y, Fin y, Ori angle, Avg angle, Fin angle, Ori zoom, Avg
zoom, Fin zoom
0.000000, 0.000000, -0.000000, 0.000000, 0.000000, -0.000000, 0.000000, 0.000000,
-0.000000, 0.000000, 0.000000, 0.000000
1.000000, 0.100000, -0.900000, 0.000000, 0.000000, -0.000000, 0.000000, 0.000000,
-0.000000, 0.000000, 0.000000, 0.000000
2.000000, 0.290000, -1.710000, -2.000000, -0.200000, 1.800000, 0.000000, 0.000000,
-0.000000, 0.000000, 0.000000
1.000000, 0.361000, -0.639000, -2.000000, -0.380000, 1.620000, 0.000000, 0.000000,
-0.000000, 0.000000, 0.000000, 0.000000
2.000000, 0.524900, -1.475100, -2.000000, -0.542000, 1.458000, 0.000000, 0.000000,
-0.000000, 0.000000, 0.000000
2.000000, 0.672410, -1.327590, -2.000000, -0.687800, 1.312200, 0.000000, 0.000000,
-0.000000, 0.000000, 0.000000
3.000000, 0.905169, -2.094831, -3.000000, -0.919020, 2.080980, 0.000000, 0.000000,
-0.000000, 0.000000, 0.000000, 0.000000
2.000000, 1.014652, -0.985348, -3.000000, -1.127118, 1.872882, 0.000000, 0.000000,
-0.000000, 0.000000, 0.000000, 0.000000
-7.512649, 0.161922, 7.674571, -6.278449, -1.642251, 4.636198, 0.001002, 0.000100,
-0.00002, 0.000000, 0.000000, 0.000000
-12.839205, -1.138191, 11.701014, 4.191670, -1.058859, -5.250529, 0.006518, 0.000742,
-0.005776, 0.000000, 0.000000, 0.000000
-14.000000, -2.424372, 11.575628, 7.000000, -0.252973, -7.252973, 0.000000, 0.000668,
0.000668, 0.000000, 0.000000, 0.000000
-7.414146, -2.923349, 4.490797, -3.836722, -0.611348, 3.225374, 0.017543, 0.002355,
-0.015187, 0.000000, 0.000000, 0.000000
-11.722030, -3.803217, 7.918812, -18.401463, -2.390359, 16.011103, 0.008674, 0.002987,
-0.005687, 0.000000, 0.000000, 0.000000
-10.890879, -4.511983, 6.378895, -18.899221, -4.041246, 14.857976, 0.010375, 0.003726,
-0.006649, 0.000000, 0.000000, 0.000000
```

log.txt

## 将颜色框添加到视频

使用 `drawbox`, 我们可以在矩形区域找到精确的坐标, 以在其中搜索运动矢量, 它用于除雾过滤器。其他用途包括各种图表, 方案等。

Video filter: drawbox

描述	在输入的选定区域绘制指定颜色和指定大小的框
语法	drawbox[=x:y:width:height:color:thickness]
	参数的描述
color, c	格式0xRRGGBB[AA]中的标准颜色名称或十六进制值
height, h	框的高度, 默认值为0
thickness, t	边框边缘的宽度以像素为单位, 默认值为4
width, w	框的宽度, 默认值为0
x, y	方框的左上角坐标, 默认值为0

例如, 要在SVGA大小的输入上添加一个尺寸为600x400像素的黄色框, 其大小为左侧150像素和顶部0像素, 我们可以使用以下命令:

```
ffmpeg -i ship.avi -vf drawbox=x=150:w=600:h=400:c=yellow ship1.avi
```



我的测试命令:

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf  
drawbox=x=150:w=600:h=400:c=yellow /Users/zhangfangtao/Desktop/test3.mp4
```

- 显示效果



## 检测帧数

如果您需要知道有多少帧包含您的视频文件，您可以使用以下命令：

```
ffmpeg -i input.mpg -f null /dev/null
```

显示输出的最后两行是：

```
frame= 250 fps=0.0 q=0.0 Lsize= 0kB time=00:00:10.00 bitrate= 0.0kbits/s
video:16kB audio:0kB subtitle:0 global headers:0kB muxing overhead -100.000000%
```

帧数为250，表示视频帧的总数，也可以从帧速率和持续时间计算，但结果并不总是准确。

我的测试命如下：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -f null /dev/null
```

- 显示效果如下：

```

Input #0, mov,mp4,m4a,3gp,3g2,mj2, from '/Users/zhangfangtao/Desktop/test.mp4':
Metadata:
    major_brand     : isom
    minor_version   : 512
    compatible_brands: isomiso2avc1mp41
    encoder        : Lavf58.13.100
Duration: 00:00:40.05, start: 0.000000, bitrate: 486 kb/s
Stream #0:0(eng): Video: h264 (High) (avc1 / 0x31637661), yuv420p, 1024x768
[SAR 1:1 DAR 4:3], 353 kb/s, 15 fps, 15 tbr, 15360 tbn, 30 tbc (default)
    Metadata:
        handler_name   : VideoHandler
    Stream #0:1(eng): Audio: aac (LC) (mp4a / 0x6134706D), 22050 Hz, stereo, flt
p, 128 kb/s (default)
        Metadata:
            handler_name   : SoundHandler
Stream mapping:
    Stream #0:0 -> #0:0 (h264 (native) -> wrapped_avframe (native))
    Stream #0:1 -> #0:1 (aac (native) -> pcm_s16le (native))
Press [q] to stop, [?] for help
Output #0, null, to '/dev/null':
Metadata:
    major_brand     : isom
    minor_version   : 512
    compatible_brands: isomiso2avc1mp41
    encoder        : Lavf58.13.100
    Stream #0:0(eng): Video: wrapped_avframe, yuv420p, 1024x768 [SAR 1:1 DAR 4:3
], q=2-31, 200 kb/s, 15 fps, 15 tbn, 15 tbc (default)
    Metadata:
        handler_name   : VideoHandler
        encoder        : Lavc58.19.100 wrapped_avframe
    Stream #0:1(eng): Audio: pcm_s16le, 22050 Hz, stereo, s16, 705 kb/s (default
)
    Metadata:
        handler_name   : SoundHandler
        encoder        : Lavc58.19.100 pcm_s16le
frame= 600 fps=0.0 q=-0.0 Lsize=N/A time=00:00:40.03 bitrate=N/A speed=51.6x
video:314kB audio:3448kB subtitle:0kB other streams:0kB global headers:0kB muxin
g overhead: unknown

```

## 检测广告，部分转换或损坏的编码

从电视，互联网等录制的较长视频可以包含带有广告，转场，不完整帧和其他不需要内容的短片。如果此部分包含黑色框架，则可以使用表中描述的黑色检测过滤器检测它们。

### Video filter: blackdetect

Description	检测几乎全黑的视频部分，并输出包含检测到的黑色间隔的开始，结束和持续时间的行，以秒为单位表示。如果日志级别设置为低于 <code>AV_LOG_INFO</code> 值，则不显示行
语法	<code>blackdetect[=d=duration:pic_th=pbr_threshold:pix_th=px_threshold]</code>

参数的描述（所有的参数都是可选的）

参数名称	单位	描述	默认值
<code>black_min_duration, d</code>	秒	正浮点数确定视频中黑色帧的最小持续时间	2.0
<code>picture_black_ratio_th, pic_th</code>	浮点数在0到1.0之间	例如，如果帧大小为400x300(总共12万像素)，12000像素不是黑色，那么这个比例是0.9	0.98

参数名称	浮点 单位 数在0 到1.0 之间	Threshold设置像素为黑色, 它等于表达式:  (absolute_threshold- luminance_minimum_value)  luminance_range_size)	默认 值
pixel_black_th, pix_th			0.1

例如, 要从源mptestsrc中检测黑帧, 命令是(控制台输出如下):

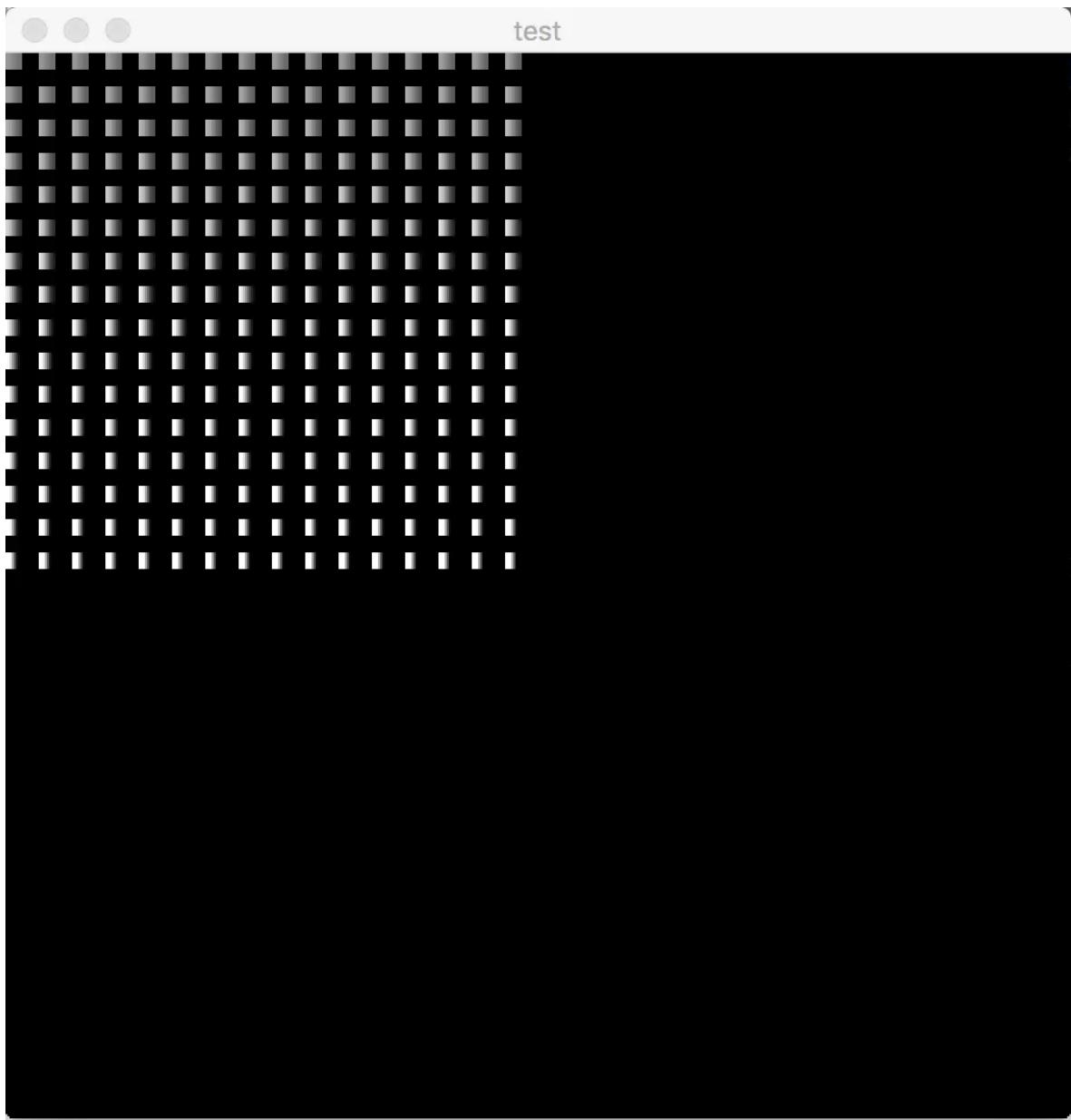
```
ffmpeg -f lavfi -i mptestsrc -vf blackdetect -f sdl 'test'
```

```
f-f lavfi -i mptestsrc -vf blackdetect -f sdl 'test'
libswresample 0. 15.100 / 0. 15.100
libpostproc 52. 0.100 / 52. 0.100
lavfi [lavfi @ 0x1f1bf20] Estimating duration from bitrate, this may be inaccurate
Input #0, lavfi, from 'mptestsrc':
Duration: N/A, start: 0.000000, bitrate: N/A
Stream #0:0: Video: rawvideo (I420 / 0x30323449), yuv420p, 512x512 [SAR 1:1 DAR 1:1],
[SDL @ 0x1f1ff0] w:512 h:512 fmt:yuv420p sar:1/1 -> w:512 h:512
Output #0, sdl, to 'test':
Metadata:
encoder : Lavf54.22.100
Stream #0:0: Video: rawvideo (I420 / 0x30323449), yuv420p, 512x512 [SAR 1:1 DAR 1:1].
Stream mapping:
Stream #0:0 -> #0:0 (rawvideo -> rawvideo)
Press [q] to stop, [?] for help
black_start:1.2 black_end:4.84 black_duration:3.6400:04.60 bitrate= 0.0kbits/s
black_start:13.2 black_end:16.84 black_duration:3.64:15.92 bitrate= 0.0kbits/s
black_start:25.2 black_end:28.84 black_duration:3.64:27.76 bitrate= 0.0kbits/s
black_start:37.2 black_end:40.84 black_duration:3.64:40.36 bitrate= 0.0kbits/s
black_start:49.2 black_end:52.84 black_duration:3.64:51.40 bitrate= 0.0kbits/s
black_start:61.2 black_end:64.84 black_duration:3.64:63.76 bitrate= 0.0kbits/s
black_start:73.2 black_end:76.84 black_duration:3.64:76.28 bitrate= 0.0kbits/s
black_start:85.2 black_end:88.84 black_duration:3.64:88.68 bitrate= 0.0kbits/s
black_start:97.2 black_end:100.84 black_duration:3.6439.68 bitrate= 0.0kbits/s
black_start:109.2 black_end:112.84 black_duration:3.642.28 bitrate= 0.0kbits/s
black_start:121.2 black_end:124.84 black_duration:3.644.76 bitrate= 0.0kbits/s
black_start:133.2 black_end:136.84 black_duration:3.645.72 bitrate= 0.0kbits/s
black_start:145.2 black_end:148.84 black_duration:3.648.40 bitrate= 0.0kbits/s
black_start:157.2 black_end:160.84 black_duration:3.640.60 bitrate= 0.0kbits/s
black_start:169.2 black_end:172.84 black_duration:3.641.56 bitrate= 0.0kbits/s
black_start:181.2 black_end:184.84 black_duration:3.644.04 bitrate= 0.0kbits/s
frame= 4773 fps= 67 q=0.0 size= 0kB time=00:03:10.92 bitrate= 0.0kbits/s
```

我的测试命令:

