# FFmpeg Filters Documentation

## Table of Contents

# 1 Description

This document describes filters, sources, and sinks provided by the libavfilter library.

# 2 Filtering Introduction

Filtering in FFmpeg is enabled through the libavfilter library.

In libavfilter, a filter can have multiple inputs and multiple outputs. To illustrate the sorts of things that are possible, we consider the following filtergraph.

```
                [main]
input --> split ---------------------> overlay --> output
            |                             ^
            |[tmp]                  [flip]|
            +-----> crop --> vflip -------+
```

This filtergraph splits the input stream in two streams, then sends one stream through the crop filter and the vflip filter, before merging it back with the other stream by overlaying it on top. You can use the following command to achieve this:

```
ffmpeg -i INPUT -vf "split [main][tmp]; [tmp] crop=iw:ih/2:0:0, vflip [flip]; [main][flip] overlay=0:H/2" OUTPUT
```

The result will be that the top half of the video is mirrored onto the bottom half of the output video.

Filters in the same linear chain are separated by commas, and distinct linear chains of filters are separated by semicolons. In our example, *crop,vflip* are in one linear chain, *split* and *overlay* are separately in another. The points where the linear chains join are labelled by names enclosed in square brackets. In the example, the split filter generates two outputs that are associated to the labels *[main]* and *[tmp]*.

The stream sent to the second output of *split*, labelled as *[tmp]*, is processed through the *crop* filter, which crops away the lower half part of the video, and then vertically flipped. The *overlay* filter takes in input the first unchanged output of the split filter (which was labelled as *[main]*), and overlay on its lower half the output generated by the *crop,vflip* filterchain.

Some filters take in input a list of parameters: they are specified after the filter name and an equal sign, and are separated from each other by a colon.

There exist so-called *source filters* that do not have an audio/video input, and *sink filters* that will not have audio/video output.

# 3 graph2dot

The `graph2dot` program included in the FFmpeg `tools` directory can be used to parse a filtergraph description and issue a corresponding textual representation in the dot language.

Invoke the command:

```
graph2dot -h
```

to see how to use `graph2dot`.

You can then pass the dot description to the `dot` program (from the graphviz suite of programs) and obtain a graphical representation of the filtergraph.

For example the sequence of commands:

```
echo GRAPH_DESCRIPTION | \
tools/graph2dot -o graph.tmp && \
dot -Tpng graph.tmp -o graph.png && \
display graph.png
```

can be used to create and display an image representing the graph described by the *GRAPH_DESCRIPTION* string. Note that this string must be a complete self-contained graph, with its inputs and outputs explicitly defined. For example if your command line is of the form:

```
ffmpeg -i infile -vf scale=640:360 outfile
```

your *GRAPH_DESCRIPTION* string will need to be of the form:

```
nullsrc,scale=640:360,nullsink
```

you may also need to set the *nullsrc* parameters and add a *format* filter in order to simulate a specific input file.

# 4 Filtergraph description

A filtergraph is a directed graph of connected filters. It can contain cycles, and there can be multiple links between a pair of filters. Each link has one input pad on one side connecting it to one filter from which it takes its input, and one output pad on the other side connecting it to one filter accepting its output.

Each filter in a filtergraph is an instance of a filter class registered in the application, which defines the features and the number of input and output pads of the filter.

A filter with no input pads is called a "source", and a filter with no output pads is called a "sink".

## 4.1 Filtergraph syntax

A filtergraph has a textual representation, which is recognized by the `-filter`/`-vf` and `-filter_complex` options in `ffmpeg` and `-vf` in `ffplay`, and by the `avfilter_graph_parse()`/`avfilter_graph_parse2()` functions defined in `libavfilter/avfilter.h`.

A filterchain consists of a sequence of connected filters, each one connected to the previous one in the sequence. A filterchain is represented by a list of ","-separated filter descriptions.

A filtergraph consists of a sequence of filterchains. A sequence of filterchains is represented by a list of ";"-separated filterchain descriptions.

A filter is represented by a string of the form:
[*in_link_1*]...[*in_link_N*]*filter_name*=*arguments*[*out_link_1*]...[*out_link_M*]

*filter_name* is the name of the filter class of which the described filter is an instance of, and has to be the name of one of the filter classes registered in the program. The name of the filter class is optionally followed by a string "=*arguments*".

*arguments* is a string which contains the parameters used to initialize the filter instance. It may have one of two forms:

- A ':'-separated list of *key=value* pairs.
- A ':'-separated list of *value*. In this case, the keys are assumed to be the option names in the order they are declared. E.g. the `fade` filter declares three options in this order – `type`, `start_frame` and `nb_frames`. Then the parameter list *in:0:30* means that the value *in* is assigned to the option `type`, *0* to `start_frame` and *30* to `nb_frames`.
- A ':'-separated list of mixed direct *value* and long *key=value* pairs. The direct *value* must precede the *key=value* pairs, and follow the same constraints order of the previous point. The following *key=value* pairs can be set in any preferred order.

If the option value itself is a list of items (e.g. the `format` filter takes a list of pixel formats), the items in the list are usually separated by '|'.

The list of arguments can be quoted using the character "'" as initial and ending mark, and the character '\' for escaping the characters within the quoted text; otherwise the argument string is considered terminated when the next special character (belonging to the set "[]=;,") is encountered.

The name and arguments of the filter are optionally preceded and followed by a list of link labels. A link label allows one to name a link and associate it to a filter output or input pad. The preceding labels *in_link_1* ... *in_link_N*, are associated to the filter input pads, the following labels *out_link_1* ... *out_link_M*, are associated to the output pads.

When two link labels with the same name are found in the filtergraph, a link between the corresponding input and output pad is created.

If an output pad is not labelled, it is linked by default to the first unlabelled input pad of the next filter in the filterchain. For example in the filterchain

```
nullsrc, split[L1], [L2]overlay, nullsink
```

the split filter instance has two output pads, and the overlay filter instance two input pads. The first output pad of split is labelled "L1", the first input pad of overlay is labelled "L2", and the second output pad of split is linked to the second input pad of overlay, which are both unlabelled.

In a complete filterchain all the unlabelled filter input and output pads must be connected. A filtergraph is considered valid if all the filter input and output pads of all the filterchains are connected.

Libavfilter will automatically insert scale filters where format conversion is required. It is possible to specify swscale flags for those automatically inserted scalers by prepending `sws_flags=flags;` to the filtergraph description.

Here is a BNF description of the filtergraph syntax:

```
NAME             ::= sequence of alphanumeric characters and '_'
LINKLABEL        ::= "[" NAME "]"
LINKLABELS       ::= LINKLABEL [LINKLABELS]
FILTER_ARGUMENTS ::= sequence of chars (possibly quoted)
FILTER           ::= [LINKLABELS] NAME ["=" FILTER_ARGUMENTS] [LINKLABELS]
FILTERCHAIN      ::= FILTER [,FILTERCHAIN]
FILTERGRAPH      ::= [sws_flags=flags;] FILTERCHAIN [;FILTERGRAPH]
```

## 4.2 Notes on filtergraph escaping

Filtergraph description composition entails several levels of escaping. See (ffmpeg-utils)the "Quoting and escaping" section in the ffmpeg-utils(1) manual for more information about the employed escaping procedure.

A first level escaping affects the content of each filter option value, which may contain the special character `:` used to separate values, or one of the escaping characters `\'`.

A second level escaping affects the whole filter description, which may contain the escaping characters `\'` or the special characters `[],;` used by the filtergraph description.

Finally, when you specify a filtergraph on a shell commandline, you need to perform a third level escaping for the shell special characters contained within it.

For example, consider the following string to be embedded in the drawtext filter description `text` value:

```
this is a 'string': may contain one, or more, special characters
```

This string contains the `'` special escaping character, and the `:` special character, so it needs to be escaped in this way:

```
text=this is a \'string\'\: may contain one, or more, special characters
```

A second level of escaping is required when embedding the filter description in a filtergraph description, in order to escape all the filtergraph special characters. Thus the example above becomes:

```
drawtext=text=this is a \\\'string\\\'\\: may contain one\, or more\, special characters
```

(note that in addition to the `\'` escaping special characters, also `,` needs to be escaped).

Finally an additional level of escaping is needed when writing the filtergraph description in a shell command, which depends on the escaping rules of the adopted shell. For example, assuming that `\` is special and needs to be escaped with another `\`, the previous string will finally result in:

```
-vf "drawtext=text=this is a \\\\\\'string\\\\\\'\\\\: may contain one\\, or more\\, special characters"
```

# 5 Timeline editing

Some filters support a generic `enable` option. For the filters supporting timeline editing, this option can be set to an expression which is evaluated before sending a frame to the filter. If the evaluation is non-zero, the filter will be enabled, otherwise the frame will be sent unchanged to the next filter in the filtergraph.

The expression accepts the following values:

‘`t`’

timestamp expressed in seconds, NAN if the input timestamp is unknown

‘`n`’

sequential number of the input frame, starting from 0

‘`pos`’

the position in the file of the input frame, NAN if unknown

‘`w`’
‘`h`’

width and height of the input frame if video

Additionally, these filters support an `enable` command that can be used to re-define the expression.

Like any other filtering option, the `enable` option follows the same rules.

For example, to enable a blur filter (smartblur) from 10 seconds to 3 minutes, and a curves filter starting at 3 seconds:

```
smartblur = enable='between(t,10,3*60)',
curves    = enable='gte(t,3)' : preset=cross_process
```

# 6 Audio Filters

When you configure your FFmpeg build, you can disable any of the existing filters using `--disable-filters`. The configure output will show the audio filters included in your build.

Below is a description of the currently available audio filters.

## 6.1 adelay

Delay one or more audio channels.

Samples in delayed channel are filled with silence.

The filter accepts the following option:

`delays`

Set list of delays in milliseconds for each channel separated by '|'. At least one delay greater than 0 should be provided. Unused delays will be silently ignored. If number of given delays is smaller than number of channels all remaining channels will not be delayed.

### 6.1.1 Examples

- Delay first channel by 1.5 seconds, the third channel by 0.5 seconds and leave the second channel (and any other channels that may be present) unchanged.

  ```
  adelay=1500|0|500
  ```

## 6.2 aecho

Apply echoing to the input audio.

Echoes are reflected sound and can occur naturally amongst mountains (and sometimes large buildings) when talking or shouting; digital echo effects emulate this behaviour and are often used to help fill out the sound of a single instrument or vocal. The time difference between the original signal and the reflection is the `delay`, and the loudness of the reflected signal is the `decay`. Multiple echoes can have different delays and decays.

A description of the accepted parameters follows.

`in_gain`

Set input gain of reflected signal. Default is `0.6`.

`out_gain`

Set output gain of reflected signal. Default is `0.3`.

`delays`

Set list of time intervals in milliseconds between original signal and reflections separated by ’|’. Allowed range for each `delay` is `(0 - 90000.0]`. Default is `1000`.

`decays`

Set list of loudnesses of reflected signals separated by ’|’. Allowed range for each `decay` is `(0 - 1.0]`. Default is `0.5`.

### 6.2.1 Examples

- Make it sound as if there are twice as many instruments as are actually playing:

  ```
  aecho=0.8:0.88:60:0.4
  ```

- If delay is very short, then it sound like a (metallic) robot playing music:

  ```
  aecho=0.8:0.88:6:0.4
  ```

- A longer delay will sound like an open air concert in the mountains:

  ```
  aecho=0.8:0.9:1000:0.3
  ```

- Same as above but with one more mountain:

  ```
  aecho=0.8:0.9:1000|1800:0.3|0.25
  ```

## 6.3 aeval

Modify an audio signal according to the specified expressions.

This filter accepts one or more expressions (one for each channel), which are evaluated and used to modify a corresponding audio signal.

It accepts the following parameters:

`exprs`

Set the ’|’-separated expressions list for each separate channel. If the number of input channels is greater than the number of expressions, the last specified expression is used for the remaining output channels.

`channel_layout, c`

Set output channel layout. If not specified, the channel layout is specified by the number of expressions. If set to ‘`same`’, it will use by default the same input channel layout.

Each expression in *exprs* can contain the following constants and functions:

`ch`

channel number of the current expression

`n`

number of the evaluated sample, starting from 0

`s`

sample rate

`t`

time of the evaluated sample expressed in seconds

`nb_in_channels`
`nb_out_channels`

input and output number of channels

`val(CH)`

the value of input channel with number *CH*

Note: this filter is slow. For faster processing you should use a dedicated filter.

### 6.3.1 Examples

- Half volume:

```
aeval=val(ch)/2:c=same
```

- Invert phase of the second channel:

```
aeval=val(0)|-val(1)
```

## 6.4 afade

Apply fade-in/out effect to input audio.

A description of the accepted parameters follows.

`type, t`

Specify the effect type, can be either `in` for fade-in, or `out` for a fade-out effect. Default is `in`.

`start_sample, ss`

Specify the number of the start sample for starting to apply the fade effect. Default is 0.

`nb_samples, ns`

Specify the number of samples for which the fade effect has to last. At the end of the fade-in effect the output audio will have the same volume as the input audio, at the end of the fade-out transition the output audio will be silence. Default is 44100.

`start_time, st`

Specify the start time of the fade effect. Default is 0. The value must be specified as a time duration; see (ffmpeg-utils)the Time duration section in the ffmpeg-utils(1) manual for the accepted syntax. If set this option is used instead of *start_sample*.

`duration, d`

Specify the duration of the fade effect. See (ffmpeg-utils)the Time duration section in the ffmpeg-utils(1) manual for the accepted syntax. At the end of the fade-in effect the output audio will have the same volume as the input audio, at the end of the fade-out transition the output audio will be silence. By default the duration is determined by *nb_samples*. If set this option is used instead of *nb_samples*.

`curve`

Set curve for fade transition.

It accepts the following values:

`tri`

select triangular, linear slope (default)

`qsin`

select quarter of sine wave

`hsin`

select half of sine wave

`esin`

select exponential sine wave

`log`

select logarithmic

`par`

select inverted parabola

`qua`

select quadratic

`cub`

select cubic

`squ`

select square root

`cbr`

select cubic root

### 6.4.1 Examples

- Fade in first 15 seconds of audio:

  ```
  afade=t=in:ss=0:d=15
  ```

- Fade out last 25 seconds of a 900 seconds audio:

  ```
  afade=t=out:st=875:d=25
  ```

## 6.5 aformat

Set output format constraints for the input audio. The framework will negotiate the most appropriate format to minimize conversions.

It accepts the following parameters:

`sample_fmts`

A ’|’-separated list of requested sample formats.

`sample_rates`

A ’|’-separated list of requested sample rates.

`channel_layouts`

A ’|’-separated list of requested channel layouts.

See (ffmpeg-utils)the Channel Layout section in the ffmpeg-utils(1) manual for the required syntax.

If a parameter is omitted, all values are allowed.

Force the output to either unsigned 8-bit or signed 16-bit stereo

```
aformat=sample_fmts=u8|s16:channel_layouts=stereo
```

## 6.6 allpass

Apply a two-pole all-pass filter with central frequency (in Hz) *frequency*, and filter-width *width*. An all-pass filter changes the audio’s frequency to phase relationship without changing its frequency to amplitude relationship.

The filter accepts the following options:

`frequency, f`

Set frequency in Hz.

`width_type`

Set method to specify band-width of filter.

`h`

Hz

`q`

Q-Factor

`o`

octave

`s`

slope

`width, w`

Specify the band-width of a filter in width_type units.

## 6.7 amerge

Merge two or more audio streams into a single multi-channel stream.

The filter accepts the following options:

`inputs`

Set the number of inputs. Default is 2.

If the channel layouts of the inputs are disjoint, and therefore compatible, the channel layout of the output will be set accordingly and the channels will be reordered as necessary. If the channel layouts of the inputs are not disjoint, the output will have all the channels of the first input then all the channels of the second input, in that order, and the channel layout of the output will be the default value corresponding to the total number of channels.

For example, if the first input is in 2.1 (FL+FR+LF) and the second input is FC+BL+BR, then the output will be in 5.1, with the channels in the following order: a1, a2, b1, a3, b2, b3 (a1 is the first channel of the first input, b1 is the first channel of the second input).

On the other hand, if both input are in stereo, the output channels will be in the default order: a1, a2, b1, b2, and the channel layout will be arbitrarily set to 4.0, which may or may not be the expected value.

All inputs must have the same sample rate, and format.

If inputs do not have the same duration, the output will stop with the shortest.

### 6.7.1 Examples

- Merge two mono files into a stereo stream:

```
amovie=left.wav [l] ; amovie=right.mp3 [r] ; [l] [r] amerge
```

- Multiple merges assuming 1 video stream and 6 audio streams in `input.mkv`:

```
ffmpeg -i input.mkv -filter_complex "[0:1][0:2][0:3][0:4][0:5][0:6] amerge=inputs=6" -c:a pcm_s16le output.mkv
```

## 6.8 amix

Mixes multiple audio inputs into a single output.

Note that this filter only supports float samples (the *amerge* and *pan* audio filters support many formats). If the *amix* input has integer samples then aresample will be automatically inserted to perform the conversion to float samples.

For example

```
ffmpeg -i INPUT1 -i INPUT2 -i INPUT3 -filter_complex amix=inputs=3:duration=first:dropout_transition=3 OUTPUT
```

will mix 3 input audio streams to a single output with the same duration as the first input and a dropout transition time of 3 seconds.

It accepts the following parameters:

`inputs`

The number of inputs. If unspecified, it defaults to 2.

`duration`

How to determine the end-of-stream.

`longest`

The duration of the longest input. (default)

`shortest`

The duration of the shortest input.

`first`

The duration of the first input.

`dropout_transition`

The transition time, in seconds, for volume renormalization when an input stream ends. The default value is 2 seconds.

## 6.9 anull

Pass the audio source unchanged to the output.

## 6.10 apad

Pad the end of an audio stream with silence.

This can be used together with `ffmpeg -shortest` to extend audio streams to the same length as the video stream.

A description of the accepted options follows.

`packet_size`

Set silence packet size. Default value is 4096.

`pad_len`

Set the number of samples of silence to add to the end. After the value is reached, the stream is terminated. This option is mutually exclusive with `whole_len`.

`whole_len`

Set the minimum total number of samples in the output audio stream. If the value is longer than the input audio length, silence is added to the end, until the value is reached. This option is mutually exclusive with `pad_len`.

If neither the `pad_len` nor the `whole_len` option is set, the filter will add silence to the end of the input stream indefinitely.

### 6.10.1 Examples

- Add 1024 samples of silence to the end of the input:

  ```
  apad=pad_len=1024
  ```

- Make sure the audio output will contain at least 10000 samples, pad the input with silence if required:

  ```
  apad=whole_len=10000
  ```

- Use `ffmpeg` to pad the audio input with silence, so that the video stream will always result the shortest and will be converted until the end in the output file when using the `shortest` option:

  ```
  ffmpeg -i VIDEO -i AUDIO -filter_complex "[1:0]apad" -shortest OUTPUT
  ```

## 6.11 aphaser

Add a phasing effect to the input audio.

A phaser filter creates series of peaks and troughs in the frequency spectrum. The position of the peaks and troughs are modulated so that they vary over time, creating a sweeping effect.

A description of the accepted parameters follows.

`in_gain`

Set input gain. Default is 0.4.

`out_gain`

Set output gain. Default is 0.74

`delay`

Set delay in milliseconds. Default is 3.0.

`decay`

Set decay. Default is 0.4.

`speed`

Set modulation speed in Hz. Default is 0.5.

`type`

Set modulation type. Default is triangular.

It accepts the following values:

‘`triangular, t`’
‘`sinusoidal, s`’

## 6.12 aresample

Resample the input audio to the specified parameters, using the libswresample library. If none are specified then the filter will automatically convert between its input and output.

This filter is also able to stretch/squeeze the audio data to make it match the timestamps or to inject silence / cut out audio to make it match the timestamps, do a combination of both or do neither.

The filter accepts the syntax [*sample_rate*:]*resampler_options*, where *sample_rate* expresses a sample rate and *resampler_options* is a list of *key*=*value* pairs, separated by ":". See the ffmpeg-resampler manual for the complete list of supported options.

### 6.12.1 Examples

- Resample the input audio to 44100Hz:

  ```
  aresample=44100
  ```

- Stretch/squeeze samples to the given timestamps, with a maximum of 1000 samples per second compensation:

  ```
  aresample=async=1000
  ```

## 6.13 asetnsamples

Set the number of samples per each output audio frame.

The last output packet may contain a different number of samples, as the filter will flush all the remaining samples when the input audio signal its end.

The filter accepts the following options:

`nb_out_samples, n`

> Set the number of frames per each output audio frame. The number is intended as the number of samples *per each channel*. Default value is 1024.

`pad, p`

> If set to 1, the filter will pad the last audio frame with zeroes, so that the last frame will contain the same number of samples as the previous ones. Default value is 1.

For example, to set the number of per-frame samples to 1234 and disable padding for the last frame, use:

```
asetnsamples=n=1234:p=0
```

## 6.14 asetrate

Set the sample rate without altering the PCM data. This will result in a change of speed and pitch.

The filter accepts the following options:

`sample_rate, r`

> Set the output sample rate. Default is 44100 Hz.

## 6.15 ashowinfo

Show a line containing various information for each input audio frame. The input audio is not modified.

The shown line contains a sequence of key/value pairs of the form *key*:*value*.

The following values are shown in the output:

`n`

> The (sequential) number of the input frame, starting from 0.

`pts`

The presentation timestamp of the input frame, in time base units; the time base depends on the filter input pad, and is usually 1/*sample_rate*.

`pts_time`

The presentation timestamp of the input frame in seconds.

`pos`

position of the frame in the input stream, -1 if this information in unavailable and/or meaningless (for example in case of synthetic audio)

`fmt`

The sample format.

`chlayout`

The channel layout.

`rate`

The sample rate for the audio frame.

`nb_samples`

The number of samples (per channel) in the frame.

`checksum`

The Adler-32 checksum (printed in hexadecimal) of the audio data. For planar audio, the data is treated as if all the planes were concatenated.

`plane_checksums`

A list of Adler-32 checksums for each data plane.

## 6.16 astats

Display time domain statistical information about the audio channels. Statistics are calculated and displayed for each audio channel and, where applicable, an overall figure is also given.

It accepts the following option:

`length`

Short window length in seconds, used for peak and trough RMS measurement. Default is `0.05` (50 miliseconds). Allowed range is `[0.1 - 10]`.

A description of each shown parameter follows:

`DC offset`

Mean amplitude displacement from zero.

`Min level`

Minimal sample level.

`Max level`

Maximal sample level.

`Peak level dB`
`RMS level dB`

Standard peak and RMS level measured in dBFS.

`RMS peak dB`
`RMS trough dB`

Peak and trough values for RMS level measured over a short window.

`Crest factor`

Standard ratio of peak to RMS level (note: not in dB).

`Flat factor`

Flatness (i.e. consecutive samples with the same value) of the signal at its peak levels (i.e. either *Min level* or *Max level*).

`Peak count`

Number of occasions (not the number of samples) that the signal attained either *Min level* or *Max level*.

## 6.17 astreamsync

Forward two audio streams and control the order the buffers are forwarded.

The filter accepts the following options:

`expr, e`

Set the expression deciding which stream should be forwarded next: if the result is negative, the first stream is forwarded; if the result is positive or zero, the second stream is forwarded. It can use the following variables:

*b1 b2*

number of buffers forwarded so far on each stream

*s1 s2*

number of samples forwarded so far on each stream

*t1 t2*

current timestamp of each stream

The default value is `t1-t2`, which means to always forward the stream that has a smaller timestamp.

### 6.17.1 Examples

Stress-test `amerge` by randomly sending buffers on the wrong input, while avoiding too much of a desynchronization:

```
amovie=file.ogg [a] ; amovie=file.mp3 [b] ;
[a] [b] astreamsync=(2*random(1))-1+tanh(5*(t1-t2)) [a2] [b2] ;
[a2] [b2] amerge
```

## 6.18 asyncts

Synchronize audio data with timestamps by squeezing/stretching it and/or dropping samples/adding silence when needed.

This filter is not built by default, please use aresample to do squeezing/stretching.

It accepts the following parameters:

`compensate`

Enable stretching/squeezing the data to make it match the timestamps. Disabled by default. When disabled, time gaps are covered with silence.

`min_delta`

The minimum difference between timestamps and audio data (in seconds) to trigger adding/dropping samples. The default value is 0.1. If you get an imperfect sync with this filter, try setting this parameter to 0.

`max_comp`

The maximum compensation in samples per second. Only relevant with compensate=1. The default value is 500.

`first_pts`

> Assume that the first PTS should be this value. The time base is 1 / sample rate. This allows for padding/trimming at the start of the stream. By default, no assumption is made about the first frame's expected PTS, so no padding or trimming is done. For example, this could be set to 0 to pad the beginning with silence if an audio stream starts after the video stream or to trim any samples with a negative PTS due to encoder delay.

## 6.19 atempo

Adjust audio tempo.

The filter accepts exactly one parameter, the audio tempo. If not specified then the filter will assume nominal 1.0 tempo. Tempo must be in the [0.5, 2.0] range.

### 6.19.1 Examples

- Slow down audio to 80% tempo:

  ```
  atempo=0.8
  ```

- To speed up audio to 125% tempo:

  ```
  atempo=1.25
  ```

## 6.20 atrim

Trim the input so that the output contains one continuous subpart of the input.

It accepts the following parameters:

`start`

> Timestamp (in seconds) of the start of the section to keep. I.e. the audio sample with the timestamp *start* will be the first sample in the output.

`end`

> Specify time of the first audio sample that will be dropped, i.e. the audio sample immediately preceding the one with the timestamp *end* will be the last sample in the output.

`start_pts`

> Same as *start*, except this option sets the start timestamp in samples instead of seconds.

`end_pts`

Same as *end*, except this option sets the end timestamp in samples instead of seconds.

`duration`

The maximum duration of the output in seconds.

`start_sample`

The number of the first sample that should be output.

`end_sample`

The number of the first sample that should be dropped.

`start`, `end`, and `duration` are expressed as time duration specifications; see (ffmpeg-utils)the Time duration section in the ffmpeg-utils(1) manual.

Note that the first two sets of the start/end options and the `duration` option look at the frame timestamp, while the _sample options simply count the samples that pass through the filter. So start/end_pts and start/end_sample will give different results when the timestamps are wrong, inexact or do not start at zero. Also note that this filter does not modify the timestamps. If you wish to have the output timestamps start at zero, insert the asetpts filter after the atrim filter.

If multiple start or end options are set, this filter tries to be greedy and keep all samples that match at least one of the specified constraints. To keep only the part that matches all the constraints at once, chain multiple atrim filters.

The defaults are such that all the input is kept. So it is possible to set e.g. just the end values to keep everything before the specified time.

Examples:

- Drop everything except the second minute of input:

  ```
  ffmpeg -i INPUT -af atrim=60:120
  ```

- Keep only the first 1000 samples:

  ```
  ffmpeg -i INPUT -af atrim=end_sample=1000
  ```

## 6.21 bandpass

Apply a two-pole Butterworth band-pass filter with central frequency *frequency*, and (3dB-point) band-width width. The *csg* option selects a constant skirt gain (peak gain = Q) instead of the default: constant 0dB peak gain. The filter roll off at 6dB per octave (20dB per decade).

The filter accepts the following options:

`frequency, f`

Set the filter’s central frequency. Default is `3000`.

`csg`

Constant skirt gain if set to 1. Defaults to 0.

`width_type`

Set method to specify band-width of filter.

`h`

Hz

`q`

Q-Factor

`o`

octave

`s`

slope

`width, w`

Specify the band-width of a filter in width_type units.

## 6.22 bandreject

Apply a two-pole Butterworth band-reject filter with central frequency *frequency*, and (3dB-point) band-width *width*. The filter roll off at 6dB per octave (20dB per decade).

The filter accepts the following options:

`frequency, f`

Set the filter’s central frequency. Default is `3000`.

`width_type`

Set method to specify band-width of filter.

`h`

Hz

`q`

Q-Factor

`o`

octave

`s`

slope

`width, w`

Specify the band-width of a filter in width_type units.

## 6.23 bass

Boost or cut the bass (lower) frequencies of the audio using a two-pole shelving filter with a response similar to that of a standard hi-fi's tone-controls. This is also known as shelving equalisation (EQ).

The filter accepts the following options:

`gain, g`

Give the gain at 0 Hz. Its useful range is about -20 (for a large cut) to +20 (for a large boost). Beware of clipping when using a positive gain.

`frequency, f`

Set the filter's central frequency and so can be used to extend or reduce the frequency range to be boosted or cut. The default value is `100` Hz.

`width_type`

Set method to specify band-width of filter.

`h`

Hz

`q`

Q-Factor

`o`

octave

`s`

slope

`width, w`

Determine how steep is the filter’s shelf transition.

## 6.24 biquad

Apply a biquad IIR filter with the given coefficients. Where *b0*, *b1*, *b2* and *a0*, *a1*, *a2* are the numerator and denominator coefficients respectively.

## 6.25 bs2b

Bauer stereo to binaural transformation, which improves headphone listening of stereo audio records.

It accepts the following parameters:

`profile`

Pre-defined crossfeed level.

`default`

Default level (fcut=700, feed=50).

`cmoy`

Chu Moy circuit (fcut=700, feed=60).

`jmeier`

Jan Meier circuit (fcut=650, feed=95).

`fcut`

Cut frequency (in Hz).

`feed`

Feed level (in Hz).

## 6.26 channelmap

Remap input channels to new locations.

It accepts the following parameters:

`channel_layout`

The channel layout of the output stream.

`map`

Map channels from input to output. The argument is a ’|’-separated list of mappings, each in the *`in_channel-out_channel`* or *in_channel* form. *in_channel* can be either the name of the input channel (e.g. FL for front left) or its index in the input channel layout. *out_channel* is the name of the output channel or its index in the output channel layout. If *out_channel* is not given then it is implicitly an index, starting with zero and increasing by one for each mapping.

If no mapping is present, the filter will implicitly map input channels to output channels, preserving indices.

For example, assuming a 5.1+downmix input MOV file,

```
ffmpeg -i in.mov -filter ’channelmap=map=DL-FL|DR-FR’ out.wav
```

will create an output WAV file tagged as stereo from the downmix channels of the input.

To fix a 5.1 WAV improperly encoded in AAC’s native channel order

```
ffmpeg -i in.wav -filter ’channelmap=1|2|0|5|3|4:channel_layout=5.1’ out.wav
```

## 6.27 channelsplit

Split each channel from an input audio stream into a separate output stream.

It accepts the following parameters:

`channel_layout`

The channel layout of the input stream. The default is "stereo".

For example, assuming a stereo input MP3 file,

```
ffmpeg -i in.mp3 -filter_complex channelsplit out.mkv
```

will create an output Matroska file with two audio streams, one containing only the left channel and the other the right channel.

Split a 5.1 WAV file into per-channel files:

```
ffmpeg -i in.wav -filter_complex
'channelsplit=channel_layout=5.1[FL][FR][FC][LFE][SL][SR]'
-map '[FL]' front_left.wav -map '[FR]' front_right.wav -map '[FC]'
front_center.wav -map '[LFE]' lfe.wav -map '[SL]' side_left.wav -map '[SR]'
side_right.wav
```

## 6.28 compand

Compress or expand the audio's dynamic range.

It accepts the following parameters:

`attacks`
`decays`

> A list of times in seconds for each channel over which the instantaneous level of the input signal is averaged to determine its volume. *attacks* refers to increase of volume and *decays* refers to decrease of volume. For most situations, the attack time (response to the audio getting louder) should be shorter than the decay time, because the human ear is more sensitive to sudden loud audio than sudden soft audio. A typical value for attack is 0.3 seconds and a typical value for decay is 0.8 seconds.

`points`

> A list of points for the transfer function, specified in dB relative to the maximum possible signal amplitude. Each key points list must be defined using the following syntax: `x0/y0|x1/y1|x2/y2|....` or `x0/y0 x1/y1 x2/y2 ....`
>
> The input values must be in strictly increasing order but the transfer function does not have to be monotonically rising. The point `0/0` is assumed but may be overridden (by `0/out-dBn`). Typical values for the transfer function are `-70/-70|-60/-20`.

`soft-knee`

> Set the curve radius in dB for all joints. It defaults to 0.01.

`gain`

> Set the additional gain in dB to be applied at all points on the transfer function. This allows for easy adjustment of the overall gain. It defaults to 0.

`volume`

> Set an initial volume, in dB, to be assumed for each channel when filtering starts. This permits the user to supply a nominal level initially, so that, for example, a very large gain is not applied to initial signal levels before the companding has begun to operate. A typical value for audio which is initially quiet is -90 dB. It defaults to 0.

`delay`

Set a delay, in seconds. The input audio is analyzed immediately, but audio is delayed before being fed to the volume adjuster. Specifying a delay approximately equal to the attack/decay times allows the filter to effectively operate in predictive rather than reactive mode. It defaults to 0.

### 6.28.1 Examples

- Make music with both quiet and loud passages suitable for listening to in a noisy environment:

  ```
  compand=.3|.3:1|1:-90/-60|-60/-40|-40/-30|-20/-20:6:0:-90:0.2
  ```

- A noise gate for when the noise is at a lower level than the signal:

  ```
  compand=.1|.1:.2|.2:-900/-900|-50.1/-900|-50/-50:.01:0:-90:.1
  ```

- Here is another noise gate, this time for when the noise is at a higher level than the signal (making it, in some ways, similar to squelch):

  ```
  compand=.1|.1:.1|.1:-45.1/-45.1|-45/-900|0/-900:.01:45:-90:.1
  ```

## 6.29 earwax

Make audio easier to listen to on headphones.

This filter adds ‘cues’ to 44.1kHz stereo (i.e. audio CD format) audio so that when listened to on headphones the stereo image is moved from inside your head (standard for headphones) to outside and in front of the listener (standard for speakers).

Ported from SoX.

## 6.30 equalizer

Apply a two-pole peaking equalisation (EQ) filter. With this filter, the signal-level at and around a selected frequency can be increased or decreased, whilst (unlike bandpass and bandreject filters) that at all other frequencies is unchanged.

In order to produce complex equalisation curves, this filter can be given several times, each with a different central frequency.

The filter accepts the following options:

`frequency, f`

Set the filter’s central frequency in Hz.

`width_type`

Set method to specify band-width of filter.

`h`

Hz

`q`

Q-Factor

`o`

octave

`s`

slope

`width, w`

Specify the band-width of a filter in width_type units.

`gain, g`

Set the required gain or attenuation in dB. Beware of clipping when using a positive gain.

### 6.30.1 Examples

- Attenuate 10 dB at 1000 Hz, with a bandwidth of 200 Hz:

```
equalizer=f=1000:width_type=h:width=200:g=-10
```

- Apply 2 dB gain at 1000 Hz with Q 1 and attenuate 5 dB at 100 Hz with Q 2:

```
equalizer=f=1000:width_type=q:width=1:g=2,equalizer=f=100:width_type=q:width=2:g=-5
```

## 6.31 flanger

Apply a flanging effect to the audio.

The filter accepts the following options:

`delay`

Set base delay in milliseconds. Range from 0 to 30. Default value is 0.

`depth`

Set added swep delay in milliseconds. Range from 0 to 10. Default value is 2.

`regen`

Set percentage regeneneration (delayed signal feedback). Range from -95 to 95. Default value is 0.

`width`

Set percentage of delayed signal mixed with original. Range from 0 to 100. Default valu is 71.

`speed`

Set sweeps per second (Hz). Range from 0.1 to 10. Default value is 0.5.

`shape`

Set swept wave shape, can be *triangular* or *sinusoidal*. Default value is *sinusoidal*.