```
ffmpeg -f lavfi -i mptestsrc -vf blackdetect -f sdl 'test'
```

\*显示效果:



## 用黑帧过滤器进行检测

检测黑帧的另一个过滤器是表中描述的黑帧过滤器：

### Video filter: blackframe

描述	检测几乎为黑色的帧，并输出包含： - 检测帧的帧数 - 黑色部分的百分比 - 如果已知则在文件中定位，否则为-1 - 时间戳
语法	blackframe[=amount:[threshold]] 所有的参数都是可选的
	参数
amount	在阈值下的像素百分比，默认值为98
threshold	下面是被认为是黑色的像素，默认值是32

过滤器 `blackdetect` 和 `blackframe` 类似，但每个显示不同的信息。 图像上显示使用与黑检测滤镜相同的视频源的黑帧滤镜输出：

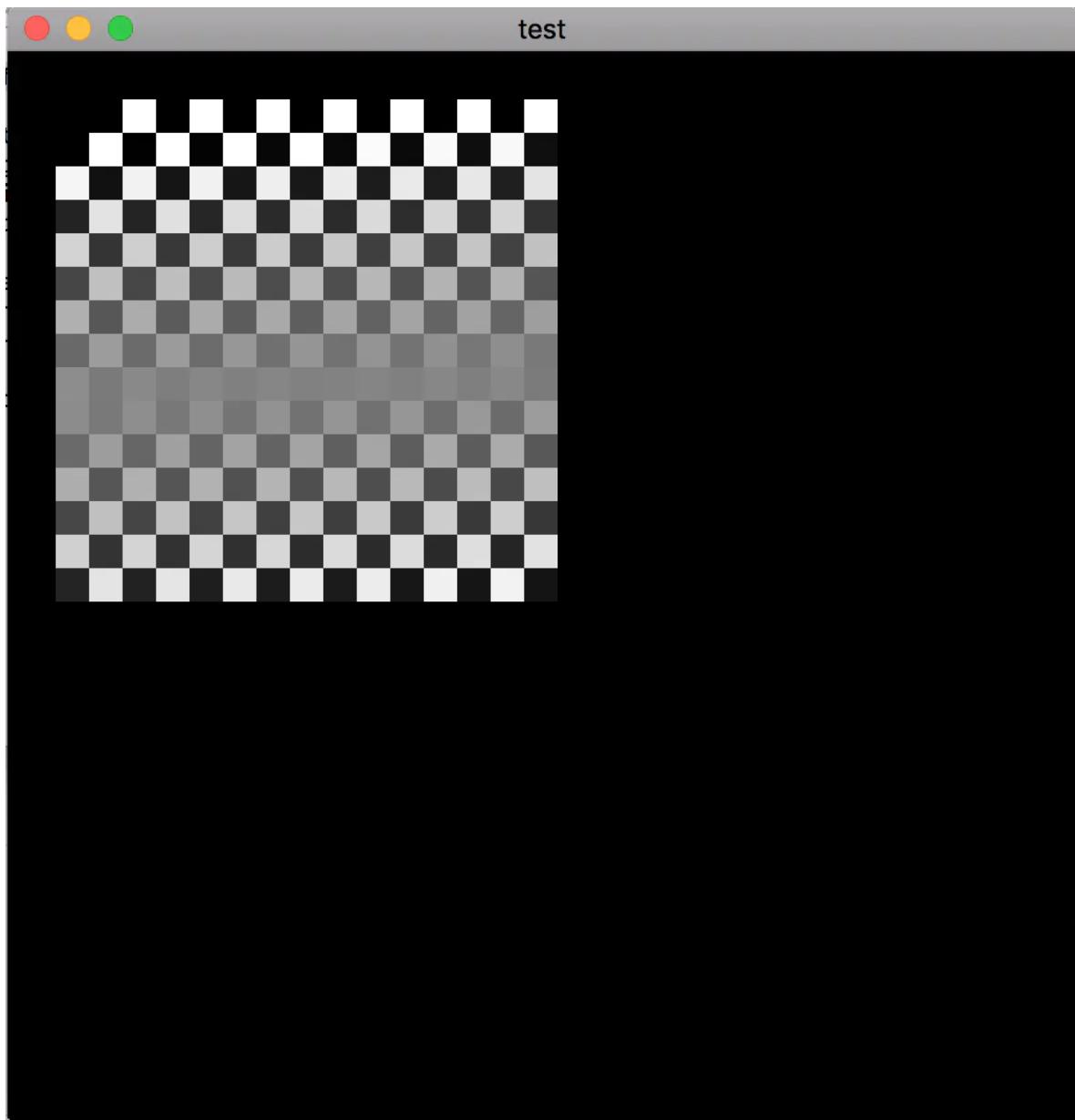
```
ffmpeg -f lavfi -i mptestsrc -vf blackframe -f sdl 'test'
```

```
[Parsed_blackframe_0 @ 0x2e31a80] frame:4017 pblack:100 pos:0 pts:4017 t:160.680000 type:I last_keyframe:4017
[Parserd_blackframe_0 @ 0x2e31a80] frame:4018 pblack:100 pos:0 pts:4018 t:160.720000 type:I last_keyframe:4018
[Parserd_blackframe_0 @ 0x2e31a80] frame:4019 pblack:100 pos:0 pts:4019 t:160.760000 type:I last_keyframe:4019
[Parserd_blackframe_0 @ 0x2e31a80] frame:4020 pblack:100 pos:0 pts:4020 t:160.800000 type:I last_keyframe:4020
frame:4054 pblack:100 pos:0 pts:4050 t:162.000000 type:I last_keyframe:4050 ipts:s
[Parserd_blackframe_0 @ 0x2e31a80] frame:4051 pblack:100 pos:0 pts:4051 t:162.040000 type:I last_keyframe:4051
[Parserd_blackframe_0 @ 0x2e31a80] frame:4052 pblack:100 pos:0 pts:4052 t:162.080000 type:I last_keyframe:4052
[Parserd_blackframe_0 @ 0x2e31a80] frame:4053 pblack:100 pos:0 pts:4053 t:162.120000 type:I last_keyframe:4053
frame:4054 pblack:100 pos:0 pts:4054 t:162.160000 type:I last_keyframe:4054 ipts:s
[Parserd_blackframe_0 @ 0x2e31a80] frame:4055 pblack:100 pos:0 pts:4055 t:162.200000 type:I last_keyframe:4055
[Parserd_blackframe_0 @ 0x2e31a80] frame:4056 pblack:100 pos:0 pts:4056 t:162.240000 type:I last_keyframe:4056
[Parserd_blackframe_0 @ 0x2e31a80] frame:4057 pblack:100 pos:0 pts:4057 t:162.280000 type:I last_keyframe:4057
[Parserd_blackframe_0 @ 0x2e31a80] frame:4058 pblack:100 pos:0 pts:4058 t:162.320000 type:I last_keyframe:4058
[Parserd_blackframe_0 @ 0x2e31a80] frame:4059 pblack:100 pos:0 pts:4059 t:162.360000 type:I last_keyframe:4059
[Parserd_blackframe_0 @ 0x2e31a80] frame:4060 pblack:100 pos:0 pts:4060 t:162.400000 type:I last_keyframe:4060
[Parserd_blackframe_0 @ 0x2e31a80] frame:4061 pblack:100 pos:0 pts:4061 t:162.440000 type:I last_keyframe:4061
[Parserd_blackframe_0 @ 0x2e31a80] frame:4062 pblack:100 pos:0 pts:4062 t:162.480000 type:I last_keyframe:4062
[Parserd_blackframe_0 @ 0x2e31a80] frame:4063 pblack:100 pos:0 pts:4063 t:162.520000 type:I last_keyframe:4063
[Parserd_blackframe_0 @ 0x2e31a80] frame:4064 pblack:100 pos:0 pts:4064 t:162.560000 type:I last_keyframe:4064
[Parserd_blackframe_0 @ 0x2e31a80] frame:4065 pblack:100 pos:0 pts:4065 t:162.600000 type:I last_keyframe:4065
[Parserd_blackframe_0 @ 0x2e31a80] frame:4066 pblack:100 pos:0 pts:4066 t:162.640000 type:I last_keyframe:4066
[Parserd_blackframe_0 @ 0x2e31a80] frame:4067 pblack:100 pos:0 pts:4067 t:162.680000 type:I last_keyframe:4067
[Parserd_blackframe_0 @ 0x2e31a80] frame:4068 pblack:100 pos:0 pts:4068 t:162.720000 type:I last_keyframe:4068
[Parserd_blackframe_0 @ 0x2e31a80] frame:4069 pblack:100 pos:0 pts:4069 t:162.760000 type:I last_keyframe:4069
[Parserd_blackframe_0 @ 0x2e31a80] frame:4070 pblack:100 pos:0 pts:4070 t:162.800000 type:I last_keyframe:4070
[Parserd_blackframe_0 @ 0x2e31a80] frame:4071 pblack:100 pos:0 pts:4071 t:162.840000 type:I last_keyframe:4071
[Parserd_blackframe_0 @ 0x2e31a80] frame:4072 pblack:100 pos:0 pts:4072 t:162.880000 type:I last_keyframe:4072
[Parserd_blackframe_0 @ 0x2e31a80] frame:4073 pblack:100 pos:0 pts:4073 t:162.920000 type:I last_keyframe:4073
[Parserd_blackframe_0 @ 0x2e31a80] frame:4074 pblack:100 pos:0 pts:4074 t:162.960000 type:I last_keyframe:4074
[Parserd_blackframe_0 @ 0x2e31a80] frame:4075 pblack:100 pos:0 pts:4075 t:163.000000 type:I last_keyframe:4075
[Parserd_blackframe_0 @ 0x2e31a80] frame:4076 pblack:100 pos:0 pts:4076 t:163.040000 type:I last_keyframe:4076
frame:4077 pblack:100 pos:0 pts:4077 t:163.080000 type:I last_keyframe:4077 ipts:s
[Parserd_blackframe_0 @ 0x2e31a80] frame:4078 pblack:100 pos:0 pts:4078 t:163.120000 type:I last_keyframe:4078
[Parserd_blackframe_0 @ 0x2e31a80] frame:4079 pblack:100 pos:0 pts:4079 t:163.160000 type:I last_keyframe:4079
[Parserd_blackframe_0 @ 0x2e31a80] frame:4080 pblack:100 pos:0 pts:4080 t:163.200000 type:I last_keyframe:4080
frame: 4110 pblack:100 pos:0 pts:4110 t:164.400000 type:I last_keyframe:4110 ipts:s
[Parserd_blackframe_0 @ 0x2e31a80] frame:4140 pblack:100 pos:0 pts:4140 t:165.600000 type:I last_keyframe:4140
frame:4170 pblack:100 pos:0 pts:4170 t:166.800000 type:I last_keyframe:4170 ipts:s
frame= 4182 fps= 55 q=-0.0 size= 0kB time=00:02:47.28 bitrate= 0.0kbit/s
```

我的测试命令如下：

```
ffmpeg -f lavfi -i mptestsrc -vf blackframe -f sdl 'test'
```

- 显示效果：



## 只选择指定的帧进行输出

特殊的多媒体过滤器可以选择音频并选择视频启用，以精确指定哪些帧将保留，哪些从输出中排除。

**Audio filter: aselect and Video filter: select**

<b>描述</b>	选择输出帧，对每个输入帧评估表达式，如果表达式的值不为零，则选择帧，否则跳过帧
<b>语法</b>	<code>select=expression expression</code> 默认值是1
	可用的参数
<b>n</b>	从0开始的过滤帧的连续编号
<b>selected_n</b>	所选帧的序号，从0开始
<b>prev_selected_n</b>	如果未定义，则最后一个选定帧的序列号为NAN
<b>TB</b>	输入时间戳的时基
<b>pts</b>	如果NAN未定义，那么经过滤波的视频帧的PTS（表示时间戳）以TB为单位表示
<b>t</b>	如果NAN未定义，则滤波的视频帧的PTS以秒表示
<b>prev_pts</b>	先前过滤的视频帧的PTS，如果未定义则为NAN
<b>prev_selected_pts</b>	如果NAN未定义，则最后一个先前过滤的视频帧的PTS
<b>prev_selected_t</b>	如果NAN未定义，则最后选择的最后一个视频帧的PTS
<b>start_pts</b>	如果NAN未定义，视频中第一个视频帧的PTS
<b>start_t</b>	如果NAN未定义，第一个视频帧的时间在视频中
<b>pict_type (只局限在视频中)</b>	过滤帧的类型，可以采用以下值之一：I ... 帧内预测帧, P ... 前向预测帧, B ... 双向预测帧, S ... 交换帧, SI ... 切换I帧, SP ... 切换P帧, BI ... 特殊帧内帧，不是关键帧 (VC-1视频编解码器)
<b>interlace_type (只局限在视频中)</b>	帧间型，可采用下列值之一: PROGRESSIVE, TOPFIRST, BOTTOMFIRST
<b>PROGRESSIVE</b>	帧是渐进的(不是交错的)
<b>TOPFIRST</b>	帧是top-field-first
<b>BOTTOMFIRST</b>	帧是bottom-field-first
<b>key</b>	如果经过筛选的帧是一个关键帧，否则为0
<b>pos</b>	如果信息不可用(例如合成视频)，在过滤帧的文件中位置为-1
<b>scene (仅局限于视频)</b>	0和1之间的值表示一个新的场景;低值反映了当前帧引入新场景的低概率，而更高的值意味着当前帧更可能是一个
<b>consumed_sample_n</b>	在当前帧之前选择的样本数目
<b>samples_n</b>	当前帧中的样本数目
<b>sample_rate</b>	输入采样率

由于**select**表达式的默认值为1，因此选择过滤器使用的接下来的两个示例会产生相同的结果 - 所有帧将被选择为输出（值如果是0将不会选择任何内容）：

```
ffmpeg -i input.avi -vf select output.avi  
ffmpeg -i input.avi -vf select=1 output.avi
```

我的测试命令如下：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf select  
/Users/zhangfangtao/Desktop/test3.mp4  
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf select=1  
/Users/zhangfangtao/Desktop/test3.mp4  
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf select=0  
/Users/zhangfangtao/Desktop/test3.mp4
```

- 实现效果，前两个命令生成的是和原来一样的视频，最后一个命令，只有音频信息，没有界面。

要选择20到25秒的部分，我们可以使用以下命令：

```
ffmpeg -i input.avi -vf select="gte(t\,20)*lte(t\,25)" output.avi
```

我的测试命令如下：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf select="gte\(\t\,20\)*lte\  
\(\t\,25\)" /Users/zhangfangtao/Desktop/test3.mp4
```

- 实现效果：

时间的限制不太准确，，，还是从头给我播放到了最后。。。

若要选择帧内仅为输出，我们可以使用以下命令：

```
ffmpeg -i input.avi -vf select="eq(pict_type\,I)" output.avi
```

我的测试命令如下：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf select="eq\(\pict_type\,I\)"  
/Users/zhangfangtao/Desktop/test3.mp4
```

- 实现的结果：

视频长短没有变，但是视频的大小几乎减少了一半。而且视频少了很多细节。

## 通过改变纵横比来缩放输入

[调整和伸缩视频](#)章节描述了缩放视频帧的缩放过滤器。另一种方法是使用改变显示宽高比DAR和样本宽高比SAR的setdar和setsar滤波器，它们的关系用公式表示（关于宽高比的细节在[词汇表](#)中）：

DAR = width/height \* SAR

## Video filters: setdar, setsar

描述	<b>setdar</b> 过滤器设置显示纵横比和 <b>setsar</b> 的样本纵横比
语法	setdar[=r=aspect_ratio[:max=number]] setsar[=aspect_ratio[:number]]
	参数的描述
r, ratio	纵横比, 值可以是浮点数或表达式, 默认值为0
max	在将纵横比设为一个有理数时, 表示分子和分母的最大整数值, 默认值为100

示例如何使用setdar和setsar过滤器:

```
ffplay -i input.avi -vf setdar=r=16/9  
ffplay -i input.avi -vf setdar=16/9  
ffplay -i input.avi -vf setsar=r=1.234  
ffplay -i input.avi -vf setsar=1.234
```

我的测试命令如下:

```
ffplay -i /Users/zhangfangtao/Desktop/test.mp4 -vf setdar=r=16/9  
ffplay -i /Users/zhangfangtao/Desktop/test.mp4 -vf setdar=16/9  
ffplay -i /Users/zhangfangtao/Desktop/test.mp4 -vf setsar=r=1.234  
ffplay -i /Users/zhangfangtao/Desktop/test.mp4 -vf setsar=1.234
```

- 显示效果如下:



屏幕抓取

为了将显示输出记录到视频文件中，例如创建一个教程，我们可以使用安装了UScreenCapture直接显示源过滤器的dshow输入设备，下面是下载地址。

<http://www.umediaserver.net/bin/UScreenCapture.zip>

为了抓取全屏内容，我们可以使用以下命令：

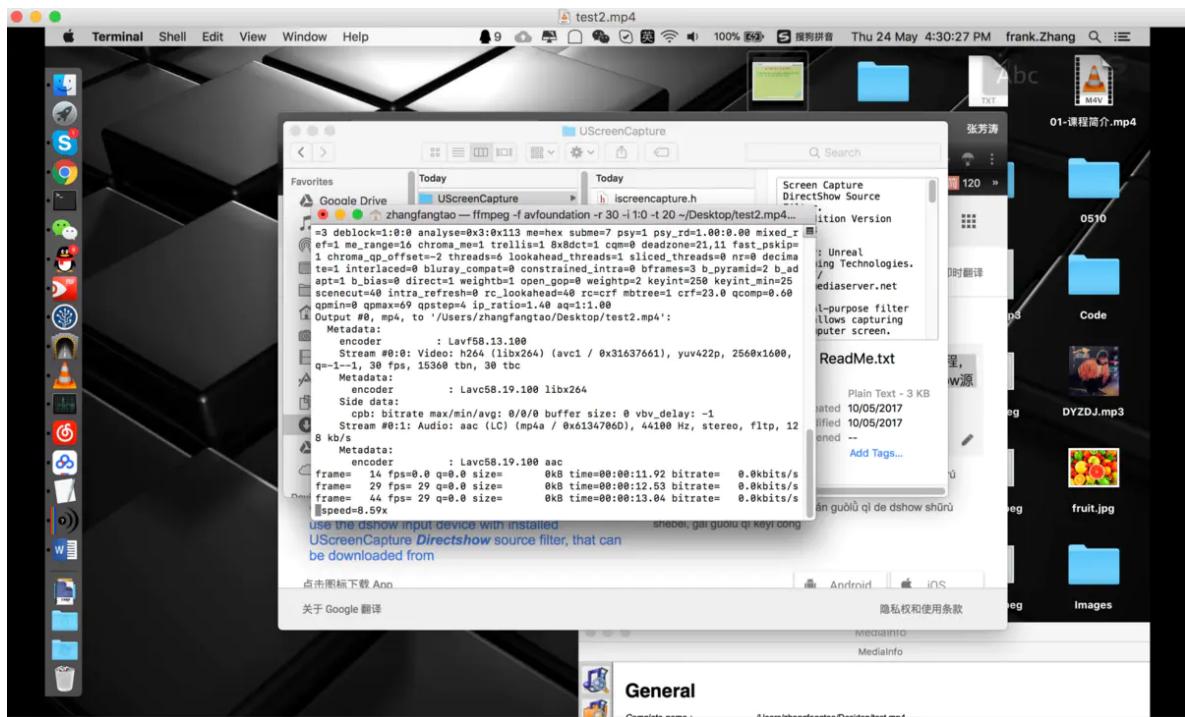
```
ffmpeg -f dshow -i video="UScreenCapture" -t 60 screen.mp4
```

如果我们想抓取一个特定的屏幕区域。我们必须使用regedit Windows工具来修改某些注册表项，相信信息在downloaded UScreenCapture.zip文件里面的README文件里面。

因为我用的MAC电脑，所以我的录屏命令如下：

```
ffmpeg -f avfoundation -r 30 -i "1:0" -t 20  
/Users/zhangfangtao/Desktop/test2.mp4
```

- 显示效果如下：



## 视频帧的详细信息

为了显示每个视频帧的信息，我们可以使用表格中描述的 showinfo 过滤器：

Video filter: showinfo

描述	显示包含有关每个输入视频帧信息的行，数据采用键：值对的形式。该过滤器没有参数，应与 <code>-report</code> 选项一起使用
语法	<code>-vf showinfo</code>
	显示参数的描述
n	输入框的序号，从0开始
pts	输入框的表示时间戳，表示为若干时间基单元;时间基单元依赖于过滤器输入板
pts_time	输入框的表示时间戳，表示为若干秒
pos	输入流中帧的位置，如果该信息不可用或没有意义(例如在合成视频中)
fmt	像素格式名称
sar	输入帧的采样宽高比，以分子/分母的形式表示
s	输入框的大小，以宽*长的形式表示
i	隔行扫描模式：P表示渐进式，T表示前场第一，B表示后场第一
iskey	如果该帧是关键帧，则为1，否则为0
type	输入帧的图像类型：I代表I帧，P代表P帧，B代表B帧，？对于未知类型（更多的信息见AVPictureType枚举的文档）
checksum	输入帧的所有平面的Adler-32校验和（十六进制）
plane_checksum	输入帧的每个平面的Adler-32校验和（十六进制），表示形式为[c0 c1 c2 c3]

例如，下一个命令会生成下面打印的信息，其中包括前三行：

```
ffmpeg -report -f lavfi -i testsrc -vf showinfo -t 10 showinfo.mpg

n:0 pts:0 pts_time:0 pos:-1 fmt:rgb24 sar:1/1 s:320x240 i:P iskey:1 type:I
checksum:88C4D19A plane_checksum:[88C4D19A]
n:1 pts:1 pts_time:0.04 pos:-1 fmt:rgb24 sar:1/1 s:320x240 i:P iskey:1 type:I
checksum:C4740AD1 plane_checksum:[C4740AD1]
n:2 pts:2 pts_time:0.08 pos:-1 fmt:rgb24 sar:1/1 s:320x240 i:P iskey:1 type:I
checksum:B6DD3DEB plane_checksum:[B6DD3DEB]
```

我的测试命令如下：

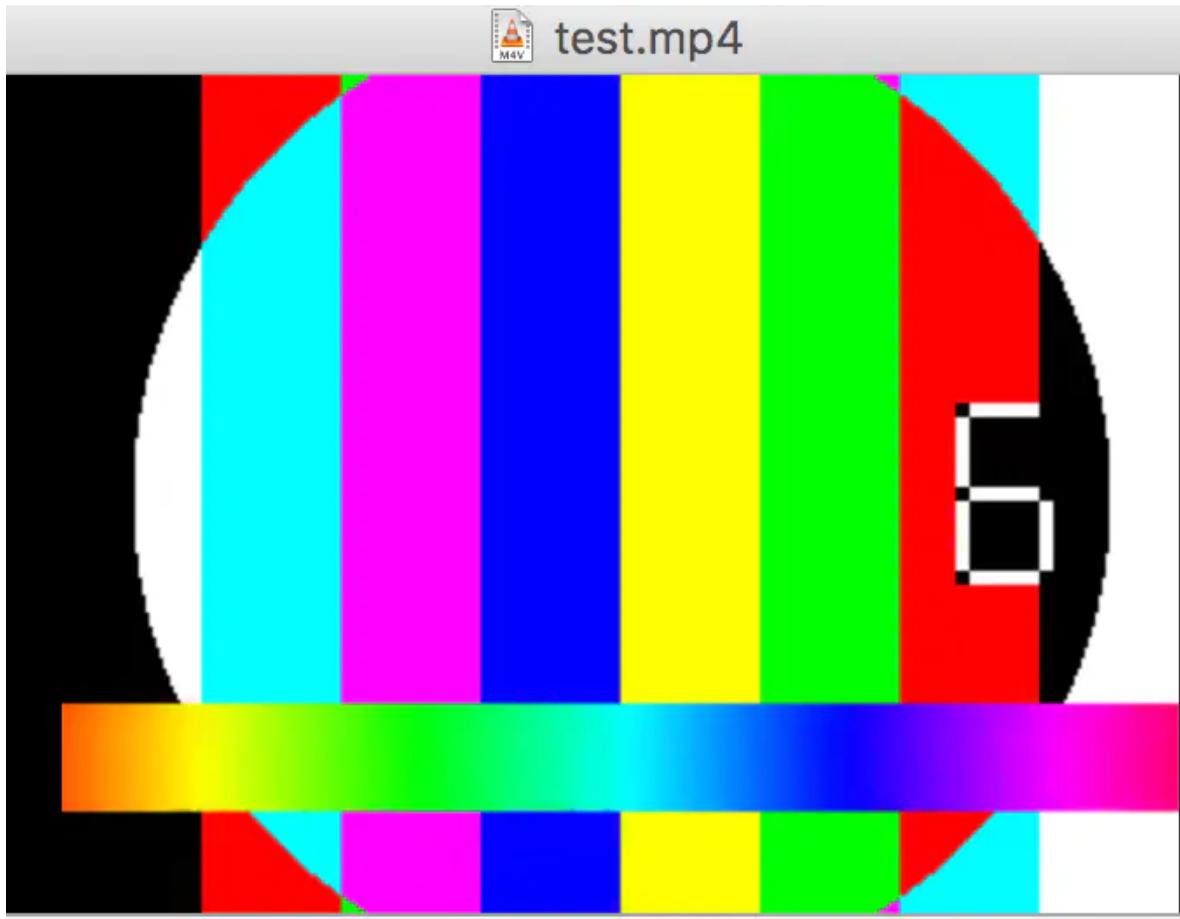
```
ffmpeg -report -f lavfi -i testsrc -vf showinfo -t 10
/Users/zhangfangtao/Desktop/test.mp4
```

- 显示效果：

```

cpb: bitrate max/min/avg: 0/0/0 buffer size: 0 vbv_delay: -1
[Parsed_showinfo_0 @ 0x7fb163f11d80] n: 1 pts: 1 pts_time:0.04 pos: -1 fmt:rgb24 sar: 1/1 s:320x240 i:P iskey:1 type:I checksum:C4740AD1 plane_checksum:[C4740AD1] mean:[127] stdev:[125.7]
[Parsed_showinfo_0 @ 0x7fb163f11d80] n: 2 pts: 2 pts_time:0.08 pos: -1 fmt:rgb24 sar: 1/1 s:320x240 i:P iskey:1 type:I checksum:B6DD3DEB plane_checksum:[B6DD3DEB] mean:[127] stdev:[125.7]
[Parsed_showinfo_0 @ 0x7fb163f11d80] n: 3 pts: 3 pts_time:0.12 pos: -1 fmt:rgb24 sar: 1/1 s:320x240 i:P iskey:1 type:I checksum:936E6BB1 plane_checksum:[936E6BB1] mean:[128] stdev:[125.7]
[Parsed_showinfo_0 @ 0x7fb163f11d80] n: 4 pts: 4 pts_time:0.16 pos: -1 fmt:rgb24 sar: 1/1 s:320x240 i:P iskey:1 type:I checksum:59759369 plane_checksum:[59759369] mean:[128] stdev:[125.7]
[Parsed_showinfo_0 @ 0x7fb163f11d80] n: 5 pts: 5 pts_time:0.2 pos: -1 fmt:rgb24 sar: 1/1 s:320x240 i:P iskey:1 type:I checksum:0930B896 plane_checksum:[0930B896] mean:[128] stdev:[125.6]
[Parsed_showinfo_0 @ 0x7fb163f11d80] n: 6 pts: 6 pts_time:0.24 pos: -1 fmt:rgb24 sar: 1/1 s:320x240 i:P iskey:1 type:I checksum:C86BD3B6 plane_checksum:[C86BD3B6] mean:[128] stdev:[125.6]
[Parsed_showinfo_0 @ 0x7fb163f11d80] n: 7 pts: 7 pts_time:0.28 pos: -1 fmt:rgb24 sar: 1/1 s:320x240 i:P iskey:1 type:I checksum:4CC2E982 plane_checksum:[4CC2E982] mean:[128] stdev:[125.6]
[Parsed_showinfo_0 @ 0x7fb163f11d80] n: 8 pts: 8 pts_time:0.32 pos: -1 fmt:rgb24 sar: 1/1 s:320x240 i:P iskey:1 type:I checksum:7B75F95F plane_checksum:[7B75F95F] mean:[128] stdev:[125.6]
[Parsed_showinfo_0 @ 0x7fb163f11d80] n: 9 pts: 9 pts_time:0.36 pos: -1 fmt:rgb24 sar: 1/1 s:320x240 i:P iskey:1 type:I checksum:15C00454 plane_checksum:[15C00454] mean:[128] stdev:[125.6]
[Parsed_showinfo_0 @ 0x7fb163f11d80] n: 10 pts: 10 pts_time:0.4 pos: -1 fmt:rgb24 sar: 1/1 s:320x240 i:P iskey:1 type:I checksum:754F0815 plane_checksum:[754F0815] mean:[128] stdev:[125.6]
[Parsed_showinfo_0 @ 0x7fb163f11d80] n: 11 pts: 11 pts_time:0.44 pos: -1 fmt:rgb24 sar: 1/1 s:320x240 i:P iskey:1 type:I checksum:08D505A9 plane_checksum:[08D505A9] mean:[128] stdev:[125.6]

```



## 音频频谱

为了使音频频谱可视化，我们可以使用表中描述的示波器滤波器：

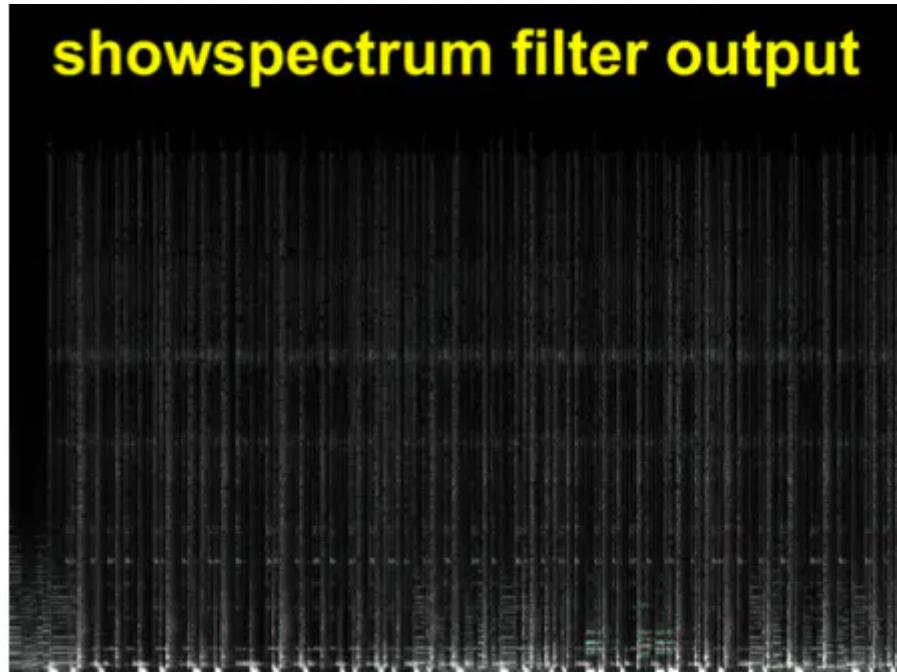
### Multimedia filter: showspectrum

描述	将音频输入转换为视频输出
语法	showspectrum[=s=widthxheight[:slide=number]]
	参数的描述
size, s	输出视频大小，默认值为640x480