`phase`

Set swept wave percentage-shift for multi channel. Range from 0 to 100. Default value is 25.

`interp`

Set delay-line interpolation, *linear* or *quadratic*. Default is *linear*.

## 6.32 highpass

Apply a high-pass filter with 3dB point frequency. The filter can be either single-pole, or double-pole (the default). The filter roll off at 6dB per pole per octave (20dB per pole per decade).

The filter accepts the following options:

`frequency, f`

Set frequency in Hz. Default is 3000.

`poles, p`

Set number of poles. Default is 2.

`width_type`

Set method to specify band-width of filter.

`h`

Hz

`q`

Q-Factor

`o`

octave

`s`

slope

`width, w`

Specify the band-width of a filter in width_type units. Applies only to double-pole filter. The default is 0.707q and gives a Butterworth response.

## 6.33 join

Join multiple input streams into one multi-channel stream.

It accepts the following parameters:

`inputs`

The number of input streams. It defaults to 2.

`channel_layout`

The desired output channel layout. It defaults to stereo.

`map`

Map channels from inputs to output. The argument is a '|'-separated list of mappings, each in the *`input_idx.in_channel-out_channel`* form. *input_idx* is the 0-based index of the input stream. *in_channel* can be either the name of the input channel (e.g. FL for front left) or its index in the specified input stream. *out_channel* is the name of the output channel.

The filter will attempt to guess the mappings when they are not specified explicitly. It does so by first trying to find an unused matching input channel and if that fails it picks the first unused input channel.

Join 3 inputs (with properly set channel layouts):

```
ffmpeg -i INPUT1 -i INPUT2 -i INPUT3 -filter_complex join=inputs=3 OUTPUT
```

Build a 5.1 output from 6 single-channel streams:

```
ffmpeg -i fl -i fr -i fc -i sl -i sr -i lfe -filter_complex
'join=inputs=6:channel_layout=5.1:map=0.0-FL|1.0-FR|2.0-FC|3.0-SL|4.0-SR|5.0-LFE'
out
```

## 6.34 ladspa

Load a LADSPA (Linux Audio Developer's Simple Plugin API) plugin.

To enable compilation of this filter you need to configure FFmpeg with `--enable-ladspa`.

`file, f`

> Specifies the name of LADSPA plugin library to load. If the environment variable `LADSPA_PATH` is defined, the LADSPA plugin is searched in each one of the directories specified by the colon separated list in `LADSPA_PATH`, otherwise in the standard LADSPA paths, which are in this order: `HOME/.ladspa/lib/`, `/usr/local/lib/ladspa/`, `/usr/lib/ladspa/`.

`plugin, p`

> Specifies the plugin within the library. Some libraries contain only one plugin, but others contain many of them. If this is not set filter will list all available plugins within the specified library.

`controls, c`

> Set the '|' separated list of controls which are zero or more floating point values that determine the behavior of the loaded plugin (for example delay, threshold or gain). Controls need to be defined using the following syntax: c0=*value0*|c1=*value1*|c2=*value2*|..., where *valuei* is the value set on the *i*-th control. If `controls` is set to `help`, all available controls and their valid ranges are printed.

`sample_rate, s`

> Specify the sample rate, default to 44100. Only used if plugin have zero inputs.

`nb_samples, n`

> Set the number of samples per channel per each output frame, default is 1024. Only used if plugin have zero inputs.

`duration, d`

> Set the minimum duration of the sourced audio. See (ffmpeg-utils)the Time duration section in the ffmpeg-utils(1) manual for the accepted syntax. Note that the resulting duration may be greater than the specified duration, as the generated audio is always cut at the end of a complete frame. If not specified, or the expressed duration is negative, the audio is supposed to be generated forever. Only used if plugin have zero inputs.

### 6.34.1 Examples

- List all available plugins within amp (LADSPA example plugin) library:

  ```
  ladspa=file=amp
  ```

- List all available controls and their valid ranges for `vcf_notch` plugin from `VCF` library:

  ```
  ladspa=f=vcf:p=vcf_notch:c=help
  ```

- Simulate low quality audio equipment using `Computer Music Toolkit` (CMT) plugin library:

  ```
  ladspa=file=cmt:plugin=lofi:controls=c0=22|c1=12|c2=12
  ```

- Add reverberation to the audio using TAP-plugins (Tom's Audio Processing plugins):

  ```
  ladspa=file=tap_reverb:tap_reverb
  ```

- Generate white noise, with 0.2 amplitude:

  ```
  ladspa=file=cmt:noise_source_white:c=c0=.2
  ```

- Generate 20 bpm clicks using plugin `C* Click - Metronome` from the `C* Audio Plugin Suite` (CAPS) library:

  ```
  ladspa=file=caps:Click:c=c1=20'
  ```

- Apply `C* Eq10X2 - Stereo 10-band equaliser` effect:

  ```
  ladspa=caps:Eq10X2:c=c0=-48|c9=-24|c3=12|c4=2
  ```

### 6.34.2 Commands

This filter supports the following commands:

`cN`

  Modify the *N*-th control value.

  If the specified value is not valid, it is ignored and prior one is kept.

## 6.35 lowpass

Apply a low-pass filter with 3dB point frequency. The filter can be either single-pole or double-pole (the default). The filter roll off at 6dB per pole per octave (20dB per pole per decade).

The filter accepts the following options:

`frequency, f`

Set frequency in Hz. Default is 500.

`poles, p`

Set number of poles. Default is 2.

`width_type`

Set method to specify band-width of filter.

`h`

Hz

`q`

Q-Factor

`o`

octave

`s`

slope

`width, w`

Specify the band-width of a filter in width_type units. Applies only to double-pole filter. The default is 0.707q and gives a Butterworth response.

## 6.36 pan

Mix channels with specific gain levels. The filter accepts the output channel layout followed by a set of channels definitions.

This filter is also designed to efficiently remap the channels of an audio stream.

The filter accepts parameters of the form: "*l*|*outdef*|*outdef*|..."

`l`

output channel layout or number of channels

`outdef`

output channel specification, of the form: "*out_name*=[*gain**]*in_name*[+[*gain**]*in_name*...]"

`out_name`

output channel to define, either a channel name (FL, FR, etc.) or a channel number (c0, c1, etc.)

`gain`

multiplicative coefficient for the channel, 1 leaving the volume unchanged

`in_name`

input channel to use, see out_name for details; it is not possible to mix named and numbered input channels

If the ‘=’ in a channel specification is replaced by ‘<’, then the gains for that specification will be renormalized so that the total is 1, thus avoiding clipping noise.

### 6.36.1 Mixing examples

For example, if you want to down-mix from stereo to mono, but with a bigger factor for the left channel:

```
pan=1c|c0=0.9*c0+0.1*c1
```

A customized down-mix to stereo that works automatically for 3-, 4-, 5- and 7-channels surround:

```
pan=stereo| FL < FL + 0.5*FC + 0.6*BL + 0.6*SL | FR < FR + 0.5*FC + 0.6*BR + 0.6*SR
```

Note that `ffmpeg` integrates a default down-mix (and up-mix) system that should be preferred (see "-ac" option) unless you have very specific needs.

### 6.36.2 Remapping examples

The channel remapping will be effective if, and only if:

- gain coefficients are zeroes or ones,
- only one input per channel output,

If all these conditions are satisfied, the filter will notify the user ("Pure channel mapping detected"), and use an optimized and lossless method to do the remapping.

For example, if you have a 5.1 source and want a stereo audio stream by dropping the extra channels:

```
pan="stereo| c0=FL | c1=FR"
```

Given the same source, you can also switch front left and front right channels and keep the input channel layout:

```
pan="5.1| c0=c1 | c1=c0 | c2=c2 | c3=c3 | c4=c4 | c5=c5"
```

If the input is a stereo audio stream, you can mute the front left channel (and still keep the stereo channel layout) with:

```
pan="stereo|c1=c1"
```

Still with a stereo audio stream input, you can copy the right channel in both front left and right:

```
pan="stereo| c0=FR | c1=FR"
```

## 6.37 replaygain

ReplayGain scanner filter. This filter takes an audio stream as an input and outputs it unchanged. At end of filtering it displays `track_gain` and `track_peak`.

## 6.38 resample

Convert the audio sample format, sample rate and channel layout. It is not meant to be used directly.

## 6.39 silencedetect

Detect silence in an audio stream.

This filter logs a message when it detects that the input audio volume is less or equal to a noise tolerance value for a duration greater or equal to the minimum detected noise duration.

The printed times and duration are expressed in seconds.

The filter accepts the following options:

`duration, d`

Set silence duration until notification (default is 2 seconds).

`noise, n`

Set noise tolerance. Can be specified in dB (in case "dB" is appended to the specified value) or amplitude ratio. Default is -60dB, or 0.001.

### 6.39.1 Examples

- Detect 5 seconds of silence with -50dB noise tolerance:

  ```
  silencedetect=n=-50dB:d=5
  ```

- Complete example with `ffmpeg` to detect silence with 0.0001 noise tolerance in `silence.mp3`:

  ```
  ffmpeg -i silence.mp3 -af silencedetect=noise=0.0001 -f null -
  ```

## 6.40 silenceremove

Remove silence from the beginning, middle or end of the audio.

The filter accepts the following options:

`start_periods`

> This value is used to indicate if audio should be trimmed at beginning of the audio. A value of zero indicates no silence should be trimmed from the beginning. When specifying a non-zero value, it trims audio up until it finds non-silence. Normally, when trimming silence from beginning of audio the *start_periods* will be `1` but it can be increased to higher values to trim all audio up to specific count of non-silence periods. Default value is `0`.

`start_duration`

> Specify the amount of time that non-silence must be detected before it stops trimming audio. By increasing the duration, bursts of noises can be treated as silence and trimmed off. Default value is `0`.

`start_threshold`

> This indicates what sample value should be treated as silence. For digital audio, a value of `0` may be fine but for audio recorded from analog, you may wish to increase the value to account for background noise. Can be specified in dB (in case "dB" is appended to the specified value) or amplitude ratio. Default value is `0`.

`stop_periods`

> Set the count for trimming silence from the end of audio. To remove silence from the middle of a file, specify a *stop_periods* that is negative. This value is then threated as a positive value and is used to indicate the effect should restart processing as specified by *start_periods*, making it suitable for removing periods of silence in the middle of the audio. Default value is `0`.

`stop_duration`

> Specify a duration of silence that must exist before audio is not copied any more. By specifying a higher duration, silence that is wanted can be left in the audio. Default value is `0`.

`stop_threshold`

> This is the same as `start_threshold` but for trimming silence from the end of audio. Can be specified in dB (in case "dB" is appended to the specified value) or amplitude ratio. Default value is `0`.

`leave_silence`

> This indicate that *stop_duration* length of audio should be left intact at the beginning of each period of silence. For example, if you want to remove long pauses between words but do not want to remove the pauses completely. Default value is `0`.

### 6.40.1 Examples

- The following example shows how this filter can be used to start a recording that does not contain the delay at the start which usually occurs between pressing the record button and the start of the performance:

  ```
  silenceremove=1:5:0.02
  ```

## 6.41 treble

Boost or cut treble (upper) frequencies of the audio using a two-pole shelving filter with a response similar to that of a standard hi-fi's tone-controls. This is also known as shelving equalisation (EQ).

The filter accepts the following options:

`gain, g`

> Give the gain at whichever is the lower of ~22 kHz and the Nyquist frequency. Its useful range is about -20 (for a large cut) to +20 (for a large boost). Beware of clipping when using a positive gain.

`frequency, f`

> Set the filter's central frequency and so can be used to extend or reduce the frequency range to be boosted or cut. The default value is `3000` Hz.

`width_type`

> Set method to specify band-width of filter.
>
> `h`
>
> > Hz
>
> `q`
>
> > Q-Factor
>
> `o`
>
> > octave
>
> `s`
>
> > slope

`width, w`

Determine how steep is the filter’s shelf transition.

## 6.42 volume

Adjust the input audio volume.

It accepts the following parameters:

`volume`

Set audio volume expression.

Output values are clipped to the maximum value.

The output audio volume is given by the relation:

*output_volume = volume * input_volume*

The default value for *volume* is "1.0".

`precision`

This parameter represents the mathematical precision.

It determines which input sample formats will be allowed, which affects the precision of the volume scaling.

`fixed`

8-bit fixed-point; this limits input sample format to U8, S16, and S32.

`float`

32-bit floating-point; this limits input sample format to FLT. (default)

`double`

64-bit floating-point; this limits input sample format to DBL.

`replaygain`

Choose the behaviour on encountering ReplayGain side data in input frames.

`drop`

Remove ReplayGain side data, ignoring its contents (the default).

`ignore`

Ignore ReplayGain side data, but leave it in the frame.

`track`

Prefer the track gain, if present.

`album`

Prefer the album gain, if present.

`replaygain_preamp`

Pre-amplification gain in dB to apply to the selected replaygain gain.

Default value for *replaygain_preamp* is 0.0.

`eval`

Set when the volume expression is evaluated.

It accepts the following values:

‘`once`’

only evaluate expression once during the filter initialization, or when the ‘`volume`’ command is sent

‘`frame`’

evaluate expression for each incoming frame

Default value is ‘`once`’.

The volume expression can contain the following parameters.

`n`

frame number (starting at zero)

`nb_channels`

number of channels

`nb_consumed_samples`

number of samples consumed by the filter

`nb_samples`

number of samples in the current frame

`pos`

original frame position in the file

`pts`

frame PTS

`sample_rate`

sample rate

`startpts`

PTS at start of stream

`startt`

time at start of stream

`t`

frame time

`tb`

timestamp timebase

`volume`

last set volume value

Note that when `eval` is set to ‘`once`’ only the *sample_rate* and *tb* variables are available, all other variables will evaluate to NAN.

### 6.42.1 Commands

This filter supports the following commands:

`volume`

Modify the volume expression. The command accepts the same syntax of the corresponding option.

If the specified expression is not valid, it is kept at its current value.

`replaygain_noclip`

Prevent clipping by limiting the gain applied.

Default value for *replaygain_noclip* is 1.

### 6.42.2 Examples

- Halve the input audio volume:

  ```
  volume=volume=0.5
  volume=volume=1/2
  volume=volume=-6.0206dB
  ```

  In all the above example the named key for `volume` can be omitted, for example like in:

  ```
  volume=0.5
  ```

- Increase input audio power by 6 decibels using fixed-point precision:

  ```
  volume=volume=6dB:precision=fixed
  ```

- Fade volume after time 10 with an annihilation period of 5 seconds:

  ```
  volume='if(lt(t,10),1,max(1-(t-10)/5,0))':eval=frame
  ```

## 6.43 volumedetect

Detect the volume of the input video.

The filter has no parameters. The input is not modified. Statistics about the volume will be printed in the log when the input stream end is reached.

In particular it will show the mean volume (root mean square), maximum volume (on a per-sample basis), and the beginning of a histogram of the registered volume values (from the maximum value to a cumulated 1/1000 of the samples).

All volumes are in decibels relative to the maximum PCM value.

### 6.43.1 Examples

Here is an excerpt of the output:

```
[Parsed_volumedetect_0  0xa23120] mean_volume: -27 dB
[Parsed_volumedetect_0  0xa23120] max_volume: -4 dB
[Parsed_volumedetect_0  0xa23120] histogram_4db: 6
[Parsed_volumedetect_0  0xa23120] histogram_5db: 62
[Parsed_volumedetect_0  0xa23120] histogram_6db: 286
[Parsed_volumedetect_0  0xa23120] histogram_7db: 1042
[Parsed_volumedetect_0  0xa23120] histogram_8db: 2551
[Parsed_volumedetect_0  0xa23120] histogram_9db: 4609
[Parsed_volumedetect_0  0xa23120] histogram_10db: 8409
```

It means that:

- The mean square energy is approximately -27 dB, or 10^-2.7.
- The largest sample is at -4 dB, or more precisely between -4 dB and -5 dB.
- There are 6 samples at -4 dB, 62 at -5 dB, 286 at -6 dB, etc.

In other words, raising the volume by +4 dB does not cause any clipping, raising it by +5 dB causes clipping for 6 samples, etc.

# 7 Audio Sources

Below is a description of the currently available audio sources.

## 7.1 abuffer

Buffer audio frames, and make them available to the filter chain.

This source is mainly intended for a programmatic use, in particular through the interface defined in `libavfilter/asrc_abuffer.h`.

It accepts the following parameters:

`time_base`

The timebase which will be used for timestamps of submitted frames. It must be either a floating-point number or in *numerator*/*denominator* form.

`sample_rate`

The sample rate of the incoming audio buffers.

`sample_fmt`

The sample format of the incoming audio buffers. Either a sample format name or its corresponging integer representation from the enum AVSampleFormat in `libavutil/samplefmt.h`

`channel_layout`

The channel layout of the incoming audio buffers. Either a channel layout name from channel_layout_map in `libavutil/channel_layout.c` or its corresponding integer representation from the AV_CH_LAYOUT_* macros in `libavutil/channel_layout.h`

`channels`

The number of channels of the incoming audio buffers. If both *channels* and *channel_layout* are specified, then they must be consistent.

### 7.1.1 Examples

```
abuffer=sample_rate=44100:sample_fmt=s16p:channel_layout=stereo
```

will instruct the source to accept planar 16bit signed stereo at 44100Hz. Since the sample format with name "s16p" corresponds to the number 6 and the "stereo" channel layout corresponds to the value 0x3, this is equivalent to:

```
abuffer=sample_rate=44100:sample_fmt=6:channel_layout=0x3
```

## 7.2 aevalsrc

Generate an audio signal specified by an expression.

This source accepts in input one or more expressions (one for each channel), which are evaluated and used to generate a corresponding audio signal.

This source accepts the following options:

`exprs`

> Set the '|'-separated expressions list for each separate channel. In case the `channel_layout` option is not specified, the selected channel layout depends on the number of provided expressions. Otherwise the last specified expression is applied to the remaining output channels.

`channel_layout, c`

> Set the channel layout. The number of channels in the specified layout must be equal to the number of specified expressions.

`duration, d`

> Set the minimum duration of the sourced audio. See (ffmpeg-utils)the Time duration section in the ffmpeg-utils(1) manual for the accepted syntax. Note that the resulting duration may be greater than the specified duration, as the generated audio is always cut at the end of a complete frame.
>
> If not specified, or the expressed duration is negative, the audio is supposed to be generated forever.

`nb_samples, n`

> Set the number of samples per channel per each output frame, default to 1024.

`sample_rate, s`

> Specify the sample rate, default to 44100.

Each expression in *exprs* can contain the following constants:

`n`

number of the evaluated sample, starting from 0

`t`

time of the evaluated sample expressed in seconds, starting from 0

`s`

sample rate

### 7.2.1 Examples

- Generate silence:

```
aevalsrc=0
```

- Generate a sin signal with frequency of 440 Hz, set sample rate to 8000 Hz:

```
aevalsrc="sin(440*2*PI*t):s=8000"
```

- Generate a two channels signal, specify the channel layout (Front Center + Back Center) explicitly:

```
aevalsrc="sin(420*2*PI*t)|cos(430*2*PI*t):c=FC|BC"
```

- Generate white noise:

```
aevalsrc="-2+random(0)"
```

- Generate an amplitude modulated signal:

```
aevalsrc="sin(10*2*PI*t)*sin(880*2*PI*t)"
```

- Generate 2.5 Hz binaural beats on a 360 Hz carrier:

```
aevalsrc="0.1*sin(2*PI*(360-2.5/2)*t) | 0.1*sin(2*PI*(360+2.5/2)*t)"
```

## 7.3 anullsrc

The null audio source, return unprocessed audio frames. It is mainly useful as a template and to be employed in analysis / debugging tools, or as the source for filters which ignore the input data (for example the sox synth filter).

This source accepts the following options:

`channel_layout, cl`

Specifies the channel layout, and can be either an integer or a string representing a channel layout. The default value of *channel_layout* is "stereo".

Check the channel_layout_map definition in `libavutil/channel_layout.c` for the mapping between strings and channel layout values.

`sample_rate, r`

Specifies the sample rate, and defaults to 44100.

`nb_samples, n`

Set the number of samples per requested frames.

### 7.3.1 Examples

- Set the sample rate to 48000 Hz and the channel layout to AV_CH_LAYOUT_MONO.

```
anullsrc=r=48000:cl=4
```

- Do the same operation with a more obvious syntax:

```
anullsrc=r=48000:cl=mono
```

All the parameters need to be explicitly defined.

## 7.4 flite

Synthesize a voice utterance using the libflite library.

To enable compilation of this filter you need to configure FFmpeg with `--enable-libflite`.

Note that the flite library is not thread-safe.

The filter accepts the following options:

`list_voices`

If set to 1, list the names of the available voices and exit immediately. Default value is 0.

`nb_samples, n`

Set the maximum number of samples per frame. Default value is 512.

`textfile`

Set the filename containing the text to speak.

`text`

Set the text to speak.

`voice, v`

Set the voice to use for the speech synthesis. Default value is `kal`. See also the *list_voices* option.

### 7.4.1 Examples

- Read from file `speech.txt`, and synthetize the text using the standard flite voice:

  ```
  flite=textfile=speech.txt
  ```

- Read the specified text selecting the `slt` voice:

  ```
  flite=text='So fare thee well, poor devil of a Sub-Sub, whose commentator I am':voice=slt
  ```

- Input text to ffmpeg:

  ```
  ffmpeg -f lavfi -i flite=text='So fare thee well, poor devil of a Sub-Sub, whose commentator I am':voice=slt
  ```

- Make `ffplay` speak the specified text, using `flite` and the `lavfi` device:

  ```
  ffplay -f lavfi flite=text='No more be grieved for which that thou hast done.'
  ```

For more information about libflite, check: http://www.speech.cs.cmu.edu/flite/

## 7.5 sine

Generate an audio signal made of a sine wave with amplitude 1/8.

The audio signal is bit-exact.

The filter accepts the following options:

`frequency, f`

Set the carrier frequency. Default is 440 Hz.

`beep_factor, b`

Enable a periodic beep every second with frequency *beep_factor* times the carrier frequency. Default is 0, meaning the beep is disabled.

`sample_rate, r`

Specify the sample rate, default is 44100.

`duration, d`

Specify the duration of the generated audio stream.

`samples_per_frame`

Set the number of samples per output frame, default is 1024.

### 7.5.1 Examples

- Generate a simple 440 Hz sine wave:

  ```
  sine
  ```

- Generate a 220 Hz sine wave with a 880 Hz beep each second, for 5 seconds:

  ```
  sine=220:4:d=5
  sine=f=220:b=4:d=5
  sine=frequency=220:beep_factor=4:duration=5
  ```

# 8 Audio Sinks

Below is a description of the currently available audio sinks.

## 8.1 abuffersink

Buffer audio frames, and make them available to the end of filter chain.

This sink is mainly intended for programmatic use, in particular through the interface defined in `libavfilter/buffersink.h` or the options system.

It accepts a pointer to an AVABufferSinkContext structure, which defines the incoming buffers' formats, to be passed as the opaque parameter to `avfilter_init_filter` for initialization.

## 8.2 anullsink

Null audio sink; do absolutely nothing with the input audio. It is mainly useful as a template and for use in analysis / debugging tools.

# 9 Video Filters

When you configure your FFmpeg build, you can disable any of the existing filters using `--disable-filters`. The configure output will show the video filters included in your build.

Below is a description of the currently available video filters.

## 9.1 alphaextract

Extract the alpha component from the input as a grayscale video. This is especially useful with the *alphamerge* filter.

## 9.2 alphamerge

Add or replace the alpha component of the primary input with the grayscale value of a second input. This is intended for use with *alphaextract* to allow the transmission or storage of frame sequences that have alpha in a format that doesn't support an alpha channel.

For example, to reconstruct full frames from a normal YUV-encoded video and a separate video created with *alphaextract*, you might use:

```
movie=in_alpha.mkv [alpha]; [in][alpha] alphamerge [out]
```

Since this filter is designed for reconstruction, it operates on frame sequences without considering timestamps, and terminates when either input reaches end of stream. This will cause problems if your encoding pipeline drops frames. If you're trying to apply an image as an overlay to a video stream, consider the *overlay* filter instead.

## 9.3 ass

Same as the subtitles filter, except that it doesn't require libavcodec and libavformat to work. On the other hand, it is limited to ASS (Advanced Substation Alpha) subtitles files.

This filter accepts the following option in addition to the common options from the subtitles filter:

`shaping`

> Set the shaping engine
>
> Available values are:
>
> '`auto`'
>
> > The default libass shaping engine, which is the best available.
>
> '`simple`'
>
> > Fast, font-agnostic shaper that can do only substitutions
>
> '`complex`'
>
> > Slower shaper using OpenType for substitutions and positioning
>
> The default is `auto`.

## 9.4 bbox

Compute the bounding box for the non-black pixels in the input frame luminance plane.

This filter computes the bounding box containing all the pixels with a luminance value greater than the minimum allowed value. The parameters describing the bounding box are printed on the filter log.

The filter accepts the following option:

`min_val`

Set the minimal luminance value. Default is `16`.

## 9.5 blackdetect

Detect video intervals that are (almost) completely black. Can be useful to detect chapter transitions, commercials, or invalid recordings. Output lines contains the time for the start, end and duration of the detected black interval expressed in seconds.

In order to display the output lines, you need to set the loglevel at least to the AV_LOG_INFO value.

The filter accepts the following options:

`black_min_duration, d`

Set the minimum detected black duration expressed in seconds. It must be a non-negative floating point number.

Default value is 2.0.

`picture_black_ratio_th, pic_th`

Set the threshold for considering a picture "black". Express the minimum value for the ratio:

*`nb_black_pixels / nb_pixels`*

for which a picture is considered black. Default value is 0.98.

`pixel_black_th, pix_th`

Set the threshold for considering a pixel "black".

The threshold expresses the maximum pixel luminance value for which a pixel is considered "black". The provided value is scaled according to the following equation:

*`absolute_threshold = luminance_minimum_value + pixel_black_th * luminance_range_size`*

*luminance_range_size* and *luminance_minimum_value* depend on the input video format, the range is [0-255] for YUV full-range formats and [16-235] for YUV non full-range formats.

Default value is 0.10.

The following example sets the maximum pixel threshold to the minimum value, and detects only black intervals of 2 or more seconds:

```
blackdetect=d=2:pix_th=0.00
```

## 9.6 blackframe

Detect frames that are (almost) completely black. Can be useful to detect chapter transitions or commercials. Output lines consist of the frame number of the detected frame, the percentage of blackness, the position in the file if known or `-1` and the timestamp in seconds.

In order to display the output lines, you need to set the loglevel at least to the AV_LOG_INFO value.

It accepts the following parameters:

`amount`

The percentage of the pixels that have to be below the threshold; it defaults to `98`.

`threshold, thresh`

The threshold below which a pixel value is considered black; it defaults to `32`.

## 9.7 blend

Blend two video frames into each other.

It takes two input streams and outputs one stream, the first input is the "top" layer and second input is "bottom" layer. Output terminates when shortest input terminates.

A description of the accepted options follows.

`c0_mode`
`c1_mode`
`c2_mode`
`c3_mode`
`all_mode`

Set blend mode for specific pixel component or all pixel components in case of *all_mode*. Default value is `normal`.

Available values for component modes are:

‘`addition`’
‘`and`’
‘`average`’
‘`burn`’

‘darken’
‘difference’
‘divide’
‘dodge’
‘exclusion’
‘hardlight’
‘lighten’
‘multiply’
‘negation’
‘normal’
‘or’
‘overlay’
‘phoenix’
‘pinlight’
‘reflect’
‘screen’
‘softlight’
‘subtract’
‘vividlight’
‘xor’

c0_opacity
c1_opacity
c2_opacity
c3_opacity
all_opacity

Set blend opacity for specific pixel component or all pixel components in case of *all_opacity*. Only used in combination with pixel component blend modes.

c0_expr
c1_expr
c2_expr
c3_expr
all_expr

Set blend expression for specific pixel component or all pixel components in case of *all_expr*. Note that related mode options will be ignored if those are set.

The expressions can use the following variables:

N

The sequential number of the filtered frame, starting from `0`.

X
Y

the coordinates of the current sample

`W`
`H`

the width and height of currently filtered plane

`SW`
`SH`

Width and height scale depending on the currently filtered plane. It is the ratio between the corresponding luma plane number of pixels and the current plane ones. E.g. for YUV4:2:0 the values are `1,1` for the luma plane, and `0.5,0.5` for chroma planes.

`T`

Time of the current frame, expressed in seconds.

`TOP, A`

Value of pixel component at current location for first video frame (top layer).

`BOTTOM, B`

Value of pixel component at current location for second video frame (bottom layer).

`shortest`

Force termination when the shortest input terminates. Default is `0`.

`repeatlast`

Continue applying the last bottom frame after the end of the stream. A value of `0` disable the filter after the last frame of the bottom layer is reached. Default is `1`.

### 9.7.1 Examples

- Apply transition from bottom layer to top layer in first 10 seconds:

```
blend=all_expr='A*(if(gte(T,10),1,T/10))+B*(1-(if(gte(T,10),1,T/10)))'
```

- Apply 1x1 checkerboard effect:

```
blend=all_expr='if(eq(mod(X,2),mod(Y,2)),A,B)'
```

- Apply uncover left effect:

```
blend=all_expr='if(gte(N*SW+X,W),A,B)'
```

- Apply uncover down effect:

```
blend=all_expr='if(gte(Y-N*SH,0),A,B)'
```

- Apply uncover up-left effect:

```
blend=all_expr='if(gte(T*SH*40+Y,H)*gte((T*40*SW+X)*W/H,W),A,B)'
```

## 9.8 boxblur

Apply a boxblur algorithm to the input video.

It accepts the following parameters:

`luma_radius, lr`
`luma_power, lp`
`chroma_radius, cr`
`chroma_power, cp`
`alpha_radius, ar`
`alpha_power, ap`

A description of the accepted options follows.

`luma_radius, lr`
`chroma_radius, cr`
`alpha_radius, ar`

Set an expression for the box radius in pixels used for blurring the corresponding input plane.

The radius value must be a non-negative number, and must not be greater than the value of the expression `min(w,h)/2` for the luma and alpha planes, and of `min(cw,ch)/2` for the chroma planes.

Default value for `luma_radius` is "2". If not specified, `chroma_radius` and `alpha_radius` default to the corresponding value set for `luma_radius`.

The expressions can contain the following constants:

`w`
`h`

The input width and height in pixels.

`cw`
`ch`

The input chroma image width and height in pixels.

`hsub`
`vsub`

The horizontal and vertical chroma subsample values. For example, for the pixel format "yuv422p", *hsub* is 2 and *vsub* is 1.

`luma_power, lp`
`chroma_power, cp`
`alpha_power, ap`

Specify how many times the boxblur filter is applied to the corresponding plane.

Default value for `luma_power` is 2. If not specified, `chroma_power` and `alpha_power` default to the corresponding value set for `luma_power`.

A value of 0 will disable the effect.

### 9.8.1 Examples

- Apply a boxblur filter with the luma, chroma, and alpha radii set to 2:

```
boxblur=luma_radius=2:luma_power=1
boxblur=2:1
```

- Set the luma radius to 2, and alpha and chroma radius to 0:

```
boxblur=2:1:cr=0:ar=0
```

- Set the luma and chroma radii to a fraction of the video dimension:

```
boxblur=luma_radius=min(h\,w)/10:luma_power=1:chroma_radius=min(cw\,ch)/10:chroma_power=1
```

## 9.9 codecview

Visualize information exported by some codecs.

Some codecs can export information through frames using side-data or other means. For example, some MPEG based codecs export motion vectors through the *export_mvs* flag in the codec `flags2` option.

The filter accepts the following option:

`mv`

Set motion vectors to visualize.

Available flags for *mv* are:

‘`pf`’

forward predicted MVs of P-frames

‘`bf`’

forward predicted MVs of B-frames

'bb'

backward predicted MVs of B-frames

### 9.9.1 Examples

- Visualizes multi-directionals MVs from P and B-Frames using `ffplay`:

```
ffplay -flags2 +export_mvs input.mpg -vf codecview=mv=pf+bf+bb
```

## 9.10 colorbalance

Modify intensity of primary colors (red, green and blue) of input frames.

The filter allows an input frame to be adjusted in the shadows, midtones or highlights regions for the red-cyan, green-magenta or blue-yellow balance.

A positive adjustment value shifts the balance towards the primary color, a negative value towards the complementary color.

The filter accepts the following options:

`rs`
`gs`
`bs`

Adjust red, green and blue shadows (darkest pixels).

`rm`
`gm`
`bm`

Adjust red, green and blue midtones (medium pixels).

`rh`
`gh`
`bh`

Adjust red, green and blue highlights (brightest pixels).

Allowed ranges for options are `[-1.0, 1.0]`. Defaults are `0`.

### 9.10.1 Examples

- Add red color cast to shadows:

  ```
  colorbalance=rs=.3
  ```

## 9.11 colorchannelmixer

Adjust video input frames by re-mixing color channels.

This filter modifies a color channel by adding the values associated to the other channels of the same pixels. For example if the value to modify is red, the output value will be:

*red*=*red***rr* + *blue***rb* + *green***rg* + *alpha***ra*

The filter accepts the following options:

`rr`
`rg`
`rb`
`ra`

Adjust contribution of input red, green, blue and alpha channels for output red channel. Default is `1` for *rr*, and `0` for *rg*, *rb* and *ra*.

`gr`
`gg`
`gb`
`ga`

Adjust contribution of input red, green, blue and alpha channels for output green channel. Default is `1` for *gg*, and `0` for *gr*, *gb* and *ga*.

`br`
`bg`
`bb`
`ba`

Adjust contribution of input red, green, blue and alpha channels for output blue channel. Default is `1` for *bb*, and `0` for *br*, *bg* and *ba*.

`ar`
`ag`
`ab`
`aa`

Adjust contribution of input red, green, blue and alpha channels for output alpha channel. Default is `1` for *aa*, and `0` for *ar*, *ag* and *ab*.

Allowed ranges for options are `[-2.0, 2.0]`.

### 9.11.1 Examples

- Convert source to grayscale:

```
colorchannelmixer=.3:.4:.3:0:.3:.4:.3:0:.3:.4:.3
```

- Simulate sepia tones:

```
colorchannelmixer=.393:.769:.189:0:.349:.686:.168:0:.272:.534:.131
```

## 9.12 colormatrix

Convert color matrix.

The filter accepts the following options:

`src`
`dst`

Specify the source and destination color matrix. Both values must be specified.

The accepted values are:

'`bt709`'

BT.709

'`bt601`'

BT.601

'`smpte240m`'

SMPTE-240M

'`fcc`'

FCC

For example to convert from BT.601 to SMPTE-240M, use the command:

```
colormatrix=bt601:smpte240m
```

## 9.13 copy

Copy the input source unchanged to the output. This is mainly useful for testing purposes.

## 9.14 crop

Crop the input video to given dimensions.

It accepts the following parameters:

`w, out_w`

The width of the output video. It defaults to `iw`. This expression is evaluated only once during the filter configuration.

`h, out_h`

The height of the output video. It defaults to `ih`. This expression is evaluated only once during the filter configuration.

`x`

The horizontal position, in the input video, of the left edge of the output video. It defaults to `(in_w-out_w)/2`. This expression is evaluated per-frame.

`y`

The vertical position, in the input video, of the top edge of the output video. It defaults to `(in_h-out_h)/2`. This expression is evaluated per-frame.

`keep_aspect`

If set to 1 will force the output display aspect ratio to be the same of the input, by changing the output sample aspect ratio. It defaults to 0.

The *out_w*, *out_h*, *x*, *y* parameters are expressions containing the following constants:

`x`
`y`

The computed values for *x* and *y*. They are evaluated for each new frame.