描述	将音频输入转换为视频输出 设置频谱是否沿窗口滑动，默认值为0
----	-----------------------------------

例如，下面的图片显示了该命令创建的声谱：

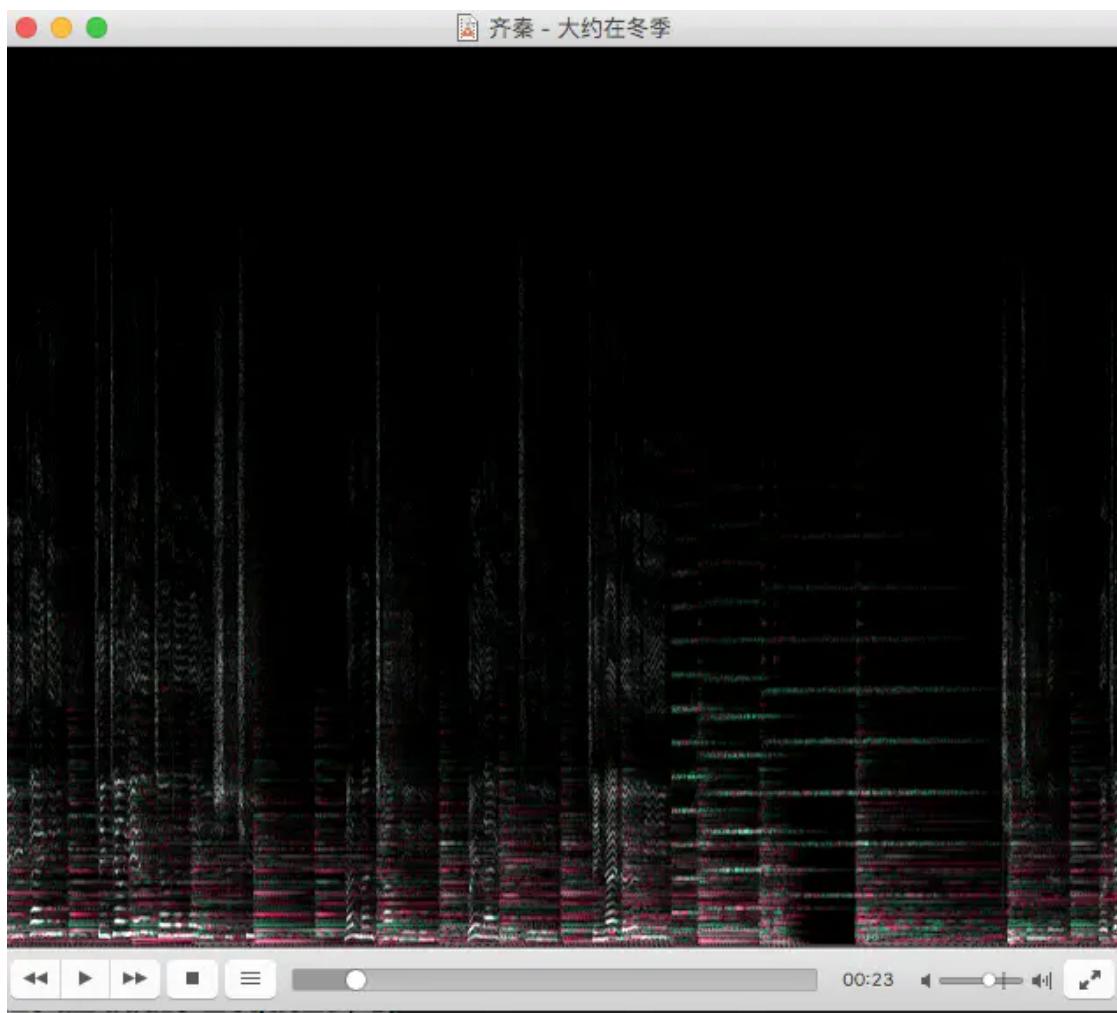
```
ffmpeg -i audio.mp3 -vf showspectrum audio_spectrum.mp4
```



我的测试命令如下：

```
ffmpeg -i /Users/zhangfangtao/Desktop/DYZDJ.mp3 -lavfi showspectrum  
/Users/zhangfangtao/Desktop/test3.mp4
```

- 效果图



## 音频波形可视化

描述	将输入音频转换为包含音频波表示的视频
语法	showwaves[=n=number[:r=rate[:s=video_size]]]
	参数的描述
n	打印在同一列上的样本数量越大，数值越大会降低帧速率，因此不能与速率参数结合使用
rate, r	帧速率，默认值为25，不能与n参数组合使用
size, s	视频大小，默认值是640x480

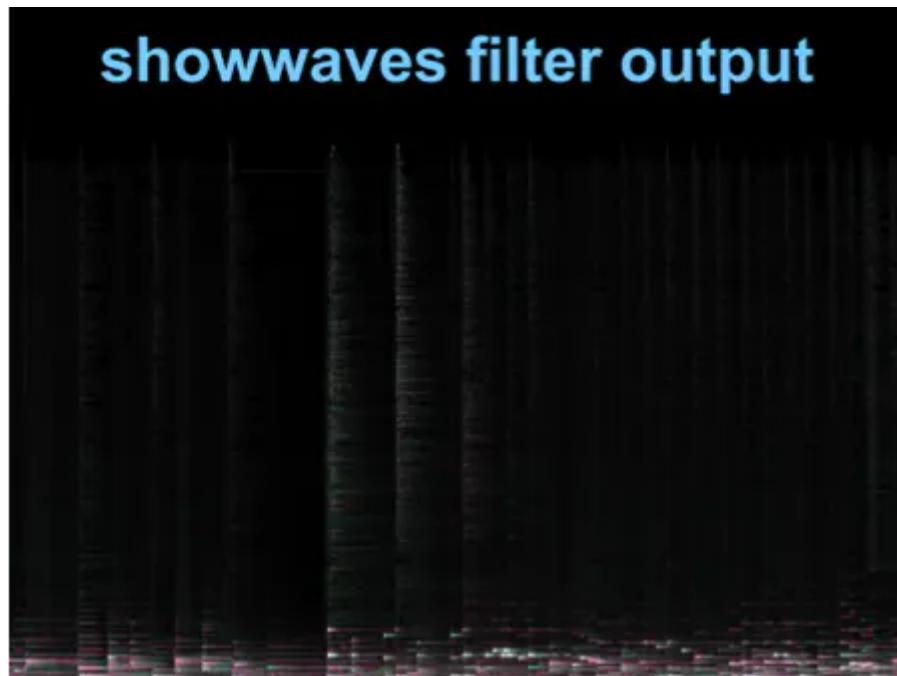
来自音频输入的波形可以通过表格中描述的showwaves滤波器进行可视化：

Multimedia filter: **showwaves**

描述	将输入音频转换为包含音频波表示的视频
语法	showwaves[=n=number[:r=rate[:s=video_size]]]
	参数的描述
n	打印在同一列上的样本数量越大，数值越大会降低帧速率，因此不能与速率参数结合使用
rate, r	帧速率，默认值为25，不能与n参数组合使用
size, s	视频大小，默认值是640x480

例如，要将music.mp3文件中的波形可视化为waves.mp4文件，我们可以使用以下命令：

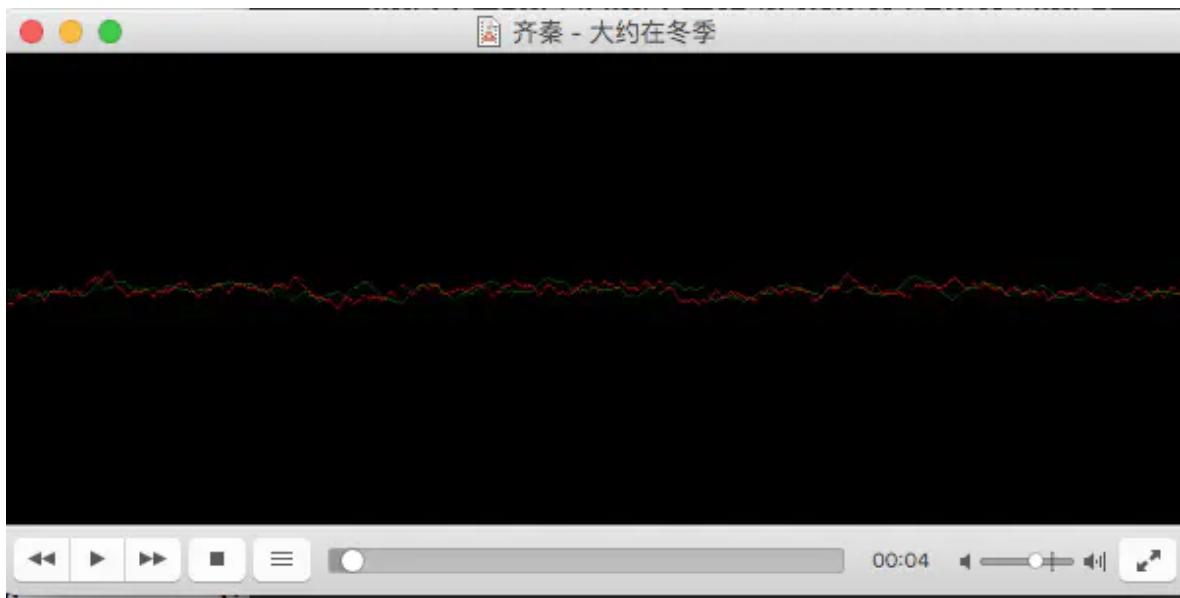
```
ffmpeg -i music.mp3 -vf showwaves waves.mp4
```



我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/NWDSL.mp3 -lavfi showwaves
/Users/zhangfangtao/Desktop/test3.mp4
```

- 显示效果如图：



## 语音合成

- 没有Windows环境，下面的没测试，不过AVFoundation做这个相当简单，[想了解AVFounadtion怎么实现的点击这里](#)

通过包含libflite外部库，人声可以与来自Flite（Festival Lite）（一种小型可嵌入TTS（文本到语音）引擎）的flite音频源合成。它由美国卡内基梅隆大学CMU Speech Group开发。Flite完全用C语言编写，重新实现了音乐节体系结构的核心部分，以实现为每个系统设计的声音之间的兼容性。爱丁堡大学的节日语音合成系统是构建语音合成系统的框架。有关Flite的更多详细信息，请访问<http://www.speech.cs.cmu.edu/flite>

Audio source: flite

描述	由于其大尺寸，使用未包含在官方Windows二进制文件中的libflite库来合成具有选定语音类型的人类语音
语法	flite="text"[ <i>:v=voice[:n=n_samples]</i> ] flite=textfile= <i>filename</i> [ <i>:v=voice[:n=n_samples]</i> ]
	参数的描述
list_voices	如果设置为1，则显示可用语音列表
<i>n</i> , <i>nb_samples</i>	每帧最大采样数，默认值为512
<i>text</i>	演讲的源文本
<i>textfile</i>	包含文本的文件名
<i>v, voice</i>	可用的声音：女 - slt, 男 - awb, kal, kal16, rms; 默认语音为kal, 其采样率（频率）为8000 Hz, 其他语音使用16000 Hz

由于flite库增加了10 MB以上的ffmpeg.exe文件，因此它不在官方二进制文件中，而Windows二进制文件可以从<http://ffmpeg.tv/flite.php>下载（Linux和OS X用户可以编译它们）。要显示可用语音列表，我们可以使用以下命令：

```
ffmpeg -f lavfi -i flite=list_voices=1
```

要让计算机使用女性声音从Message.txt文件中读取文本，该命令为：

```
ffplay -f lavfi -i flite=textfile=Message.txt:v=slt
```

例如，要将文字“Happy New Year to all”保存到wish.wav文件中，我们可以使用以下命令：

```
ffmpeg -f lavfi -i flite=text="Happy New Year to all":v=kal16 wish.wav
```

如果我们想减慢讲话速度以获得更好的听力，我们可以使用以下命令：

```
ffmpeg -f lavfi -i flite=textfile=text.txt -af atempo=0.5 speech.mp3
```

## 一次将输出保存为多种格式

虽然从第一章中解释的命令语法可以清楚看出，但我们可以使用一个命令将处理结果保存为多种格式，例如我们可以将flite语音引擎的输出保存为MP3，WAV和WMA格式在一个命令中：

```
ffmpeg -f lavfi -i flite=textfile=speech.txt speech.mp3 speech.wav speech.wma
```

我们还可以结合音频和视频格式，如果我们从视频输入格式指定音频格式，则只包含音频流，下一个示例中的文件clip.mp3只包含音频流：

```
ffmpeg -i clip.avi clip.flv clip.mov clip.mp3 clip.mp4 clip.webm
```

```
Stream mapping:  
Stream #0:0 -> #0:0 (mjpeg -> flv)  
Stream #0:1 -> #0:1 (pcm_s16le -> libmp3lame)  
Stream #0:0 -> #1:0 (mjpeg -> libx264)  
Stream #0:1 -> #1:1 (pcm_s16le -> libvo_aacenc)  
Stream #0:1 -> #2:0 (pcm_s16le -> libmp3lame)  
Stream #0:0 -> #3:0 (mjpeg -> libx264)  
Stream #0:1 -> #3:1 (pcm_s16le -> libvo_aacenc)  
Stream #0:0 -> #4:0 (mjpeg -> libvpx)  
Stream #0:1 -> #4:1 (pcm_s16le -> libvorbis)  
  
ffmpeg -i clip.avi clip.flv clip.mov clip.mp3 clip.mp4 clip.webm
```

## 额外的媒体输入到filtergraph

默认情况下，在任何具有-i选项的过滤器之前指定输入文件，并且第一个输入在带有[in]链接标签的过滤器图中可用。如果我们想要过滤额外的文件，我们可以使用amovie来源作为视频文件的音频和电影来源，它们在表格中描述：

#### Audio source: amovie & Video source: movie

描述	从媒体（电影）容器读取音频和/或视频流。必需参数是媒体文件的文件名，可选键=值对由冒号分隔
语法	movie=video_name[:options] amovie=audio_name[:options]
	Available key = 值选项参数中的值对
f, format_name	视频容器或输入设备的格式，如果未指定，则从扩展名确定或者探测
loop	按顺序读取数据流的次数，如果为-1，则选择最佳视频（具有amovie的音频）数据流
sp, seek_point	以秒为单位查找点，如果设置，则输入从给定时间开始
s, streams	- 要选择的流，用+符号指定多个流，顺序很重要 - 特殊名称dv（电影）和da（amovie）指定默认（最佳）视频/音频流 - 第一章介绍了如何指定特定流的语法
si, stream_index	要读取的流的索引，如果为-1，则选择最佳流，这是默认值（不建议使用，s参数为首选）

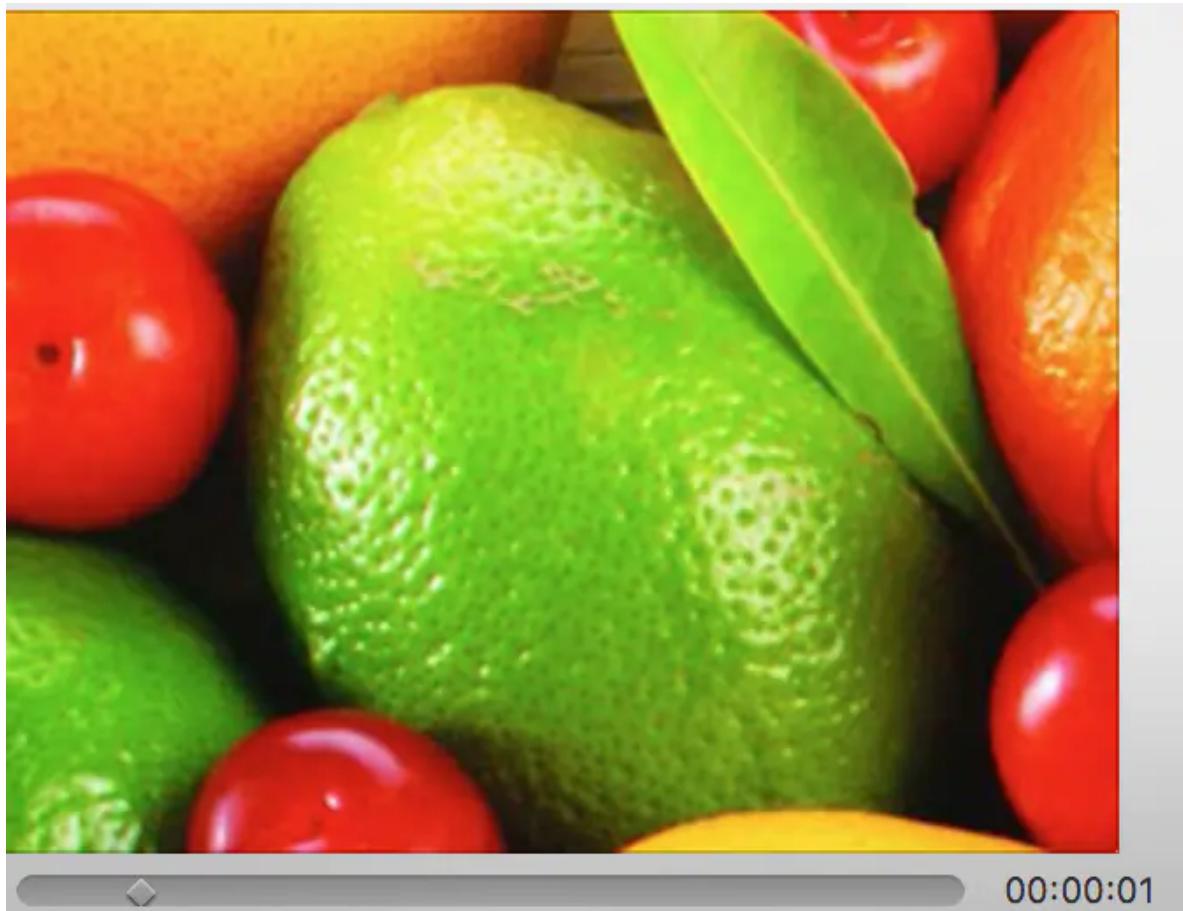
例如，要在输入视频上显示徽标，我们可以使用以下命令：

```
ffmpeg -i video.mpg -vf movie=logo.png[a];[in][a]overlay video1.mp4
```

我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/test.mp4 -vf  
"movie=/Users/zhangfangtao/Desktop/fruit.jpg[a];[in][a]overlay"  
/Users/zhangfangtao/Desktop/test3.mp4
```

\*显示效果(logo太大了，把视频盖住了)：



00:00:01

例如，sp (seek\_point) 选项设置为5时，徽标将从开始5秒后显示：

```
ffmpeg -i video.mpg -vf movie=logo.png:sp=5[a];[in][a]overlay video1.mp4
```

我的测试命令：

```
ffmpeg -i /Users/zhangfangtao/Desktop/001.mp4 -vf  
"movie=/Users/zhangfangtao/Desktop/fruit.jpg:sp=0[a];[in][a]overlay"  
/Users/zhangfangtao/Desktop/test3.mp4
```

- 报错：

```
[mov,mp4,m4a,3gp,3g2,mj2 @ 0x7fe477000000] Format mov,mp4,m4a,3gp,3g2,mj2 detect  
ed only with low score of 1, misdetection possible!  
[mov,mp4,m4a,3gp,3g2,mj2 @ 0x7fe477000000] moov atom not found  
/Users/zhangfangtao/Desktop/001.mp4: Invalid data found when processing input  
zhangfangtaodeMacBook-Pro:~ zhangfangtao$ ]
```

## 24-网页视频

由于其无所不在，互联网是用ffmpeg创建或编辑视频的最佳媒介。除了上传到YouTube, Vimeo等流行的视频分享网站之外，了解如何将媒体文件包含到网页中也很有用。为确保具有不同浏览器和媒体支持的用户可以收听并查看音频和视频，建议您为HTML5和Adobe Flash Player提供所有支持的格式的媒体文件。

## 主流的浏览器对HTML5的支持情况

使用HTML5在网络上添加媒体文件相对容易，并且有支持HTML5的设备，但不支持Flash Player，因此了解各种浏览器支持的媒体格式非常有用。FFmpeg能够将您的音频和视频转换为任何指定的HTML5格式。OGG容器格式的文件使用Theora视频编解码器和Vorbis音频编解码器，这些编解码器可以在商业项目中免费使用，WebM格式也提供相同的免费使用。请注意，默认情况下，ffmpeg使用FLAC编解码器编码OGG音频，这些浏览器无法播放，必须包含-acodec libvorbis选项。

### HTML5 Audio Support

浏览器	MP3	OGG*	WAV
Apple Safari 5+	yes	no	yes
Firefox 3.6+	no	yes	yes
Google Chrome 6+	yes	yes	yes
Internet Explorer 9+	yes	no	no
Opera 10.6+	yes	yes	yes
Maxthon 3+	yes	yes	yes

### HTML5 Video Support

浏览器	MP4	OGG	WEBM
Apple Safari 5+	yes	no	no
Firefox 3.6+	no	yes	yes
Google Chrome 6+	yes	yes	yes
Internet Explorer 9+	yes	no	no
Opera 10.6+	no	yes	yes
Maxthon 3+	yes	yes	yes
如果设置，则显示控件：播放，暂停，寻找，音量			
对于Internet Explorer的HTML5支持可以从版本9获得，以前的版本6,7和8可以从中安装Google Chrome Frame插件			

<https://developers.google.com/chrome/chrome-frame>

在线测试您的浏览器如何支持特定的位于网络上的HTML5功能

<http://html5test.com>

## 使用HTML5添加音频

为了在任何主流浏览器上播放我们的音频，最通用的格式是在除Firefox以外的所有浏览器上支持MP3，而对于Firefox我们提供OGG或WAV格式。用于音频包含的HTML5中的新标签是表中描述的标签：

属性	值	描述
autoplay	autoplay	如果设置，音频开始准备就绪时播放
controls	controls	如果设置，则显示控件：播放，暂停，寻找，音量
loop	loop	如果设置，音频播放一遍又一遍
preload	auto metadata none	auto - 整个音频文件被加载 metadata - 只加载元数据 none - 音频文件未与网页一起加载 不要将它与自动播放属性一起使用；它最近在IE和Opera中不支持
src	URL	音频文件的绝对或相对URL

因为我们希望为同一个音频指定至少两个不同格式的文件，所以标签的src属性不会被使用，并且在打开和关闭</ audio>标签之间会添加多个标签。浏览器将扫描包含的媒体文件，并选择它支持播放的第一个文件。

标签（所有的属性都是可选的）

属性	值	描述
media	media_query	现在没有浏览器支持它，描述了媒体资源的类型
src	URL	音频文件的绝对或相对URL
type	MIME_type	最近媒体资源的MIME类型：audio:音频/ mpeg，音频/ ogg，音频/ wav video:视频/ mp4，视频/ ogg，视频/ webm

下一个HTML代码包括带有显示控件和循环的音频文件，它被保存到文本文件中，并与其他HTML元素（如文档类型，头，标题，正文，div等）一起调用audio.htm。

```
<audio controls='controls' loop='loop'>
    <source src='music.mp3' type='audio/mpeg' />
    <source src='music.ogg' type='audio/ogg' />
    Audio element is not supported in your browser, please update.
</audio>
```

要开始自动播放，我们可以添加属性autoplay ='autoplay'。

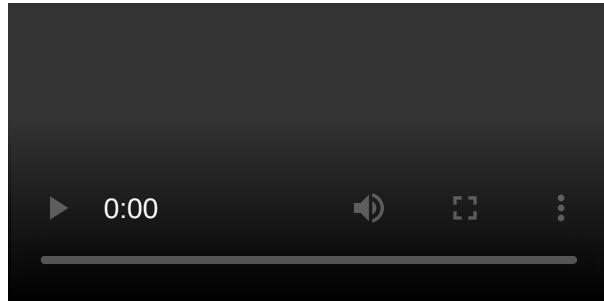
各种浏览器中的音频控制

浏览器	音频播放器
Firefox 4	
Google Chrome 6	
Internet Explorer 9	
Maxthon 3	
Opera 12	

## 使用HTML5添加视频

HTML5中的视频标签是一个

标签，其属性自动播放，控制，循环和静音使用等于属性名称的值（例如loop ='loop'），但许多浏览器接受跳过此值，生产使用我们可以使用



。

属性	值	描述
autoplay	autoplay	如果已设置，视频在准备就绪时开始播放
controls	controls	如果设置，则显示按钮控制：播放，暂停，寻找，音量，切换全屏，字幕等
height	像素	视频播放器的高度
loop	loop	如果设置，视频会一遍又一遍播放
muted	静音	如果设置，音频流静音，最近在Apple Safari和Internet Explorer中不支持
poster	URL	在视频下载期间显示的图像文件的URL（如果不存在）显示的是视频的第一帧
preload	auto metadata none	auto -整个音频文件被加载 metadata -只加载元数据 none - 音频文件未与网页一起加载 不要将它与自动播放属性一起使用；现在它不适用于IE
src	URL	视频文件的绝对或相对URL

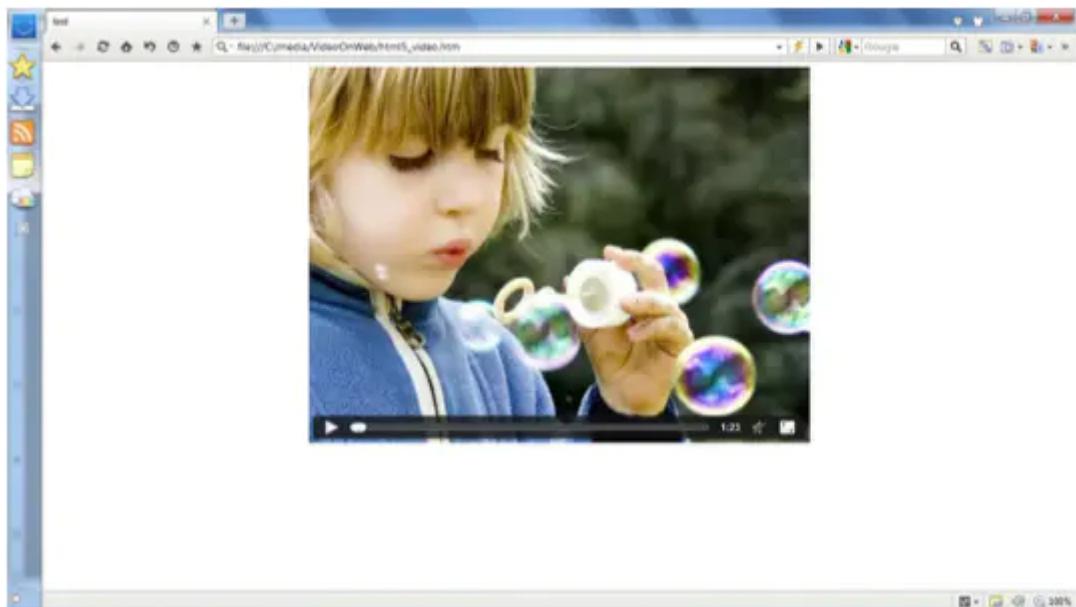
为了在所有主流浏览器上都能看到视频，我们必须提供至少2种不同的格式，最好的选择是MP4和WEBM。因此，不会使用

标签的src属性，并在开始的

和关闭</ video>标签之间添加前面章节中描述的多个标签。浏览器将扫描包含的媒体文件，并选择它支持播放的第一个文件。

例如，要包含带有显示控件和循环的视频文件，我们可以使用HTML代码：

```
<video controls='controls' loop='loop' width='640' height='480'>
    <source src='videoclip.mp4' type='video/mp4' />
    <source src='videoclip.webm' type='video/webm' />
    video element is not supported in your browser, please update.
</video>
```



## 为Flash Player添加视频

对于不支持HTML5的浏览器，我们可以在

标签中包含SWF格式的标签（ffmpeg -f videotape videotape.swf）。标签包含不支持标签的浏览器的标签和标签。

```
<object width='400' height='300'>
  <param name='src' value='videoclip.swf' />
  <param name='loop' value='true' />
  <embed src='videoclip.swf' width='400' height='300' loop='true' />
</object>
```

## 视频分享网站

YouTube成功推出视频共享服务之后，还有许多英文和其他语言的类似网站。YouTube仍然是最受欢迎的，但其他一些服务器提供了更多功能。几乎所有的视频共享网站都支持以下媒体格式：3gp, avi, asf, flv, mkv, mp4, mpegps, mov, ogg, wmv等。最流行的视频共享网站列表位于表格中。  
最受欢迎的视频分享网站

名称	每月访问者	描述(每月的访客数量仅来自美国，并在增长)
YouTube <a href="http://youtube.com">youtube.com</a>	800,000,000	- 最受欢迎的视频网站，总访问量排名第三的网站，每天的视频浏览量超过40亿次 - 1080p高清视频，最大2 GB和15分钟 - 支持4k格式的3D视频和视频（4096x3072分辨率 - 可用于手机，iPod，PlayStation，Xbox等 - Flash Player和HTML5视频 - 视频编辑器，字幕等 - 用户评论，评分，视频回复等 - 不支持的图像和音频文件
DailyMotion <a href="http://dailymotion.com">dailymotion.com</a>	61,000,000	高清视频，最大文件大小2 GB和60分钟，音频90 kbps MP3或AAC，Flash Player或HTML5 用户可以从图像创建幻灯片，最多30幅图像，MP4输出
Vimeo <a href="http://vimeo.com">vimeo.com</a>	17,000,000	高清视频（1920x1080），最高文件大小5 GB，无限期 - Flash Player，HTML5 - 注册用户超过800万，每月独立访问者达到6500万 - 编码视频比特率最高（平均2000 kbps，最高5000
Metacafe <a href="http://metacafe.com">metacafe.com</a>	9,200,000	- 短片娱乐视频（电影，游戏，音乐，体育，电视剪辑等） - 最大。文件大小100 MB，视频转换为320x240 FLV，VP6，比特率330 kbps，MP3音频 - 每天1700万次观看，全球每月4000万独立访客

名称	每月访问者	描述 (每月的访客数量仅来自美国，并在增长)
Break break.com	6,800,000	- 有趣的视频（流行文化，生活方式，交通，游戏等） - 被推荐到主页的视频将获得奖励：首先是\$ 400，第二个\$ 500，第三个和接下来是\$ 600
Veoh veoh.com	6,100,000	- 视频可以按照系列和频道进行组织 - 在几个国家被封锁（被墙了。。。。）
RuTube rutube.com	4,000,000	最大的文件大小300 MB, VGA分辨率, Flash Player; 主要用户在俄罗斯，交互界面还是不错的，每个月的独立访客30万，该网站也可以登录Facebook
Internet Archive archive.org	1,600,000	- 用户可以上传视频，音频，文档，免费书籍等 - 永久存储
Multiply.com	695,000	用户档案，非常受欢迎
Qik.com	505,000	基于移动设备
Phanfare.com	323,000	照片和视频
Sevenload.com	192,000	在几个国家被封锁
OpenFilm.com	114,000	电影，音乐，社区
ScienceStage.com	100,000	以科学为导向的媒体门户，用户还可以上传mp3, vob和swf格式的文件

## Web服务器上的视频处理

由于ffmpeg和视频共享网站的流行，一些虚拟主机公司提供对ffmpeg服务器上视频处理的支持，这需要比传统网站更大的CPU负载。包含他们提供的参数的几个虚拟主机的预览在表中：