`in_w`
`in_h`

The input width and height.

`iw`
`ih`

These are the same as *in_w* and *in_h*.

`out_w`
`out_h`

The output (cropped) width and height.

`ow`
`oh`

These are the same as *out_w* and *out_h*.

`a`

same as *iw* / *ih*

`sar`

input sample aspect ratio

`dar`

input display aspect ratio, it is the same as (*iw* / *ih*) * *sar*

`hsub`
`vsub`

horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

`n`

The number of the input frame, starting from 0.

`pos`

the position in the file of the input frame, NAN if unknown

`t`

The timestamp expressed in seconds. It's NAN if the input timestamp is unknown.

The expression for *out_w* may depend on the value of *out_h*, and the expression for *out_h* may depend on *out_w*, but they cannot depend on *x* and *y*, as *x* and *y* are evaluated after *out_w* and *out_h*.

The *x* and *y* parameters specify the expressions for the position of the top-left corner of the output (non-cropped) area. They are evaluated for each frame. If the evaluated value is not valid, it is approximated to the nearest valid value.

The expression for *x* may depend on *y*, and the expression for *y* may depend on *x*.

### 9.14.1 Examples

- Crop area with size 100x100 at position (12,34).

  ```
  crop=100:100:12:34
  ```

  Using named options, the example above becomes:

  ```
  crop=w=100:h=100:x=12:y=34
  ```

- Crop the central input area with size 100x100:

  ```
  crop=100:100
  ```

- Crop the central input area with size 2/3 of the input video:

  ```
  crop=2/3*in_w:2/3*in_h
  ```

- Crop the input video central square:

  ```
  crop=out_w=in_h
  crop=in_h
  ```

- Delimit the rectangle with the top-left corner placed at position 100:100 and the right-bottom corner corresponding to the right-bottom corner of the input image.

  ```
  crop=in_w-100:in_h-100:100:100
  ```

- Crop 10 pixels from the left and right borders, and 20 pixels from the top and bottom borders

  ```
  crop=in_w-2*10:in_h-2*20
  ```

- Keep only the bottom right quarter of the input image:

  ```
  crop=in_w/2:in_h/2:in_w/2:in_h/2
  ```

- Crop height for getting Greek harmony:

  ```
  crop=in_w:1/PHI*in_w
  ```

- Appply trembling effect:

  ```
  crop=in_w/2:in_h/2:(in_w-out_w)/2+((in_w-out_w)/2)*sin(n/10):(in_h-out_h)/2 +((in_h-out_h)/2)*sin(n/7)
  ```

- Apply erratic camera effect depending on timestamp:

```
crop=in_w/2:in_h/2:(in_w-out_w)/2+((in_w-out_w)/2)*sin(t*10):(in_h-out_h)/2 +((in_h-out_h)/2)*sin(t*13)"
```

- Set x depending on the value of y:

```
crop=in_w/2:in_h/2:y:10+10*sin(n/10)
```

## 9.15 cropdetect

Auto-detect the crop size.

It calculates the necessary cropping parameters and prints the recommended parameters via the logging system. The detected dimensions correspond to the non-black area of the input video.

It accepts the following parameters:

`limit`

Set higher black value threshold, which can be optionally specified from nothing (0) to everything (255). An intensity value greater to the set value is considered non-black. It defaults to 24.

`round`

The value which the width/height should be divisible by. It defaults to 16. The offset is automatically adjusted to center the video. Use 2 to get only even dimensions (needed for 4:2:2 video). 16 is best when encoding to most video codecs.

`reset_count, reset`

Set the counter that determines after how many frames cropdetect will reset the previously detected largest video area and start over to detect the current optimal crop area. Default value is 0.

This can be useful when channel logos distort the video area. 0 indicates 'never reset', and returns the largest area encountered during playback.

## 9.16 curves

Apply color adjustments using curves.

This filter is similar to the Adobe Photoshop and GIMP curves tools. Each component (red, green and blue) has its values defined by *N* key points tied from each other using a smooth curve. The x-axis represents the pixel values from the input frame, and the y-axis the new pixel values to be set for the output frame.

By default, a component curve is defined by the two points *(0;0)* and *(1;1)*. This creates a straight line where each original pixel value is "adjusted" to its own value, which means no change to the image.

The filter allows you to redefine these two points and add some more. A new curve (using a natural cubic spline interpolation) will be define to pass smoothly through all these new coordinates. The new defined points needs to be strictly increasing over the x-axis, and their *x* and *y* values must be in the *[0;1]* interval.

If the computed curves happened to go outside the vector spaces, the values will be clipped accordingly.

If there is no key point defined in `x=0`, the filter will automatically insert a *(0;0)* point. In the same way, if there is no key point defined in `x=1`, the filter will automatically insert a *(1;1)* point.

The filter accepts the following options:

`preset`

> Select one of the available color presets. This option can be used in addition to the `r`, `g`, `b` parameters; in this case, the later options takes priority on the preset values. Available presets are:
>
> ‘`none`’
> ‘`color_negative`’
> ‘`cross_process`’
> ‘`darker`’
> ‘`increase_contrast`’
> ‘`lighter`’
> ‘`linear_contrast`’
> ‘`medium_contrast`’
> ‘`negative`’
> ‘`strong_contrast`’
> ‘`vintage`’
>
> Default is `none`.

`master, m`

> Set the master key points. These points will define a second pass mapping. It is sometimes called a "luminance" or "value" mapping. It can be used with `r`, `g`, `b` or `all` since it acts like a post-processing LUT.

`red, r`

> Set the key points for the red component.

`green, g`

> Set the key points for the green component.

`blue, b`

> Set the key points for the blue component.

`all`

> Set the key points for all components (not including master). Can be used in addition to the other key points component options. In this case, the unset component(s) will fallback on this `all` setting.

`psfile`

Specify a Photoshop curves file (`.asv`) to import the settings from.

To avoid some filtergraph syntax conflicts, each key points list need to be defined using the following syntax: `x0/y0 x1/y1 x2/y2 ....`

### 9.16.1 Examples

- Increase slightly the middle level of blue:

  ```
  curves=blue='0.5/0.58'
  ```

- Vintage effect:

  ```
  curves=r='0/0.11 .42/.51 1/0.95':g='0.50/0.48':b='0/0.22 .49/.44 1/0.8'
  ```

  Here we obtain the following coordinates for each components:

  *red*

  `(0;0.11) (0.42;0.51) (1;0.95)`

  *green*

  `(0;0) (0.50;0.48) (1;1)`

  *blue*

  `(0;0.22) (0.49;0.44) (1;0.80)`

- The previous example can also be achieved with the associated built-in preset:

  ```
  curves=preset=vintage
  ```

- Or simply:

  ```
  curves=vintage
  ```

- Use a Photoshop preset and redefine the points of the green component:

  ```
  curves=psfile='MyCurvesPresets/purple.asv':green='0.45/0.53'
  ```

## 9.17 dctdnoiz

Denoise frames using 2D DCT (frequency domain filtering).

This filter is not designed for real time.

The filter accepts the following options:

`sigma, s`

> Set the noise sigma constant.
>
> This *sigma* defines a hard threshold of `3 * sigma`; every DCT coefficient (absolute value) below this threshold with be dropped.
>
> If you need a more advanced filtering, see `expr`.
>
> Default is `0`.

`overlap`

> Set number overlapping pixels for each block. Since the filter can be slow, you may want to reduce this value, at the cost of a less effective filter and the risk of various artefacts.
>
> If the overlapping value doesn't allow to process the whole input width or height, a warning will be displayed and according borders won't be denoised.
>
> Default value is *blocksize*-1, which is the best possible setting.

`expr, e`

> Set the coefficient factor expression.
>
> For each coefficient of a DCT block, this expression will be evaluated as a multiplier value for the coefficient.
>
> If this is option is set, the `sigma` option will be ignored.
>
> The absolute value of the coefficient can be accessed through the *c* variable.

`n`

> Set the *blocksize* using the number of bits. `1<<`*`n`* defines the *blocksize*, which is the width and height of the processed blocks.
>
> The default value is *3* (8x8) and can be raised to *4* for a *blocksize* of 16x16. Note that changing this setting has huge consequences on the speed processing. Also, a larger block size does not necessarily means a better de-noising.

### 9.17.1 Examples

Apply a denoise with a `sigma` of `4.5`:

```
dctdnoiz=4.5
```

The same operation can be achieved using the expression system:

```
dctdnoiz=e='gte(c, 4.5*3)'
```

Violent denoise using a block size of `16x16`:

```
dctdnoiz=15:n=4
```

## 9.18 decimate

Drop duplicated frames at regular intervals.

The filter accepts the following options:

`cycle`

: Set the number of frames from which one will be dropped. Setting this to *N* means one frame in every batch of *N* frames will be dropped. Default is `5`.

`dupthresh`

: Set the threshold for duplicate detection. If the difference metric for a frame is less than or equal to this value, then it is declared as duplicate. Default is `1.1`

`scthresh`

: Set scene change threshold. Default is `15`.

`blockx`
`blocky`

: Set the size of the x and y-axis blocks used during metric calculations. Larger blocks give better noise suppression, but also give worse detection of small movements. Must be a power of two. Default is `32`.

`ppsrc`

: Mark main input as a pre-processed input and activate clean source input stream. This allows the input to be pre-processed with various filters to help the metrics calculation while keeping the frame selection lossless. When set to `1`, the first stream is for the pre-processed input, and the second stream is the clean source from where the kept frames are chosen. Default is `0`.

`chroma`

: Set whether or not chroma is considered in the metric calculations. Default is `1`.

## 9.19 dejudder

Remove judder produced by partially interlaced telecined content.

Judder can be introduced, for instance, by pullup filter. If the original source was partially telecined content then the output of `pullup,dejudder` will have a variable frame rate. May change the recorded frame rate of the container. Aside from that change, this filter will not affect constant frame rate video.

The option available in this filter is:

`cycle`

> Specify the length of the window over which the judder repeats.
>
> Accepts any integer greater than 1. Useful values are:
>
> ‘`4`’
>
> > If the original was telecined from 24 to 30 fps (Film to NTSC).
>
> ‘`5`’
>
> > If the original was telecined from 25 to 30 fps (PAL to NTSC).
>
> ‘`20`’
>
> > If a mixture of the two.
>
> The default is ‘`4`’.

## 9.20 delogo

Suppress a TV station logo by a simple interpolation of the surrounding pixels. Just set a rectangle covering the logo and watch it disappear (and sometimes something even uglier appear - your mileage may vary).

It accepts the following parameters:

`x`
`y`

> Specify the top left corner coordinates of the logo. They must be specified.

`w`
`h`

> Specify the width and height of the logo to clear. They must be specified.

`band, t`

Specify the thickness of the fuzzy edge of the rectangle (added to *w* and *h*). The default value is 4.

`show`

When set to 1, a green rectangle is drawn on the screen to simplify finding the right *x*, *y*, *w*, and *h* parameters. The default value is 0.

The rectangle is drawn on the outermost pixels which will be (partly) replaced with interpolated values. The values of the next pixels immediately outside this rectangle in each direction will be used to compute the interpolated pixel values inside the rectangle.

### 9.20.1 Examples

- Set a rectangle covering the area with top left corner coordinates 0,0 and size 100x77, and a band of size 10:

  ```
  delogo=x=0:y=0:w=100:h=77:band=10
  ```

## 9.21 deshake

Attempt to fix small changes in horizontal and/or vertical shift. This filter helps remove camera shake from hand-holding a camera, bumping a tripod, moving on a vehicle, etc.

The filter accepts the following options:

`x`
`y`
`w`
`h`

Specify a rectangular area where to limit the search for motion vectors. If desired the search for motion vectors can be limited to a rectangular area of the frame defined by its top left corner, width and height. These parameters have the same meaning as the drawbox filter which can be used to visualise the position of the bounding box.

This is useful when simultaneous movement of subjects within the frame might be confused for camera motion by the motion vector search.

If any or all of *x*, *y*, *w* and *h* are set to -1 then the full frame is used. This allows later options to be set without specifying the bounding box for the motion vector search.

Default - search the whole frame.

`rx`
`ry`

Specify the maximum extent of movement in x and y directions in the range 0-64 pixels. Default 16.

`edge`

Specify how to generate pixels to fill blanks at the edge of the frame. Available values are:

‘`blank, 0`’

Fill zeroes at blank locations

‘`original, 1`’

Original image at blank locations

‘`clamp, 2`’

Extruded edge value at blank locations

‘`mirror, 3`’

Mirrored edge at blank locations

Default value is ‘`mirror`’.

`blocksize`

Specify the blocksize to use for motion search. Range 4-128 pixels, default 8.

`contrast`

Specify the contrast threshold for blocks. Only blocks with more than the specified contrast (difference between darkest and lightest pixels) will be considered. Range 1-255, default 125.

`search`

Specify the search strategy. Available values are:

‘`exhaustive, 0`’

Set exhaustive search

‘`less, 1`’

Set less exhaustive search.

Default value is ‘`exhaustive`’.

`filename`

If set then a detailed log of the motion search is written to the specified file.

`opencl`

If set to 1, specify using OpenCL capabilities, only available if FFmpeg was configured with `--enable-opencl`. Default value is 0.

## 9.22 drawbox

Draw a colored box on the input image.

It accepts the following parameters:

`x`
`y`

The expressions which specify the top left corner coordinates of the box. It defaults to 0.

`width, w`
`height, h`

The expressions which specify the width and height of the box; if 0 they are interpreted as the input width and height. It defaults to 0.

`color, c`

Specify the color of the box to write. For the general syntax of this option, check the "Color" section in the ffmpeg-utils manual. If the special value `invert` is used, the box edge color is the same as the video with inverted luma.

`thickness, t`

The expression which sets the thickness of the box edge. Default value is `3`.

See below for the list of accepted constants.

The parameters for *x*, *y*, *w* and *h* and *t* are expressions containing the following constants:

`dar`

The input display aspect ratio, it is the same as (*w* / *h*) * *sar*.

`hsub`
`vsub`

horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

`in_h, ih`
`in_w, iw`

The input width and height.

`sar`

The input sample aspect ratio.

`x`
`y`

The x and y offset coordinates where the box is drawn.

`w`
`h`

The width and height of the drawn box.

`t`

The thickness of the drawn box.

These constants allow the *x*, *y*, *w*, *h* and *t* expressions to refer to each other, so you may for example specify `y=x/dar` or `h=w/dar`.

### 9.22.1 Examples

- Draw a black box around the edge of the input image:

  ```
  drawbox
  ```

- Draw a box with color red and an opacity of 50%:

  ```
  drawbox=10:20:200:60:red@0.5
  ```

  The previous example can be specified as:

  ```
  drawbox=x=10:y=20:w=200:h=60:color=red@0.5
  ```

- Fill the box with pink color:

  ```
  drawbox=x=10:y=10:w=100:h=100:color=pink@0.5:t=max
  ```

- Draw a 2-pixel red 2.40:1 mask:

  ```
  drawbox=x=-t:y=0.5*(ih-iw/2.4)-t:w=iw+t*2:h=iw/2.4+t*2:t=2:c=red
  ```

## 9.23 drawgrid

Draw a grid on the input image.

It accepts the following parameters:

`x`
`y`

The expressions which specify the coordinates of some point of grid intersection (meant to configure offset). Both default to 0.

`width, w`
`height, h`

The expressions which specify the width and height of the grid cell, if 0 they are interpreted as the input width and height, respectively, minus `thickness`, so image gets framed. Default to 0.

`color, c`

Specify the color of the grid. For the general syntax of this option, check the "Color" section in the ffmpeg-utils manual. If the special value `invert` is used, the grid color is the same as the video with inverted luma.

`thickness, t`

The expression which sets the thickness of the grid line. Default value is `1`.

See below for the list of accepted constants.

The parameters for *x*, *y*, *w* and *h* and *t* are expressions containing the following constants:

`dar`

The input display aspect ratio, it is the same as (*w* / *h*) * *sar*.

`hsub`
`vsub`

horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

`in_h, ih`
`in_w, iw`

The input grid cell width and height.

`sar`

The input sample aspect ratio.

`x`
`y`

The x and y coordinates of some point of grid intersection (meant to configure offset).

`w`
`h`

The width and height of the drawn cell.

`t`

The thickness of the drawn cell.

These constants allow the *x*, *y*, *w*, *h* and *t* expressions to refer to each other, so you may for example specify `y=x/dar` or `h=w/dar`.

### 9.23.1 Examples

- Draw a grid with cell 100x100 pixels, thickness 2 pixels, with color red and an opacity of 50%:

```
drawgrid=width=100:height=100:thickness=2:color=red@0.5
```

- Draw a white 3x3 grid with an opacity of 50%:

```
drawgrid=w=iw/3:h=ih/3:t=2:c=white@0.5
```

## 9.24 drawtext

Draw a text string or text from a specified file on top of a video, using the libfreetype library.

To enable compilation of this filter, you need to configure FFmpeg with `--enable-libfreetype`. To enable default font fallback and the *font* option you need to configure FFmpeg with `--enable-libfontconfig`. To enable the *text_shaping* option, you need to configure FFmpeg with `--enable-libfribidi`.

### 9.24.1 Syntax

It accepts the following parameters:

`box`

Used to draw a box around text using the background color. The value must be either 1 (enable) or 0 (disable). The default value of *box* is 0.

`boxcolor`

The color to be used for drawing box around text. For the syntax of this option, check the "Color" section in the ffmpeg-utils manual.

The default value of *boxcolor* is "white".

`borderw`

Set the width of the border to be drawn around the text using *bordercolor*. The default value of *borderw* is 0.

`bordercolor`

Set the color to be used for drawing border around text. For the syntax of this option, check the "Color" section in the ffmpeg-utils manual.

The default value of *bordercolor* is "black".

`expansion`

Select how the *text* is expanded. Can be either `none`, `strftime` (deprecated) or `normal` (default). See the Text expansion section below for details.

`fix_bounds`

If true, check and fix text coords to avoid clipping.

`fontcolor`

The color to be used for drawing fonts. For the syntax of this option, check the "Color" section in the ffmpeg-utils manual.

The default value of *fontcolor* is "black".

`fontcolor_expr`

String which is expanded the same way as *text* to obtain dynamic *fontcolor* value. By default this option has empty value and is not processed. When this option is set, it overrides *fontcolor* option.

`font`

The font family to be used for drawing text. By default Sans.

`fontfile`

The font file to be used for drawing text. The path must be included. This parameter is mandatory if the fontconfig support is disabled.

`fontsize`

The font size to be used for drawing text. The default value of *fontsize* is 16.

`text_shaping`

If set to 1, attempt to shape the text (for example, reverse the order of right-to-left text and join Arabic characters) before drawing it. Otherwise, just draw the text exactly as given. By default 1 (if supported).

`ft_load_flags`

The flags to be used for loading the fonts.

The flags map the corresponding flags supported by libfreetype, and are a combination of the following values:

*default*
*no_scale*
*no_hinting*
*render*
*no_bitmap*
*vertical_layout*
*force_autohint*
*crop_bitmap*
*pedantic*
*ignore_global_advance_width*
*no_recurse*
*ignore_transform*
*monochrome*
*linear_design*
*no_autohint*

Default value is "default".

For more information consult the documentation for the FT_LOAD_* libfreetype flags.

`shadowcolor`

The color to be used for drawing a shadow behind the drawn text. For the syntax of this option, check the "Color" section in the ffmpeg-utils manual.

The default value of *shadowcolor* is "black".

`shadowx`
`shadowy`

The x and y offsets for the text shadow position with respect to the position of the text. They can be either positive or negative values. The default value for both is "0".

`start_number`

The starting frame number for the n/frame_num variable. The default value is "0".

`tabsize`

The size in number of spaces to use for rendering the tab. Default value is 4.

`timecode`

Set the initial timecode representation in "hh:mm:ss[:;.]ff" format. It can be used with or without text parameter. *timecode_rate* option must be specified.

`timecode_rate, rate, r`

Set the timecode frame rate (timecode only).

`text`

The text string to be drawn. The text must be a sequence of UTF-8 encoded characters. This parameter is mandatory if no file is specified with the parameter *textfile*.

`textfile`

A text file containing text to be drawn. The text must be a sequence of UTF-8 encoded characters.

This parameter is mandatory if no text string is specified with the parameter *text*.

If both *text* and *textfile* are specified, an error is thrown.

`reload`

If set to 1, the *textfile* will be reloaded before each frame. Be sure to update it atomically, or it may be read partially, or even fail.

`x`
`y`

The expressions which specify the offsets where text will be drawn within the video frame. They are relative to the top/left border of the output image.

The default value of *x* and *y* is "0".

See below for the list of accepted constants and functions.

The parameters for *x* and *y* are expressions containing the following constants and functions:

`dar`

input display aspect ratio, it is the same as (*w* / *h*) * *sar*

`hsub`
`vsub`

horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

`line_h, lh`

the height of each text line

`main_h, h, H`

the input height

`main_w, w, W`

the input width

`max_glyph_a, ascent`

the maximum distance from the baseline to the highest/upper grid coordinate used to place a glyph outline point, for all the rendered glyphs. It is a positive value, due to the grid's orientation with the Y axis upwards.

`max_glyph_d, descent`

the maximum distance from the baseline to the lowest grid coordinate used to place a glyph outline point, for all the rendered glyphs. This is a negative value, due to the grid's orientation, with the Y axis upwards.

`max_glyph_h`

maximum glyph height, that is the maximum height for all the glyphs contained in the rendered text, it is equivalent to *ascent* - *descent*.

`max_glyph_w`

maximum glyph width, that is the maximum width for all the glyphs contained in the rendered text

`n`

the number of input frame, starting from 0 `rand(min, max)`

return a random number included between *min* and *max*

`sar`

The input sample aspect ratio.

`t`

timestamp expressed in seconds, NAN if the input timestamp is unknown

`text_h, th`

the height of the rendered text

`text_w, tw`

the width of the rendered text

`x`
`y`

the x and y offset coordinates where the text is drawn.

These parameters allow the *x* and *y* expressions to refer each other, so you can for example specify `y=x/dar`.

### 9.24.2 Text expansion

If `expansion` is set to `strftime`, the filter recognizes strftime() sequences in the provided text and expands them accordingly. Check the documentation of strftime(). This feature is deprecated.

If `expansion` is set to `none`, the text is printed verbatim.

If `expansion` is set to `normal` (which is the default), the following expansion mechanism is used.

The backslash character '\', followed by any character, always expands to the second character.

Sequence of the form `%{...}` are expanded. The text between the braces is a function name, possibly followed by arguments separated by ':'. If the arguments contain special characters or delimiters (':' or '}'), they should be escaped.

Note that they probably must also be escaped as the value for the `text` option in the filter argument string and as the filter argument in the filtergraph description, and possibly also for the shell, that makes up to four levels of escaping; using a text file avoids these problems.

The following functions are available:

`expr, e`

> The expression evaluation result.
>
> It must take one argument specifying the expression to be evaluated, which accepts the same constants and functions as the *x* and *y* values. Note that not all constants should be used, for example the text size is not known when evaluating the expression, so the constants *text_w* and *text_h* will have an undefined value.

`expr_int_format, eif`

> Evaluate the expression's value and output as formatted integer.
>
> The first argument is the expression to be evaluated, just as for the *expr* function. The second argument specifies the output format. Allowed values are 'x', 'X', 'd' and 'u'. They are treated exactly as in the printf function. The third parameter is optional and sets the number of positions taken by the output. It can be used to add padding with zeros from the left.

`gmtime`

> The time at which the filter is running, expressed in UTC. It can accept an argument: a strftime() format string.

`localtime`

> The time at which the filter is running, expressed in the local time zone. It can accept an argument: a strftime() format string.

`metadata`

> Frame metadata. It must take one argument specifying metadata key.

`n, frame_num`

> The frame number, starting from 0.

`pict_type`

> A 1 character description of the current picture type.

`pts`

> The timestamp of the current frame. It can take up to two arguments.
>
> The first argument is the format of the timestamp; it defaults to `flt` for seconds as a decimal number with microsecond accuracy; `hms` stands for a formatted *[-]HH:MM:SS.mmm* timestamp with millisecond accuracy.

The second argument is an offset added to the timestamp.

### 9.24.3 Examples

- Draw "Test Text" with font FreeSerif, using the default values for the optional parameters.

  ```
  drawtext="fontfile=/usr/share/fonts/truetype/freefont/FreeSerif.ttf: text='Test Text'"
  ```

- Draw 'Test Text' with font FreeSerif of size 24 at position x=100 and y=50 (counting from the top-left corner of the screen), text is yellow with a red box around it. Both the text and the box have an opacity of 20%.

  ```
  drawtext="fontfile=/usr/share/fonts/truetype/freefont/FreeSerif.ttf: text='Test Text':\
            x=100: y=50: fontsize=24: fontcolor=yellow@0.2: box=1: boxcolor=red@0.2"
  ```

  Note that the double quotes are not necessary if spaces are not used within the parameter list.

- Show the text at the center of the video frame:

  ```
  drawtext="fontsize=30:fontfile=FreeSerif.ttf:text='hello world':x=(w-text_w)/2:y=(h-text_h-line_h)/2"
  ```

- Show a text line sliding from right to left in the last row of the video frame. The file `LONG_LINE` is assumed to contain a single line with no newlines.

  ```
  drawtext="fontsize=15:fontfile=FreeSerif.ttf:text=LONG_LINE:y=h-line_h:x=-50*t"
  ```

- Show the content of file `CREDITS` off the bottom of the frame and scroll up.

  ```
  drawtext="fontsize=20:fontfile=FreeSerif.ttf:textfile=CREDITS:y=h-20*t"
  ```

- Draw a single green letter "g", at the center of the input video. The glyph baseline is placed at half screen height.

  ```
  drawtext="fontsize=60:fontfile=FreeSerif.ttf:fontcolor=green:text=g:x=(w-max_glyph_w)/2:y=h/2-ascent"
  ```

- Show text for 1 second every 3 seconds:

  ```
  drawtext="fontfile=FreeSerif.ttf:fontcolor=white:x=100:y=x/dar:enable=lt(mod(t\,3)\,1):text='blink'"
  ```

- Use fontconfig to set the font. Note that the colons need to be escaped.

  ```
  drawtext='fontfile=Linux Libertine O-40\:style=Semibold:text=FFmpeg'
  ```

- Print the date of a real-time encoding (see strftime(3)):

  ```
  drawtext='fontfile=FreeSans.ttf:text=%{localtime\:%a %b %d %Y}'
  ```

- Show text fading in and out (appearing/disappearing):

  ```
  #!/bin/sh
  DS=1.0 # display start
  DE=10.0 # display end
  FID=1.5 # fade in duration
  FOD=5 # fade out duration
  ffplay -f lavfi "color,drawtext=text=TEST:fontsize=50:fontfile=FreeSerif.ttf:fontcolor_expr=ff0000%{eif\\\\: clip(255*(1*between(t\\, $DS + $FID\\, $DE - $FOD) + ((t - $DS)/$FID)*between(t\\, $DS\\, $DS + $FID) + (-(t - $DE)/$FOD)*between(t\\, $DE - $FOD\\, $DE) )\\, 0\\, 255) \\\\: x\\\\: 2 }"
  ```

For more information about libfreetype, check: http://www.freetype.org/.

For more information about fontconfig, check: http://freedesktop.org/software/fontconfig/fontconfig-user.html.

For more information about libfribidi, check: http://fribidi.org/.

## 9.25 edgedetect

Detect and draw edges. The filter uses the Canny Edge Detection algorithm.

The filter accepts the following options:

`low`
`high`

> Set low and high threshold values used by the Canny thresholding algorithm.
>
> The high threshold selects the "strong" edge pixels, which are then connected through 8-connectivity with the "weak" edge pixels selected by the low threshold.
>
> *low* and *high* threshold values must be chosen in the range [0,1], and *low* should be lesser or equal to *high*.
>
> Default value for *low* is `20/255`, and default value for *high* is `50/255`.

`mode`

> Define the drawing mode.
>
> ‘`wires`’
>
> > Draw white/gray wires on black background.
>
> ‘`colormix`’
>
> > Mix the colors to create a paint/cartoon effect.
>
> Default value is *wires*.

### 9.25.1 Examples

- Standard edge detection with custom values for the hysteresis thresholding:

  ```
  edgedetect=low=0.1:high=0.4
  ```

- Painting effect without thresholding:

```
edgedetect=mode=colormix:high=0
```

## 9.26 extractplanes

Extract color channel components from input video stream into separate grayscale video streams.

The filter accepts the following option:

`planes`

Set plane(s) to extract.

Available values for planes are:

‘y’
‘u’
‘v’
‘a’
‘r’
‘g’
‘b’

Choosing planes not available in the input will result in an error. That means you cannot select `r`, `g`, `b` planes with `y`, `u`, `v` planes at same time.

### 9.26.1 Examples

- Extract luma, u and v color channel component from input video frame into 3 grayscale outputs:

```
ffmpeg -i video.avi -filter_complex 'extractplanes=y+u+v[y][u][v]' -map '[y]' y.avi -map '[u]' u.avi -map '[v]' v.avi
```

## 9.27 elbg

Apply a posterize effect using the ELBG (Enhanced LBG) algorithm.

For each input image, the filter will compute the optimal mapping from the input to the output given the codebook length, that is the number of distinct output colors.

This filter accepts the following options.

`codebook_length, l`

Set codebook length. The value must be a positive integer, and represents the number of distinct output colors. Default value is 256.

`nb_steps, n`

Set the maximum number of iterations to apply for computing the optimal mapping. The higher the value the better the result and the higher the computation time. Default value is 1.

`seed, s`

Set a random seed, must be an integer included between 0 and UINT32_MAX. If not specified, or if explicitly set to -1, the filter will try to use a good random seed on a best effort basis.

## 9.28 fade

Apply a fade-in/out effect to the input video.

It accepts the following parameters:

`type, t`

The effect type can be either "in" for a fade-in, or "out" for a fade-out effect. Default is `in`.

`start_frame, s`

Specify the number of the frame to start applying the fade effect at. Default is 0.

`nb_frames, n`

The number of frames that the fade effect lasts. At the end of the fade-in effect, the output video will have the same intensity as the input video. At the end of the fade-out transition, the output video will be filled with the selected `color`. Default is 25.

`alpha`

If set to 1, fade only alpha channel, if one exists on the input. Default value is 0.

`start_time, st`

Specify the timestamp (in seconds) of the frame to start to apply the fade effect. If both start_frame and start_time are specified, the fade will start at whichever comes last. Default is 0.

`duration, d`

The number of seconds for which the fade effect has to last. At the end of the fade-in effect the output video will have the same intensity as the input video, at the end of the fade-out transition the output video will be filled with the selected `color`. If both duration and nb_frames are specified, duration is used. Default is 0.

`color, c`

Specify the color of the fade. Default is "black".

### 9.28.1 Examples

- Fade in the first 30 frames of video:

  ```
  fade=in:0:30
  ```

  The command above is equivalent to:

  ```
  fade=t=in:s=0:n=30
  ```

- Fade out the last 45 frames of a 200-frame video:

  ```
  fade=out:155:45
  fade=type=out:start_frame=155:nb_frames=45
  ```

- Fade in the first 25 frames and fade out the last 25 frames of a 1000-frame video:

  ```
  fade=in:0:25, fade=out:975:25
  ```

- Make the first 5 frames yellow, then fade in from frame 5-24:

  ```
  fade=in:5:20:color=yellow
  ```

- Fade in alpha over first 25 frames of video:

  ```
  fade=in:0:25:alpha=1
  ```

- Make the first 5.5 seconds black, then fade in for 0.5 seconds:

  ```
  fade=t=in:st=5.5:d=0.5
  ```

## 9.29 field

Extract a single field from an interlaced image using stride arithmetic to avoid wasting CPU time. The output frames are marked as non-interlaced.

The filter accepts the following options:

`type`

  Specify whether to extract the top (if the value is `0` or `top`) or the bottom field (if the value is `1` or `bottom`).

## 9.30 fieldmatch

Field matching filter for inverse telecine. It is meant to reconstruct the progressive frames from a telecined stream. The filter does not drop duplicated frames, so to achieve a complete inverse telecine `fieldmatch` needs to be followed by a decimation filter such as decimate in the filtergraph.

The separation of the field matching and the decimation is notably motivated by the possibility of inserting a de-interlacing filter fallback between the two. If the source has mixed telecined and real interlaced content, `fieldmatch` will not be able to match fields for the interlaced parts. But these remaining combed frames will be marked as interlaced, and thus can be de-interlaced by a later filter such as yadif before decimation.

In addition to the various configuration options, `fieldmatch` can take an optional second stream, activated through the `ppsrc` option. If enabled, the frames reconstruction will be based on the fields and frames from this second stream. This allows the first input to be pre-processed in order to help the various algorithms of the filter, while keeping the output lossless (assuming the fields are matched properly). Typically, a field-aware denoiser, or brightness/contrast adjustments can help.

Note that this filter uses the same algorithms as TIVTC/TFM (AviSynth project) and VIVTC/VFM (VapourSynth project). The later is a light clone of TFM from which `fieldmatch` is based on. While the semantic and usage are very close, some behaviour and options names can differ.

The decimate filter currently only works for constant frame rate input. Do not use `fieldmatch` and decimate if your input has mixed telecined and progressive content with changing framerate.

The filter accepts the following options:

`order`

Specify the assumed field order of the input stream. Available values are:

‘`auto`’

Auto detect parity (use FFmpeg’s internal parity value).

‘`bff`’

Assume bottom field first.

‘`tff`’

Assume top field first.

Note that it is sometimes recommended not to trust the parity announced by the stream.

Default value is *auto*.

`mode`

Set the matching mode or strategy to use. `pc` mode is the safest in the sense that it won’t risk creating jerkiness due to duplicate frames when possible, but if there are bad edits or blended fields it will end up outputting combed frames when a good match might actually exist. On the other hand, `pcn_ub` mode is the most risky in terms of creating jerkiness, but will almost always find a good frame if there is one. The other values are all somewhere in between `pc` and `pcn_ub` in terms of risking jerkiness and creating duplicate frames versus finding good matches in sections with bad edits, orphaned fields, blended fields, etc.

More details about p/c/n/u/b are available in p/c/n/u/b meaning section.

Available values are:

'`pc`'

    2-way matching (p/c)

'`pc_n`'

    2-way matching, and trying 3rd match if still combed (p/c + n)

'`pc_u`'

    2-way matching, and trying 3rd match (same order) if still combed (p/c + u)

'`pc_n_ub`'

    2-way matching, trying 3rd match if still combed, and trying 4th/5th matches if still combed (p/c + n + u/b)

'`pcn`'

    3-way matching (p/c/n)

'`pcn_ub`'

    3-way matching, and trying 4th/5th matches if all 3 of the original matches are detected as combed (p/c/n + u/b)

The parenthesis at the end indicate the matches that would be used for that mode assuming `order`=*tff* (and `field` on *auto* or *top*).

In terms of speed `pc` mode is by far the fastest and `pcn_ub` is the slowest.

Default value is *pc_n*.

`ppsrc`

Mark the main input stream as a pre-processed input, and enable the secondary input stream as the clean source to pick the fields from. See the filter introduction for more details. It is similar to the `clip2` feature from VFM/TFM.

Default value is `0` (disabled).

`field`

Set the field to match from. It is recommended to set this to the same value as `order` unless you experience matching failures with that setting. In certain circumstances changing the field that is used to match from can have a large impact on matching performance. Available values are:

‘`auto`’

Automatic (same value as `order`).

‘`bottom`’

Match from the bottom field.

‘`top`’

Match from the top field.

Default value is *auto*.