带有FFmpeg支持的虚拟主机服务

Name URL	描述
CirtexHosting <a href="http://www.cirtexhosting.com">www.cirtexhosting.com</a>	支持: FFmpeg, FFmpeg-PHP, Mplayer + Mencoder + Yamdi + Yasm, flv2tool + GD库, Xvidcore + Faac + Faad2, Libogg + Libvorbis + Libtheora, Libx264 + Libopencore-amrnb + Libopencore-amrwb, LAME MP3 Encoder
GlowHost <a href="http://www.glowhost.com">www.glowhost.com</a>	可用模块: FFmpeg和FFmpeg-PHP, GD Library 2+, MPlayer和MEncoder, FAAD / FAAC, FLVTool2, Libogg和Libvorbis, LAME MP3编码器, x264 / H.264, MPEG-4 AVC
HostUpon <a href="http://www.hostupon.com">www.hostupon.com</a>	所有模块启动视频网站, Youtube克隆或社交网络与视频上传。 FFmpeg托管脚本: Boonex Dolphin, PHPMotion, 社交引擎, ABKsoft脚本, Joomla视频插件, Clipshare, ClipBucket, 社交媒体, Rayzz, Vidi脚本等
PacificHost.com <a href="http://www.pacifichost.com">www.pacifichost.com</a>	可以选择创建和运行YouTube等在线视频分享网站。他们使用软件来转换视频: ffmpeg-php, mplayer, mencoder, flvtool2, lame, libogg, libvorbis, xvid, theora, faac, phpshield加载器。 PacificHost的FFmpeg包含以下模块: libfaac, libfaad, libxvid, libamr-nb, libamr-wb, libgsm, libogg, libtheora和libvorbis
VPSDeploy <a href="https://vpsdeploy.com/whm-cpanel-ssd-hosting.php">https://vpsdeploy.com/whm-cpanel-ssd-hosting.php</a>	托管: FFmpeg支持, flvtool2, X.264插件, libogg, flac和LibTheora用于videostreaming

### FFmpeg Web Hosting

GlowHost offers FFmpeg web hosting support for your video sharing applications. FFmpeg's popularity stems from its accessibility. Users don't have to know much about multimedia formats. They just upload whatever file they have and FFmpeg automatically converts it to a file format which will play in any web browser that has Flash Player enabled. If you have seen video on YouTube, you have seen the end result of FFmpeg in action. Your app will allow your users to upload video to your web site in all of the most popular video formats, and will pump them out into something anyone can watch, even full High Definition video.



#### FFmpeg & Associated Modules Include:

- |                         |                        |
|-------------------------|------------------------|
| ✓ FFmpeg and FFmpeg-PHP | ✓ FLVTool2             |
| ✓ GD Library 2+         | ✓ Libogg and Libvorbis |
| ✓ MPlayer and MEncoder  | ✓ LAME MP3 Encoder     |
| ✓ FAAD/FAAC             | ✓ x264 / H.264         |
|                         | ✓ MPEG-4 AVC           |

## 通过视频上传获利

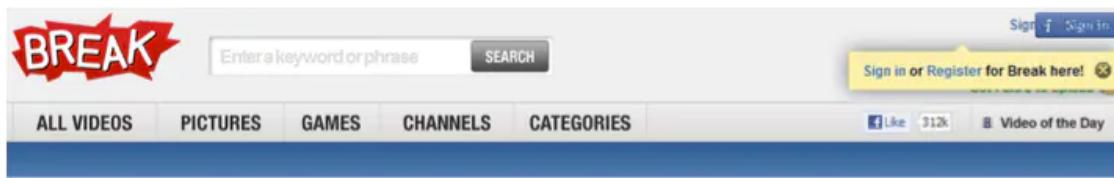
一些视频分享网站提供了查看上传视频的付款方式:

- YouTube合作伙伴计划为创作者提供工具和程序, 以构建受众群体并通过视频获利。获利的视频会显示增加, 更多信息位于网页[http://www.youtube.com/account\\_monetization](http://www.youtube.com/account_monetization)
- [Blip.tv](#)从包含的附加费中支付50%, 但创作者必须上传原始视频系列, 而不是所有系列都被接受, 有关更多信息, 请访问<http://blip.tv/users/apply>
- [NowVideo.eu](#)为每1000个视频流支付10美元 (完整的访问者浏览量), 详情请见<http://www.nowvideo.eu/affiliate.php>
- Break.com

从用户中选择有趣的视频并将它们包含到其主页中。这些视频的创作者将为每个视频从200美元到600美元进行支付，金额取决于用户是否决定出售或许可视频和其他条件，更多信息请访问

<http://info.break.com/break/html/>

。upload\_videos.html



The screenshot shows a section of the Break website dedicated to video uploads. It features a large image of a silver digital camcorder on the left. To the right, there are two main sections: "UPLOAD" (number 1) and "UPLOAD MORE VIDEOS" (number 2). The "UPLOAD" section includes text about payment for featured content and a link to the homepage. The "UPLOAD MORE VIDEOS" section includes information about payment for multiple uploads and links to "UPLOAD NOW" and "CHECK OUT THE WINNERS PAGE". A note at the bottom specifies that original videos must be filmed by the user and not contain copyrighted material. A "Get Started Now" button is located at the bottom left of the upload area.

## 25-调试和测试

为了检测错误并测试各种输入，参数，性能等，我们可以使用多个FFmpeg过滤器，选项和来源。当控制台输出很长时，-report选项会将测试结果保存到名为ffmpeg-yyyyymmdd-hhmmss.log的文件中，其中斜体部分表示当前日期和时间。

### debug, debug\_ts and fdebug选项

FFmpeg中的基本调试工具是一个-debug选项，其中在下表中描述了17个可能值：

<b>描述</b>	打印关于所选音频, 字幕或视频流的特定调试信息
<b>语法</b>	<code>-debug[:stream_specifier]</code>
	可用值的描述
<b>pict</b>	图片信息
<b>rc</b>	速率控制
<b>bitstream</b>	比特流
<b>mb_type</b>	宏块 (MB) 类型
<b>qp</b>	每块量化参数 (QP)
<b>mv</b>	运动矢量
<b>dct_coeff</b>	DCT系数
<b>skip</b>	进度跳跃
<b>startcode</b>	起始码
<b>pts</b>	演示文稿时间戳
<b>er</b>	错误识别
<b>mmco</b>	内存管理控制操作 (H.264)
<b>bugs</b>	bugs(错误)
<b>vis_qp</b>	可视化量化参数 (QP), 较低的QP是更绿色的
<b>vis_mb_type</b>	可视化块类型
<b>buffers</b>	图片缓冲区分配
<b>thread_ops</b>	线程操作

例如, 我们将mptestsrc源的短输出保存为具有mmco值的MP4 (H.264) 文件:

```
ffmpeg -debug mmco -f lavfi -i mptestsrc -t 0.5 output.mp4
```

向控制台输出添加了12行描述各个帧; 包含的术语说明:

- QP - 量化参数
- NAL - 网络抽象层单元
- 切片: B - 双向预测, I - 帧内编码, P - 预测

```
[libx264 @ 03dec60] frame=  0 QP=13.00 NAL=3 Slice:I Poc:0   I:1024 P:0     SKIP:0      size=93 bytes
[libx264 @ 03dec60] frame=  1 QP=31.03 NAL=2 Slice:I Poc:2   I:1024 P:0     SKIP:0      size=1376 bytes
[libx264 @ 03dec60] frame=  2 QP=21.76 NAL=2 Slice:P Poc:10  I:1       P:222    SKIP:801      size=253 bytes
[libx264 @ 03dec60] frame=  3 QP=24.00 NAL=2 Slice:B Poc:6   I:1       P:8      SKIP:1014      size=44 bytes
[libx264 @ 03dec60] frame=  4 QP=21.00 NAL=0 Slice:B Poc:4   I:1       P:2      SKIP:1021      size=26 bytes
[libx264 @ 03dec60] frame=  5 QP=22.94 NAL=0 Slice:B Poc:8   I:1       P:5      SKIP:1018      size=32 bytes
[libx264 @ 03dec60] frame=  6 QP=25.26 NAL=2 Slice:P Poc:18  I:5       P:158    SKIP:861      size=238 bytes
[libx264 @ 03dec60] frame=  7 QP=23.51 NAL=2 Slice:B Poc:14  I:2       P:7      SKIP:1015      size=45 bytes
[libx264 @ 03dec60] frame=  8 QP=23.99 NAL=0 Slice:B Poc:12  I:3       P:6      SKIP:1015      size=46 bytes
[libx264 @ 03dec60] frame=  9 QP=25.47 NAL=0 Slice:B Poc:16  I:1       P:9      SKIP:1012      size=46 bytes
[libx264 @ 03dec60] frame= 10 QP=28.09 NAL=2 Slice:P Poc:24 I:6       P:39    SKIP:979      size=118 bytes
[libx264 @ 03dec60] frame= 11 QP=26.42 NAL=2 Slice:B Poc:22 I:1       P:13    SKIP:1009      size=63 bytes
[libx264 @ 03dec60] frame= 12 QP=25.49 NAL=0 Slice:B Poc:20 I:4       P:9      SKIP:1011      size=62 bytes
```

另一个调试选项是`-debug_ts`, 它可以在处理期间打印时间戳信息, 例如我们可以修改前面的示例并仅使用0.1秒(3帧) :

```
ffmpeg -debug_ts -f lavfi -i mp4testsrc -t 0.1 output.mp4
```

向控制台输出添加下一行:

```
demuxer  -> ist_index:0 type:video next_dts:NPTS next_dts_time:NPTS next_pts:NPTS
next_pts_time:NPTS pkt_pts:0 pkt_pts_time:0 pkt_dts:0 pkt_dts_time:0 off:0
decoder  -> ist_index:0 type:video frame_pts:0 frame_pts_time:0 best_effort_ts:0
best_effort_ts_time:0 keyframe:1 frame_type:1
[libx264 @ 023ef4c0] using mv_range_thread = 88
demuxer  -> ist_index:0 type:video next_dts:40000 next_dts_time:0.04 next_pts:40000
next_pts_time:0.04 pkt_pts:1 pkt_pts_time:0.04 pkt_dts:1 pkt_dts_time:0.04 off:0
decoder  -> ist_index:0 type:video frame_pts:1 frame_pts_time:0.04 best_effort_ts:1
best_effort_ts_time:0.04 keyframe:1 frame_type:1
demuxer  -> ist_index:0 type:video next_dts:80000 next_dts_time:0.08 next_pts:80000
next_pts_time:0.08 pkt_pts:2 pkt_pts_time:0.08 pkt_dts:2 pkt_dts_time:0.08 off:0
decoder  -> ist_index:0 type:video frame_pts:2 frame_pts_time:0.08 best_effort_ts:2
best_effort_ts_time:0.08 keyframe:1 frame_type:1
demuxer  -> ist_index:0 type:video next_dts:120000 next_dts_time:0.12 next_pts:120000
next_pts_time:0.12 pkt_pts:3 pkt_pts_time:0.12 pkt_dts:3 pkt_dts_time:0.12 off:0
decoder  -> ist_index:0 type:video frame_pts:3 frame_pts_time:0.12 best_effort_ts:3
best_effort_ts_time:0.12 keyframe:1 frame_type:1
No more output streams to write to, finishing.
[libx264 @ 023ef4c0] scene cut at 1 Icost:252652 Pcost:248376 ratio:0.0169 bias:0.0250 gop:1 (imb:0
pmb:900)
[libx264 @ 023ef4c0] frame=  0 QP=13.00 NAL=3 Slice:I Poc:0   I:1024 P:0     SKIP:0      size=93
bytes
muxer <- type:video pkt_pts:0 pkt_pts_time:0 pkt_dts:-1024 pkt_dts_time:-0.08 size:782
[libx264 @ 023ef4c0] frame=  1 QP=21.48 NAL=2 Slice:I Poc:2   I:1024 P:0     SKIP:0      size=926
bytes
muxer <- type:video pkt_pts:512 pkt_pts_time:0.04 pkt_dts:-512 pkt_dts_time:-0.04 size:926
[libx264 @ 023ef4c0] frame=  2 QP=26.98 NAL=2 Slice:P Poc:4   I:1       P:6      SKIP:1017      size=50
bytes
muxer <- type:video pkt_pts:1024 pkt_pts_time:0.08 pkt_dts:0 pkt_dts_time:0 size:50
```

选项`fdebug`只有1个可能的值`ts`, 通常与`-debug_ts`选项一起用于各种测试, 例如调试DTS(解码时间戳)和PTS(演示时间戳)关系。使用上例中的修改后的命令, 控制台输出显示命令后列出的添加行:

```
ffmpeg -fdebug ts -f lavfi -i mp4testsrc -t 0.1 output.mp4
```

```

[libx264 @ 0206cb00] using mv_range_thread = 88
[lavfi @ 0229c2c0] ff_read_packet stream=0, pts=1, dts=NPTS, size=393216, duration=0, flags=0
[lavfi @ 0229c2c0] read_frame_internal stream=0, pts=1, dts=1, size=393216, duration=1, flags=1
[lavfi @ 0229c2c0] ff_read_packet stream=0, pts=2, dts=NPTS, size=393216, duration=0, flags=0
[lavfi @ 0229c2c0] read_frame_internal stream=0, pts=2, dts=2, size=393216, duration=1, flags=1
[lavfi @ 0229c2c0] ff_read_packet stream=0, pts=3, dts=NPTS, size=393216, duration=0, flags=0
[lavfi @ 0229c2c0] read_frame_internal stream=0, pts=3, dts=3, size=393216, duration=1, flags=1
No more output streams to write to, finishing.
[libx264 @ 0206cb00] scene cut at 1 Icost:252652 Pcost:248376 ratio:0.0169 bias:0.0250 gop:1 (imb:0 pmb:900)
[libx264 @ 0206cb00] frame=   0 QP=13.00 NAL=3 Slice:I Poc:0   I:1024 P:0    SKIP:0    size=93 bytes
[libx264 @ 0206cb00] frame=   1 QP=21.48 NAL=2 Slice:I Poc:2   I:1024 P:0    SKIP:0    size=926 bytes
[libx264 @ 0206cb00] frame=   2 QP=26.98 NAL=2 Slice:P Poc:4   I:1      P:6    SKIP:1017 size=50 bytes

```

## 用于错误检测的标志

检测ffmpeg处理中的错误可以通过表中描述的-err\_detect选项指定：

描述	检测一个错误，该标志指定了哪种类型
语法	-err_detect[:stream_specifier] flag
	可用标志的描述
aggressive	考虑一个理智的编码器不应该做的错误
bitstream	检测比特流指定偏差
buffer	检测不合适的比特流长度
careful	考虑违反规范并且没有被视为错误的东西
compliant	将所有规范不合规视为错误
crccheck	验证嵌入式CRC
explode	终止对较小错误检测的解码

例如，要检测不正确的比特流长度，我们可以使用以下命令：

```
ffmpeg -report -err_detect buffer -i input.avi output.mp4
```

## 日志记录级别设置

日志记录级别确定处理过程中控制台输出中显示的内容，可用的修改值包括：安静，恐慌，致命，错误，警告，信息，详细，调试。要设置日志记录级别，我们可以使用选项-v或-loglevel选项，例如对于详细级别，我们可以使用以下命令：

```
ffmpeg -loglevel verbose -i input.avi output.mp4
```

## 时间基配置测试

过滤器asettb和settb用于测试时基配置，asettb用于音频输入并用于视频输入的建立。两个过滤器都具有相同的参数，并在公共表中进行了描述：

### Audio filter: asettb & Video filter: settb

描述	两个滤镜都设置时基，它将用于输出帧时间戳。此设置用于测试时基配置和类似功能。 两个过滤器的语法和参数都相同
语法	settb=expr expr的结果是一个有理数，可以包含下面描述的变量
	表达式中可用的变量
AVTB	设置默认时基值 (AVTB =默认时基)
intb	输入时基
sr	采样率，仅适用于asettb

下面的例子设置时基，第一个设置为AVTB，第二个设置为0.3，第三个设置为输入时基的1.5倍。

```
ffmpeg input.mpg -vf settb=AVTB output.mpg  
ffmpeg input.mpg -vf settb=0.3 output.mpg  
ffmpeg input.mpg -vf settb=1.5*intb output.mpg
```

## 测试编码功能

要为离散余弦亮度，色度，亮度和色度的频率和幅度等生成各种测试图案，我们可以使用MPlayer项目中的mptestsrc滤镜，该滤镜在表中描述：

### Video filter: mptestsrc

描述	生成与色度，亮度和其他视频属性相关的各种测试。如果没有参数使用，则会执行所有测试，直到用户停止该过程
语法	mptestsrc[=t=test_type[:d=duration[:r=rate]]]
	参数的描述
test, t	- 所选测试的名称，可用测试为dc_luma, dc_chroma, freq_luma, freq_chroma, amp_luma, amp_chroma, cbp, mv, ring1, ring2 - 默认值是"all"
duration, d	以秒为单位或HH: MM: SS格式的测试持续时间
r	帧率，默认值是25

下表说明了特定测试值的样本。

模式	句法	图片
DC亮度	mptest=t=dc_luma	
DC色度	mptest=t=dc_chroma	
亮度频率	mptest=t=freq_luma	
色度频率	mptest=t=freq_chroma	
亮度幅度	mptest=t=amp_luma	
色度幅度	mptest=t=amp_chroma	
编码块模式 (CBP)	mptest=t=cbp	
运动矢量 (MV)	mptest=t=mv	
test ring 1	mptest=t=ring1	
test ring 2	mptest=t=ring2	

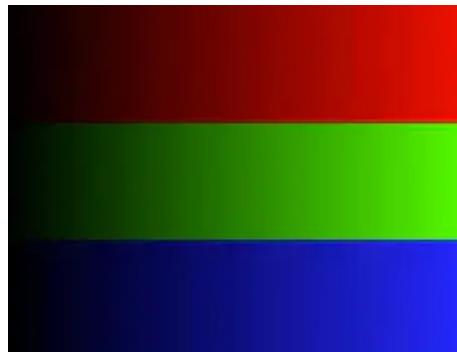
## 测试模式

检测各种错误并为视频测试提供源FFmpeg包含下面列出的3个特殊视频源。除颜色参数外，它们与“图像处理”章节“创建图像”一节中介绍的颜色来源共享相同的参数。

## RGB测试模式

要测试可用的RGB和BGR色彩空间是名为rgbttests src的视频源

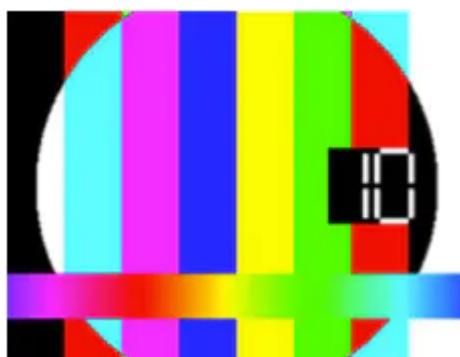
```
ffplay -f lavfi -i rgbttests src
```



## 滚动渐变和时间戳的颜色模式

要生成类似于TV模式的视频模式，可以使用testsrc视频源与命令：

```
ffplay -f lavfi -i testsrc
```



## SMPTE条形图案模式

可以使用以下命令创建来自电影和电视工程师协会（SMPTE）的彩条图案：

```
ffplay -f lavfi -i testsrc
```



## 简单的数据包转储或有效负载（十六进制）

为了更精确的调试，也可以使用`-hex`选项转储有效负载，通常使用`-report`选项将结果保存到当前目录中的文件。使用此选项，处理速度非常慢，报告文件更大。该命令的示例是：

```
ffmpeg -dump -hex -report -i input.mpg output.flv
```

## CPU使用时间和内存消耗

要在处理过程中显示使用的CPU时间和内存消耗，我们可以使用产生类似输出的`-benchmark`或`-benchmark_all`选项：

- `-benchmark`选项显示编码后的结果
- `-benchmark_all`以各种步骤显示编码过程中的结果

不支持最大内存消耗数据的计算机系统将显示0而不是数值。这两个选项都是全局选项，并在命令的开头输入，例如：

```
ffmpeg -benchmark -i input.avi output.webm
```

```
frame= 900 fps= 35 q=0.0 Lsize= 931kB time=00:00:30.00 bitrate= 254.2kbits/s
video:799kB audio:115kB subtitle:0 global headers:3kB muxing overhead 1.579193%
bench: utime=25.849s maxrss=36808kB
```

在控制台输出的末尾添加一条从工作台开始的行：`utime`表示在处理过程中CPU（计算机的中央处理单元）使用的时间。`benchmark_all`选项显示处理过程中的结果，完成后的屏幕显示在下图中。

```
ffmpeg -benchmark_all -i input.avi output.mpg
```

```
bench: 0 decode_audio 0.1
bench: 0 decode_video 0.0
bench: 0 decode_audio 0.1
bench: 468003 flush Video 0.0
bench: 0 flush Audio 0.1
frame= 900 fps= 21 q=0.0 Lsize= 931kB time=00:00:30.00 bitrate= 254.2kbits/s
video:799kB audio:115kB subtitle:0 global headers:3kB muxing overhead 1.579193%
```

## 26-词汇表

### 4cc 或者 fourcc

fourcc（也是FourCC - 四字符码）是媒体文件中使用的视频编解码器，压缩格式，颜色或像素格式的标识符。一个字符表示一个1字节（8位）的值，所以4cc在文件中总是有32位（4字节）。fourcc中的4个字符通常限制在ASCII表格中的人类可读字符内，因此很容易传达和传达媒体文件中的四个字符。AVI文件是最普遍的，或者是第一种被广泛使用的媒体文件格式，它使用4cc标识符来编码用于压缩文件内各种视频/音频流的编解码器。一些更为人熟知的fourccs包括DIVX, XVID, H264, DX50等。在FFmpeg中使用fourcc的示例：

- -vtag（或-tag: v）选项设置视频流的fourcc值
- -atag（或-tag: a）选项设置音频流的fourcc值
- -tag选项设置所有流的fourcc值

### 长宽比

长宽比是图像或视频帧的宽度和高度之间的比率。

长宽比的种类

类型	缩写	描述
Display Aspect Ratio	DAR	显示的图像和视频的纵横比 替代名称是图像纵横比 (IAR) 和图片纵横比
Storage Aspect Ratio	SAR	图像或视频帧的长宽比取决于视频源
Pixel Aspect Ratio	PAR	像素宽度与其高度之比, 在大多数LCD中它是1: 1, 但是一些模拟设备使用矩形像素
Original Aspect Ratio	OAR	从家庭影院标准术语来说, 表示视频最初创建的方面, 许多电影具有广泛的方面
Modified Aspect Ratio	MAR	家庭影院标准术语, 表示调整OAR以符合可用输出屏幕长宽比的方面, 通常为4: 3或16: 9

DAR, SAR和PAR之间的关系可以用等式表示。

$$\text{DAR} = \text{PAR} \times \text{SAR}$$

在ffmpeg中有一个-aspect选项可用于指定输出视频:

```
-aspect[:stream_specifier]
```

托马斯·阿尔瓦·爱迪生和威廉·迪克森于1892年创建了一种35mm宽的薄膜通用标准, 其中穿孔确定了24.89x18.67毫米的框架尺寸, 大约为4: 3或1.33: 1的比例。从那时起, 许多宽高比被创建, 其中一些将在下一个表中描述。

<b>1.33:1 或者 4:3</b>	原来的电影宽高比今天广泛用于液晶显示器, 电视屏幕, 摄像机等, 它也是 <b>MPEG-2</b> 压缩的标准
1.618:1	黄金比例, FFmpeg将其作为内置恒定PHI, 其中 $\text{PHI} = (1 + \sqrt{5}) / 2$
1.77:1或 16:9 (也可以 1.78:1)	用于高清晰度电视, 液晶显示器, 摄像机等的宽屏标准。它是MPEG-2标准中3种宽高比中的一种。比例16: 9可以从4: 3导出, 其中 $4^2 = 16$ 和 $3^2 = 9$
2.2:1 或 11:5	50年代开发的70毫米胶片标准, 它也是MPEG-2标准的一部分, 其值为2.21: 1
2.37:1	2010年推出的所谓“21: 9电影放映”新标准。该比率从4: 3推导出, 其中 $4:3:3 = 64:27 = 2.37$
2.39:1	35毫米胶片标准是变形的, 1970年用作‘范围’或Panavision格式。蓝光光盘将此比例用作2.4: 1, 并以1920x800的分辨率录制电影

## CIE

国际照明委员会（法国国际照明委员会 - 国际照明委员会）是关于照明, 照明, 色彩空间等的机构和权威机构, 更多地参见色彩校正章节。

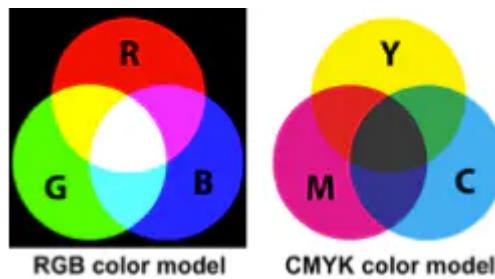
## 颜色深度, 位深度, 每像素位数

颜色深度, 位深度和每像素位数是描述使用多少位来指定图像中每个像素组件的颜色属性的术语。表中使用最多颜色深度:

每像素的位数	颜色数量	ffmpeg中可用的对应像素格式
1 (B&W)	$2^1 = 2$	单声道, 单声道
2 (CGA)	$2^2 = 4$	没有像素格式 (在早期的计算机中使用了2 bpp)
3	$2^3 = 8$	没有像素格式 (用于具有电视显示的早期家用电脑)
4 (EGA)	$2^4 = 16$	rgb4, rgb4_byte, bgr4, bgr4_byte
8 (VGA)	$2^8 = 256$	gray, pal8, bgr8, rgb8
12	$2^{12} = 4096$	yuv420p, yuv411p, yuvj420p, nv12, nv21, rgb444be, rgb444le
16 high color	$2^{16} = 65,536$	yuyv422, yuv422p, yuvj422p, yuv440p, yuvj440p, rgb565be等
24 true color	$2^{24} = 16,777,216$ (我怀疑这个数据的正确性。。。)	rgb24, bgr24, yuv444p, yuvj444p, gbrp, 0rgb, rgb0, yuva422p, 等
32 true+alpha	24-bit with alpha	argb, rgba, abgr, bgra, yuva444p, yuv422p16be, yuv422p16le
30 deep color	$2^{30} = 1,073,741,824$	yuv444p10be, yuv444p10le, gbrp10be, gbrp10le
36 deep color	$2^{36} = 68,719,476,736$	yuva444p9be, yuva444p9le, gbrp12be, gbrp12le, yuv444p12be等
48 deep color	超过280万亿	rgb48be, rgb48le, bgr48be, bgr48le, yuva422p16be, yuva422p16le

## 颜色模型和颜色空间

颜色空间是从颜色模型派生而来的。色彩模型描述了可用于模拟人类视觉的方法。当这种描述如何用颜色表示数字的数学模型辅以用于解释这些数字的精确定义时, 这样的一组颜色被认为是一种颜色空间。数字颜色表示通常使用3或4个组件来描述每种颜色。更精确的规格要求更大的数字, 详情请参见上一节。5种颜色模型被认为是主要的: CIE, CMYK, HSL / HSV, RGB和YUV以及视频技术通常使用RGB和YUV。计算机显示器使用像素来显示信息, 颜色空间以像素格式实现, 他们的列表在第二章中介绍。



## 色彩视觉

尽管人耳对从16到20,000赫兹频率的振动敏感，但人眼能够感知频率从大约405 THz (740纳米) 到 789 THz (380纳米) 的电磁波。没有颜色的暗光被称为圆锥体的特殊眼睛细胞感知，并且三色视觉由棒状细胞提供。3种不同类型的棒状细胞对3种基本颜色敏感：红色，绿色和蓝色：

- L (长波) rods - 对大约564到580纳米的波敏感 - 红色
  - M (中波) rods - 对约534至545纳米波长敏感 - 绿色
  - S (短波) rods - 对大约420至440纳米的波敏感 - 蓝色
- rods的峰值灵敏度为约498nm的波，因此绿色感觉比红色好，甚至比蓝色更好，更多细节在隔行视频一章中。

## DCT - 离散余弦变换

离散余弦变换用于利用有损编解码器对音频和视频数据进行压缩，从而减少多次存储的数据量。它使用一个以不同频率振荡的余弦函数来将复信号转换为复信号。DCT是离散傅立叶变换（DFT）的一种特殊类型，其中输入和输出采样是实数（DFT与复数相匹配）。使用FFmpeg的DCT示例：

- 选项-dct [: streamSpecifier]为任何编解码器设置DCT算法
- 选项-idct [: streamSpecifier]为任何编解码器设置DCT实现
- 值为ildct（使用隔行DCT）的任何编解码器的标志选项
- 值为任何编解码器的-debug选项中的dct\_coeff
- 对于任何编解码器，在-cmp, -subcmp, -mbcmp, -ildctcmp, -skipcmp和-precmp选项中的值dct（绝对DCT转换差的总和）和值dctmax
- 为任何编解码器选项-skip\_idct [: streamSpecifier]选项

## 解码器

在转码过程中，解码器处理来自分路器的编码数据包，并为下一个处理生成未压缩帧。从FFmpeg文档定义：

“解码器是FFmpeg中的配置元素，可以解码多媒体流。”解码器列表可以使用以下命令显示：

```
ffmpeg -decoders
```

## 分路器 (demux)

在转码过程中，解复用器（也是解复用器和解复用器）读取输入文件并生成编码数据包，发送到解码器进行解码。

从FFmpeg文档定义：

“分解器是FFmpeg中的配置元素，它允许从特定类型的文件读取多媒体流。”

分解器在可用格式中列出，详细信息在第二章中。有关特定分路器的信息可以使用以下命令显示：

```
ffmpeg -h demuxer=demuxer_name
```

## 编码器

在转码期间，编码器处理未压缩的帧，并根据选定的编解码器将它们编码为发送到解复用器的数据包，通常采用某种压缩，有损或无损压缩。

FFmpeg文档中编码器的定义：

“编码器是FFmpeg中的配置元素，它允许编码多媒体流。”