`mchroma`

Set whether or not chroma is included during the match comparisons. In most cases it is recommended to leave this enabled. You should set this to `0` only if your clip has bad chroma problems such as heavy rainbowing or other artifacts. Setting this to `0` could also be used to speed things up at the cost of some accuracy.

Default value is `1`.

`y0`
`y1`

These define an exclusion band which excludes the lines between `y0` and `y1` from being included in the field matching decision. An exclusion band can be used to ignore subtitles, a logo, or other things that may interfere with the matching. `y0` sets the starting scan line and `y1` sets the ending line; all lines in between `y0` and `y1` (including `y0` and `y1`) will be ignored. Setting `y0` and `y1` to the same value will disable the feature. `y0` and `y1` defaults to `0`.

`scthresh`

Set the scene change detection threshold as a percentage of maximum change on the luma plane. Good values are in the `[8.0, 14.0]` range. Scene change detection is only relevant in case `combmatch`=*sc*. The range for `scthresh` is `[0.0, 100.0]`.

Default value is `12.0`.

`combmatch`

When `combatch` is not *none*, `fieldmatch` will take into account the combed scores of matches when deciding what match to use as the final match. Available values are:

‘none’

No final matching based on combed scores.

‘sc’

Combed scores are only used when a scene change is detected.

‘full’

Use combed scores all the time.

Default is *sc*.

combdbg

Force fieldmatch to calculate the combed metrics for certain matches and print them. This setting is known as micout in TFM/VFM vocabulary. Available values are:

‘none’

No forced calculation.

‘pcn’

Force p/c/n calculations.

‘pcnub’

Force p/c/n/u/b calculations.

Default value is *none*.

cthresh

This is the area combing threshold used for combed frame detection. This essentially controls how "strong" or "visible" combing must be to be detected. Larger values mean combing must be more visible and smaller values mean combing can be less visible or strong and still be detected. Valid settings are from -1 (every pixel will be detected as combed) to 255 (no pixel will be detected as combed). This is basically a pixel difference value. A good range is [8, 12].

Default value is 9.

chroma

Sets whether or not chroma is considered in the combed frame decision. Only disable this if your source has chroma problems (rainbowing, etc.) that are causing problems for the combed frame detection with chroma enabled. Actually, using chroma=*0* is usually more reliable, except for the case where there is chroma only combing in the source.

Default value is `0`.

`blockx`
`blocky`

Respectively set the x-axis and y-axis size of the window used during combed frame detection. This has to do with the size of the area in which `combpel` pixels are required to be detected as combed for a frame to be declared combed. See the `combpel` parameter description for more info. Possible values are any number that is a power of 2 starting at 4 and going up to 512.

Default value is `16`.

`combpel`

The number of combed pixels inside any of the `blocky` by `blockx` size blocks on the frame for the frame to be detected as combed. While `cthresh` controls how "visible" the combing must be, this setting controls "how much" combing there must be in any localized area (a window defined by the `blockx` and `blocky` settings) on the frame. Minimum value is `0` and maximum is `blocky x blockx` (at which point no frames will ever be detected as combed). This setting is known as `MI` in TFM/VFM vocabulary.

Default value is `80`.

## 9.30.1 p/c/n/u/b meaning

### 9.30.1.1 p/c/n

We assume the following telecined stream:

```
Top fields:     1 2 2 3 4
Bottom fields:  1 2 3 4 4
```

The numbers correspond to the progressive frame the fields relate to. Here, the first two frames are progressive, the 3rd and 4th are combed, and so on.

When `fieldmatch` is configured to run a matching from bottom (`field`=*bottom*) this is how this input stream get transformed:

```
Input stream:
                T     1 2 2 3 4
                B     1 2 3 4 4   <-- matching reference

Matches:              c c n n c

Output stream:
                T     1 2 3 4 4
                B     1 2 3 4 4
```

As a result of the field matching, we can see that some frames get duplicated. To perform a complete inverse telecine, you need to rely on a decimation filter after this operation. See for instance the decimate filter.

The same operation now matching from top fields (`field`=*top*) looks like this:

```
Input stream:
                T     1 2 2 3 4   <-- matching reference
                B     1 2 3 4 4

Matches:              c c p p c

Output stream:
                T     1 2 2 3 4
                B     1 2 2 3 4
```

In these examples, we can see what *p*, *c* and *n* mean; basically, they refer to the frame and field of the opposite parity:

- *p* matches the field of the opposite parity in the previous frame
- *c* matches the field of the opposite parity in the current frame
- *n* matches the field of the opposite parity in the next frame

### 9.30.1.2 u/b

The *u* and *b* matching are a bit special in the sense that they match from the opposite parity flag. In the following examples, we assume that we are currently matching the 2nd frame (Top:2, bottom:2). According to the match, a 'x' is placed above and below each matched fields.

With bottom matching (`field`=*bottom*):

```
Match:           c         p           n          b          u

                 x       x               x        x          x
  Top          1 2 2     1 2 2       1 2 2      1 2 2      1 2 2
  Bottom       1 2 3     1 2 3       1 2 3      1 2 3      1 2 3
                 x         x           x        x              x

Output frames:
                 2          1          2          2          2
                 2          2          2          1          3
```

With top matching (`field`=*top*):

```
Match:           c         p           n          b          u

                 x         x           x        x              x
  Top          1 2 2     1 2 2       1 2 2      1 2 2      1 2 2
  Bottom       1 2 3     1 2 3       1 2 3      1 2 3      1 2 3
                 x       x               x        x          x

Output frames:
                 2          2          2          1          2
                 2          1          3          2          2
```

### 9.30.2 Examples

Simple IVTC of a top field first telecined stream:

```
fieldmatch=order=tff:combmatch=none, decimate
```

Advanced IVTC, with fallback on yadif for still combed frames:

```
fieldmatch=order=tff:combmatch=full, yadif=deint=interlaced, decimate
```

## 9.31 fieldorder

Transform the field order of the input video.

It accepts the following parameters:

`order`

> The output field order. Valid values are *tff* for top field first or *bff* for bottom field first.

The default value is ‘`tff`’.

The transformation is done by shifting the picture content up or down by one line, and filling the remaining line with appropriate picture content. This method is consistent with most broadcast field order converters.

If the input video is not flagged as being interlaced, or it is already flagged as being of the required output field order, then this filter does not alter the incoming video.

It is very useful when converting to or from PAL DV material, which is bottom field first.

For example:

```
ffmpeg -i in.vob -vf "fieldorder=bff" out.dv
```

## 9.32 fifo

Buffer input images and send them when they are requested.

It is mainly useful when auto-inserted by the libavfilter framework.

It does not take parameters.

## 9.33 format

Convert the input video to one of the specified pixel formats. Libavfilter will try to pick one that is suitable as input to the next filter.

It accepts the following parameters:

`pix_fmts`

A '|'-separated list of pixel format names, such as "pix_fmts=yuv420p|monow|rgb24".

### 9.33.1 Examples

- Convert the input video to the *yuv420p* format

  ```
  format=pix_fmts=yuv420p
  ```

  Convert the input video to any of the formats in the list

  ```
  format=pix_fmts=yuv420p|yuv444p|yuv410p
  ```

## 9.34 fps

Convert the video to specified constant frame rate by duplicating or dropping frames as necessary.

It accepts the following parameters:

`fps`

The desired output frame rate. The default is `25`.

`round`

Rounding method.

Possible values are:

`zero`

zero round towards 0

`inf`

round away from 0

`down`

round towards -infinity

`up`

round towards +infinity

`near`

round to nearest

The default is `near`.

`start_time`

Assume the first PTS should be the given value, in seconds. This allows for padding/trimming at the start of stream. By default, no assumption is made about the first frame's expected PTS, so no padding or trimming is done. For example, this could be set to 0 to pad the beginning with duplicates of the first frame if a video stream starts after the audio stream or to trim any frames with a negative PTS.

Alternatively, the options can be specified as a flat string: *fps*[:*round*].

See also the setpts filter.

### 9.34.1 Examples

- A typical usage in order to set the fps to 25:

  ```
  fps=fps=25
  ```

- Sets the fps to 24, using abbreviation and rounding method to round to nearest:

  ```
  fps=fps=film:round=near
  ```

## 9.35 framepack

Pack two different video streams into a stereoscopic video, setting proper metadata on supported codecs. The two views should have the same size and framerate and processing will stop when the shorter video ends. Please note that you may conveniently adjust view properties with the scale and fps filters.

It accepts the following parameters:

`format`

The desired packing format. Supported values are:

`sbs`

The views are next to each other (default).

`tab`

The views are on top of each other.

`lines`

The views are packed by line.

`columns`

The views are packed by column.

`frameseq`

The views are temporally interleaved.

Some examples:

```
# Convert left and right views into a frame-sequential video
ffmpeg -i LEFT -i RIGHT -filter_complex framepack=frameseq OUTPUT

# Convert views into a side-by-side video with the same output resolution as the input
ffmpeg -i LEFT -i RIGHT -filter_complex [0:v]scale=w=iw/2[left],[1:v]scale=w=iw/2[right],[left][right]framepack=sbs OUTPUT
```

## 9.36 framestep

Select one frame every N-th frame.

This filter accepts the following option:

`step`

Select frame after every `step` frames. Allowed values are positive integers higher than 0. Default value is `1`.

## 9.37 frei0r

Apply a frei0r effect to the input video.

To enable the compilation of this filter, you need to install the frei0r header and configure FFmpeg with `--enable-frei0r`.

It accepts the following parameters:

`filter_name`

The name of the frei0r effect to load. If the environment variable `FREI0R_PATH` is defined, the frei0r effect is searched for in each of the directories specified by the colon-separated list in `FREIOR_PATH`. Otherwise, the standard frei0r paths are searched, in this order: `HOME/.frei0r-1/lib/`, `/usr/local/lib/frei0r-1/`, `/usr/lib/frei0r-1/`.

`filter_params`

A '|'-separated list of parameters to pass to the frei0r effect.

A frei0r effect parameter can be a boolean (its value is either "y" or "n"), a double, a color (specified as *R*/*G*/*B*, where *R*, *G*, and *B* are floating point numbers between 0.0 and 1.0, inclusive) or by a color description specified in the "Color" section in the ffmpeg-utils manual), a position (specified as *X*/*Y*, where *X* and *Y* are floating point numbers) and/or a string.

The number and types of parameters depend on the loaded effect. If an effect parameter is not specified, the default value is set.

### 9.37.1 Examples

- Apply the distort0r effect, setting the first two double parameters:

  ```
  frei0r=filter_name=distort0r:filter_params=0.5|0.01
  ```

- Apply the colordistance effect, taking a color as the first parameter:

  ```
  frei0r=colordistance:0.2/0.3/0.4
  frei0r=colordistance:violet
  frei0r=colordistance:0x112233
  ```

- Apply the perspective effect, specifying the top left and top right image positions:

  ```
  frei0r=perspective:0.2/0.2|0.8/0.2
  ```

For more information, see http://frei0r.dyne.org

## 9.38 geq

The filter accepts the following options:

`lum_expr, lum`

Set the luminance expression.

`cb_expr, cb`

Set the chrominance blue expression.

`cr_expr, cr`

Set the chrominance red expression.

`alpha_expr, a`

Set the alpha expression.

`red_expr, r`

Set the red expression.

`green_expr, g`

Set the green expression.

`blue_expr, b`

Set the blue expression.

The colorspace is selected according to the specified options. If one of the `lum_expr`, `cb_expr`, or `cr_expr` options is specified, the filter will automatically select a YCbCr colorspace. If one of the `red_expr`, `green_expr`, or `blue_expr` options is specified, it will select an RGB colorspace.

If one of the chrominance expression is not defined, it falls back on the other one. If no alpha expression is specified it will evaluate to opaque value. If none of chrominance expressions are specified, they will evaluate to the luminance expression.

The expressions can use the following variables and functions:

`N`

The sequential number of the filtered frame, starting from `0`.

`X`
`Y`

The coordinates of the current sample.

`W`
`H`

The width and height of the image.

`SW`
`SH`

Width and height scale depending on the currently filtered plane. It is the ratio between the corresponding luma plane number of pixels and the current plane ones. E.g. for YUV4:2:0 the values are `1,1` for the luma plane, and `0.5,0.5` for chroma planes.

`T`

Time of the current frame, expressed in seconds.

`p(x, y)`

Return the value of the pixel at location (*x*,*y*) of the current plane.

`lum(x, y)`

Return the value of the pixel at location (*x*,*y*) of the luminance plane.

`cb(x, y)`

Return the value of the pixel at location (*x*,*y*) of the blue-difference chroma plane. Return 0 if there is no such plane.

`cr(x, y)`

Return the value of the pixel at location (*x*,*y*) of the red-difference chroma plane. Return 0 if there is no such plane.

`r(x, y)`
`g(x, y)`
`b(x, y)`

Return the value of the pixel at location (*x*,*y*) of the red/green/blue component. Return 0 if there is no such component.

`alpha(x, y)`

Return the value of the pixel at location (*x*,*y*) of the alpha plane. Return 0 if there is no such plane.

For functions, if *x* and *y* are outside the area, the value will be automatically clipped to the closer edge.

### 9.38.1 Examples

- Flip the image horizontally:

```
geq=p(W-X\,Y)
```

- Generate a bidimensional sine wave, with angle `PI/3` and a wavelength of 100 pixels:

  ```
  geq=128 + 100*sin(2*(PI/100)*(cos(PI/3)*(X-50*T) + sin(PI/3)*Y)):128:128
  ```

- Generate a fancy enigmatic moving light:

  ```
  nullsrc=s=256x256,geq=random(1)/hypot(X-cos(N*0.07)*W/2-W/2\,Y-sin(N*0.09)*H/2-H/2)^2*1000000*sin(N*0.02):128:128
  ```

- Generate a quick emboss effect:

  ```
  format=gray,geq=lum_expr='(p(X,Y)+(256-p(X-4,Y-4)))/2'
  ```

- Modify RGB components depending on pixel position:

  ```
  geq=r='X/W*r(X,Y)':g='(1-X/W)*g(X,Y)':b='(H-Y)/H*b(X,Y)'
  ```

- Create a radial gradient that is the same size as the input (also see the vignette filter):

  ```
  geq=lum=255*gauss((X/W-0.5)*3)*gauss((Y/H-0.5)*3)/gauss(0)/gauss(0),format=gray
  ```

- Create a linear gradient to use as a mask for another filter, then compose with overlay. In this example the video will gradually become more blurry from the top to the bottom of the y-axis as defined by the linear gradient:

  ```
  ffmpeg -i input.mp4 -filter_complex "geq=lum=255*(Y/H),format=gray[grad];[0:v]boxblur=4[blur];[blur][grad]alphamerge[alpha];[0:v][alpha]overlay" output.mp4
  ```

## 9.39 gradfun

Fix the banding artifacts that are sometimes introduced into nearly flat regions by truncation to 8bit color depth. Interpolate the gradients that should go where the bands are, and dither them.

It is designed for playback only. Do not use it prior to lossy compression, because compression tends to lose the dither and bring back the bands.

It accepts the following parameters:

`strength`

  The maximum amount by which the filter will change any one pixel. This is also the threshold for detecting nearly flat regions. Acceptable values range from .51 to 64; the default value is 1.2. Out-of-range values will be clipped to the valid range.

`radius`

  The neighborhood to fit the gradient to. A larger radius makes for smoother gradients, but also prevents the filter from modifying the pixels near detailed regions. Acceptable values are 8-32; the default value is 16. Out-of-range values will be clipped to the valid range.

Alternatively, the options can be specified as a flat string: *strength*[:*radius*]

### 9.39.1 Examples

- Apply the filter with a `3.5` strength and radius of `8`:

  ```
  gradfun=3.5:8
  ```

- Specify radius, omitting the strength (which will fall-back to the default value):

  ```
  gradfun=radius=8
  ```

## 9.40 haldclut

Apply a Hald CLUT to a video stream.

First input is the video stream to process, and second one is the Hald CLUT. The Hald CLUT input can be a simple picture or a complete video stream.

The filter accepts the following options:

`shortest`

> Force termination when the shortest input terminates. Default is `0`.

`repeatlast`

> Continue applying the last CLUT after the end of the stream. A value of `0` disable the filter after the last frame of the CLUT is reached. Default is `1`.

`haldclut` also has the same interpolation options as lut3d (both filters share the same internals).

More information about the Hald CLUT can be found on Eskil Steenberg's website (Hald CLUT author) at http://www.quelsolaar.com/technology/clut.html.

### 9.40.1 Workflow examples

#### 9.40.1.1 Hald CLUT video stream

Generate an identity Hald CLUT stream altered with various effects:

```
ffmpeg -f lavfi -i haldclutsrc=8 -vf "hue=H=2*PI*t:s=sin(2*PI*t)+1, curves=cross_process" -t 10 -c:v ffv1 clut.nut
```

Note: make sure you use a lossless codec.

Then use it with `haldclut` to apply it on some random stream:

```
ffmpeg -f lavfi -i mandelbrot -i clut.nut -filter_complex '[0][1] haldclut' -t 20 mandelclut.mkv
```

The Hald CLUT will be applied to the 10 first seconds (duration of `clut.nut`), then the latest picture of that CLUT stream will be applied to the remaining frames of the `mandelbrot` stream.

#### 9.40.1.2 Hald CLUT with preview

A Hald CLUT is supposed to be a squared image of `Level*Level*Level` by `Level*Level*Level` pixels. For a given Hald CLUT, FFmpeg will select the biggest possible square starting at the top left of the picture. The remaining padding pixels (bottom or right) will be ignored. This area can be used to add a preview of the Hald CLUT.

Typically, the following generated Hald CLUT will be supported by the `haldclut` filter:

```
ffmpeg -f lavfi -i haldclutsrc=8 -vf "
   pad=iw+320 [padded_clut];
   smptebars=s=320x256, split [a][b];
   [padded_clut][a] overlay=W-320:h, curves=color_negative [main];
   [main][b] overlay=W-320" -frames:v 1 clut.png
```

It contains the original and a preview of the effect of the CLUT: SMPTE color bars are displayed on the right-top, and below the same color bars processed by the color changes.

Then, the effect of this Hald CLUT can be visualized with:

```
ffplay input.mkv -vf "movie=clut.png, [in] haldclut"
```

## 9.41 hflip

Flip the input video horizontally.

For example, to horizontally flip the input video with `ffmpeg`:

```
ffmpeg -i in.avi -vf "hflip" out.avi
```

## 9.42 histeq

This filter applies a global color histogram equalization on a per-frame basis.

It can be used to correct video that has a compressed range of pixel intensities. The filter redistributes the pixel intensities to equalize their distribution across the intensity range. It may be viewed as an "automatically adjusting contrast filter". This filter is useful only for correcting degraded or poorly captured source video.

The filter accepts the following options:

`strength`

: Determine the amount of equalization to be applied. As the strength is reduced, the distribution of pixel intensities more-and-more approaches that of the input frame. The value must be a float number in the range [0,1] and defaults to 0.200.

`intensity`

Set the maximum intensity that can generated and scale the output values appropriately. The strength should be set as desired and then the intensity can be limited if needed to avoid washing-out. The value must be a float number in the range [0,1] and defaults to 0.210.

`antibanding`

Set the antibanding level. If enabled the filter will randomly vary the luminance of output pixels by a small amount to avoid banding of the histogram. Possible values are `none`, `weak` or `strong`. It defaults to `none`.

## 9.43 histogram

Compute and draw a color distribution histogram for the input video.

The computed histogram is a representation of the color component distribution in an image.

The filter accepts the following options:

`mode`

Set histogram mode.

It accepts the following values:

‘`levels`’

Standard histogram that displays the color components distribution in an image. Displays color graph for each color component. Shows distribution of the Y, U, V, A or R, G, B components, depending on input format, in the current frame. Below each graph a color component scale meter is shown.

‘`color`’

Displays chroma values (U/V color placement) in a two dimensional graph (which is called a vectorscope). The brighter a pixel in the vectorscope, the more pixels of the input frame correspond to that pixel (i.e., more pixels have this chroma value). The V component is displayed on the horizontal (X) axis, with the leftmost side being V = 0 and the rightmost side being V = 255. The U component is displayed on the vertical (Y) axis, with the top representing U = 0 and the bottom representing U = 255.

The position of a white pixel in the graph corresponds to the chroma value of a pixel of the input clip. The graph can therefore be used to read the hue (color flavor) and the saturation (the dominance of the hue in the color). As the hue of a color changes, it moves around the square. At the center of the square the saturation is zero, which means that the corresponding pixel has no color. If the amount of a specific color is increased (while leaving the other colors unchanged) the saturation increases, and the indicator moves towards the edge of the square.

‘color2’

Chroma values in vectorscope, similar as `color` but actual chroma values are displayed.

‘waveform’

Per row/column color component graph. In row mode, the graph on the left side represents color component value 0 and the right side represents value = 255. In column mode, the top side represents color component value = 0 and bottom side represents value = 255.

Default value is `levels`.

`level_height`

Set height of level in `levels`. Default value is `200`. Allowed range is [50, 2048].

`scale_height`

Set height of color scale in `levels`. Default value is `12`. Allowed range is [0, 40].

`step`

Set step for `waveform` mode. Smaller values are useful to find out how many values of the same luminance are distributed across input rows/columns. Default value is `10`. Allowed range is [1, 255].

`waveform_mode`

Set mode for `waveform`. Can be either `row`, or `column`. Default is `row`.

`waveform_mirror`

Set mirroring mode for `waveform`. `0` means unmirrored, `1` means mirrored. In mirrored mode, higher values will be represented on the left side for `row` mode and at the top for `column` mode. Default is `0` (unmirrored).

`display_mode`

Set display mode for `waveform` and `levels`. It accepts the following values:

‘parade’

Display separate graph for the color components side by side in `row` waveform mode or one below the other in `column` waveform mode for `waveform` histogram mode. For `levels` histogram mode, per color component graphs are placed below each other.

Using this display mode in `waveform` histogram mode makes it easy to spot color casts in the highlights and shadows of an image, by comparing the contours of the top and the bottom graphs of each waveform. Since whites, grays, and blacks are characterized by exactly equal amounts of red, green, and blue, neutral areas of the picture should display three waveforms of roughly equal width/height. If not, the correction is easy to perform by making level adjustments the three waveforms.

'overlay'

Presents information identical to that in the parade, except that the graphs representing color components are superimposed directly over one another.

This display mode in waveform histogram mode makes it easier to spot relative differences or similarities in overlapping areas of the color components that are supposed to be identical, such as neutral whites, grays, or blacks.

Default is parade.

levels_mode

Set mode for levels. Can be either linear, or logarithmic. Default is linear.

### 9.43.1 Examples

- Calculate and draw histogram:

```
ffplay -i input -vf histogram
```

## 9.44 hqdn3d

This is a high precision/quality 3d denoise filter. It aims to reduce image noise, producing smooth images and making still images really still. It should enhance compressibility.

It accepts the following optional parameters:

luma_spatial

A non-negative floating point number which specifies spatial luma strength. It defaults to 4.0.

chroma_spatial

A non-negative floating point number which specifies spatial chroma strength. It defaults to 3.0**luma_spatial*/4.0.

luma_tmp

A floating point number which specifies luma temporal strength. It defaults to 6.0**luma_spatial*/4.0.

chroma_tmp

A floating point number which specifies chroma temporal strength. It defaults to *luma_tmp***chroma_spatial*/*luma_spatial*.

## 9.45 hqx

Apply a high-quality magnification filter designed for pixel art. This filter was originally created by Maxim Stepin.

It accepts the following option:

`n`

Set the scaling dimension: `2` for `hq2x`, `3` for `hq3x` and `4` for `hq4x`. Default is `3`.

## 9.46 hue

Modify the hue and/or the saturation of the input.

It accepts the following parameters:

`h`

Specify the hue angle as a number of degrees. It accepts an expression, and defaults to "0".

`s`

Specify the saturation in the [-10,10] range. It accepts an expression and defaults to "1".

`H`

Specify the hue angle as a number of radians. It accepts an expression, and defaults to "0".

`b`

Specify the brightness in the [-10,10] range. It accepts an expression and defaults to "0".

`h` and `H` are mutually exclusive, and can’t be specified at the same time.

The `b`, `h`, `H` and `s` option values are expressions containing the following constants:

`n`

frame count of the input frame starting from 0

`pts`

presentation timestamp of the input frame expressed in time base units

`r`

frame rate of the input video, NAN if the input frame rate is unknown

`t`

timestamp expressed in seconds, NAN if the input timestamp is unknown

`tb`

time base of the input video

### 9.46.1 Examples

- Set the hue to 90 degrees and the saturation to 1.0:

  ```
  hue=h=90:s=1
  ```

- Same command but expressing the hue in radians:

  ```
  hue=H=PI/2:s=1
  ```

- Rotate hue and make the saturation swing between 0 and 2 over a period of 1 second:

  ```
  hue="H=2*PI*t: s=sin(2*PI*t)+1"
  ```

- Apply a 3 seconds saturation fade-in effect starting at 0:

  ```
  hue="s=min(t/3\,1)"
  ```

  The general fade-in expression can be written as:

  ```
  hue="s=min(0\, max((t-START)/DURATION\, 1))"
  ```

- Apply a 3 seconds saturation fade-out effect starting at 5 seconds:

  ```
  hue="s=max(0\, min(1\, (8-t)/3))"
  ```

  The general fade-out expression can be written as:

  ```
  hue="s=max(0\, min(1\, (START+DURATION-t)/DURATION))"
  ```

### 9.46.2 Commands

This filter supports the following commands:

`b`
`s`
`h`
`H`

Modify the hue and/or the saturation and/or brightness of the input video. The command accepts the same syntax of the corresponding option.

If the specified expression is not valid, it is kept at its current value.

## 9.47 idet

Detect video interlacing type.

This filter tries to detect if the input frames as interlaced, progressive, top or bottom field first. It will also try and detect fields that are repeated between adjacent frames (a sign of telecine).

Single frame detection considers only immediately adjacent frames when classifying each frame. Multiple frame detection incorporates the classification history of previous frames.

The filter will log these metadata values:

`single.current_frame`

Detected type of current frame using single-frame detection. One of: “tff” (top field first), “bff” (bottom field first), “progressive”, or “undetermined”

`single.tff`

Cumulative number of frames detected as top field first using single-frame detection.

`multiple.tff`

Cumulative number of frames detected as top field first using multiple-frame detection.

`single.bff`

Cumulative number of frames detected as bottom field first using single-frame detection.

`multiple.current_frame`

Detected type of current frame using multiple-frame detection. One of: “tff” (top field first), “bff” (bottom field first), “progressive”, or “undetermined”

`multiple.bff`

Cumulative number of frames detected as bottom field first using multiple-frame detection.

`single.progressive`

Cumulative number of frames detected as progressive using single-frame detection.

`multiple.progressive`

Cumulative number of frames detected as progressive using multiple-frame detection.

`single.undetermined`

Cumulative number of frames that could not be classified using single-frame detection.

`multiple.undetermined`

Cumulative number of frames that could not be classified using multiple-frame detection.

`repeated.current_frame`

Which field in the current frame is repeated from the last. One of "neither", "top", or "bottom".

`repeated.neither`

Cumulative number of frames with no repeated field.

`repeated.top`

Cumulative number of frames with the top field repeated from the previous frame's top field.

`repeated.bottom`

Cumulative number of frames with the bottom field repeated from the previous frame's bottom field.

The filter accepts the following options:

`intl_thres`

Set interlacing threshold.

`prog_thres`

Set progressive threshold.

`repeat_thres`

Threshold for repeated field detection.

`half_life`

Number of frames after which a given frame's contribution to the statistics is halved (i.e., it contributes only 0.5 to it's classification). The default of 0 means that all frames seen are given full weight of 1.0 forever.

## 9.48 il

Deinterleave or interleave fields.

This filter allows one to process interlaced images fields without deinterlacing them. Deinterleaving splits the input frame into 2 fields (so called half pictures). Odd lines are moved to the top half of the output image, even lines to the bottom half. You can process (filter) them independently and then re-interleave them.

The filter accepts the following options:

`luma_mode, l`
`chroma_mode, c`
`alpha_mode, a`

Available values for *luma_mode*, *chroma_mode* and *alpha_mode* are:

'`none`'

Do nothing.

'`deinterleave, d`'

Deinterleave fields, placing one above the other.

'`interleave, i`'

Interleave fields. Reverse the effect of deinterleaving.

Default value is `none`.

`luma_swap, ls`
`chroma_swap, cs`
`alpha_swap, as`

Swap luma/chroma/alpha fields. Exchange even & odd lines. Default value is `0`.

## 9.49 interlace

Simple interlacing filter from progressive contents. This interleaves upper (or lower) lines from odd frames with lower (or upper) lines from even frames, halving the frame rate and preserving image height.

```
   Original        Original             New Frame
   Frame 'j'      Frame 'j+1'             (tff)
  ==========      ===========       ==================
    Line 0  -------------------->    Frame 'j' Line 0
    Line 1          Line 1  ---->   Frame 'j+1' Line 1
    Line 2 --------------------->    Frame 'j' Line 2
    Line 3          Line 3  ---->   Frame 'j+1' Line 3
     ...             ...                   ...
New Frame + 1 will be generated by Frame 'j+2' and Frame 'j+3' and so on
```

It accepts the following optional parameters:

`scan`

This determines whether the interlaced frame is taken from the even (tff - default) or odd (bff) lines of the progressive frame.

`lowpass`

Enable (default) or disable the vertical lowpass filter to avoid twitter interlacing and reduce moire patterns.

## 9.50 kerndeint

Deinterlace input video by applying Donald Graft's adaptive kernel deinterling. Work on interlaced parts of a video to produce progressive frames.

The description of the accepted parameters follows.

`thresh`

Set the threshold which affects the filter's tolerance when determining if a pixel line must be processed. It must be an integer in the range [0,255] and defaults to 10. A value of 0 will result in applying the process on every pixels.

`map`

Paint pixels exceeding the threshold value to white if set to 1. Default is 0.

`order`

Set the fields order. Swap fields if set to 1, leave fields alone if 0. Default is 0.

`sharp`

Enable additional sharpening if set to 1. Default is 0.

`twoway`

Enable twoway sharpening if set to 1. Default is 0.

### 9.50.1 Examples

- Apply default values:

  ```
  kerndeint=thresh=10:map=0:order=0:sharp=0:twoway=0
  ```

- Enable additional sharpening:

```
kerndeint=sharp=1
```

- Paint processed pixels in white:

```
kerndeint=map=1
```

## 9.51 lenscorrection

Correct radial lens distortion

This filter can be used to correct for radial distortion as can result from the use of wide angle lenses, and thereby re-rectify the image. To find the right parameters one can use tools available for example as part of opencv or simply trial-and-error. To use opencv use the calibration sample (under samples/cpp) from the opencv sources and extract the k1 and k2 coefficients from the resulting matrix.

Note that effectively the same filter is available in the open-source tools Krita and Digikam from the KDE project.

In contrast to the vignette filter, which can also be used to compensate lens errors, this filter corrects the distortion of the image, whereas vignette corrects the brightness distribution, so you may want to use both filters together in certain cases, though you will have to take care of ordering, i.e. whether vignetting should be applied before or after lens correction.

### 9.51.1 Options

The filter accepts the following options:

`cx`

Relative x-coordinate of the focal point of the image, and thereby the center of the distrortion. This value has a range [0,1] and is expressed as fractions of the image width.

`cy`

Relative y-coordinate of the focal point of the image, and thereby the center of the distrortion. This value has a range [0,1] and is expressed as fractions of the image height.

`k1`

Coefficient of the quadratic correction term. 0.5 means no correction.

`k2`

Coefficient of the double quadratic correction term. 0.5 means no correction.

The formula that generates the correction is:

*r_src* = *r_tgt* * (1 + *k1* * (*r_tgt* / *r_0*)^2 + *k2* * (*r_tgt* / *r_0*)^4)

where *r_0* is halve of the image diagonal and *r_src* and *r_tgt* are the distances from the focal point in the source and target images, respectively.

## 9.52 lut3d

Apply a 3D LUT to an input video.

The filter accepts the following options:

`file`

Set the 3D LUT file name.

Currently supported formats:

‘`3dl`’

AfterEffects

‘`cube`’

Iridas

‘`dat`’

DaVinci

‘`m3d`’

Pandora

`interp`

Select interpolation mode.

Available values are:

‘`nearest`’

Use values from the nearest defined point.

‘`trilinear`’

Interpolate values using the 8 points defining a cube.

‘`tetrahedral`’

Interpolate values using a tetrahedron.

## 9.53 lut, lutrgb, lutyuv

Compute a look-up table for binding each pixel component input value to an output value, and apply it to the input video.

*lutyuv* applies a lookup table to a YUV input video, *lutrgb* to an RGB input video.

These filters accept the following parameters:

`c0`

set first pixel component expression

`c1`

set second pixel component expression

`c2`

set third pixel component expression

`c3`

set fourth pixel component expression, corresponds to the alpha component

`r`

set red component expression

`g`

set green component expression

`b`

set blue component expression

`a`

alpha component expression

`y`

set Y/luminance component expression

`u`

set U/Cb component expression

`v`

set V/Cr component expression

Each of them specifies the expression to use for computing the lookup table for the corresponding pixel component values.

The exact component associated to each of the *c** options depends on the format in input.

The *lut* filter requires either YUV or RGB pixel formats in input, *lutrgb* requires RGB pixel formats in input, and *lutyuv* requires YUV.

The expressions can contain the following constants and functions:

`w`
`h`

The input width and height.

`val`

The input value for the pixel component.

`clipval`

The input value, clipped to the *minval*-*maxval* range.

`maxval`

The maximum value for the pixel component.

`minval`

The minimum value for the pixel component.

`negval`

The negated value for the pixel component value, clipped to the *minval*-*maxval* range; it corresponds to the expression "maxval-clipval+minval".

`clip(val)`

The computed value in *val*, clipped to the *minval*-*maxval* range.

`gammaval(gamma)`

The computed gamma correction value of the pixel component value, clipped to the *minval*-*maxval* range. It corresponds to the expression "pow((clipval-minval)/(maxval-minval)\,*gamma*)*(maxval-minval)+minval"

All expressions default to "val".

### 9.53.1 Examples

- Negate input video:

  ```
  lutrgb="r=maxval+minval-val:g=maxval+minval-val:b=maxval+minval-val"
  lutyuv="y=maxval+minval-val:u=maxval+minval-val:v=maxval+minval-val"
  ```

  The above is the same as:

  ```
  lutrgb="r=negval:g=negval:b=negval"
  lutyuv="y=negval:u=negval:v=negval"
  ```

- Negate luminance:

  ```
  lutyuv=y=negval
  ```

- Remove chroma components, turning the video into a graytone image:

  ```
  lutyuv="u=128:v=128"
  ```

- Apply a luma burning effect:

  ```
  lutyuv="y=2*val"
  ```

- Remove green and blue components:

  ```
  lutrgb="g=0:b=0"
  ```

- Set a constant alpha channel value on input:

  ```
  format=rgba,lutrgb=a="maxval-minval/2"
  ```

- Correct luminance gamma by a factor of 0.5:

  ```
  lutyuv=y=gammaval(0.5)
  ```

- Discard least significant bits of luma:

  ```
  lutyuv=y='bitand(val, 128+64+32)'
  ```

## 9.54 mergeplanes

Merge color channel components from several video streams.

The filter accepts up to 4 input streams, and merge selected input planes to the output video.

This filter accepts the following options:

`mapping`

Set input to output plane mapping. Default is `0`.