编码器列表位于第二章中，可以使用以下命令显示：

```
ffmpeg -encoders
```

要显示关于特定编码器的详细信息，我们可以使用以下命令：

```
ffmpeg -h encoder=encoder_name
```

## FFmpeg配置

原生FFmpeg源代码的可能性通过包含来自其他开源项目的附加软件库代码得到了改进。在使用名为configure的配置文件的--enable-library选项进行编译之前，会包含这些库。在每个控制台使用ffmpeg的情况下，ffplay和ffprobe工具都会显示实际的FFmpeg配置并启用其他库，如下一个控制台输出中所示：

```
configuration: --enable-gpl --enable-version3 --disable-pthreads --enable-runtime-cpudetect --enable-avisynth --enable-bzlib --enable-frei0r --enable-libass --enable-libcelt --enable-libopencore-amrnb --enable-libopencore-amrwb --enable-libfreetype --enable-libgsm --enable-libmp3lame --enable-libnut --enable-libopenjpeg --enable-librtmp --enable-libschroedinger --enable-libspeex --enable-libtheora --enable-libutvideo --enable-libvo-aacenc --enable-libvo-amrwbenc --enable-libvorbis --enable-libvpx --enable-libx264 --enable-libxavs --enable-libxvid --enable-zlib
libavutil      51. 73.101 / 51. 73.101
libavcodec     54. 56.100 / 54. 56.100
libavformat    54. 27.101 / 54. 27.101
libavdevice    54.  2.100 / 54.  2.100
libavfilter     3. 16.104 / 3. 16.104
libswscale      2. 1.101 / 2. 1.101
libswresample   0. 15.100 / 0. 15.100
libpostproc    52.  0.100 / 52.  0.100
```

显示的还有FFmpeg本地库的版本：libavutil, libavcodec等

## JPEG

JPEG通常有2种含义：

- 联合图像专家组是移动图像专家组（MPEG）的前身。JPEG小组成立于1986年，并创建了JPEG, JPEG 2000和JPEG XR标准。

- 有压缩比为10: 1的图像和视频帧的有损压缩方法没有明显的质量下降，它被用于许多图像格式，包括在网页和数码相机中广泛使用的JPG图像。

由于JPEG格式的有损压缩不适合多次编辑和某些技术任务。创建JPEG图像的段以各种标记（SOI, SOF0, SOF2, DHT等）分隔。在编码过程中，内容被转换为YCbCr（FFmpeg像素格式的YUV）色彩空间，其中Y表示亮度通道，Cb和Cr两个色度通道。色度通道可选择降采样，每个通道被分成8×8块，通过类型II的归一化2维离散余弦变换（DCT）转换为频域。

在FFmpeg中使用JPEG和派生标准的示例：

- mjpeg2jpeg比特流筛选器转换MJPEG / AVI1数据包以完成JPEG / JFIF数据包
- mjpegadump比特流过滤器用于转换为MJPEG文件格式
- 解码器和基于JPEG的编码器：jpeg2000, jpegls, ljpeg, mjpeg, mjpegb（仅解码器），nuv（RTJPEG），sp5x（Sunplus JPEG），adpcm\_ima\_smjpeg（音频，仅解码器）
- 文件格式：ingenient（原始MJPEG，仅解码），mjpeg, mpjpeg（MIME多部分，仅编码），smjpeg（Loki SDL MJPEG）

## Macroblock

图像或视频帧到宏块的划分是使用离散余弦变换（DCT）进行编码的一部分。在FFmpeg中，各种AVC编解码器上下文选项，H.263编码器选项，ProRes编码器选项，libx264选项，libxavs选项等都使用宏块。

Coding of Macroblocks for DCT (JPEG images and video frames)							
ADDR	TYPE	QUANT	VECTOR	CBP	b0	b1	...b5
<b>ADDR - address of a block in the image (frame)</b>							
<b>TYPE - macroblock type:</b> intra, inter (predicted), inter (bidirect.)							
<b>QUANT - quantization parameter</b>							
<b>VECTOR - motion vector (2D vector for inter prediction)</b>							
<b>CBP - Coded Block Pattern (bit mask for blocks coefficients)</b>							
<b>b0...b3 - 4 Y blocks for luminance</b>							
<b>b4 - 1 block for Cr</b>							
<b>b5 - 1 block for Cb</b>							

## 运动矢量

运动矢量根据另一图片（参考图片）中宏块的位置来表示图片中的宏块。运动矢量是视频压缩过程中运动估计的主要元素。H.264标准的官方定义是：

用于帧间预测的二维矢量，其提供从解码图片中的坐标到参考图片中的坐标的偏移。

运动矢量在FFmpeg中的使用示例：

- 值为mv4（在宏块（mp4）中使用四个运动矢量）in -flags选项用于任何编解码器
- 任何一个编解码器的-ec（错误隐藏策略）选项中的guess\_mvs（迭代运动矢量（MV）搜索）
- 值为mv（运动矢量）的任何编解码器的调试选项
- -任何编解码器的-vismv [: stream\_specifier]（可视化运动矢量）选项
- 任何编解码器的-me\_range [: stream\_specifier]（限制运动矢量范围）选项
- 任何编解码器的-bidir\_refine选项（细化双向宏块中使用的2个运动向量）

- 去抖滤波器使用x, y, w和h参数指定矩形区域来搜索运动矢量

## MPEG

MPEG（发音“em-peg”）代表Moving Pictures Experts Group，它是ISO / IEC于1988年建立的多媒体专家团队，负责制定音频和视频压缩和传输标准。MPEG标准包括：

- MPEG-1
- MPEG-2
- MPEG-4
- MPEG-7
- MPEG-21

MPEG被分成称为工作组的团队（暂且先这么翻译，原文是：MPEG is divided to the teams called working groups.）。

## MPEG-1

该标准的官方定义是：“以高达约1.5 Mbps的速度为数字存储媒体编码运动图像和相关音频。”它在1993年进行了标准化，主要目标是编码视频和声音以存储在光盘上。MPEG-1用于音频CD, VCD和可选的SVCD和低质量DVD。在MPEG-2之前，它也被用于数字卫星和有线电视网络。该标准的一部分是流行的音频格式MP3，MP3是MPEG-1 Audio Layer III的缩写。在ffmpeg中，您可以使用-f mpeg1video选项来选择此格式，例如：

```
ffmpeg -i input.avi -f mpeg1video output.mpg
```

## MPEG-2

MPEG-2标准的官方定义是：运动图像和相关音频信息的通用编码（ISO / IEC 13818）。这个广泛的标准于1995年发布，包含广播电视的传输，视频和音频规范。其压缩方案用于：

- 地面数字电视，即ATSC, DVB和ISDB
- 卫星数字电视，例如Dish Network
- 数字有线电视
- 超级视频CD, DVD视频和偶尔在蓝光光盘上播放

在ffmpeg中，您可以使用-f mpeg2video选项来选择此格式，例如：

```
ffmpeg -i input.avi -f mpeg2video output.mpg
```

## MPEG-4

ISO / IEC对标准的正式定义是：“视听对象的编码”。它在1998年得到了标准化，其主要特点是：

- 新的编码算法ACE（高级编码效率），与MPEG-2相比，其数据存储/带宽减少了约11倍
- 解码器是渲染处理器，压缩比特流描述3D形状和表面纹理
- 支持MPEG-J（Java API），是定制交互式多媒体应用开发的程序的解决方案
- 支持使用专有技术和保护DRM（数字版权管理）等内容的IPMP（知识产权管理和保护）

对于带有MP4扩展名的输出文件，ffmpeg会自动选择h264编码器和yuv420p像素格式。

## 复用器（mux）

在转码过程中，复用器（也是多路复用器或多路复用器）处理编码的数据包，并生成指定格式的文件。

FFmpeg文档中复用器的定义：

“**Muxers**是FFmpeg中的配置元素，允许将多媒体流写入特定类型的文件。”

muxers的列表在可用格式的第二章中。要显示有关特定复用器的详细信息，我们可以使用以下命令：

```
ffmpeg -h muxer=muxer_name
```

## 像素

像素或像素来源于图像元素，并表示数字图像或视频帧的最小可控元素。像素的形状通常是正方形，但一些帧分辨率使用矩形像素。像素宽度和高度之间的比例是像素宽高比，通常缩写为PAR。ffmpeg包含包含1个、3个或4个组件的像素格式，并使用以下命令显示：

```
ffmpeg -pix_fmts
```

## 协议

计算机术语中的术语协议通常意味着一组用于数据接收和传输的通信规则。FFmpeg文档中协议的定义：

“**协议**是FFmpeg中的配置元素，允许访问需要使用特定协议的资源。”

可用协议列表在第二章中，可以使用以下命令显示：

```
ffmpeg -protocols
```

协议的例子是http（超文本传输协议），rtmp（实时消息传输协议），tcp（传输控制协议），udp（用户数据报协议）等。

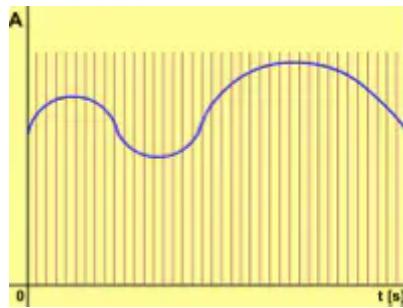
## 量化

数字信号的量化涉及将一系列值减小到代表性的单个值。媒体处理包括音频和视频量化，音频中量化的声音幅度和视频编码涉及颜色和频率量化。音频和视频量化都可以利用DCT变换。与量化相关的ffmpeg选项的例子：

- 值为任何编解码器的标记选项（归一化自适应量化）
- 任何编解码器的-debug选项中的qp（量化参数）和vis\_qp（可视化QP）值
- -trellis [: stream\_specifier]（速率 - 失真最优量化）选项

## 采样和采样率

采样连续（模拟）信号意味着将其降低到离散信号，例如声波（图中所示）被转换为一系列采样。这些样本的数值以数字表示，这种数字形式可以保存到计算机文件中。采样率（或频率）决定了每秒采样的数量，典型的音频采样率是8000 Hz和11025 Hz的倍数。人耳灵敏度范围从16 Hz到20 kHz，由于采样定理，至少需要40 kHz频率才能表示所有可听声音，选择44,100 Hz作为CD音频的标准。



## 视频

术语视频是在电视被发明时创建的，并且表示一种以电子形式处理运动图像的技术（与电影 - 光化学形式相比），可以包括：

- 捕获
- 记录
- 压缩
- 编码
- 解码
- 传送（广播）等。

主要的视频功能是：

- 帧率
- 长宽比
- 存储类型（模拟或数字）
- 色彩空间
- 交错式或渐进式
- 质量（由用户感知）
- 每像素位数（色彩深度）
- 编解码器（仅数字视频）
- 3维（3D）

视频表示以帧速率指定的快速序列投影以产生连续运动的假象的静止图像，当人眼看到连续场景时的最小帧速率大约为每秒15帧图像（帧）。视频来源于电影，这是一种处理运动图片的机械技术，最初是为电影和电视开发的，它使用隔行扫描来消除闪烁。FFmpeg适用于第二章中列出的许多视频格式。一些格式能够存储多个不同的运动图像序列，并且这些序列被称为视频流；它们的编号是从零开始的，第一个流编号为0，第二个编号为1，等等

## 视频过滤器

FFmpeg中的过滤器是通过libavfilter库实现的。为获得最佳性能，它们通常用于过滤链（逗号分隔过滤器的说明）和过滤器图形（分号分隔的过滤链）。使用filtergraphs可以使用带标签的链接替换后面的filterchains中的输入，默认情况下会创建一个特殊的[in]链接标签，并表示使用-i选项输入的输入。将过滤器与过滤链和过滤器图组合在一起比重复处理更加优选，所述重复处理涉及由压缩算法引起的轻微变化。过滤器列表位于第2章中，由以下内容显示：ffmpeg -filters视频过滤器可根据多个标准进行划分，总体分类见下表：

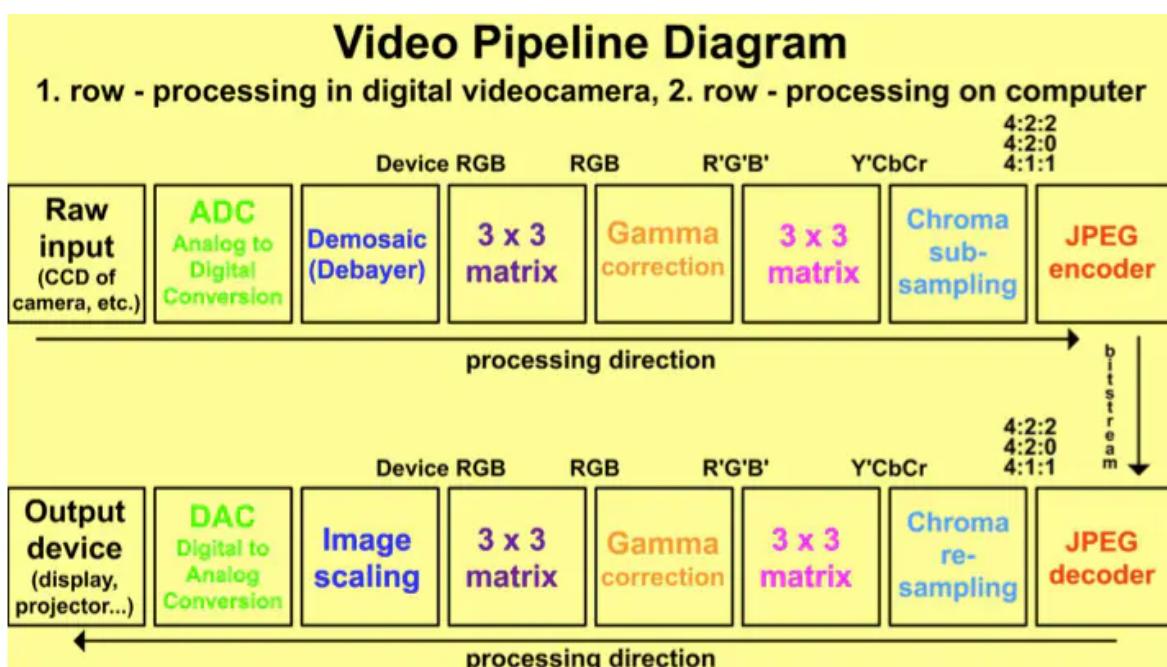
视频过滤器（一般分类）

过滤器类型	描述	示例
预过滤器	在编码之前使用	对比度调整 去闪烁 逐行扫描 降噪 缩放（缩减采样，上采样）
Intrafilters	用于编码（通常是视频编解码器的一部分）	解封装
Postfilters	解码后使用	解封装，逐行扫描，dering

注意：一些像隔行扫描一样的滤镜被列入2类，因为输入视频流的修改取决于软件实现。

## 视频管道

视频流水线描述了从原始视频输入到显示设备上最终输出的视频帧处理过程，图中显示了一个简化的视频流水线。ffmpeg包含2个特殊输入设备dv1394和iec61883，可直接从数码摄像机使用的FireWire端口录制。



## 关于作者

Frantisek Korbel是Zend认证工程师，他的工作包括编程，视频编辑和网页设计。2004年，他创建了一个使用Macromedia Flash的急救基础免费软件，从那时起他经常使用动画和视频。他在2009年使用Adobe AIR开发了化学元素学习周期表。他的大部分活动都专注于志愿工作，主要用于发展中国家的教育和社区项目（非洲 - nkolfoulou.org, oyoko.org），印度（kidedu.org）等。他参加了联合国志愿工

作的各种项目，由联合国开发计划署协调的WaterWiki.net网站设计。为了这本书，他创建了一个支持网站ffmpeg.tv。

## 链接地址

---

书主页: <http://ffmpeg.tv>

Facebook: <http://ffmpeg.tv/facebook>

witter: <http://twitter.com/FFmpeg>

YouTube: <http://youtube.com/FFmpegTv>



是不是感觉作者萌萌哒

转载自简书-[张芳涛](#)

原地址: <https://www.jianshu.com/p/835aee0a5a0a>

由loongmonkey整理并排版。如有错漏，欢迎给我写邮件[loongmonkey@qq.com](mailto:loongmonkey@qq.com)，

或在我的开源项目<https://github.com/loongmonkey/Multimedia-development>上提意见

著作版权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。