The mappings is specified as a bitmap. It should be specified as a hexadecimal number in the form 0xAa[Bb[Cc[Dd]]]. 'Aa' describes the mapping for the first plane of the output stream. 'A' sets the number of the input stream to use (from 0 to 3), and 'a' the plane number of the corresponding input to use (from 0 to 3). The rest of the mappings is similar, 'Bb' describes the mapping for the output stream second plane, 'Cc' describes the mapping for the output stream third plane and 'Dd' describes the mapping for the output stream fourth plane.

`format`

Set output pixel format. Default is `yuva444p`.

### 9.54.1 Examples

- Merge three gray video streams of same width and height into single video stream:

  ```
  [a0][a1][a2]mergeplanes=0x001020:yuv444p
  ```

- Merge 1st yuv444p stream and 2nd gray video stream into yuva444p video stream:

  ```
  [a0][a1]mergeplanes=0x00010210:yuva444p
  ```

- Swap Y and A plane in yuva444p stream:

  ```
  format=yuva444p,mergeplanes=0x03010200:yuva444p
  ```

- Swap U and V plane in yuv420p stream:

  ```
  format=yuv420p,mergeplanes=0x000201:yuv420p
  ```

- Cast a rgb24 clip to yuv444p:

  ```
  format=rgb24,mergeplanes=0x000102:yuv444p
  ```

## 9.55 mcdeint

Apply motion-compensation deinterlacing.

It needs one field per frame as input and must thus be used together with yadif=1/3 or equivalent.

This filter accepts the following options:

`mode`

Set the deinterlacing mode.

It accepts one of the following values:

‘`fast`’
‘`medium`’
‘`slow`’

use iterative motion estimation

‘`extra_slow`’

like ‘`slow`’, but use multiple reference frames.

Default value is ‘`fast`’.

`parity`

Set the picture field parity assumed for the input video. It must be one of the following values:

‘`0, tff`’

assume top field first

‘`1, bff`’

assume bottom field first

Default value is ‘`bff`’.

`qp`

Set per-block quantization parameter (QP) used by the internal encoder.

Higher values should result in a smoother motion vector field but less optimal individual vectors. Default value is 1.

## 9.56 mp

Apply an MPlayer filter to the input video.

This filter provides a wrapper around some of the filters of MPlayer/MEncoder.

This wrapper is considered experimental. Some of the wrapped filters may not work properly and we may drop support for them, as they will be implemented natively into FFmpeg. Thus you should avoid depending on them when writing portable scripts.

The filter accepts the parameters: *filter_name*[:=]*filter_params*

*filter_name* is the name of a supported MPlayer filter, *filter_params* is a string containing the parameters accepted by the named filter.

The list of the currently supported filters follows:

*eq2*
*eq*
*fspp*
*ilpack*
*pp7*
*softpulldown*
*uspp*

The parameter syntax and behavior for the listed filters are the same of the corresponding MPlayer filters. For detailed instructions check the "VIDEO FILTERS" section in the MPlayer manual.

### 9.56.1 Examples

- Adjust gamma, brightness, contrast:

  ```
  mp=eq2=1.0:2:0.5
  ```

See also mplayer(1), http://www.mplayerhq.hu/.

## 9.57 mpdecimate

Drop frames that do not differ greatly from the previous frame in order to reduce frame rate.

The main use of this filter is for very-low-bitrate encoding (e.g. streaming over dialup modem), but it could in theory be used for fixing movies that were inverse-telecined incorrectly.

A description of the accepted options follows.

`max`

> Set the maximum number of consecutive frames which can be dropped (if positive), or the minimum interval between dropped frames (if negative). If the value is 0, the frame is dropped unregarding the number of previous sequentially dropped frames.
>
> Default value is 0.

`hi`
`lo`
`frac`

Set the dropping threshold values.

Values for `hi` and `lo` are for 8x8 pixel blocks and represent actual pixel value differences, so a threshold of 64 corresponds to 1 unit of difference for each pixel, or the same spread out differently over the block.

A frame is a candidate for dropping if no 8x8 blocks differ by more than a threshold of `hi`, and if no more than `frac` blocks (1 meaning the whole image) differ by more than a threshold of `lo`.

Default value for `hi` is 64*12, default value for `lo` is 64*5, and default value for `frac` is 0.33.

## 9.58 negate

Negate input video.

It accepts an integer in input; if non-zero it negates the alpha component (if available). The default value in input is 0.

## 9.59 noformat

Force libavfilter not to use any of the specified pixel formats for the input to the next filter.

It accepts the following parameters:

`pix_fmts`

A ’|’-separated list of pixel format names, such as apix_fmts=yuv420p|monow|rgb24".

### 9.59.1 Examples

- Force libavfilter to use a format different from *yuv420p* for the input to the vflip filter:

```
noformat=pix_fmts=yuv420p,vflip
```

- Convert the input video to any of the formats not contained in the list:

```
noformat=yuv420p|yuv444p|yuv410p
```

## 9.60 noise

Add noise on video input frame.

The filter accepts the following options:

`all_seed`
`c0_seed`
`c1_seed`

`c2_seed`
`c3_seed`

Set noise seed for specific pixel component or all pixel components in case of *all_seed*. Default value is `123457`.

`all_strength, alls`
`c0_strength, c0s`
`c1_strength, c1s`
`c2_strength, c2s`
`c3_strength, c3s`

Set noise strength for specific pixel component or all pixel components in case *all_strength*. Default value is `0`. Allowed range is [0, 100].

`all_flags, allf`
`c0_flags, c0f`
`c1_flags, c1f`
`c2_flags, c2f`
`c3_flags, c3f`

Set pixel component flags or set flags for all components if *all_flags*. Available values for component flags are:

‘`a`’

averaged temporal noise (smoother)

‘`p`’

mix random noise with a (semi)regular pattern

‘`t`’

temporal noise (noise pattern changes between frames)

‘`u`’

uniform noise (gaussian otherwise)

### 9.60.1 Examples

Add temporal and uniform noise to input video:

```
noise=alls=20:allf=t+u
```

## 9.61 null

Pass the video source unchanged to the output.

## 9.62 ocv

Apply a video transform using libopencv.

To enable this filter, install the libopencv library and headers and configure FFmpeg with `--enable-libopencv`.

It accepts the following parameters:

`filter_name`

The name of the libopencv filter to apply.

`filter_params`

The parameters to pass to the libopencv filter. If not specified, the default values are assumed.

Refer to the official libopencv documentation for more precise information: http://docs.opencv.org/master/modules/imgproc/doc/filtering.html

Several libopencv filters are supported; see the following subsections.

### 9.62.1 dilate

Dilate an image by using a specific structuring element. It corresponds to the libopencv function `cvDilate`.

It accepts the parameters: *struct_el*|*nb_iterations*.

*struct_el* represents a structuring element, and has the syntax: *cols*x*rows*+*anchor_x*x*anchor_y*/*shape*

*cols* and *rows* represent the number of columns and rows of the structuring element, *anchor_x* and *anchor_y* the anchor point, and *shape* the shape for the structuring element. *shape* must be "rect", "cross", "ellipse", or "custom".

If the value for *shape* is "custom", it must be followed by a string of the form "=*filename*". The file with name *filename* is assumed to represent a binary image, with each printable character corresponding to a bright pixel. When a custom *shape* is used, *cols* and *rows* are ignored, the number or columns and rows of the read file are assumed instead.

The default value for *struct_el* is "3x3+0x0/rect".

*nb_iterations* specifies the number of times the transform is applied to the image, and defaults to 1.

Some examples:

```
# Use the default values
ocv=dilate

# Dilate using a structuring element with a 5x5 cross, iterating two times
ocv=filter_name=dilate:filter_params=5x5+2x2/cross|2

# Read the shape from the file diamond.shape, iterating two times.
# The file diamond.shape may contain a pattern of characters like this
#   *
#  ***
# *****
#  ***
#   *
# The specified columns and rows are ignored
# but the anchor point coordinates are not
ocv=dilate:0x0+2x2/custom=diamond.shape|2
```

### 9.62.2 erode

Erode an image by using a specific structuring element. It corresponds to the libopencv function `cvErode`.

It accepts the parameters: *struct_el*:*nb_iterations*, with the same syntax and semantics as the dilate filter.

### 9.62.3 smooth

Smooth the input video.

The filter takes the following parameters: *type*|*param1*|*param2*|*param3*|*param4*.

*type* is the type of smooth filter to apply, and must be one of the following values: "blur", "blur_no_scale", "median", "gaussian", or "bilateral". The default value is "gaussian".

The meaning of *param1*, *param2*, *param3*, and *param4* depend on the smooth type. *param1* and *param2* accept integer positive values or 0. *param3* and *param4* accept floating point values.

The default value for *param1* is 3. The default value for the other parameters is 0.

These parameters correspond to the parameters assigned to the libopencv function `cvSmooth`.

## 9.63 overlay

Overlay one video on top of another.

It takes two inputs and has one output. The first input is the "main" video on which the second input is overlayed.

It accepts the following parameters:

A description of the accepted options follows.

`x`
`y`

Set the expression for the x and y coordinates of the overlayed video on the main video. Default value is "0" for both expressions. In case the expression is invalid, it is set to a huge value (meaning that the overlay will not be displayed within the output visible area).

`eof_action`

The action to take when EOF is encountered on the secondary input; it accepts one of the following values:

`repeat`

Repeat the last frame (the default).

`endall`

End both streams.

`pass`

Pass the main input through.

`eval`

Set when the expressions for `x`, and `y` are evaluated.

It accepts the following values:

‘`init`’

only evaluate expressions once during the filter initialization or when a command is processed

‘`frame`’

evaluate expressions for each incoming frame

Default value is ‘`frame`’.

`shortest`

If set to 1, force the output to terminate when the shortest input terminates. Default value is 0.

`format`

Set the format for the output video.

It accepts the following values:

‘`yuv420`’

force YUV420 output

‘`yuv422`’

force YUV422 output

‘`yuv444`’

force YUV444 output

‘`rgb`’

force RGB output

Default value is ‘`yuv420`’.

`rgb` *(deprecated)*

If set to 1, force the filter to accept inputs in the RGB color space. Default value is 0. This option is deprecated, use `format` instead.

`repeatlast`

If set to 1, force the filter to draw the last overlay frame over the main input until the end of the stream. A value of 0 disables this behavior. Default value is 1.

The `x`, and `y` expressions can contain the following parameters.

`main_w, W`
`main_h, H`

The main input width and height.

`overlay_w, w`
`overlay_h, h`

The overlay input width and height.

`x`
`y`

The computed values for *x* and *y*. They are evaluated for each new frame.

`hsub`
`vsub`

horizontal and vertical chroma subsample values of the output format. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

`n`

the number of input frame, starting from 0

`pos`

the position in the file of the input frame, NAN if unknown

`t`

The timestamp, expressed in seconds. It’s NAN if the input timestamp is unknown.

Note that the *n*, *pos*, *t* variables are available only when evaluation is done *per frame*, and will evaluate to NAN when `eval` is set to ‘`init`’.

Be aware that frames are taken from each input video in timestamp order, hence, if their initial timestamps differ, it is a good idea to pass the two inputs through a *setpts=PTS-STARTPTS* filter to have them begin in the same zero timestamp, as the example for the *movie* filter does.

You can chain together more overlays but you should test the efficiency of such approach.

### 9.63.1 Commands

This filter supports the following commands:

`x`
`y`

Modify the x and y of the overlay input. The command accepts the same syntax of the corresponding option.

If the specified expression is not valid, it is kept at its current value.

### 9.63.2 Examples

- Draw the overlay at 10 pixels from the bottom right corner of the main video:

```
overlay=main_w-overlay_w-10:main_h-overlay_h-10
```

Using named options the example above becomes:

```
overlay=x=main_w-overlay_w-10:y=main_h-overlay_h-10
```

- Insert a transparent PNG logo in the bottom left corner of the input, using the `ffmpeg` tool with the `-filter_complex` option:

```
ffmpeg -i input -i logo -filter_complex 'overlay=10:main_h-overlay_h-10' output
```

- Insert 2 different transparent PNG logos (second logo on bottom right corner) using the `ffmpeg` tool:

```
ffmpeg -i input -i logo1 -i logo2 -filter_complex 'overlay=x=10:y=H-h-10,overlay=x=W-w-10:y=H-h-10' output
```

- Add a transparent color layer on top of the main video; `WxH` must specify the size of the main input to the overlay filter:

```
color=color=red@.3:size=WxH [over]; [in][over] overlay [out]
```

- Play an original video and a filtered version (here with the deshake filter) side by side using the `ffplay` tool:

```
ffplay input.avi -vf 'split[a][b]; [a]pad=iw*2:ih[src]; [b]deshake[filt]; [src][filt]overlay=w'
```

The above command is the same as:

```
ffplay input.avi -vf 'split[b], pad=iw*2[src], [b]deshake, [src]overlay=w'
```

- Make a sliding overlay appearing from the left to the right top part of the screen starting since time 2:

```
overlay=x='if(gte(t,2), -w+(t-2)*20, NAN)':y=0
```

- Compose output by putting two input videos side to side:

```
ffmpeg -i left.avi -i right.avi -filter_complex "
nullsrc=size=200x100 [background];
[0:v] setpts=PTS-STARTPTS, scale=100x100 [left];
[1:v] setpts=PTS-STARTPTS, scale=100x100 [right];
[background][left]       overlay=shortest=1       [background+left];
[background+left][right] overlay=shortest=1:x=100 [left+right]
"
```

- Mask 10-20 seconds of a video by applying the delogo filter to a section

```
ffmpeg -i test.avi -codec:v:0 wmv2 -ar 11025 -b:v 9000k
-vf '[in]split[split_main][split_delogo];[split_delogo]trim=start=360:end=371,delogo=0:0:640:480[delogoed];[split_main][delogoed]overlay=eof_action=pass[out]'
masked.avi
```

- Chain several overlays in cascade:

```
nullsrc=s=200x200 [bg];
testsrc=s=100x100, split=4 [in0][in1][in2][in3];
[in0] lutrgb=r=0, [bg]   overlay=0:0     [mid0];
[in1] lutrgb=g=0, [mid0] overlay=100:0   [mid1];
[in2] lutrgb=b=0, [mid1] overlay=0:100   [mid2];
[in3] null,       [mid2] overlay=100:100 [out0]
```

## 9.64 owdenoise

Apply Overcomplete Wavelet denoiser.

The filter accepts the following options:

`depth`

Set depth.

Larger depth values will denoise lower frequency components more, but slow down filtering.

Must be an int in the range 8-16, default is `8`.

`luma_strength, ls`

Set luma strength.

Must be a double value in the range 0-1000, default is `1.0`.

`chroma_strength, cs`

Set chroma strength.

Must be a double value in the range 0-1000, default is `1.0`.

## 9.65 pad

Add paddings to the input image, and place the original input at the provided *x*, *y* coordinates.

It accepts the following parameters:

`width, w`
`height, h`

Specify an expression for the size of the output image with the paddings added. If the value for *width* or *height* is 0, the corresponding input size is used for the output.

The *width* expression can reference the value set by the *height* expression, and vice versa.

The default value of *width* and *height* is 0.

`x`
`y`

Specify the offsets to place the input image at within the padded area, with respect to the top/left border of the output image.

The *x* expression can reference the value set by the *y* expression, and vice versa.

The default value of *x* and *y* is 0.

`color`

Specify the color of the padded area. For the syntax of this option, check the "Color" section in the ffmpeg-utils manual.

The default value of *color* is "black".

The value for the *width*, *height*, *x*, and *y* options are expressions containing the following constants:

`in_w`
`in_h`

The input video width and height.

`iw`
`ih`

These are the same as *in_w* and *in_h*.

`out_w`
`out_h`

The output width and height (the size of the padded area), as specified by the *width* and *height* expressions.

`ow`
`oh`

These are the same as *out_w* and *out_h*.

`x`
`y`

The x and y offsets as specified by the *x* and *y* expressions, or NAN if not yet specified.

`a`

same as *iw* / *ih*

`sar`

input sample aspect ratio

`dar`

input display aspect ratio, it is the same as (*iw* / *ih*) * *sar*

`hsub`
`vsub`

The horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

### 9.65.1 Examples

- Add paddings with the color "violet" to the input video. The output video size is 640x480, and the top-left corner of the input video is placed at column 0, row 40

  ```
  pad=640:480:0:40:violet
  ```

  The example above is equivalent to the following command:

  ```
  pad=width=640:height=480:x=0:y=40:color=violet
  ```

- Pad the input to get an output with dimensions increased by 3/2, and put the input video at the center of the padded area:

  ```
  pad="3/2*iw:3/2*ih:(ow-iw)/2:(oh-ih)/2"
  ```

- Pad the input to get a squared output with size equal to the maximum value between the input width and height, and put the input video at the center of the padded area:

  ```
  pad="max(iw\,ih):ow:(ow-iw)/2:(oh-ih)/2"
  ```

- Pad the input to get a final w/h ratio of 16:9:

  ```
  pad="ih*16/9:ih:(ow-iw)/2:(oh-ih)/2"
  ```

- In case of anamorphic video, in order to set the output display aspect correctly, it is necessary to use *sar* in the expression, according to the relation:

  ```
  (ih * X / ih) * sar = output_dar
  X = output_dar / sar
  ```

  Thus the previous example needs to be modified to:

  ```
  pad="ih*16/9/sar:ih:(ow-iw)/2:(oh-ih)/2"
  ```

- Double the output size and put the input video in the bottom-right corner of the output padded area:

  ```
  pad="2*iw:2*ih:ow-iw:oh-ih"
  ```

## 9.66 perspective

Correct perspective of video not recorded perpendicular to the screen.

A description of the accepted parameters follows.

`x0`
`y0`
`x1`
`y1`
`x2`
`y2`
`x3`
`y3`

> Set coordinates expression for top left, top right, bottom left and bottom right corners. Default values are `0:0:W:0:0:H:W:H` with which perspective will remain unchanged. If the `sense` option is set to `source`, then the specified points will be sent to the corners of the destination. If the `sense` option is set to `destination`, then the corners of the source will be sent to the specified coordinates.
>
> The expressions can use the following variables:
>
> `W`
> `H`
>
> > the width and height of video frame.

`interpolation`

> Set interpolation for perspective correction.
>
> It accepts the following values:
>
> ‘`linear`’
> ‘`cubic`’
>
> Default value is ‘`linear`’.

`sense`

> Set interpretation of coordinate options.
>
> It accepts the following values:
>
> ‘`0, source`’

Send point in the source specified by the given coordinates to the corners of the destination.

'`1, destination`'

Send the corners of the source to the point in the destination specified by the given coordinates.

Default value is '`source`'.

## 9.67 phase

Delay interlaced video by one field time so that the field order changes.

The intended use is to fix PAL movies that have been captured with the opposite field order to the film-to-video transfer.

A description of the accepted parameters follows.

`mode`

Set phase mode.

It accepts the following values:

'`t`'

Capture field order top-first, transfer bottom-first. Filter will delay the bottom field.

'`b`'

Capture field order bottom-first, transfer top-first. Filter will delay the top field.

'`p`'

Capture and transfer with the same field order. This mode only exists for the documentation of the other options to refer to, but if you actually select it, the filter will faithfully do nothing.

'`a`'

Capture field order determined automatically by field flags, transfer opposite. Filter selects among '`t`' and '`b`' modes on a frame by frame basis using field flags. If no field information is available, then this works just like '`u`'.

'`u`'

Capture unknown or varying, transfer opposite. Filter selects among '`t`' and '`b`' on a frame by frame basis by analyzing the images and selecting the alternative that produces best match between the fields.

‘T’

Capture top-first, transfer unknown or varying. Filter selects among ‘t’ and ‘p’ using image analysis.

‘B’

Capture bottom-first, transfer unknown or varying. Filter selects among ‘b’ and ‘p’ using image analysis.

‘A’

Capture determined by field flags, transfer unknown or varying. Filter selects among ‘t’, ‘b’ and ‘p’ using field flags and image analysis. If no field information is available, then this works just like ‘U’. This is the default mode.

‘U’

Both capture and transfer unknown or varying. Filter selects among ‘t’, ‘b’ and ‘p’ using image analysis only.

## 9.68 pixdesctest

Pixel format descriptor test filter, mainly useful for internal testing. The output video should be equal to the input video.

For example:

```
format=monow, pixdesctest
```

can be used to test the monowhite pixel format descriptor definition.

## 9.69 pp

Enable the specified chain of postprocessing subfilters using libpostproc. This library should be automatically selected with a GPL build (`--enable-gpl`). Subfilters must be separated by ’/’ and can be disabled by prepending a ’-’. Each subfilter and some options have a short and a long name that can be used interchangeably, i.e. dr/dering are the same.

The filters accept the following options:

`subfilters`

Set postprocessing subfilters string.

All subfilters share common options to determine their scope:

`a/autoq`

Honor the quality commands for this subfilter.

`c/chrom`

Do chrominance filtering, too (default).

`y/nochrom`

Do luminance filtering only (no chrominance).

`n/noluma`

Do chrominance filtering only (no luminance).

These options can be appended after the subfilter name, separated by a ’|’.

Available subfilters are:

`hb/hdeblock[|difference[|flatness]]`

Horizontal deblocking filter

`difference`

Difference factor where higher values mean more deblocking (default: `32`).

`flatness`

Flatness threshold where lower values mean more deblocking (default: `39`).

`vb/vdeblock[|difference[|flatness]]`

Vertical deblocking filter

`difference`

Difference factor where higher values mean more deblocking (default: `32`).

`flatness`

Flatness threshold where lower values mean more deblocking (default: `39`).

`ha/hadeblock[|difference[|flatness]]`

Accurate horizontal deblocking filter

`difference`

Difference factor where higher values mean more deblocking (default: `32`).

`flatness`

Flatness threshold where lower values mean more deblocking (default: `39`).

`va/vadeblock[|difference[|flatness]]`

Accurate vertical deblocking filter

`difference`

Difference factor where higher values mean more deblocking (default: `32`).

`flatness`

Flatness threshold where lower values mean more deblocking (default: `39`).

The horizontal and vertical deblocking filters share the difference and flatness values so you cannot set different horizontal and vertical thresholds.

`h1/x1hdeblock`

Experimental horizontal deblocking filter

`v1/x1vdeblock`

Experimental vertical deblocking filter

`dr/dering`

Deringing filter

`tn/tmpnoise[|threshold1[|threshold2[|threshold3]]], temporal noise reducer`

`threshold1`

larger -> stronger filtering

`threshold2`

larger -> stronger filtering

`threshold3`

larger -> stronger filtering

`al/autolevels[:f/fullyrange], automatic brightness / contrast correction` `f/fullyrange`

Stretch luminance to `0-255`.

`lb/linblenddeint`

Linear blend deinterlacing filter that deinterlaces the given block by filtering all lines with a `(1 2 1)` filter.

`li/linipoldeint`

Linear interpolating deinterlacing filter that deinterlaces the given block by linearly interpolating every second line.

`ci/cubicipoldeint`

Cubic interpolating deinterlacing filter deinterlaces the given block by cubically interpolating every second line.

`md/mediandeint`

Median deinterlacing filter that deinterlaces the given block by applying a median filter to every second line.

`fd/ffmpegdeint`

FFmpeg deinterlacing filter that deinterlaces the given block by filtering every second line with a `(-1 4 2 4 -1)` filter.

`l5/lowpass5`

Vertically applied FIR lowpass deinterlacing filter that deinterlaces the given block by filtering all lines with a `(-1 2 6 2 -1)` filter.

`fq/forceQuant[|quantizer]`

Overrides the quantizer table from the input with the constant quantizer you specify.

`quantizer`

Quantizer to use

`de/default`

Default pp filter combination (`hb|a,vb|a,dr|a`)

`fa/fast`

Fast pp filter combination (`h1|a,v1|a,dr|a`)

`ac`

High quality pp filter combination (`ha|a|128|7,va|a,dr|a`)

### 9.69.1 Examples

- Apply horizontal and vertical deblocking, deringing and automatic brightness/contrast:

```
pp=hb/vb/dr/al
```

- Apply default filters without brightness/contrast correction:

```
pp=de/-al
```

- Apply default filters and temporal denoiser:

```
pp=default/tmpnoise|1|2|3
```

- Apply deblocking on luminance only, and switch vertical deblocking on or off automatically depending on available CPU time:

```
pp=hb|y/vb|a
```

## 9.70 psnr

Obtain the average, maximum and minimum PSNR (Peak Signal to Noise Ratio) between two input videos.

This filter takes in input two input videos, the first input is considered the "main" source and is passed unchanged to the output. The second input is used as a "reference" video for computing the PSNR.

Both video inputs must have the same resolution and pixel format for this filter to work correctly. Also it assumes that both inputs have the same number of frames, which are compared one by one.

The obtained average PSNR is printed through the logging system.

The filter stores the accumulated MSE (mean squared error) of each frame, and at the end of the processing it is averaged across all frames equally, and the following formula is applied to obtain the PSNR:

```
PSNR = 10*log10(MAX^2/MSE)
```

Where MAX is the average of the maximum values of each component of the image.

The description of the accepted parameters follows.

`stats_file, f`

If specified the filter will use the named file to save the PSNR of each individual frame.

The file printed if *stats_file* is selected, contains a sequence of key/value pairs of the form *key*:*value* for each compared couple of frames.

A description of each shown parameter follows:

`n`

sequential number of the input frame, starting from 1

`mse_avg`

Mean Square Error pixel-by-pixel average difference of the compared frames, averaged over all the image components.

`mse_y, mse_u, mse_v, mse_r, mse_g, mse_g, mse_a`

Mean Square Error pixel-by-pixel average difference of the compared frames for the component specified by the suffix.

`psnr_y, psnr_u, psnr_v, psnr_r, psnr_g, psnr_b, psnr_a`

Peak Signal to Noise ratio of the compared frames for the component specified by the suffix.

For example:

```
movie=ref_movie.mpg, setpts=PTS-STARTPTS [main];
[main][ref] psnr="stats_file=stats.log" [out]
```

On this example the input file being processed is compared with the reference file `ref_movie.mpg`. The PSNR of each individual frame is stored in `stats.log`.

## 9.71 pullup

Pulldown reversal (inverse telecine) filter, capable of handling mixed hard-telecine, 24000/1001 fps progressive, and 30000/1001 fps progressive content.

The pullup filter is designed to take advantage of future context in making its decisions. This filter is stateless in the sense that it does not lock onto a pattern to follow, but it instead looks forward to the following fields in order to identify matches and rebuild progressive frames.

To produce content with an even framerate, insert the fps filter after pullup, use `fps=24000/1001` if the input frame rate is 29.97fps, `fps=24` for 30fps and the (rare) telecined 25fps input.

The filter accepts the following options:

`jl`
`jr`
`jt`
`jb`

These options set the amount of "junk" to ignore at the left, right, top, and bottom of the image, respectively. Left and right are in units of 8 pixels, while top and bottom are in units of 2 lines. The default is 8 pixels on each side.

`sb`

Set the strict breaks. Setting this option to 1 will reduce the chances of filter generating an occasional mismatched frame, but it may also cause an excessive number of frames to be dropped during high motion sequences. Conversely, setting it to `-1` will make filter match fields more easily. This may help processing of video where there is slight blurring between the fields, but may also cause there to be interlaced frames in the output. Default value is `0`.

`mp`

Set the metric plane to use. It accepts the following values:

‘`l`’

Use luma plane.

‘`u`’

Use chroma blue plane.

‘`v`’

Use chroma red plane.

This option may be set to use chroma plane instead of the default luma plane for doing filter’s computations. This may improve accuracy on very clean source material, but more likely will decrease accuracy, especially if there is chroma noise (rainbow effect) or any grayscale video. The main purpose of setting `mp` to a chroma plane is to reduce CPU load and make pullup usable in realtime on slow machines.

For best results (without duplicated frames in the output file) it is necessary to change the output frame rate. For example, to inverse telecine NTSC input:

```
ffmpeg -i input -vf pullup -r 24000/1001 ...
```

## 9.72 removelogo

Suppress a TV station logo, using an image file to determine which pixels comprise the logo. It works by filling in the pixels that comprise the logo with neighboring pixels.

The filter accepts the following options:

`filename, f`

> Set the filter bitmap file, which can be any image format supported by libavformat. The width and height of the image file must match those of the video stream being processed.

Pixels in the provided bitmap image with a value of zero are not considered part of the logo, non-zero pixels are considered part of the logo. If you use white (255) for the logo and black (0) for the rest, you will be safe. For making the filter bitmap, it is recommended to take a screen capture of a black frame with the logo visible, and then using a threshold filter followed by the erode filter once or twice.

If needed, little splotches can be fixed manually. Remember that if logo pixels are not covered, the filter quality will be much reduced. Marking too many pixels as part of the logo does not hurt as much, but it will increase the amount of blurring needed to cover over the image and will destroy more information than necessary, and extra pixels will slow things down on a large logo.

## 9.73 rotate

Rotate video by an arbitrary angle expressed in radians.

The filter accepts the following options:

A description of the optional parameters follows.

`angle, a`

> Set an expression for the angle by which to rotate the input video clockwise, expressed as a number of radians. A negative value will result in a counter-clockwise rotation. By default it is set to "0".
>
> This expression is evaluated for each frame.

`out_w, ow`

> Set the output width expression, default value is "iw". This expression is evaluated just once during configuration.

`out_h, oh`

> Set the output height expression, default value is "ih". This expression is evaluated just once during configuration.

`bilinear`

Enable bilinear interpolation if set to 1, a value of 0 disables it. Default value is 1.

`fillcolor, c`

Set the color used to fill the output area not covered by the rotated image. For the generalsyntax of this option, check the "Color" section in the ffmpeg-utils manual. If the special value "none" is selected then no background is printed (useful for example if the background is never shown).

Default value is "black".

The expressions for the angle and the output size can contain the following constants and functions:

`n`

sequential number of the input frame, starting from 0. It is always NAN before the first frame is filtered.

`t`

time in seconds of the input frame, it is set to 0 when the filter is configured. It is always NAN before the first frame is filtered.

`hsub`
`vsub`

horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

`in_w, iw`
`in_h, ih`

the input video width and height

`out_w, ow`
`out_h, oh`

the output width and height, that is the size of the padded area as specified by the *width* and *height* expressions

`rotw(a)`
`roth(a)`

the minimal width/height required for completely containing the input video rotated by *a* radians.

These are only available when computing the `out_w` and `out_h` expressions.

### 9.73.1 Examples

- Rotate the input by PI/6 radians clockwise:

  ```
  rotate=PI/6
  ```

- Rotate the input by PI/6 radians counter-clockwise:

  ```
  rotate=-PI/6
  ```

- Rotate the input by 45 degrees clockwise:

  ```
  rotate=45*PI/180
  ```

- Apply a constant rotation with period T, starting from an angle of PI/3:

  ```
  rotate=PI/3+2*PI*t/T
  ```

- Make the input video rotation oscillating with a period of T seconds and an amplitude of A radians:

  ```
  rotate=A*sin(2*PI/T*t)
  ```

- Rotate the video, output size is chosen so that the whole rotating input video is always completely contained in the output:

  ```
  rotate='2*PI*t:ow=hypot(iw,ih):oh=ow'
  ```

- Rotate the video, reduce the output size so that no background is ever shown:

  ```
  rotate=2*PI*t:ow='min(iw,ih)/sqrt(2)':oh=ow:c=none
  ```

### 9.73.2 Commands

The filter supports the following commands:

`a, angle`

Set the angle expression. The command accepts the same syntax of the corresponding option.

If the specified expression is not valid, it is kept at its current value.

## 9.74 sab

Apply Shape Adaptive Blur.

The filter accepts the following options:

`luma_radius, lr`

Set luma blur filter strength, must be a value in range 0.1-4.0, default value is 1.0. A greater value will result in a more blurred image, and in slower processing.

`luma_pre_filter_radius, lpfr`

Set luma pre-filter radius, must be a value in the 0.1-2.0 range, default value is 1.0.

`luma_strength, ls`

Set luma maximum difference between pixels to still be considered, must be a value in the 0.1-100.0 range, default value is 1.0.

`chroma_radius, cr`

Set chroma blur filter strength, must be a value in range 0.1-4.0. A greater value will result in a more blurred image, and in slower processing.

`chroma_pre_filter_radius, cpfr`

Set chroma pre-filter radius, must be a value in the 0.1-2.0 range.

`chroma_strength, cs`

Set chroma maximum difference between pixels to still be considered, must be a value in the 0.1-100.0 range.

Each chroma option value, if not explicitly specified, is set to the corresponding luma option value.

## 9.75 scale

Scale (resize) the input video, using the libswscale library.

The scale filter forces the output display aspect ratio to be the same of the input, by changing the output sample aspect ratio.

If the input image format is different from the format requested by the next filter, the scale filter will convert the input to the requested format.

### 9.75.1 Options

The filter accepts the following options, or any of the options supported by the libswscale scaler.

See (ffmpeg-scaler)the ffmpeg-scaler manual for the complete list of scaler options.

`width, w`
`height, h`

Set the output video dimension expression. Default value is the input dimension.

If the value is 0, the input width is used for the output.

If one of the values is -1, the scale filter will use a value that maintains the aspect ratio of the input image, calculated from the other specified dimension. If both of them are -1, the input size is used

If one of the values is -n with n > 1, the scale filter will also use a value that maintains the aspect ratio of the input image, calculated from the other specified dimension. After that it will, however, make sure that the calculated dimension is divisible by n and adjust the value if necessary.

See below for the list of accepted constants for use in the dimension expression.

`interl`

Set the interlacing mode. It accepts the following values:

‘`1`’

Force interlaced aware scaling.

‘`0`’

Do not apply interlaced scaling.

‘`-1`’

Select interlaced aware scaling depending on whether the source frames are flagged as interlaced or not.

Default value is ‘`0`’.

`flags`

Set libswscale scaling flags. See (ffmpeg-scaler)the ffmpeg-scaler manual for the complete list of values. If not explicitly specified the filter applies the default flags.

`size, s`

Set the video size. For the syntax of this option, check the "Video size" section in the ffmpeg-utils manual.

`in_color_matrix`
`out_color_matrix`

Set in/output YCbCr color space type.

This allows the autodetected value to be overridden as well as allows forcing a specific value used for the output and encoder.

If not specified, the color space type depends on the pixel format.

Possible values:

‘`auto`’

Choose automatically.

‘`bt709`’

Format conforming to International Telecommunication Union (ITU) Recommendation BT.709.

‘`fcc`’

Set color space conforming to the United States Federal Communications Commission (FCC) Code of Federal Regulations (CFR) Title 47 (2003) 73.682 (a).

‘`bt601`’

Set color space conforming to:

- ITU Radiocommunication Sector (ITU-R) Recommendation BT.601
- ITU-R Rec. BT.470-6 (1998) Systems B, B1, and G
- Society of Motion Picture and Television Engineers (SMPTE) ST 170:2004

‘`smpte240m`’

Set color space conforming to SMPTE ST 240:1999.

`in_range`
`out_range`

Set in/output YCbCr sample range.

This allows the autodetected value to be overridden as well as allows forcing a specific value used for the output and encoder. If not specified, the range depends on the pixel format. Possible values:

‘`auto`’

Choose automatically.

‘`jpeg/full/pc`’

Set full range (0-255 in case of 8-bit luma).

‘`mpeg/tv`’

Set "MPEG" range (16-235 in case of 8-bit luma).

`force_original_aspect_ratio`

Enable decreasing or increasing output video width or height if necessary to keep the original aspect ratio. Possible values:

‘`disable`’

Scale the video as specified and disable this feature.

‘`decrease`’

The output video dimensions will automatically be decreased if needed.

‘`increase`’

The output video dimensions will automatically be increased if needed.

One useful instance of this option is that when you know a specific device’s maximum allowed resolution, you can use this to limit the output video to that, while retaining the aspect ratio. For example, device A allows 1280x720 playback, and your video is 1920x800. Using this option (set it to decrease) and specifying 1280x720 to the command line makes the output 1280x533.

Please note that this is a different thing than specifying `-1` for `w` or `h`, you still need to specify the output resolution for this option to work.

The values of the `w` and `h` options are expressions containing the following constants:

*in_w*
*in_h*

The input width and height

*iw*
*ih*

These are the same as *in_w* and *in_h*.

*out_w*
*out_h*

The output (scaled) width and height

*ow*
*oh*

These are the same as *out_w* and *out_h*

*a*

The same as *iw* / *ih*

*sar*

input sample aspect ratio

*dar*

The input display aspect ratio. Calculated from `(iw / ih) * sar`.

*hsub*
*vsub*

horizontal and vertical input chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

*ohsub*
*ovsub*

horizontal and vertical output chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

## 9.75.2 Examples

- Scale the input video to a size of 200x100

  ```
  scale=w=200:h=100
  ```

  This is equivalent to:

  ```
  scale=200:100
  ```

  or:

  ```
  scale=200x100
  ```

- Specify a size abbreviation for the output size:

  ```
  scale=qcif
  ```

  which can also be written as:

  ```
  scale=size=qcif
  ```

- Scale the input to 2x:

  ```
  scale=w=2*iw:h=2*ih
  ```

- The above is the same as:

```
scale=2*in_w:2*in_h
```

- Scale the input to 2x with forced interlaced scaling:

```
scale=2*iw:2*ih:interl=1
```

- Scale the input to half size:

```
scale=w=iw/2:h=ih/2
```

- Increase the width, and set the height to the same size:

```
scale=3/2*iw:ow
```

- Seek Greek harmony:

```
scale=iw:1/PHI*iw
scale=ih*PHI:ih
```

- Increase the height, and set the width to 3/2 of the height:

```
scale=w=3/2*oh:h=3/5*ih
```

- Increase the size, making the size a multiple of the chroma subsample values:

```
scale="trunc(3/2*iw/hsub)*hsub:trunc(3/2*ih/vsub)*vsub"
```

- Increase the width to a maximum of 500 pixels, keeping the same aspect ratio as the input:

```
scale=w='min(500\, iw*3/2):h=-1'
```

## 9.76 separatefields

The `separatefields` takes a frame-based video input and splits each frame into its components fields, producing a new half height clip with twice the frame rate and twice the frame count.

This filter use field-dominance information in frame to decide which of each pair of fields to place first in the output. If it gets it wrong use setfield filter before `separatefields` filter.

## 9.77 setdar, setsar

The `setdar` filter sets the Display Aspect Ratio for the filter output video.

This is done by changing the specified Sample (aka Pixel) Aspect Ratio, according to the following equation:

```
DAR = HORIZONTAL_RESOLUTION / VERTICAL_RESOLUTION * SAR
```

Keep in mind that the `setdar` filter does not modify the pixel dimensions of the video frame. Also, the display aspect ratio set by this filter may be changed by later filters in the filterchain, e.g. in case of scaling or if another "setdar" or a "setsar" filter is applied.

The `setsar` filter sets the Sample (aka Pixel) Aspect Ratio for the filter output video.

Note that as a consequence of the application of this filter, the output display aspect ratio will change according to the equation above.

Keep in mind that the sample aspect ratio set by the `setsar` filter may be changed by later filters in the filterchain, e.g. if another "setsar" or a "setdar" filter is applied.

It accepts the following parameters:

`r, ratio, dar (setdar only), sar (setsar only)`

> Set the aspect ratio used by the filter.
>
> The parameter can be a floating point number string, an expression, or a string of the form *num*:*den*, where *num* and *den* are the numerator and denominator of the aspect ratio. If the parameter is not specified, it is assumed the value "0". In case the form "*num*:*den*" is used, the `:` character should be escaped.

`max`

> Set the maximum integer value to use for expressing numerator and denominator when reducing the expressed aspect ratio to a rational. Default value is `100`.

The parameter *sar* is an expression containing the following constants:

`E, PI, PHI`

> These are approximated values for the mathematical constants e (Euler's number), pi (Greek pi), and phi (the golden ratio).

`w, h`

> The input width and height.

`a`

> These are the same as *w* / *h*.

`sar`

> The input sample aspect ratio.

`dar`

> The input display aspect ratio. It is the same as (*w* / *h*) * *sar*.

`hsub, vsub`

Horizontal and vertical chroma subsample values. For example, for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

### 9.77.1 Examples

- To change the display aspect ratio to 16:9, specify one of the following:

  ```
  setdar=dar=1.77777
  setdar=dar=16/9
  setdar=dar=1.77777
  ```

- To change the sample aspect ratio to 10:11, specify:

  ```
  setsar=sar=10/11
  ```

- To set a display aspect ratio of 16:9, and specify a maximum integer value of 1000 in the aspect ratio reduction, use the command:

  ```
  setdar=ratio=16/9:max=1000
  ```

## 9.78 setfield

Force field for the output video frame.

The `setfield` filter marks the interlace type field for the output frames. It does not change the input frame, but only sets the corresponding property, which affects how the frame is treated by following filters (e.g. `fieldorder` or `yadif`).

The filter accepts the following options:

`mode`

Available values are:

‘`auto`’

Keep the same field property.

‘`bff`’

Mark the frame as bottom-field-first.

‘`tff`’

Mark the frame as top-field-first.

‘`prog`’

Mark the frame as progressive.

## 9.79 showinfo

Show a line containing various information for each input video frame. The input video is not modified.

The shown line contains a sequence of key/value pairs of the form *key*:*value*.

The following values are shown in the output:

`n`

The (sequential) number of the input frame, starting from 0.

`pts`

The Presentation TimeStamp of the input frame, expressed as a number of time base units. The time base unit depends on the filter input pad.

`pts_time`

The Presentation TimeStamp of the input frame, expressed as a number of seconds.

`pos`

The position of the frame in the input stream, or -1 if this information is unavailable and/or meaningless (for example in case of synthetic video).

`fmt`

The pixel format name.

`sar`

The sample aspect ratio of the input frame, expressed in the form *num*/*den*.

`s`

The size of the input frame. For the syntax of this option, check the "Video size" section in the ffmpeg-utils manual.

`i`

The type of interlaced mode ("P" for "progressive", "T" for top field first, "B" for bottom field first).

`iskey`

This is 1 if the frame is a key frame, 0 otherwise.

`type`

The picture type of the input frame ("I" for an I-frame, "P" for a P-frame, "B" for a B-frame, or "?" for an unknown type). Also refer to the documentation of the `AVPictureType` enum and of the `av_get_picture_type_char` function defined in `libavutil/avutil.h`.

`checksum`

The Adler-32 checksum (printed in hexadecimal) of all the planes of the input frame.

`plane_checksum`

The Adler-32 checksum (printed in hexadecimal) of each plane of the input frame, expressed in the form "[*c0 c1 c2 c3*]".

## 9.80 shuffleplanes

Reorder and/or duplicate video planes.

It accepts the following parameters:

`map0`

The index of the input plane to be used as the first output plane.

`map1`

The index of the input plane to be used as the second output plane.

`map2`

The index of the input plane to be used as the third output plane.

`map3`

The index of the input plane to be used as the fourth output plane.

The first plane has the index 0. The default is to keep the input unchanged.

Swap the second and third planes of the input:

```
ffmpeg -i INPUT -vf shuffleplanes=0:2:1:3 OUTPUT
```

## 9.81 signalstats

Evaluate various visual metrics that assist in determining issues associated with the digitization of analog video media.

By default the filter will log these metadata values:

`YMIN`

Display the minimal Y value contained within the input frame. Expressed in range of [0-255].

`YLOW`

Display the Y value at the 10% percentile within the input frame. Expressed in range of [0-255].

`YAVG`

Display the average Y value within the input frame. Expressed in range of [0-255].

`YHIGH`

Display the Y value at the 90% percentile within the input frame. Expressed in range of [0-255].

`YMAX`

Display the maximum Y value contained within the input frame. Expressed in range of [0-255].

`UMIN`

Display the minimal U value contained within the input frame. Expressed in range of [0-255].

`ULOW`

Display the U value at the 10% percentile within the input frame. Expressed in range of [0-255].

`UAVG`

Display the average U value within the input frame. Expressed in range of [0-255].

`UHIGH`

Display the U value at the 90% percentile within the input frame. Expressed in range of [0-255].

`UMAX`

Display the maximum U value contained within the input frame. Expressed in range of [0-255].

`VMIN`

Display the minimal V value contained within the input frame. Expressed in range of [0-255].

`VLOW`

Display the V value at the 10% percentile within the input frame. Expressed in range of [0-255].

`VAVG`

Display the average V value within the input frame. Expressed in range of [0-255].

`VHIGH`

Display the V value at the 90% percentile within the input frame. Expressed in range of [0-255].

`VMAX`

Display the maximum V value contained within the input frame. Expressed in range of [0-255].

`SATMIN`

Display the minimal saturation value contained within the input frame. Expressed in range of [0-~181.02].

`SATLOW`

Display the saturation value at the 10% percentile within the input frame. Expressed in range of [0-~181.02].

`SATAVG`

Display the average saturation value within the input frame. Expressed in range of [0-~181.02].

`SATHIGH`

Display the saturation value at the 90% percentile within the input frame. Expressed in range of [0-~181.02].

`SATMAX`

Display the maximum saturation value contained within the input frame. Expressed in range of [0-~181.02].

`HUEMED`

Display the median value for hue within the input frame. Expressed in range of [0-360].

`HUEAVG`

Display the average value for hue within the input frame. Expressed in range of [0-360].

`YDIF`

Display the average of sample value difference between all values of the Y plane in the current frame and corresponding values of the previous input frame. Expressed in range of [0-255].

`UDIF`

Display the average of sample value difference between all values of the U plane in the current frame and corresponding values of the previous input frame. Expressed in range of [0-255].

`VDIF`

Display the average of sample value difference between all values of the V plane in the current frame and corresponding values of the previous input frame. Expressed in range of [0-255].

The filter accepts the following options:

`stat`
`out`

`stat` specify an additional form of image analysis. `out` output video with the specified type of pixel highlighted.

Both options accept the following values:

'`tout`'

Identify *temporal outliers* pixels. A *temporal outlier* is a pixel unlike the neighboring pixels of the same field. Examples of temporal outliers include the results of video dropouts, head clogs, or tape tracking issues.

'`vrep`'

Identify *vertical line repetition*. Vertical line repetition includes similar rows of pixels within a frame. In born-digital video vertical line repetition is common, but this pattern is uncommon in video digitized from an analog source. When it occurs in video that results from the digitization of an analog source it can indicate concealment from a dropout compensator.

'`brng`'

Identify pixels that fall outside of legal broadcast range.

`color, c`

Set the highlight color for the `out` option. The default color is yellow.

### 9.81.1 Examples

- Output data of various video metrics:

```
ffprobe -f lavfi movie=example.mov,signalstats="stat=tout+vrep+brng" -show_frames
```

- Output specific data about the minimum and maximum values of the Y plane per frame:

```
ffprobe -f lavfi movie=example.mov,signalstats -show_entries frame_tags=lavfi.signalstats.YMAX,lavfi.signalstats.YMIN
```

- Playback video while highlighting pixels that are outside of broadcast range in red.

```
ffplay example.mov -vf signalstats="out=brng:color=red"
```

- Playback video with signalstats metadata drawn over the frame.

```
ffplay example.mov -vf signalstats=stat=brng+vrep+tout,drawtext=fontfile=FreeSerif.ttf:textfile=signalstat_drawtext.txt
```

  The contents of signalstat_drawtext.txt used in the command are:

```
time %{pts:hms}
Y (%{metadata:lavfi.signalstats.YMIN}-%{metadata:lavfi.signalstats.YMAX})
U (%{metadata:lavfi.signalstats.UMIN}-%{metadata:lavfi.signalstats.UMAX})
V (%{metadata:lavfi.signalstats.VMIN}-%{metadata:lavfi.signalstats.VMAX})
saturation maximum: %{metadata:lavfi.signalstats.SATMAX}
```

## 9.82 smartblur

Blur the input video without impacting the outlines.

It accepts the following options:

`luma_radius, lr`

Set the luma radius. The option value must be a float number in the range [0.1,5.0] that specifies the variance of the gaussian filter used to blur the image (slower if larger). Default value is 1.0.

`luma_strength, ls`

Set the luma strength. The option value must be a float number in the range [-1.0,1.0] that configures the blurring. A value included in [0.0,1.0] will blur the image whereas a value included in [-1.0,0.0] will sharpen the image. Default value is 1.0.

`luma_threshold, lt`

Set the luma threshold used as a coefficient to determine whether a pixel should be blurred or not. The option value must be an integer in the range [-30,30]. A value of 0 will filter all the image, a value included in [0,30] will filter flat areas and a value included in [-30,0] will filter edges. Default value is 0.

`chroma_radius, cr`

> Set the chroma radius. The option value must be a float number in the range [0.1,5.0] that specifies the variance of the gaussian filter used to blur the image (slower if larger). Default value is 1.0.

`chroma_strength, cs`

> Set the chroma strength. The option value must be a float number in the range [-1.0,1.0] that configures the blurring. A value included in [0.0,1.0] will blur the image whereas a value included in [-1.0,0.0] will sharpen the image. Default value is 1.0.

`chroma_threshold, ct`

> Set the chroma threshold used as a coefficient to determine whether a pixel should be blurred or not. The option value must be an integer in the range [-30,30]. A value of 0 will filter all the image, a value included in [0,30] will filter flat areas and a value included in [-30,0] will filter edges. Default value is 0.

If a chroma option is not explicitly set, the corresponding luma value is set.

## 9.83 stereo3d

Convert between different stereoscopic image formats.

The filters accept the following options:

`in`

> Set stereoscopic image format of input.
>
> Available values for input image formats are:
>
> ‘`sbsl`’
>
> > side by side parallel (left eye left, right eye right)
>
> ‘`sbsr`’
>
> > side by side crosseye (right eye left, left eye right)
>
> ‘`sbs2l`’
>
> > side by side parallel with half width resolution (left eye left, right eye right)
>
> ‘`sbs2r`’
>
> > side by side crosseye with half width resolution (right eye left, left eye right)

‘`abl`’

above-below (left eye above, right eye below)

‘`abr`’

above-below (right eye above, left eye below)

‘`ab2l`’

above-below with half height resolution (left eye above, right eye below)

‘`ab2r`’

above-below with half height resolution (right eye above, left eye below)

‘`al`’

alternating frames (left eye first, right eye second)

‘`ar`’

alternating frames (right eye first, left eye second)

Default value is ‘`sbsl`’.

`out`

Set stereoscopic image format of output.

Available values for output image formats are all the input formats as well as:

‘`arbg`’

anaglyph red/blue gray (red filter on left eye, blue filter on right eye)

‘`argg`’

anaglyph red/green gray (red filter on left eye, green filter on right eye)

‘`arcg`’

anaglyph red/cyan gray (red filter on left eye, cyan filter on right eye)

‘`arch`’

anaglyph red/cyan half colored (red filter on left eye, cyan filter on right eye)

‘`arcc`’

anaglyph red/cyan color (red filter on left eye, cyan filter on right eye)

'arcd'

anaglyph red/cyan color optimized with the least squares projection of dubois (red filter on left eye, cyan filter on right eye)

'agmg'

anaglyph green/magenta gray (green filter on left eye, magenta filter on right eye)

'agmh'

anaglyph green/magenta half colored (green filter on left eye, magenta filter on right eye)

'agmc'

anaglyph green/magenta colored (green filter on left eye, magenta filter on right eye)

'agmd'

anaglyph green/magenta color optimized with the least squares projection of dubois (green filter on left eye, magenta filter on right eye)

'aybg'

anaglyph yellow/blue gray (yellow filter on left eye, blue filter on right eye)

'aybh'

anaglyph yellow/blue half colored (yellow filter on left eye, blue filter on right eye)

'aybc'

anaglyph yellow/blue colored (yellow filter on left eye, blue filter on right eye)

'aybd'

anaglyph yellow/blue color optimized with the least squares projection of dubois (yellow filter on left eye, blue filter on right eye)

'irl'

interleaved rows (left eye has top row, right eye starts on next row)

'irr'

interleaved rows (right eye has top row, left eye starts on next row) ‘ml’

mono output (left eye only)

‘mr’

mono output (right eye only)

Default value is ‘arcd’.

### 9.83.1 Examples

- Convert input video from side by side parallel to anaglyph yellow/blue dubois:

```
stereo3d=sbsl:aybd
```

- Convert input video from above bellow (left eye above, right eye below) to side by side crosseye.

```
stereo3d=abl:sbsr
```

## 9.84 spp

Apply a simple postprocessing filter that compresses and decompresses the image at several (or - in the case of quality level 6 - all) shifts and average the results.

The filter accepts the following options:

quality

Set quality. This option defines the number of levels for averaging. It accepts an integer in the range 0-6. If set to 0, the filter will have no effect. A value of 6 means the higher quality. For each increment of that value the speed drops by a factor of approximately 2. Default value is 3.

qp

Force a constant quantization parameter. If not set, the filter will use the QP from the video stream (if available).

mode

Set thresholding mode. Available modes are:

‘hard’

Set hard thresholding (default).

‘soft’

Set soft thresholding (better de-ringing effect, but likely blurrier).

`use_bframe_qp`

Enable the use of the QP from the B-Frames if set to `1`. Using this option may cause flicker since the B-Frames have often larger QP. Default is `0` (not enabled).

## 9.85 subtitles

Draw subtitles on top of input video using the libass library.

To enable compilation of this filter you need to configure FFmpeg with `--enable-libass`. This filter also requires a build with libavcodec and libavformat to convert the passed subtitles file to ASS (Advanced Substation Alpha) subtitles format.

The filter accepts the following options:

`filename, f`

Set the filename of the subtitle file to read. It must be specified.

`original_size`

Specify the size of the original video, the video for which the ASS file was composed. For the syntax of this option, check the "Video size" section in the ffmpeg-utils manual. Due to a misdesign in ASS aspect ratio arithmetic, this is necessary to correctly scale the fonts if the aspect ratio has been changed.

`charenc`

Set subtitles input character encoding. `subtitles` filter only. Only useful if not UTF-8.

`stream_index, si`

Set subtitles stream index. `subtitles` filter only.

If the first key is not specified, it is assumed that the first value specifies the `filename`.

For example, to render the file `sub.srt` on top of the input video, use the command:

```
subtitles=sub.srt
```

which is equivalent to:

```
subtitles=filename=sub.srt
```

To render the default subtitles stream from file `video.mkv`, use:

```
subtitles=video.mkv
```

To render the second subtitles stream from that file, use:

```
subtitles=video.mkv:si=1
```

## 9.86 super2xsai

Scale the input by 2x and smooth using the Super2xSaI (Scale and Interpolate) pixel art scaling algorithm.

Useful for enlarging pixel art images without reducing sharpness.

## 9.87 swapuv

Swap U & V plane.

## 9.88 telecine

Apply telecine process to the video.

This filter accepts the following options:

`first_field`

: ‘`top, t`’

top field first

‘`bottom, b`’

bottom field first The default value is `top`.

`pattern`

A string of numbers representing the pulldown pattern you wish to apply. The default value is `23`.

```
Some typical patterns:

NTSC output (30i):
27.5p: 32222
24p: 23 (classic)
24p: 2332 (preferred)
20p: 33
18p: 334
16p: 3444

PAL output (25i):
27.5p: 12222
24p: 222222222223 ("Euro pulldown")
16.67p: 33
16p: 33333334
```

## 9.89 thumbnail

Select the most representative frame in a given sequence of consecutive frames.

The filter accepts the following options:

`n`

> Set the frames batch size to analyze; in a set of *n* frames, the filter will pick one of them, and then handle the next batch of *n* frames until the end. Default is `100`.

Since the filter keeps track of the whole frames sequence, a bigger *n* value will result in a higher memory usage, so a high value is not recommended.

### 9.89.1 Examples

- Extract one picture each 50 frames:

  ```
  thumbnail=50
  ```

- Complete example of a thumbnail creation with `ffmpeg`:

  ```
  ffmpeg -i in.avi -vf thumbnail,scale=300:200 -frames:v 1 out.png
  ```

## 9.90 tile

Tile several successive frames together.

The filter accepts the following options:

`layout`

> Set the grid size (i.e. the number of lines and columns). For the syntax of this option, check the "Video size" section in the ffmpeg-utils manual.

`nb_frames`

> Set the maximum number of frames to render in the given area. It must be less than or equal to *w*x*h*. The default value is `0`, meaning all the area will be used.

`margin`

> Set the outer border margin in pixels.

`padding`

> Set the inner border thickness (i.e. the number of pixels between frames). For more advanced padding options (such as having different values for the edges), refer to the pad video filter.

`color`

Specify the color of the unused areaFor the syntax of this option, check the "Color" section in the ffmpeg-utils manual. The default value of *color* is "black".

### 9.90.1 Examples

- Produce 8x8 PNG tiles of all keyframes (`-skip_frame nokey`) in a movie:

  ```
  ffmpeg -skip_frame nokey -i file.avi -vf 'scale=128:72,tile=8x8' -an -vsync 0 keyframes%03d.png
  ```

  The `-vsync 0` is necessary to prevent `ffmpeg` from duplicating each output frame to accommodate the originally detected frame rate.

- Display `5` pictures in an area of `3x2` frames, with `7` pixels between them, and `2` pixels of initial margin, using mixed flat and named options:

  ```
  tile=3x2:nb_frames=5:padding=7:margin=2
  ```

## 9.91 tinterlace

Perform various types of temporal field interlacing.

Frames are counted starting from 1, so the first input frame is considered odd.

The filter accepts the following options:

`mode`

Specify the mode of the interlacing. This option can also be specified as a value alone. See below for a list of values for this option.

Available values are:

'`merge, 0`'

Move odd frames into the upper field, even into the lower field, generating a double height frame at half frame rate.

'`drop_odd, 1`'

Only output even frames, odd frames are dropped, generating a frame with unchanged height at half frame rate.

'`drop_even, 2`'

Only output odd frames, even frames are dropped, generating a frame with unchanged height at half frame rate.

'pad, 3'

Expand each frame to full height, but pad alternate lines with black, generating a frame with double height at the same input frame rate.

'interleave_top, 4'

Interleave the upper field from odd frames with the lower field from even frames, generating a frame with unchanged height at half frame rate.

'interleave_bottom, 5'

Interleave the lower field from odd frames with the upper field from even frames, generating a frame with unchanged height at half frame rate.

'interlacex2, 6'

Double frame rate with unchanged height. Frames are inserted each containing the second temporal field from the previous input frame and the first temporal field from the next input frame. This mode relies on the top_field_first flag. Useful for interlaced video displays with no field synchronisation.

Numeric values are deprecated but are accepted for backward compatibility reasons.

Default mode is `merge`.

`flags`

Specify flags influencing the filter process.

Available value for *flags* is:

`low_pass_filter, vlfp`

Enable vertical low-pass filtering in the filter. Vertical low-pass filtering is required when creating an interlaced destination from a progressive source which contains high-frequency vertical detail. Filtering will reduce interlace 'twitter' and Moire patterning.

Vertical low-pass filtering can only be enabled for `mode` *interleave_top* and *interleave_bottom*.

## 9.92 transpose

Transpose rows with columns in the input video and optionally flip it.

It accepts the following parameters:

`dir`

Specify the transposition direction.

Can assume the following values:

'`0, 4, cclock_flip`'

Rotate by 90 degrees counterclockwise and vertically flip (default), that is:

```
L.R     L.l
. . ->  . .
l.r     R.r
```

'`1, 5, clock`'

Rotate by 90 degrees clockwise, that is:

```
L.R     l.L
. . ->  . .
l.r     r.R
```

'`2, 6, cclock`'

Rotate by 90 degrees counterclockwise, that is:

```
L.R     R.r
. . ->  . .
l.r     L.l
```

'`3, 7, clock_flip`'

Rotate by 90 degrees clockwise and vertically flip, that is:

```
L.R     r.R
. . ->  . .
l.r     l.L
```

For values between 4-7, the transposition is only done if the input video geometry is portrait and not landscape. These values are deprecated, the `passthrough` option should be used instead.

Numerical values are deprecated, and should be dropped in favor of symbolic constants.

`passthrough`

Do not apply the transposition if the input geometry matches the one specified by the specified value. It accepts the following values:

'`none`'

Always apply transposition.

'`portrait`'

Preserve portrait geometry (when *height* >= *width*).

'`landscape`'

Preserve landscape geometry (when *width* >= *height*).

Default value is `none`.

For example to rotate by 90 degrees clockwise and preserve portrait layout:

```
transpose=dir=1:passthrough=portrait
```

The command above can also be specified as:

```
transpose=1:portrait
```

## 9.93 trim

Trim the input so that the output contains one continuous subpart of the input.

It accepts the following parameters:

`start`

Specify the time of the start of the kept section, i.e. the frame with the timestamp *start* will be the first frame in the output.

`end`

Specify the time of the first frame that will be dropped, i.e. the frame immediately preceding the one with the timestamp *end* will be the last frame in the output.

`start_pts`

This is the same as *start*, except this option sets the start timestamp in timebase units instead of seconds.

`end_pts`

This is the same as *end*, except this option sets the end timestamp in timebase units instead of seconds.

`duration`

The maximum duration of the output in seconds.

`start_frame`

The number of the first frame that should be passed to the output.

`end_frame`

The number of the first frame that should be dropped.

`start`, `end`, and `duration` are expressed as time duration specifications; see (ffmpeg-utils)the Time duration section in the ffmpeg-utils(1) manual for the accepted syntax.

Note that the first two sets of the start/end options and the `duration` option look at the frame timestamp, while the _frame variants simply count the frames that pass through the filter. Also note that this filter does not modify the timestamps. If you wish for the output timestamps to start at zero, insert a setpts filter after the trim filter.

If multiple start or end options are set, this filter tries to be greedy and keep all the frames that match at least one of the specified constraints. To keep only the part that matches all the constraints at once, chain multiple trim filters.

The defaults are such that all the input is kept. So it is possible to set e.g. just the end values to keep everything before the specified time.

Examples:

- Drop everything except the second minute of input:

```
ffmpeg -i INPUT -vf trim=60:120
```

- Keep only the first second:

```
ffmpeg -i INPUT -vf trim=duration=1
```

## 9.94 unsharp

Sharpen or blur the input video.

It accepts the following parameters:

`luma_msize_x, lx`

Set the luma matrix horizontal size. It must be an odd integer between 3 and 63. The default value is 5.

`luma_msize_y, ly`

Set the luma matrix vertical size. It must be an odd integer between 3 and 63. The default value is 5.

`luma_amount, la`

Set the luma effect strength. It must be a floating point number, reasonable values lay between -1.5 and 1.5.

Negative values will blur the input video, while positive values will sharpen it, a value of zero will disable the effect.

Default value is 1.0.

`chroma_msize_x, cx`

Set the chroma matrix horizontal size. It must be an odd integer between 3 and 63. The default value is 5.

`chroma_msize_y, cy`

Set the chroma matrix vertical size. It must be an odd integer between 3 and 63. The default value is 5.

`chroma_amount, ca`

Set the chroma effect strength. It must be a floating point number, reasonable values lay between -1.5 and 1.5.

Negative values will blur the input video, while positive values will sharpen it, a value of zero will disable the effect.

Default value is 0.0.

`opencl`

If set to 1, specify using OpenCL capabilities, only available if FFmpeg was configured with `--enable-opencl`. Default value is 0.

All parameters are optional and default to the equivalent of the string '5:5:1.0:5:5:0.0'.

### 9.94.1 Examples

- Apply strong luma sharpen effect:

```
unsharp=luma_msize_x=7:luma_msize_y=7:luma_amount=2.5
```

- Apply a strong blur of both luma and chroma parameters:

```
unsharp=7:7:-2:7:7:-2
```

## 9.95 vidstabdetect

Analyze video stabilization/deshaking. Perform pass 1 of 2, see vidstabtransform for pass 2.

This filter generates a file with relative translation and rotation transform information about subsequent frames, which is then used by the vidstabtransform filter.

To enable compilation of this filter you need to configure FFmpeg with `--enable-libvidstab`.

This filter accepts the following options:

`result`

> Set the path to the file used to write the transforms information. Default value is `transforms.trf`.

`shakiness`

> Set how shaky the video is and how quick the camera is. It accepts an integer in the range 1-10, a value of 1 means little shakiness, a value of 10 means strong shakiness. Default value is 5.

`accuracy`

> Set the accuracy of the detection process. It must be a value in the range 1-15. A value of 1 means low accuracy, a value of 15 means high accuracy. Default value is 15.

`stepsize`

> Set stepsize of the search process. The region around minimum is scanned with 1 pixel resolution. Default value is 6.

`mincontrast`

> Set minimum contrast. Below this value a local measurement field is discarded. Must be a floating point value in the range 0-1. Default value is 0.3.

`tripod`

> Set reference frame number for tripod mode.
>
> If enabled, the motion of the frames is compared to a reference frame in the filtered stream, identified by the specified number. The idea is to compensate all movements in a more-or-less static scene and keep the camera view absolutely still.
>
> If set to 0, it is disabled. The frames are counted starting from 1.

`show`

Show fields and transforms in the resulting frames. It accepts an integer in the range 0-2. Default value is 0, which disables any visualization.

### 9.95.1 Examples

- Use default values:

```
vidstabdetect
```

- Analyze strongly shaky movie and put the results in file `mytransforms.trf`:

```
vidstabdetect=shakiness=10:accuracy=15:result="mytransforms.trf"
```

- Visualize the result of internal transformations in the resulting video:

```
vidstabdetect=show=1
```

- Analyze a video with medium shakiness using `ffmpeg`:

```
ffmpeg -i input -vf vidstabdetect=shakiness=5:show=1 dummy.avi
```

## 9.96 vidstabtransform

Video stabilization/deshaking: pass 2 of 2, see vidstabdetect for pass 1.

Read a file with transform information for each frame and apply/compensate them. Together with the vidstabdetect filter this can be used to deshake videos. See also http://public.hronopik.de/vid.stab. It is important to also use the unsharp filter, see below.

To enable compilation of this filter you need to configure FFmpeg with `--enable-libvidstab`.

### 9.96.1 Options

`input`

Set path to the file used to read the transforms. Default value is `transforms.trf`.

`smoothing`

Set the number of frames (value*2 + 1) used for lowpass filtering the camera movements. Default value is 10.

For example a number of 10 means that 21 frames are used (10 in the past and 10 in the future) to smoothen the motion in the video. A larger value leads to a smoother video, but limits the acceleration of the camera (pan/tilt movements). 0 is a special case where a static camera is simulated.

`optalgo`

Set the camera path optimization algorithm.

Accepted values are:

'`gauss`'

gaussian kernel low-pass filter on camera motion (default)

'`avg`'

averaging on transformations

`maxshift`

Set maximal number of pixels to translate frames. Default value is -1, meaning no limit.

`maxangle`

Set maximal angle in radians (degree*PI/180) to rotate frames. Default value is -1, meaning no limit.

`crop`

Specify how to deal with borders that may be visible due to movement compensation.

Available values are:

'`keep`'

keep image information from previous frame (default)

'`black`'

fill the border black

`invert`

Invert transforms if set to 1. Default value is 0.

`relative`

Consider transforms as relative to previous frame if set to 1, absolute if set to 0. Default value is 0.

`zoom`

Set percentage to zoom. A positive value will result in a zoom-in effect, a negative value in a zoom-out effect. Default value is 0 (no zoom).

`optzoom`

Set optimal zooming to avoid borders.

Accepted values are:

‘`0`’

disabled

‘`1`’

optimal static zoom value is determined (only very strong movements will lead to visible borders) (default)

‘`2`’

optimal adaptive zoom value is determined (no borders will be visible), see `zoomspeed`

Note that the value given at zoom is added to the one calculated here.

`zoomspeed`

Set percent to zoom maximally each frame (enabled when `optzoom` is set to 2). Range is from 0 to 5, default value is 0.25.

`interpol`

Specify type of interpolation.

Available values are:

‘`no`’

no interpolation

‘`linear`’

linear only horizontal

‘`bilinear`’

linear in both directions (default)

‘`bicubic`’

cubic in both directions (slow)

`tripod`

Enable virtual tripod mode if set to 1, which is equivalent to `relative=0:smoothing=0`. Default value is 0.

Use also `tripod` option of vidstabdetect.

`debug`

Increase log verbosity if set to 1. Also the detected global motions are written to the temporary file `global_motions.trf`. Default value is 0.

### 9.96.2 Examples

- Use `ffmpeg` for a typical stabilization with default values:

  ```
  ffmpeg -i inp.mpeg -vf vidstabtransform,unsharp=5:5:0.8:3:3:0.4 inp_stabilized.mpeg
  ```

  Note the use of the unsharp filter which is always recommended.

- Zoom in a bit more and load transform data from a given file:

  ```
  vidstabtransform=zoom=5:input="mytransforms.trf"
  ```

- Smoothen the video even more:

  ```
  vidstabtransform=smoothing=30
  ```

## 9.97 vflip

Flip the input video vertically.

For example, to vertically flip a video with `ffmpeg`:

```
ffmpeg -i in.avi -vf "vflip" out.avi
```

## 9.98 vignette

Make or reverse a natural vignetting effect.

The filter accepts the following options:

`angle, a`

Set lens angle expression as a number of radians.

The value is clipped in the `[0,PI/2]` range.

Default value: `"PI/5"`

`x0`
`y0`

Set center coordinates expressions. Respectively `"w/2"` and `"h/2"` by default.

`mode`

Set forward/backward mode.

Available modes are:

‘`forward`’

The larger the distance from the central point, the darker the image becomes.

‘`backward`’

The larger the distance from the central point, the brighter the image becomes. This can be used to reverse a vignette effect, though there is no automatic detection to extract the lens `angle` and other settings (yet). It can also be used to create a burning effect.

Default value is ‘`forward`’.

`eval`

Set evaluation mode for the expressions (`angle`, `x0`, `y0`).

It accepts the following values:

‘`init`’

Evaluate expressions only once during the filter initialization.

‘`frame`’

Evaluate expressions for each incoming frame. This is way slower than the ‘`init`’ mode since it requires all the scalers to be re-computed, but it allows advanced dynamic expressions.

Default value is ‘`init`’.

`dither`

Set dithering to reduce the circular banding effects. Default is `1` (enabled).

`aspect`

Set vignette aspect. This setting allows one to adjust the shape of the vignette. Setting this value to the SAR of the input will make a rectangular vignetting following the dimensions of the video.

Default is `1/1`.

### 9.98.1 Expressions

The `alpha`, `x0` and `y0` expressions can contain the following parameters.

`w`
`h`

input width and height

`n`

the number of input frame, starting from 0

`pts`

the PTS (Presentation TimeStamp) time of the filtered video frame, expressed in *TB* units, NAN if undefined

`r`

frame rate of the input video, NAN if the input frame rate is unknown

`t`

the PTS (Presentation TimeStamp) of the filtered video frame, expressed in seconds, NAN if undefined

`tb`

time base of the input video

### 9.98.2 Examples

- Apply simple strong vignetting effect:

  ```
  vignette=PI/4
  ```

- Make a flickering vignetting:

  ```
  vignette='PI/4+random(1)*PI/50':eval=frame
  ```

## 9.99 w3fdif

Deinterlace the input video ("w3fdif" stands for "Weston 3 Field Deinterlacing Filter").

Based on the process described by Martin Weston for BBC R&D, and implemented based on the de-interlace algorithm written by Jim Easterbrook for BBC R&D, the Weston 3 field deinterlacing filter uses filter coefficients calculated by BBC R&D.

There are two sets of filter coefficients, so called "simple": and "complex". Which set of filter coefficients is used can be set by passing an optional parameter:

`filter`

Set the interlacing filter coefficients. Accepts one of the following values:

‘`simple`’

Simple filter coefficient set.

‘`complex`’

More-complex filter coefficient set.

Default value is ‘`complex`’.

`deint`

Specify which frames to deinterlace. Accept one of the following values:

‘`all`’

Deinterlace all frames,

‘`interlaced`’

Only deinterlace frames marked as interlaced.

Default value is ‘`all`’.

## 9.100 xbr

Apply the xBR high-quality magnification filter which is designed for pixel art. It follows a set of edge-detection rules, see http://www.libretro.com/forums/viewtopic.php?f=6&t=134.

It accepts the following option:

`n`

Set the scaling dimension: `2` for `2xBR`, `3` for `3xBR` and `4` for `4xBR`. Default is `3`.

## 9.101 yadif

Deinterlace the input video ("yadif" means "yet another deinterlacing filter").

It accepts the following parameters:

`mode`

The interlacing mode to adopt. It accepts one of the following values:

`0, send_frame`

Output one frame for each frame.

`1, send_field`

Output one frame for each field.

`2, send_frame_nospatial`

Like `send_frame`, but it skips the spatial interlacing check.

`3, send_field_nospatial`

Like `send_field`, but it skips the spatial interlacing check.

The default value is `send_frame`.

`parity`

The picture field parity assumed for the input interlaced video. It accepts one of the following values:

`0, tff`

Assume the top field is first.

`1, bff`

Assume the bottom field is first.

`-1, auto`

Enable automatic detection of field parity.

The default value is `auto`. If the interlacing is unknown or the decoder does not export this information, top field first will be assumed.

`deint`

Specify which frames to deinterlace. Accept one of the following values:

`0, all`

Deinterlace all frames.

`1, interlaced`

Only deinterlace frames marked as interlaced.

The default value is `all`.

## 9.102 zoompan

Apply Zoom & Pan effect.

This filter accepts the following options:

`zoom, z`

Set the zoom expression. Default is 1.

`x`
`y`

Set the x and y expression. Default is 0.

`d`

Set the duration expression in number of frames. This sets for how many number of frames effect will last for single input image.

`s`

Set the output image size, default is 'hd720'.

Each expression can contain the following constants:

`in_w, iw`

Input width.

`in_h, ih`

Input height.

`out_w, ow`

Output width.

`out_h, oh`

Output height.

`in`

Input frame count.

`on`

Output frame count.

`x`
`y`

Last calculated ’x’ and ’y’ position from ’x’ and ’y’ expression for current input frame.

`px`
`py`

’x’ and ’y’ of last output frame of previous input frame or 0 when there was not yet such frame (first input frame).

`zoom`

Last calculated zoom from ’z’ expression for current input frame.

`pzoom`

Last calculated zoom of last output frame of previous input frame.

`duration`

Number of output frames for current input frame. Calculated from ’d’ expression for each input frame.

`pduration`

number of output frames created for previous input frame

`a`

Rational number: input width / input height

`sar`

sample aspect ratio

`dar`

display aspect ratio

### 9.102.1 Examples

- Zoom-in up to 1.5 and pan at same time to some spot near center of picture:

```
zoompan=z='min(zoom+0.0015,1.5)':d=700:x='if(gte(zoom,1.5),x,x+1/a)':y='if(gte(zoom,1.5),y,y+1)':s=640x360
```

# 10 Video Sources

Below is a description of the currently available video sources.

## 10.1 buffer

Buffer video frames, and make them available to the filter chain.

This source is mainly intended for a programmatic use, in particular through the interface defined in `libavfilter/vsrc_buffer.h`.

It accepts the following parameters:

`video_size`

Specify the size (width and height) of the buffered video frames. For the syntax of this option, check the "Video size" section in the ffmpeg-utils manual.

`width`

The input video width.

`height`

The input video height.

`pix_fmt`

A string representing the pixel format of the buffered video frames. It may be a number corresponding to a pixel format, or a pixel format name.

`time_base`

Specify the timebase assumed by the timestamps of the buffered frames.

`frame_rate`

Specify the frame rate expected for the video stream.

`pixel_aspect, sar`

The sample (pixel) aspect ratio of the input video.

`sws_param`

Specify the optional parameters to be used for the scale filter which is automatically inserted when an input change is detected in the input size or format.

For example:

```
buffer=width=320:height=240:pix_fmt=yuv410p:time_base=1/24:sar=1
```

will instruct the source to accept video frames with size 320x240 and with format "yuv410p", assuming 1/24 as the timestamps timebase and square pixels (1:1 sample aspect ratio). Since the pixel format with name "yuv410p" corresponds to the number 6 (check the enum AVPixelFormat definition in `libavutil/pixfmt.h`), this example corresponds to:

```
buffer=size=320x240:pixfmt=6:time_base=1/24:pixel_aspect=1/1
```

Alternatively, the options can be specified as a flat string, but this syntax is deprecated:

*width*:*height*:*pix_fmt*:*time_base.num*:*time_base.den*:*pixel_aspect.num*:*pixel_aspect.den*[:*sws_param*]

## 10.2 cellauto

Create a pattern generated by an elementary cellular automaton.

The initial state of the cellular automaton can be defined through the `filename`, and `pattern` options. If such options are not specified an initial state is created randomly.

At each new frame a new row in the video is filled with the result of the cellular automaton next generation. The behavior when the whole frame is filled is defined by the `scroll` option.

This source accepts the following options:

`filename, f`

Read the initial cellular automaton state, i.e. the starting row, from the specified file. In the file, each non-whitespace character is considered an alive cell, a newline will terminate the row, and further characters in the file will be ignored.

`pattern, p`

Read the initial cellular automaton state, i.e. the starting row, from the specified string.

Each non-whitespace character in the string is considered an alive cell, a newline will terminate the row, and further characters in the string will be ignored.

`rate, r`

Set the video rate, that is the number of frames generated per second. Default is 25.

`random_fill_ratio, ratio`

Set the random fill ratio for the initial cellular automaton row. It is a floating point number value ranging from 0 to 1, defaults to 1/PHI.

This option is ignored when a file or a pattern is specified.

`random_seed, seed`

Set the seed for filling randomly the initial row, must be an integer included between 0 and UINT32_MAX. If not specified, or if explicitly set to -1, the filter will try to use a good random seed on a best effort basis.

`rule`

Set the cellular automaton rule, it is a number ranging from 0 to 255. Default value is 110.

`size, s`

Set the size of the output video. For the syntax of this option, check the "Video size" section in the ffmpeg-utils manual.

If `filename` or `pattern` is specified, the size is set by default to the width of the specified initial state row, and the height is set to *width* * PHI.

If `size` is set, it must contain the width of the specified pattern string, and the specified pattern will be centered in the larger row.

If a filename or a pattern string is not specified, the size value defaults to "320x518" (used for a randomly generated initial state).

`scroll`

If set to 1, scroll the output upward when all the rows in the output have been already filled. If set to 0, the new generated row will be written over the top row just after the bottom row is filled. Defaults to 1.

`start_full, full`

If set to 1, completely fill the output with generated rows before outputting the first frame. This is the default behavior, for disabling set the value to 0.

`stitch`

If set to 1, stitch the left and right row edges together. This is the default behavior, for disabling set the value to 0.

### 10.2.1 Examples

- Read the initial state from `pattern`, and specify an output of size 200x400.

  ```
  cellauto=f=pattern:s=200x400
  ```

- Generate a random initial row with a width of 200 cells, with a fill ratio of 2/3:

  ```
  cellauto=ratio=2/3:s=200x200
  ```

- Create a pattern generated by rule 18 starting by a single alive cell centered on an initial row with width 100:

  ```
  cellauto=p=@:s=100x400:full=0:rule=18
  ```

- Specify a more elaborated initial pattern:

  ```
  cellauto=p='@@ @ @@':s=100x400:full=0:rule=18
  ```

## 10.3 mandelbrot

Generate a Mandelbrot set fractal, and progressively zoom towards the point specified with *start_x* and *start_y*.

This source accepts the following options:

`end_pts`

Set the terminal pts value. Default value is 400.

`end_scale`

Set the terminal scale value. Must be a floating point value. Default value is 0.3.

`inner`

Set the inner coloring mode, that is the algorithm used to draw the Mandelbrot fractal internal region.

It shall assume one of the following values:

`black`

Set black mode.

`convergence`

Show time until convergence.

`mincol`

Set color based on point closest to the origin of the iterations.

`period`

Set period mode.

Default value is *mincol*.

`bailout`

Set the bailout value. Default value is 10.0.

`maxiter`

Set the maximum of iterations performed by the rendering algorithm. Default value is 7189.

`outer`

Set outer coloring mode. It shall assume one of following values:

`iteration_count`

Set iteration cound mode.

`normalized_iteration_count`

set normalized iteration count mode.

Default value is *normalized_iteration_count*.

`rate, r`

Set frame rate, expressed as number of frames per second. Default value is "25".

`size, s`

Set frame size. For the syntax of this option, check the "Video size" section in the ffmpeg-utils manual. Default value is "640x480".

`start_scale`

Set the initial scale value. Default value is 3.0.

`start_x`

Set the initial x position. Must be a floating point value between -100 and 100. Default value is -0.743643887037158704752191506114774.

start_y

Set the initial y position. Must be a floating point value between -100 and 100. Default value is -0.131825904205311970493132056385139.

## 10.4 mptestsrc

Generate various test patterns, as generated by the MPlayer test filter.

The size of the generated video is fixed, and is 256x256. This source is useful in particular for testing encoding features.

This source accepts the following options:

rate, r

Specify the frame rate of the sourced video, as the number of frames generated per second. It has to be a string in the format *frame_rate_num*/*frame_rate_den*, an integer number, a floating point number or a valid video frame rate abbreviation. The default value is "25".

duration, d

Set the duration of the sourced video. See (ffmpeg-utils)the Time duration section in the ffmpeg-utils(1) manual for the accepted syntax.

If not specified, or the expressed duration is negative, the video is supposed to be generated forever.

test, t

Set the number or the name of the test to perform. Supported tests are:

dc_luma
dc_chroma
freq_luma
freq_chroma
amp_luma
amp_chroma
cbp
mv
ring1
ring2
all

Default value is "all", which will cycle through the list of all tests.

Some examples:

```
mptestsrc=t=dc_luma
```

will generate a "dc_luma" test pattern.

## 10.5 frei0r_src

Provide a frei0r source.

To enable compilation of this filter you need to install the frei0r header and configure FFmpeg with `--enable-frei0r`.

This source accepts the following parameters:

`size`

> The size of the video to generate. For the syntax of this option, check the "Video size" section in the ffmpeg-utils manual.

`framerate`

> The framerate of the generated video. It may be a string of the form *num*/*den* or a frame rate abbreviation.

`filter_name`

> The name to the frei0r source to load. For more information regarding frei0r and how to set the parameters, read the frei0r section in the video filters documentation.

`filter_params`

> A ’|’-separated list of parameters to pass to the frei0r source.

For example, to generate a frei0r partik0l source with size 200x200 and frame rate 10 which is overlayed on the overlay filter main input:

```
frei0r_src=size=200x200:framerate=10:filter_name=partik0l:filter_params=1234 [overlay]; [in][overlay] overlay
```

## 10.6 life

Generate a life pattern.

This source is based on a generalization of John Conway’s life game.

The sourced input represents a life grid, each pixel represents a cell which can be in one of two possible states, alive or dead. Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent.

At each interaction the grid evolves according to the adopted rule, which specifies the number of neighbor alive cells which will make a cell stay alive or born. The `rule` option allows one to specify the rule to adopt.

This source accepts the following options:

`filename, f`

Set the file from which to read the initial grid state. In the file, each non-whitespace character is considered an alive cell, and newline is used to delimit the end of each row.

If this option is not specified, the initial grid is generated randomly.

`rate, r`

Set the video rate, that is the number of frames generated per second. Default is 25.

`random_fill_ratio, ratio`

Set the random fill ratio for the initial random grid. It is a floating point number value ranging from 0 to 1, defaults to 1/PHI. It is ignored when a file is specified.

`random_seed, seed`

Set the seed for filling the initial random grid, must be an integer included between 0 and UINT32_MAX. If not specified, or if explicitly set to -1, the filter will try to use a good random seed on a best effort basis.

`rule`

Set the life rule.

A rule can be specified with a code of the kind "S*NS*/B*NB*", where *NS* and *NB* are sequences of numbers in the range 0-8, *NS* specifies the number of alive neighbor cells which make a live cell stay alive, and *NB* the number of alive neighbor cells which make a dead cell to become alive (i.e. to "born"). "s" and "b" can be used in place of "S" and "B", respectively.

Alternatively a rule can be specified by an 18-bits integer. The 9 high order bits are used to encode the next cell state if it is alive for each number of neighbor alive cells, the low order bits specify the rule for "borning" new cells. Higher order bits encode for an higher number of neighbor cells. For example the number 6153 = `(12<<9)+9` specifies a stay alive rule of 12 and a born rule of 9, which corresponds to "S23/B03".

Default value is "S23/B3", which is the original Conway's game of life rule, and will keep a cell alive if it has 2 or 3 neighbor alive cells, and will born a new cell if there are three alive cells around a dead cell.

size, s

Set the size of the output video. For the syntax of this option, check the "Video size" section in the ffmpeg-utils manual.

If `filename` is specified, the size is set by default to the same size of the input file. If `size` is set, it must contain the size specified in the input file, and the initial grid defined in that file is centered in the larger resulting area.

If a filename is not specified, the size value defaults to "320x240" (used for a randomly generated initial grid).

stitch

If set to 1, stitch the left and right grid edges together, and the top and bottom edges also. Defaults to 1.

mold

Set cell mold speed. If set, a dead cell will go from `death_color` to `mold_color` with a step of `mold`. `mold` can have a value from 0 to 255.

life_color

Set the color of living (or new born) cells.

death_color

Set the color of dead cells. If `mold` is set, this is the first color used to represent a dead cell.

mold_color

Set mold color, for definitely dead and moldy cells.

For the syntax of these 3 color options, check the "Color" section in the ffmpeg-utils manual.

### 10.6.1 Examples

- Read a grid from `pattern`, and center it on a grid of size 300x300 pixels:

```
life=f=pattern:s=300x300
```

- Generate a random grid of size 200x200, with a fill ratio of 2/3:

```
life=ratio=2/3:s=200x200
```

- Specify a custom rule for evolving a randomly generated grid:

```
life=rule=S14/B34
```

- Full example with slow death effect (mold) using `ffplay`:

```
ffplay -f lavfi life=s=300x200:mold=10:r=60:ratio=0.1:death_color=#C83232:life_color=#00ff00,scale=1200:800:flags=16
```

## 10.7 color, haldclutsrc, nullsrc, rgbtestsrc, smptebars, smptehdbars, testsrc

The `color` source provides an uniformly colored input.

The `haldclutsrc` source provides an identity Hald CLUT. See also haldclut filter.

The `nullsrc` source returns unprocessed video frames. It is mainly useful to be employed in analysis / debugging tools, or as the source for filters which ignore the input data.

The `rgbtestsrc` source generates an RGB test pattern useful for detecting RGB vs BGR issues. You should see a red, green and blue stripe from top to bottom.

The `smptebars` source generates a color bars pattern, based on the SMPTE Engineering Guideline EG 1-1990.

The `smptehdbars` source generates a color bars pattern, based on the SMPTE RP 219-2002.

The `testsrc` source generates a test video pattern, showing a color pattern, a scrolling gradient and a timestamp. This is mainly intended for testing purposes.

The sources accept the following parameters:

`color, c`

> Specify the color of the source, only available in the `color` source. For the syntax of this option, check the "Color" section in the ffmpeg-utils manual.

`level`

> Specify the level of the Hald CLUT, only available in the `haldclutsrc` source. A level of `N` generates a picture of `N*N*N` by `N*N*N` pixels to be used as identity matrix for 3D lookup tables. Each component is coded on a `1/(N*N)` scale.

`size, s`

> Specify the size of the sourced video. For the syntax of this option, check the "Video size" section in the ffmpeg-utils manual. The default value is "320x240".
>
> This option is not available with the `haldclutsrc` filter.

`rate, r`

Specify the frame rate of the sourced video, as the number of frames generated per second. It has to be a string in the format *frame_rate_num*/*frame_rate_den*, an integer number, a floating point number or a valid video frame rate abbreviation. The default value is "25".

`sar`

Set the sample aspect ratio of the sourced video.

`duration, d`

Set the duration of the sourced video. See (ffmpeg-utils)the Time duration section in the ffmpeg-utils(1) manual for the accepted syntax.

If not specified, or the expressed duration is negative, the video is supposed to be generated forever.

`decimals, n`

Set the number of decimals to show in the timestamp, only available in the `testsrc` source.

The displayed timestamp value will correspond to the original timestamp value multiplied by the power of 10 of the specified value. Default value is 0.

For example the following:

```
testsrc=duration=5.3:size=qcif:rate=10
```

will generate a video with a duration of 5.3 seconds, with size 176x144 and a frame rate of 10 frames per second.

The following graph description will generate a red source with an opacity of 0.2, with size "qcif" and a frame rate of 10 frames per second.

```
color=c=red@0.2:s=qcif:r=10
```

If the input content is to be ignored, `nullsrc` can be used. The following command generates noise in the luminance plane by employing the `geq` filter:

```
nullsrc=s=256x256, geq=random(1)*255:128:128
```

### 10.7.1 Commands

The `color` source supports the following commands:

`c, color`

Set the color of the created image. Accepts the same syntax of the corresponding `color` option.

# 11 Video Sinks

Below is a description of the currently available video sinks.

## 11.1 buffersink

Buffer video frames, and make them available to the end of the filter graph.

This sink is mainly intended for programmatic use, in particular through the interface defined in `libavfilter/buffersink.h` or the options system.

It accepts a pointer to an AVBufferSinkContext structure, which defines the incoming buffers' formats, to be passed as the opaque parameter to `avfilter_init_filter` for initialization.

## 11.2 nullsink

Null video sink: do absolutely nothing with the input video. It is mainly useful as a template and for use in analysis / debugging tools.

# 12 Multimedia Filters

Below is a description of the currently available multimedia filters.

## 12.1 avectorscope

Convert input audio to a video output, representing the audio vector scope.

The filter is used to measure the difference between channels of stereo audio stream. A monoaural signal, consisting of identical left and right signal, results in straight vertical line. Any stereo separation is visible as a deviation from this line, creating a Lissajous figure. If the straight (or deviation from it) but horizontal line appears this indicates that the left and right channels are out of phase.

The filter accepts the following options:

`mode, m`

Set the vectorscope mode.

Available values are:

'`lissajous`'

Lissajous rotated by 45 degrees.

'`lissajous_xy`'

Same as above but not rotated.

Default value is ‘`lissajous`’.

`size, s`

Set the video size for the output. For the syntax of this option, check the "Video size" section in the ffmpeg-utils manual. Default value is `400x400`.

`rate, r`

Set the output frame rate. Default value is `25`.

`rc`
`gc`
`bc`

Specify the red, green and blue contrast. Default values are `40`, `160` and `80`. Allowed range is `[0, 255]`.

`rf`
`gf`
`bf`

Specify the red, green and blue fade. Default values are `15`, `10` and `5`. Allowed range is `[0, 255]`.

`zoom`

Set the zoom factor. Default value is `1`. Allowed range is `[1, 10]`.

### 12.1.1 Examples

- Complete example using `ffplay`:

```
ffplay -f lavfi 'amovie=input.mp3, asplit [a][out1];
             [a] avectorscope=zoom=1.3:rc=2:gc=200:bc=10:rf=1:gf=8:bf=7 [out0]'
```

## 12.2 concat

Concatenate audio and video streams, joining them together one after the other.

The filter works on segments of synchronized video and audio streams. All segments must have the same number of streams of each type, and that will also be the number of streams at output.

The filter accepts the following options:

`n`

Set the number of segments. Default is 2.

`v`

Set the number of output video streams, that is also the number of video streams in each segment. Default is 1.

`a`

Set the number of output audio streams, that is also the number of audio streams in each segment. Default is 0.

`unsafe`

Activate unsafe mode: do not fail if segments have a different format.

The filter has *v*+*a* outputs: first *v* video outputs, then *a* audio outputs.

There are *n*x(*v*+*a*) inputs: first the inputs for the first segment, in the same order as the outputs, then the inputs for the second segment, etc.

Related streams do not always have exactly the same duration, for various reasons including codec frame size or sloppy authoring. For that reason, related synchronized streams (e.g. a video and its audio track) should be concatenated at once. The concat filter will use the duration of the longest stream in each segment (except the last one), and if necessary pad shorter audio streams with silence.

For this filter to work correctly, all segments must start at timestamp 0.

All corresponding streams must have the same parameters in all segments; the filtering system will automatically select a common pixel format for video streams, and a common sample format, sample rate and channel layout for audio streams, but other settings, such as resolution, must be converted explicitly by the user.

Different frame rates are acceptable but will result in variable frame rate at output; be sure to configure the output file to handle it.

### 12.2.1 Examples

- Concatenate an opening, an episode and an ending, all in bilingual version (video in stream 0, audio in streams 1 and 2):

```
ffmpeg -i opening.mkv -i episode.mkv -i ending.mkv -filter_complex \
  '[0:0] [0:1] [0:2] [1:0] [1:1] [1:2] [2:0] [2:1] [2:2]
   concat=n=3:v=1:a=2 [v] [a1] [a2]' \
  -map '[v]' -map '[a1]' -map '[a2]' output.mkv
```

- Concatenate two parts, handling audio and video separately, using the (a)movie sources, and adjusting the resolution:

```
movie=part1.mp4, scale=512:288 [v1] ; amovie=part1.mp4 [a1] ;
movie=part2.mp4, scale=512:288 [v2] ; amovie=part2.mp4 [a2] ;
[v1] [v2] concat [outv] ; [a1] [a2] concat=v=0:a=1 [outa]
```

Note that a desync will happen at the stitch if the audio and video streams do not have exactly the same duration in the first file.

## 12.3 ebur128

EBU R128 scanner filter. This filter takes an audio stream as input and outputs it unchanged. By default, it logs a message at a frequency of 10Hz with the Momentary loudness (identified by `M`), Short-term loudness (`S`), Integrated loudness (`I`) and Loudness Range (`LRA`).

The filter also has a video output (see the *video* option) with a real time graph to observe the loudness evolution. The graphic contains the logged message mentioned above, so it is not printed anymore when this option is set, unless the verbose logging is set. The main graphing area contains the short-term loudness (3 seconds of analysis), and the gauge on the right is for the momentary loudness (400 milliseconds).

More information about the Loudness Recommendation EBU R128 on http://tech.ebu.ch/loudness.

The filter accepts the following options:

`video`

Activate the video output. The audio stream is passed unchanged whether this option is set or no. The video stream will be the first output stream if activated. Default is `0`.

`size`

Set the video size. This option is for video only. For the syntax of this option, check the "Video size" section in the ffmpeg-utils manual. Default and minimum resolution is `640x480`.

`meter`

Set the EBU scale meter. Default is `9`. Common values are `9` and `18`, respectively for EBU scale meter +9 and EBU scale meter +18. Any other integer value between this range is allowed.

`metadata`

Set metadata injection. If set to `1`, the audio input will be segmented into 100ms output frames, each of them containing various loudness information in metadata. All the metadata keys are prefixed with `lavfi.r128.`.

Default is `0`.

`framelog`

Force the frame logging level.

Available values are:

'`info`'

information logging level

'`verbose`'

verbose logging level

By default, the logging level is set to *info*. If the `video` or the `metadata` options are set, it switches to *verbose*.

`peak`

Set peak mode(s).

Available modes can be cumulated (the option is a `flag` type). Possible values are:

'`none`'

Disable any peak mode (default).

'`sample`'

Enable sample-peak mode.

Simple peak mode looking for the higher sample value. It logs a message for sample-peak (identified by `SPK`).

'`true`'

Enable true-peak mode.

If enabled, the peak lookup is done on an over-sampled version of the input stream for better peak accuracy. It logs a message for true-peak. (identified by `TPK`) and true-peak per frame (identified by `FTPK`). This mode requires a build with `libswresample`.

### 12.3.1 Examples

- Real-time graph using `ffplay`, with a EBU scale meter +18:

```
ffplay -f lavfi -i "amovie=input.mp3,ebur128=video=1:meter=18 [out0][out1]"
```

- Run an analysis with `ffmpeg`:

```
ffmpeg -nostats -i input.mp3 -filter_complex ebur128 -f null -
```

## 12.4 interleave, ainterleave

Temporally interleave frames from several inputs.

`interleave` works with video inputs, `ainterleave` with audio.

These filters read frames from several inputs and send the oldest queued frame to the output.

Input streams must have a well defined, monotonically increasing frame timestamp values.

In order to submit one frame to output, these filters need to enqueue at least one frame for each input, so they cannot work in case one input is not yet terminated and will not receive incoming frames.

For example consider the case when one input is a `select` filter which always drop input frames. The `interleave` filter will keep reading from that input, but it will never be able to send new frames to output until the input will send an end-of-stream signal.

Also, depending on inputs synchronization, the filters will drop frames in case one input receives more frames than the other ones, and the queue is already filled.

These filters accept the following options:

`nb_inputs, n`

Set the number of different inputs, it is 2 by default.

### 12.4.1 Examples

- Interleave frames belonging to different streams using `ffmpeg`:

  ```
  ffmpeg -i bambi.avi -i pr0n.mkv -filter_complex "[0:v][1:v] interleave" out.avi
  ```

- Add flickering blur effect:

  ```
  select='if(gt(random(0), 0.2), 1, 2)':n=2 [tmp], boxblur=2:2, [tmp] interleave
  ```

## 12.5 perms, aperms

Set read/write permissions for the output frames.

These filters are mainly aimed at developers to test direct path in the following filter in the filtergraph.

The filters accept the following options:

`mode`

Select the permissions mode.

It accepts the following values:

'none'

Do nothing. This is the default.

'ro'

Set all the output frames read-only.

'rw'

Set all the output frames directly writable.

'toggle'

Make the frame read-only if writable, and writable if read-only.

'random'

Set each output frame read-only or writable randomly.

seed

Set the seed for the *random* mode, must be an integer included between `0` and `UINT32_MAX`. If not specified, or if explicitly set to `-1`, the filter will try to use a good random seed on a best effort basis.

Note: in case of auto-inserted filter between the permission filter and the following one, the permission might not be received as expected in that following filter. Inserting a format or aformat filter before the perms/aperms filter can avoid this problem.

## 12.6 select, aselect

Select frames to pass in output.

This filter accepts the following options:

expr, e

Set expression, which is evaluated for each input frame.

If the expression is evaluated to zero, the frame is discarded.

If the evaluation result is negative or NaN, the frame is sent to the first output; otherwise it is sent to the output with index `ceil(val)-1`, assuming that the input index starts from 0.

For example a value of `1.2` corresponds to the output with index `ceil(1.2)-1 = 2-1 = 1`, that is the second output.

`outputs, n`

Set the number of outputs. The output to which to send the selected frame is based on the result of the evaluation. Default value is 1.

The expression can contain the following constants:

`n`

The (sequential) number of the filtered frame, starting from 0.

`selected_n`

The (sequential) number of the selected frame, starting from 0.

`prev_selected_n`

The sequential number of the last selected frame. It’s NAN if undefined.

`TB`

The timebase of the input timestamps.

`pts`

The PTS (Presentation TimeStamp) of the filtered video frame, expressed in *TB* units. It’s NAN if undefined.

`t`

The PTS of the filtered video frame, expressed in seconds. It’s NAN if undefined.

`prev_pts`

The PTS of the previously filtered video frame. It’s NAN if undefined.

`prev_selected_pts`

The PTS of the last previously filtered video frame. It’s NAN if undefined.

`prev_selected_t`

The PTS of the last previously selected video frame. It’s NAN if undefined.

`start_pts`

The PTS of the first video frame in the video. It’s NAN if undefined.

`start_t`

The time of the first video frame in the video. It’s NAN if undefined.

`pict_type` *(video only)*

The type of the filtered frame. It can assume one of the following values:

`I`
`P`
`B`
`S`
`SI`
`SP`
`BI`

`interlace_type` *(video only)*

The frame interlace type. It can assume one of the following values:

`PROGRESSIVE`

The frame is progressive (not interlaced).

`TOPFIRST`

The frame is top-field-first.

`BOTTOMFIRST`

The frame is bottom-field-first.

`consumed_sample_n` *(audio only)*

the number of selected samples before the current frame

`samples_n` *(audio only)*

the number of samples in the current frame

`sample_rate` *(audio only)*

the input sample rate

`key`

This is 1 if the filtered frame is a key-frame, 0 otherwise.

`pos`

the position in the file of the filtered frame, -1 if the information is not available (e.g. for synthetic video)

`scene` *(video only)*

value between 0 and 1 to indicate a new scene; a low value reflects a low probability for the current frame to introduce a new scene, while a higher value means the current frame is more likely to be one (see the example below)

The default value of the select expression is "1".

### 12.6.1 Examples

- Select all frames in input:

  ```
  select
  ```

  The example above is the same as:

  ```
  select=1
  ```

- Skip all frames:

  ```
  select=0
  ```

- Select only I-frames:

  ```
  select='eq(pict_type\,I)'
  ```

- Select one frame every 100:

  ```
  select='not(mod(n\,100))'
  ```

- Select only frames contained in the 10-20 time interval:

  ```
  select=between(t\,10\,20)
  ```

- Select only I frames contained in the 10-20 time interval:

  ```
  select=between(t\,10\,20)*eq(pict_type\,I)
  ```

- Select frames with a minimum distance of 10 seconds:

  ```
  select='isnan(prev_selected_t)+gte(t-prev_selected_t\,10)'
  ```

- Use aselect to select only audio frames with samples number > 100:

  ```
  aselect='gt(samples_n\,100)'
  ```

- Create a mosaic of the first scenes:

```
ffmpeg -i video.avi -vf select='gt(scene\,0.4)',scale=160:120,tile -frames:v 1 preview.png
```

Comparing *scene* against a value between 0.3 and 0.5 is generally a sane choice.

- Send even and odd frames to separate outputs, and compose them:

```
select=n=2:e='mod(n, 2)+1' [odd][even]; [odd] pad=h=2*ih [tmp]; [tmp][even] overlay=y=h
```

## 12.7 sendcmd, asendcmd

Send commands to filters in the filtergraph.

These filters read commands to be sent to other filters in the filtergraph.

`sendcmd` must be inserted between two video filters, `asendcmd` must be inserted between two audio filters, but apart from that they act the same way.

The specification of commands can be provided in the filter arguments with the *commands* option, or in a file specified by the *filename* option.

These filters accept the following options:

`commands, c`

Set the commands to be read and sent to the other filters.

`filename, f`

Set the filename of the commands to be read and sent to the other filters.

### 12.7.1 Commands syntax

A commands description consists of a sequence of interval specifications, comprising a list of commands to be executed when a particular event related to that interval occurs. The occurring event is typically the current frame time entering or leaving a given time interval.

An interval is specified by the following syntax:

*START*[-*END*] *COMMANDS*;

The time interval is specified by the *START* and *END* times. *END* is optional and defaults to the maximum time.

The current frame time is considered within the specified interval if it is included in the interval [*START*, *END*), that is when the time is greater or equal to *START* and is lesser than *END*.

*COMMANDS* consists of a sequence of one or more command specifications, separated by ",", relating to that interval. The syntax of a command specification is given by:

```
[FLAGS] TARGET COMMAND ARG
```

*FLAGS* is optional and specifies the type of events relating to the time interval which enable sending the specified command, and must be a non-null sequence of identifier flags separated by "+" or "|" and enclosed between "[" and "]".

The following flags are recognized:

`enter`

> The command is sent when the current frame timestamp enters the specified interval. In other words, the command is sent when the previous frame timestamp was not in the given interval, and the current is.

`leave`

> The command is sent when the current frame timestamp leaves the specified interval. In other words, the command is sent when the previous frame timestamp was in the given interval, and the current is not.

If *FLAGS* is not specified, a default value of `[enter]` is assumed.

*TARGET* specifies the target of the command, usually the name of the filter class or a specific filter instance name.

*COMMAND* specifies the name of the command for the target filter.

*ARG* is optional and specifies the optional list of argument for the given *COMMAND*.

Between one interval specification and another, whitespaces, or sequences of characters starting with # until the end of line, are ignored and can be used to annotate comments.

A simplified BNF description of the commands specification syntax follows:

```
COMMAND_FLAG  ::= "enter" | "leave"
COMMAND_FLAGS ::= COMMAND_FLAG [(+|"|")COMMAND_FLAG]
COMMAND       ::= ["[" COMMAND_FLAGS "]"] TARGET COMMAND [ARG]
COMMANDS      ::= COMMAND [,COMMANDS]
INTERVAL      ::= START[-END] COMMANDS
INTERVALS     ::= INTERVAL[;INTERVALS]
```

## 12.7.2 Examples

- Specify audio tempo change at second 4:

  ```
  asendcmd=c='4.0 atempo tempo 1.5',atempo
  ```

- Specify a list of drawtext and hue commands in a file.

```
# show text in the interval 5-10
5.0-10.0 [enter] drawtext reinit 'fontfile=FreeSerif.ttf:text=hello world',
         [leave] drawtext reinit 'fontfile=FreeSerif.ttf:text=';

# desaturate the image in the interval 15-20
15.0-20.0 [enter] hue s 0,
          [enter] drawtext reinit 'fontfile=FreeSerif.ttf:text=nocolor',
          [leave] hue s 1,
          [leave] drawtext reinit 'fontfile=FreeSerif.ttf:text=color';

# apply an exponential saturation fade-out effect, starting from time 25
25 [enter] hue s exp(25-t)
```

A filtergraph allowing to read and process the above command list stored in a file `test.cmd`, can be specified with:

```
sendcmd=f=test.cmd,drawtext=fontfile=FreeSerif.ttf:text='',hue
```

## 12.8 setpts, asetpts

Change the PTS (presentation timestamp) of the input frames.

`setpts` works on video frames, `asetpts` on audio frames.

This filter accepts the following options:

`expr`

The expression which is evaluated for each frame to construct its timestamp.

The expression is evaluated through the eval API and can contain the following constants:

`FRAME_RATE`

frame rate, only defined for constant frame-rate video

`PTS`

The presentation timestamp in input

`N`

The count of the input frame for video or the number of consumed samples, not including the current frame for audio, starting from 0.

`NB_CONSUMED_SAMPLES`

The number of consumed samples, not including the current frame (only audio)

`NB_SAMPLES, S`

The number of samples in the current frame (only audio)

`SAMPLE_RATE, SR`

The audio sample rate.

`STARTPTS`

The PTS of the first frame.

`STARTT`

the time in seconds of the first frame

`INTERLACED`

State whether the current frame is interlaced.

`T`

the time in seconds of the current frame

`POS`

original position in the file of the frame, or undefined if undefined for the current frame

`PREV_INPTS`

The previous input PTS.

`PREV_INT`

previous input time in seconds

`PREV_OUTPTS`

The previous output PTS.

`PREV_OUTT`

previous output time in seconds

`RTCTIME`

The wallclock (RTC) time in microseconds.. This is deprecated, use time(0) instead.

`RTCSTART`

The wallclock (RTC) time at the start of the movie in microseconds.

`TB`

The timebase of the input timestamps.

### 12.8.1 Examples

- Start counting PTS from zero

  ```
  setpts=PTS-STARTPTS
  ```

- Apply fast motion effect:

  ```
  setpts=0.5*PTS
  ```

- Apply slow motion effect:

  ```
  setpts=2.0*PTS
  ```

- Set fixed rate of 25 frames per second:

  ```
  setpts=N/(25*TB)
  ```

- Set fixed rate 25 fps with some jitter:

  ```
  setpts='1/(25*TB) * (N + 0.05 * sin(N*2*PI/25))'
  ```

- Apply an offset of 10 seconds to the input PTS:

  ```
  setpts=PTS+10/TB
  ```

- Generate timestamps from a "live source" and rebase onto the current timebase:

  ```
  setpts='(RTCTIME - RTCSTART) / (TB * 1000000)'
  ```

- Generate timestamps by counting samples:

  ```
  asetpts=N/SR/TB
  ```

## 12.9 settb, asettb

Set the timebase to use for the output frames timestamps. It is mainly useful for testing timebase configuration.

It accepts the following parameters:

`expr, tb`

The expression which is evaluated into the output timebase.

The value for `tb` is an arithmetic expression representing a rational. The expression can contain the constants "AVTB" (the default timebase), "intb" (the input timebase) and "sr" (the sample rate, audio only). Default value is "intb".

### 12.9.1 Examples

- Set the timebase to 1/25:

  ```
  settb=expr=1/25
  ```

- Set the timebase to 1/10:

  ```
  settb=expr=0.1
  ```

- Set the timebase to 1001/1000:

  ```
  settb=1+0.001
  ```

- Set the timebase to 2*intb:

  ```
  settb=2*intb
  ```

- Set the default timebase value:

  ```
  settb=AVTB
  ```

## 12.10 showcqt

Convert input audio to a video output representing frequency spectrum logarithmically (using constant Q transform with Brown-Puckette algorithm), with musical tone scale, from E0 to D#10 (10 octaves).

The filter accepts the following options:

`volume`

Specify transform volume (multiplier) expression. The expression can contain variables:

`frequency, freq, f`

the frequency where transform is evaluated

`timeclamp, tc`

value of timeclamp option

and functions:

`a_weighting(f)`

A-weighting of equal loudness

`b_weighting(f)`

B-weighting of equal loudness

`c_weighting(f)`

C-weighting of equal loudness

Default value is `16`.

`tlength`

Specify transform length expression. The expression can contain variables:

`frequency, freq, f`

the frequency where transform is evaluated

`timeclamp, tc`

value of timeclamp option

Default value is `384/f*tc/(384/f+tc)`.

`timeclamp`

Specify the transform timeclamp. At low frequency, there is trade-off between accuracy in time domain and frequency domain. If timeclamp is lower, event in time domain is represented more accurately (such as fast bass drum), otherwise event in frequency domain is represented more accurately (such as bass guitar). Acceptable value is [0.1, 1.0]. Default value is `0.17`.

`coeffclamp`

Specify the transform coeffclamp. If coeffclamp is lower, transform is more accurate, otherwise transform is faster. Acceptable value is [0.1, 10.0]. Default value is `1.0`.

`gamma`

Specify gamma. Lower gamma makes the spectrum more contrast, higher gamma makes the spectrum having more range. Acceptable value is [1.0, 7.0]. Default value is `3.0`.

`fontfile`

Specify font file for use with freetype. If not specified, use embedded font.

`fontcolor`

Specify font color expression. This is arithmetic expression that should return integer value 0xRRGGBB. The expression can contain variables:

`frequency, freq, f`

the frequency where transform is evaluated

`timeclamp, tc`

value of timeclamp option

and functions:

`midi(f)`

midi number of frequency f, some midi numbers: E0(16), C1(24), C2(36), A4(69)

`r(x), g(x), b(x)`

red, green, and blue value of intensity x

Default value is `st(0, (midi(f)-59.5)/12); st(1, if(between(ld(0),0,1), 0.5-0.5*cos(2*PI*ld(0)), 0)); r(1-ld(1)) + b(ld(1))`

`fullhd`

If set to 1 (the default), the video size is 1920x1080 (full HD), if set to 0, the video size is 960x540. Use this option to make CPU usage lower.

`fps`

Specify video fps. Default value is `25`.

`count`

Specify number of transform per frame, so there are fps*count transforms per second. Note that audio data rate must be divisible by fps*count. Default value is `6`.

### 12.10.1 Examples

- Playing audio while showing the spectrum:

```
ffplay -f lavfi 'amovie=a.mp3, asplit [a][out1]; [a] showcqt [out0]'
```

- Same as above, but with frame rate 30 fps:

```
ffplay -f lavfi 'amovie=a.mp3, asplit [a][out1]; [a] showcqt=fps=30:count=5 [out0]'
```

- Playing at 960x540 and lower CPU usage:

```
ffplay -f lavfi 'amovie=a.mp3, asplit [a][out1]; [a] showcqt=fullhd=0:count=3 [out0]'
```

- A1 and its harmonics: A1, A2, (near)E3, A3:

```
ffplay -f lavfi 'aevalsrc=0.1*sin(2*PI*55*t)+0.1*sin(4*PI*55*t)+0.1*sin(6*PI*55*t)+0.1*sin(8*PI*55*t),
                 asplit[a][out1]; [a] showcqt [out0]'
```

- Same as above, but with more accuracy in frequency domain (and slower):

```
ffplay -f lavfi 'aevalsrc=0.1*sin(2*PI*55*t)+0.1*sin(4*PI*55*t)+0.1*sin(6*PI*55*t)+0.1*sin(8*PI*55*t),
                 asplit[a][out1]; [a] showcqt=timeclamp=0.5 [out0]'
```

- B-weighting of equal loudness

```
volume=16*b_weighting(f)
```

- Lower Q factor

```
tlength=100/f*tc/(100/f+tc)
```

- Custom fontcolor, C-note is colored green, others are colored blue

```
fontcolor='if(mod(floor(midi(f)+0.5),12), 0x0000FF, g(1))'
```

## 12.11 showspectrum

Convert input audio to a video output, representing the audio frequency spectrum.

The filter accepts the following options:

`size, s`

Specify the video size for the output. For the syntax of this option, check the "Video size" section in the ffmpeg-utils manual. Default value is `640x512`.

`slide`

Specify how the spectrum should slide along the window.

It accepts the following values:

'`replace`'

the samples start again on the left when they reach the right

'`scroll`'

the samples scroll from right to left

'`fullframe`'

frames are only produced when the samples reach the right

Default value is `replace`.

`mode`

Specify display mode.

It accepts the following values:

‘`combined`’

all channels are displayed in the same row

‘`separate`’

all channels are displayed in separate rows

Default value is ‘`combined`’.

`color`

Specify display color mode.

It accepts the following values:

‘`channel`’

each channel is displayed in a separate color

‘`intensity`’

each channel is is displayed using the same color scheme

Default value is ‘`channel`’.

`scale`

Specify scale used for calculating intensity color values.

It accepts the following values:

‘`lin`’

linear

‘`sqrt`’

square root, default

‘cbrt’

cubic root

‘log’

logarithmic

Default value is ‘sqrt’.

saturation

Set saturation modifier for displayed colors. Negative values provide alternative color scheme. 0 is no saturation at all. Saturation must be in [-10.0, 10.0] range. Default value is 1.

win_func

Set window function.

It accepts the following values:

‘none’

No samples pre-processing (do not expect this to be faster)

‘hann’

Hann window

‘hamming’

Hamming window

‘blackman’

Blackman window

Default value is hann.

The usage is very similar to the showwaves filter; see the examples in that section.

### 12.11.1 Examples

- Large window with logarithmic color scaling:

```
showspectrum=s=1280x480:scale=log
```

- Complete example for a colored and sliding spectrum per channel using ffplay:

```
ffplay -f lavfi 'amovie=input.mp3, asplit [a][out1];
             [a] showspectrum=mode=separate:color=intensity:slide=1:scale=cbrt [out0]'
```

## 12.12 showwaves

Convert input audio to a video output, representing the samples waves.

The filter accepts the following options:

`size, s`

Specify the video size for the output. For the syntax of this option, check the "Video size" section in the ffmpeg-utils manual. Default value is "600x240".

`mode`

Set display mode.

Available values are:

'`point`'

Draw a point for each sample.

'`line`'

Draw a vertical line for each sample.

'`p2p`'

Draw a point for each sample and a line between them.

'`cline`'

Draw a centered vertical line for each sample.

Default value is `point`.

`n`

Set the number of samples which are printed on the same column. A larger value will decrease the frame rate. Must be a positive integer. This option can be set only if the value for *rate* is not explicitly specified.

`rate, r`

Set the (approximate) output frame rate. This is done by setting the option *n*. Default value is "25".

`split_channels`

Set if channels should be drawn separately or overlap. Default value is 0.

### 12.12.1 Examples

- Output the input file audio and the corresponding video representation at the same time:

```
amovie=a.mp3,asplit[out0],showwaves[out1]
```

- Create a synthetic signal and show it with showwaves, forcing a frame rate of 30 frames per second:

```
aevalsrc=sin(1*2*PI*t)*sin(880*2*PI*t):cos(2*PI*200*t),asplit[out0],showwaves=r=30[out1]
```

## 12.13 split, asplit

Split input into several identical outputs.

`asplit` works with audio input, `split` with video.

The filter accepts a single parameter which specifies the number of outputs. If unspecified, it defaults to 2.

### 12.13.1 Examples

- Create two separate outputs from the same input:

```
[in] split [out0][out1]
```

- To create 3 or more outputs, you need to specify the number of outputs, like in:

```
[in] asplit=3 [out0][out1][out2]
```

- Create two separate outputs from the same input, one cropped and one padded:

```
[in] split [splitout1][splitout2];
[splitout1] crop=100:100:0:0    [cropout];
[splitout2] pad=200:200:100:100 [padout];
```

- Create 5 copies of the input audio with `ffmpeg`:

```
ffmpeg -i INPUT -filter_complex asplit=5 OUTPUT
```

## 12.14 zmq, azmq

Receive commands sent through a libzmq client, and forward them to filters in the filtergraph.

`zmq` and `azmq` work as a pass-through filters. `zmq` must be inserted between two video filters, `azmq` between two audio filters.

To enable these filters you need to install the libzmq library and headers and configure FFmpeg with `--enable-libzmq`.

For more information about libzmq see: http://www.zeromq.org/

The `zmq` and `azmq` filters work as a libzmq server, which receives messages sent through a network interface defined by the `bind_address` option.

The received message must be in the form:

```
TARGET COMMAND [ARG]
```

*TARGET* specifies the target of the command, usually the name of the filter class or a specific filter instance name.

*COMMAND* specifies the name of the command for the target filter.

*ARG* is optional and specifies the optional argument list for the given *COMMAND*.

Upon reception, the message is processed and the corresponding command is injected into the filtergraph. Depending on the result, the filter will send a reply to the client, adopting the format:

```
ERROR_CODE ERROR_REASON
MESSAGE
```

*MESSAGE* is optional.

## 12.14.1 Examples

Look at `tools/zmqsend` for an example of a zmq client which can be used to send commands processed by these filters.

Consider the following filtergraph generated by `ffplay`

```
ffplay -dumpgraph 1 -f lavfi "
color=s=100x100:c=red  [l];
color=s=100x100:c=blue [r];
nullsrc=s=200x100, zmq [bg];
[bg][l]   overlay      [bg+l];
[bg+l][r] overlay=x=100 "
```

To change the color of the left side of the video, the following command can be used:

```
echo Parsed_color_0 c yellow | tools/zmqsend
```

To change the right side:

```
echo Parsed_color_1 c pink | tools/zmqsend
```

# 13 Multimedia Sources

Below is a description of the currently available multimedia sources.

## 13.1 amovie

This is the same as movie source, except it selects an audio stream by default.

## 13.2 movie

Read audio and/or video stream(s) from a movie container.

It accepts the following parameters:

`filename`

> The name of the resource to read (not necessarily a file; it can also be a device or a stream accessed through some protocol).

`format_name, f`

> Specifies the format assumed for the movie to read, and can be either the name of a container or an input device. If not specified, the format is guessed from *movie_name* or by probing.

`seek_point, sp`

> Specifies the seek point in seconds. The frames will be output starting from this seek point. The parameter is evaluated with `av_strtod`, so the numerical value may be suffixed by an IS postfix. The default value is "0".

`streams, s`

> Specifies the streams to read. Several streams can be specified, separated by "+". The source will then have as many outputs, in the same order. The syntax is explained in the “Stream specifiers” section in the ffmpeg manual. Two special names, "dv" and "da" specify respectively the default (best suited) video and audio stream. Default is "dv", or "da" if the filter is called as "amovie".

`stream_index, si`

> Specifies the index of the video stream to read. If the value is -1, the most suitable video stream will be automatically selected. The default value is "-1". Deprecated. If the filter is called "amovie", it will select audio instead of video.

`loop`

> Specifies how many times to read the stream in sequence. If the value is less than 1, the stream will be read again and again. Default value is "1".

Note that when the movie is looped the source timestamps are not changed, so it will generate non monotonically increasing timestamps.

It allows overlaying a second video on top of the main input of a filtergraph, as shown in this graph:

```
input -----------> deltapts0 --> overlay --> output
                                    ^
                                    |
movie --> scale--> deltapts1 -------+
```

### 13.2.1 Examples

- Skip 3.2 seconds from the start of the AVI file in.avi, and overlay it on top of the input labelled "in":

  ```
  movie=in.avi:seek_point=3.2, scale=180:-1, setpts=PTS-STARTPTS [over];
  [in] setpts=PTS-STARTPTS [main];
  [main][over] overlay=16:16 [out]
  ```

- Read from a video4linux2 device, and overlay it on top of the input labelled "in":

  ```
  movie=/dev/video0:f=video4linux2, scale=180:-1, setpts=PTS-STARTPTS [over];
  [in] setpts=PTS-STARTPTS [main];
  [main][over] overlay=16:16 [out]
  ```

- Read the first video stream and the audio stream with id 0x81 from dvd.vob; the video is connected to the pad named "video" and the audio is connected to the pad named "audio":

  ```
  movie=dvd.vob:s=v:0+#0x81 [video] [audio]
  ```

# 14 See Also

ffmpeg, ffplay, ffprobe, ffserver, libavfilter

# 15 Authors

The FFmpeg developers.

For details about the authorship, see the Git history of the project (git://source.ffmpeg.org/ffmpeg), e.g. by typing the command `git log` in the FFmpeg source directory, or browsing the online repository at http://source.ffmpeg.org.

Maintainers for the specific components are listed in the file `MAINTAINERS` in the source code tree.

This document was generated on *January 10, 2015* using *makeinfo*.

# FFmpeg Filters Documentation

## Table of Contents

# 1 Description

This document describes filters, sources, and sinks provided by the libavfilter library.

# 2 Filtering Introduction

Filtering in FFmpeg is enabled through the libavfilter library.

In libavfilter, a filter can have multiple inputs and multiple outputs. To illustrate the sorts of things that are possible, we consider the following filtergraph.

```
                [main]
input --> split ---------------------> overlay --> output
            |                             ^
            |[tmp]                  [flip]|
            +-----> crop --> vflip -------+
```

This filtergraph splits the input stream in two streams, then sends one stream through the crop filter and the vflip filter, before merging it back with the other stream by overlaying it on top. You can use the following command to achieve this:

```
ffmpeg -i INPUT -vf "split [main][tmp]; [tmp] crop=iw:ih/2:0:0, vflip [flip]; [main][flip] overlay=0:H/2" OUTPUT
```

The result will be that the top half of the video is mirrored onto the bottom half of the output video.

Filters in the same linear chain are separated by commas, and distinct linear chains of filters are separated by semicolons. In our example, *crop,vflip* are in one linear chain, *split* and *overlay* are separately in another. The points where the linear chains join are labelled by names enclosed in square brackets. In the example, the split filter generates two outputs that are associated to the labels *[main]* and *[tmp]*.

The stream sent to the second output of *split*, labelled as *[tmp]*, is processed through the *crop* filter, which crops away the lower half part of the video, and then vertically flipped. The *overlay* filter takes in input the first unchanged output of the split filter (which was labelled as *[main]*), and overlay on its lower half the output generated by the *crop,vflip* filterchain.

Some filters take in input a list of parameters: they are specified after the filter name and an equal sign, and are separated from each other by a colon.

There exist so-called *source filters* that do not have an audio/video input, and *sink filters* that will not have audio/video output.

# 3 graph2dot

The `graph2dot` program included in the FFmpeg `tools` directory can be used to parse a filtergraph description and issue a corresponding textual representation in the dot language.

Invoke the command:

```
graph2dot -h
```

to see how to use `graph2dot`.

You can then pass the dot description to the `dot` program (from the graphviz suite of programs) and obtain a graphical representation of the filtergraph.

For example the sequence of commands:

```
echo GRAPH_DESCRIPTION | \
tools/graph2dot -o graph.tmp && \
dot -Tpng graph.tmp -o graph.png && \
display graph.png
```

can be used to create and display an image representing the graph described by the *GRAPH_DESCRIPTION* string. Note that this string must be a complete self-contained graph, with its inputs and outputs explicitly defined. For example if your command line is of the form:

```
ffmpeg -i infile -vf scale=640:360 outfile
```

your *GRAPH_DESCRIPTION* string will need to be of the form:

```
nullsrc,scale=640:360,nullsink
```

you may also need to set the *nullsrc* parameters and add a *format* filter in order to simulate a specific input file.

# 4 Filtergraph description

A filtergraph is a directed graph of connected filters. It can contain cycles, and there can be multiple links between a pair of filters. Each link has one input pad on one side connecting it to one filter from which it takes its input, and one output pad on the other side connecting it to one filter accepting its output.

Each filter in a filtergraph is an instance of a filter class registered in the application, which defines the features and the number of input and output pads of the filter.

A filter with no input pads is called a "source", and a filter with no output pads is called a "sink".

## 4.1 Filtergraph syntax

A filtergraph has a textual representation, which is recognized by the `-filter`/`-vf` and `-filter_complex` options in `ffmpeg` and `-vf` in `ffplay`, and by the `avfilter_graph_parse()`/`avfilter_graph_parse2()` functions defined in `libavfilter/avfilter.h`.

A filterchain consists of a sequence of connected filters, each one connected to the previous one in the sequence. A filterchain is represented by a list of ","-separated filter descriptions.

A filtergraph consists of a sequence of filterchains. A sequence of filterchains is represented by a list of ";"-separated filterchain descriptions.

A filter is represented by a string of the form:
[*in_link_1*]...[*in_link_N*]*filter_name*=*arguments*[*out_link_1*]...[*out_link_M*]

*filter_name* is the name of the filter class of which the described filter is an instance of, and has to be the name of one of the filter classes registered in the program. The name of the filter class is optionally followed by a string "=*arguments*".

*arguments* is a string which contains the parameters used to initialize the filter instance. It may have one of two forms:

- A ':'-separated list of *key=value* pairs.
- A ':'-separated list of *value*. In this case, the keys are assumed to be the option names in the order they are declared. E.g. the `fade` filter declares three options in this order – `type`, `start_frame` and `nb_frames`. Then the parameter list *in:0:30* means that the value *in* is assigned to the option `type`, *0* to `start_frame` and *30* to `nb_frames`.
- A ':'-separated list of mixed direct *value* and long *key=value* pairs. The direct *value* must precede the *key=value* pairs, and follow the same constraints order of the previous point. The following *key=value* pairs can be set in any preferred order.

If the option value itself is a list of items (e.g. the `format` filter takes a list of pixel formats), the items in the list are usually separated by '|'.

The list of arguments can be quoted using the character "'" as initial and ending mark, and the character '\' for escaping the characters within the quoted text; otherwise the argument string is considered terminated when the next special character (belonging to the set "[]=;,") is encountered.

The name and arguments of the filter are optionally preceded and followed by a list of link labels. A link label allows one to name a link and associate it to a filter output or input pad. The preceding labels *in_link_1* ... *in_link_N*, are associated to the filter input pads, the following labels *out_link_1* ... *out_link_M*, are associated to the output pads.

When two link labels with the same name are found in the filtergraph, a link between the corresponding input and output pad is created.

If an output pad is not labelled, it is linked by default to the first unlabelled input pad of the next filter in the filterchain. For example in the filterchain

```
nullsrc, split[L1], [L2]overlay, nullsink
```

the split filter instance has two output pads, and the overlay filter instance two input pads. The first output pad of split is labelled "L1", the first input pad of overlay is labelled "L2", and the second output pad of split is linked to the second input pad of overlay, which are both unlabelled.

In a complete filterchain all the unlabelled filter input and output pads must be connected. A filtergraph is considered valid if all the filter input and output pads of all the filterchains are connected.

Libavfilter will automatically insert scale filters where format conversion is required. It is possible to specify swscale flags for those automatically inserted scalers by prepending `sws_flags=flags;` to the filtergraph description.

Here is a BNF description of the filtergraph syntax:

```
NAME             ::= sequence of alphanumeric characters and '_'
LINKLABEL        ::= "[" NAME "]"
LINKLABELS       ::= LINKLABEL [LINKLABELS]
FILTER_ARGUMENTS ::= sequence of chars (possibly quoted)
FILTER           ::= [LINKLABELS] NAME ["=" FILTER_ARGUMENTS] [LINKLABELS]
FILTERCHAIN      ::= FILTER [,FILTERCHAIN]
FILTERGRAPH      ::= [sws_flags=flags;] FILTERCHAIN [;FILTERGRAPH]
```

## 4.2 Notes on filtergraph escaping

Filtergraph description composition entails several levels of escaping. See (ffmpeg-utils)the "Quoting and escaping" section in the ffmpeg-utils(1) manual for more information about the employed escaping procedure.

A first level escaping affects the content of each filter option value, which may contain the special character `:` used to separate values, or one of the escaping characters `\'`.

A second level escaping affects the whole filter description, which may contain the escaping characters `\'` or the special characters `[],;` used by the filtergraph description.

Finally, when you specify a filtergraph on a shell commandline, you need to perform a third level escaping for the shell special characters contained within it.

For example, consider the following string to be embedded in the drawtext filter description `text` value:

```
this is a 'string': may contain one, or more, special characters
```

This string contains the `'` special escaping character, and the `:` special character, so it needs to be escaped in this way:

```
text=this is a \'string\'\: may contain one, or more, special characters
```

A second level of escaping is required when embedding the filter description in a filtergraph description, in order to escape all the filtergraph special characters. Thus the example above becomes:

```
drawtext=text=this is a \\\'string\\\'\\: may contain one\, or more\, special characters
```

(note that in addition to the `\'` escaping special characters, also `,` needs to be escaped).

Finally an additional level of escaping is needed when writing the filtergraph description in a shell command, which depends on the escaping rules of the adopted shell. For example, assuming that `\` is special and needs to be escaped with another `\`, the previous string will finally result in:

```
-vf "drawtext=text=this is a \\\\\\'string\\\\\\'\\\\: may contain one\\, or more\\, special characters"
```

# 5 Timeline editing

Some filters support a generic `enable` option. For the filters supporting timeline editing, this option can be set to an expression which is evaluated before sending a frame to the filter. If the evaluation is non-zero, the filter will be enabled, otherwise the frame will be sent unchanged to the next filter in the filtergraph.

The expression accepts the following values:

'`t`'

: timestamp expressed in seconds, NAN if the input timestamp is unknown

'`n`'

: sequential number of the input frame, starting from 0

'`pos`'

: the position in the file of the input frame, NAN if unknown

'`w`'
'`h`'

: width and height of the input frame if video

Additionally, these filters support an `enable` command that can be used to re-define the expression.

Like any other filtering option, the `enable` option follows the same rules.

For example, to enable a blur filter (smartblur) from 10 seconds to 3 minutes, and a curves filter starting at 3 seconds:

```
smartblur = enable='between(t,10,3*60)',
curves    = enable='gte(t,3)' : preset=cross_process
```

# 6 Audio Filters

When you configure your FFmpeg build, you can disable any of the existing filters using `--disable-filters`. The configure output will show the audio filters included in your build.

Below is a description of the currently available audio filters.

## 6.1 adelay

Delay one or more audio channels.

Samples in delayed channel are filled with silence.

The filter accepts the following option:

`delays`

> Set list of delays in milliseconds for each channel separated by '|'. At least one delay greater than 0 should be provided. Unused delays will be silently ignored. If number of given delays is smaller than number of channels all remaining channels will not be delayed.

### 6.1.1 Examples

- Delay first channel by 1.5 seconds, the third channel by 0.5 seconds and leave the second channel (and any other channels that may be present) unchanged.

  ```
  adelay=1500|0|500
  ```

## 6.2 aecho

Apply echoing to the input audio.

Echoes are reflected sound and can occur naturally amongst mountains (and sometimes large buildings) when talking or shouting; digital echo effects emulate this behaviour and are often used to help fill out the sound of a single instrument or vocal. The time difference between the original signal and the reflection is the `delay`, and the loudness of the reflected signal is the `decay`. Multiple echoes can have different delays and decays.

A description of the accepted parameters follows.

`in_gain`

> Set input gain of reflected signal. Default is `0.6`.

`out_gain`

Set output gain of reflected signal. Default is `0.3`.

`delays`

Set list of time intervals in milliseconds between original signal and reflections separated by ’|’. Allowed range for each `delay` is `(0 - 90000.0]`. Default is `1000`.

`decays`

Set list of loudnesses of reflected signals separated by ’|’. Allowed range for each `decay` is `(0 - 1.0]`. Default is `0.5`.

### 6.2.1 Examples

- Make it sound as if there are twice as many instruments as are actually playing:

```
aecho=0.8:0.88:60:0.4
```

- If delay is very short, then it sound like a (metallic) robot playing music:

```
aecho=0.8:0.88:6:0.4
```

- A longer delay will sound like an open air concert in the mountains:

```
aecho=0.8:0.9:1000:0.3
```

- Same as above but with one more mountain:

```
aecho=0.8:0.9:1000|1800:0.3|0.25
```

## 6.3 aeval

Modify an audio signal according to the specified expressions.

This filter accepts one or more expressions (one for each channel), which are evaluated and used to modify a corresponding audio signal.

It accepts the following parameters:

`exprs`

Set the ’|’-separated expressions list for each separate channel. If the number of input channels is greater than the number of expressions, the last specified expression is used for the remaining output channels.

`channel_layout, c`

Set output channel layout. If not specified, the channel layout is specified by the number of expressions. If set to ‘`same`’, it will use by default the same input channel layout.

Each expression in *exprs* can contain the following constants and functions:

`ch`

channel number of the current expression

`n`

number of the evaluated sample, starting from 0

`s`

sample rate

`t`

time of the evaluated sample expressed in seconds

`nb_in_channels`
`nb_out_channels`

input and output number of channels

`val(CH)`

the value of input channel with number *CH*

Note: this filter is slow. For faster processing you should use a dedicated filter.

### 6.3.1 Examples

- Half volume:

```
aeval=val(ch)/2:c=same
```

- Invert phase of the second channel:

```
aeval=val(0)|-val(1)
```

## 6.4 afade

Apply fade-in/out effect to input audio.

A description of the accepted parameters follows.

`type, t`

Specify the effect type, can be either `in` for fade-in, or `out` for a fade-out effect. Default is `in`.

`start_sample, ss`

Specify the number of the start sample for starting to apply the fade effect. Default is 0.

`nb_samples, ns`

Specify the number of samples for which the fade effect has to last. At the end of the fade-in effect the output audio will have the same volume as the input audio, at the end of the fade-out transition the output audio will be silence. Default is 44100.

`start_time, st`

Specify the start time of the fade effect. Default is 0. The value must be specified as a time duration; see (ffmpeg-utils)the Time duration section in the ffmpeg-utils(1) manual for the accepted syntax. If set this option is used instead of *start_sample*.

`duration, d`

Specify the duration of the fade effect. See (ffmpeg-utils)the Time duration section in the ffmpeg-utils(1) manual for the accepted syntax. At the end of the fade-in effect the output audio will have the same volume as the input audio, at the end of the fade-out transition the output audio will be silence. By default the duration is determined by *nb_samples*. If set this option is used instead of *nb_samples*.

`curve`

Set curve for fade transition.

It accepts the following values:

`tri`

select triangular, linear slope (default)

`qsin`

select quarter of sine wave

`hsin`

select half of sine wave

`esin`

select exponential sine wave

`log`

select logarithmic

`par`

select inverted parabola

`qua`

select quadratic

`cub`

select cubic

`squ`

select square root

`cbr`

select cubic root

### 6.4.1 Examples

- Fade in first 15 seconds of audio:

  ```
  afade=t=in:ss=0:d=15
  ```

- Fade out last 25 seconds of a 900 seconds audio:

  ```
  afade=t=out:st=875:d=25
  ```

## 6.5 aformat

Set output format constraints for the input audio. The framework will negotiate the most appropriate format to minimize conversions.

It accepts the following parameters:

`sample_fmts`

A ’|’-separated list of requested sample formats.

`sample_rates`

A ’|’-separated list of requested sample rates.

`channel_layouts`

A ’|’-separated list of requested channel layouts.

See (ffmpeg-utils)the Channel Layout section in the ffmpeg-utils(1) manual for the required syntax.

If a parameter is omitted, all values are allowed.

Force the output to either unsigned 8-bit or signed 16-bit stereo

```
aformat=sample_fmts=u8|s16:channel_layouts=stereo
```

## 6.6 allpass

Apply a two-pole all-pass filter with central frequency (in Hz) *frequency*, and filter-width *width*. An all-pass filter changes the audio’s frequency to phase relationship without changing its frequency to amplitude relationship.

The filter accepts the following options:

`frequency, f`

Set frequency in Hz.

`width_type`

Set method to specify band-width of filter.

`h`

Hz

`q`

Q-Factor

`o`

octave

`s`

slope

`width, w`

Specify the band-width of a filter in width_type units.

## 6.7 amerge

Merge two or more audio streams into a single multi-channel stream.

The filter accepts the following options:

`inputs`

Set the number of inputs. Default is 2.

If the channel layouts of the inputs are disjoint, and therefore compatible, the channel layout of the output will be set accordingly and the channels will be reordered as necessary. If the channel layouts of the inputs are not disjoint, the output will have all the channels of the first input then all the channels of the second input, in that order, and the channel layout of the output will be the default value corresponding to the total number of channels.

For example, if the first input is in 2.1 (FL+FR+LF) and the second input is FC+BL+BR, then the output will be in 5.1, with the channels in the following order: a1, a2, b1, a3, b2, b3 (a1 is the first channel of the first input, b1 is the first channel of the second input).

On the other hand, if both input are in stereo, the output channels will be in the default order: a1, a2, b1, b2, and the channel layout will be arbitrarily set to 4.0, which may or may not be the expected value.

All inputs must have the same sample rate, and format.

If inputs do not have the same duration, the output will stop with the shortest.

### 6.7.1 Examples

- Merge two mono files into a stereo stream:

  ```
  amovie=left.wav [l] ; amovie=right.mp3 [r] ; [l] [r] amerge
  ```

- Multiple merges assuming 1 video stream and 6 audio streams in `input.mkv`:

  ```
  ffmpeg -i input.mkv -filter_complex "[0:1][0:2][0:3][0:4][0:5][0:6] amerge=inputs=6" -c:a pcm_s16le output.mkv
  ```

## 6.8 amix

Mixes multiple audio inputs into a single output.

Note that this filter only supports float samples (the *amerge* and *pan* audio filters support many formats). If the *amix* input has integer samples then aresample will be automatically inserted to perform the conversion to float samples.

For example

```
ffmpeg -i INPUT1 -i INPUT2 -i INPUT3 -filter_complex amix=inputs=3:duration=first:dropout_transition=3 OUTPUT
```

will mix 3 input audio streams to a single output with the same duration as the first input and a dropout transition time of 3 seconds.

It accepts the following parameters:

`inputs`

The number of inputs. If unspecified, it defaults to 2.

`duration`

How to determine the end-of-stream.

`longest`

The duration of the longest input. (default)

`shortest`

The duration of the shortest input.

`first`

The duration of the first input.

`dropout_transition`

The transition time, in seconds, for volume renormalization when an input stream ends. The default value is 2 seconds.

## 6.9 anull

Pass the audio source unchanged to the output.

## 6.10 apad

Pad the end of an audio stream with silence.

This can be used together with `ffmpeg -shortest` to extend audio streams to the same length as the video stream.

A description of the accepted options follows.

`packet_size`

Set silence packet size. Default value is 4096.

`pad_len`

Set the number of samples of silence to add to the end. After the value is reached, the stream is terminated. This option is mutually exclusive with `whole_len`.

`whole_len`

Set the minimum total number of samples in the output audio stream. If the value is longer than the input audio length, silence is added to the end, until the value is reached. This option is mutually exclusive with `pad_len`.

If neither the `pad_len` nor the `whole_len` option is set, the filter will add silence to the end of the input stream indefinitely.

### 6.10.1 Examples

- Add 1024 samples of silence to the end of the input:

  ```
  apad=pad_len=1024
  ```

- Make sure the audio output will contain at least 10000 samples, pad the input with silence if required:

  ```
  apad=whole_len=10000
  ```

- Use `ffmpeg` to pad the audio input with silence, so that the video stream will always result the shortest and will be converted until the end in the output file when using the `shortest` option:

  ```
  ffmpeg -i VIDEO -i AUDIO -filter_complex "[1:0]apad" -shortest OUTPUT
  ```

## 6.11 aphaser

Add a phasing effect to the input audio.

A phaser filter creates series of peaks and troughs in the frequency spectrum. The position of the peaks and troughs are modulated so that they vary over time, creating a sweeping effect.

A description of the accepted parameters follows.

`in_gain`

Set input gain. Default is 0.4.

`out_gain`

Set output gain. Default is 0.74

`delay`

Set delay in milliseconds. Default is 3.0.

`decay`

Set decay. Default is 0.4.

`speed`

Set modulation speed in Hz. Default is 0.5.

`type`

Set modulation type. Default is triangular.

It accepts the following values:

‘`triangular, t`’
‘`sinusoidal, s`’

## 6.12 aresample

Resample the input audio to the specified parameters, using the libswresample library. If none are specified then the filter will automatically convert between its input and output.

This filter is also able to stretch/squeeze the audio data to make it match the timestamps or to inject silence / cut out audio to make it match the timestamps, do a combination of both or do neither.

The filter accepts the syntax [*sample_rate*:]*resampler_options*, where *sample_rate* expresses a sample rate and *resampler_options* is a list of *key*=*value* pairs, separated by ":". See the ffmpeg-resampler manual for the complete list of supported options.

### 6.12.1 Examples

- Resample the input audio to 44100Hz:

```
aresample=44100
```

- Stretch/squeeze samples to the given timestamps, with a maximum of 1000 samples per second compensation:

```
aresample=async=1000
```

## 6.13 asetnsamples

Set the number of samples per each output audio frame.

The last output packet may contain a different number of samples, as the filter will flush all the remaining samples when the input audio signal its end.

The filter accepts the following options:

`nb_out_samples, n`

> Set the number of frames per each output audio frame. The number is intended as the number of samples *per each channel*. Default value is 1024.

`pad, p`

> If set to 1, the filter will pad the last audio frame with zeroes, so that the last frame will contain the same number of samples as the previous ones. Default value is 1.

For example, to set the number of per-frame samples to 1234 and disable padding for the last frame, use:

```
asetnsamples=n=1234:p=0
```

## 6.14 asetrate

Set the sample rate without altering the PCM data. This will result in a change of speed and pitch.

The filter accepts the following options:

`sample_rate, r`

> Set the output sample rate. Default is 44100 Hz.

## 6.15 ashowinfo

Show a line containing various information for each input audio frame. The input audio is not modified.

The shown line contains a sequence of key/value pairs of the form *key*:*value*.

The following values are shown in the output:

`n`

> The (sequential) number of the input frame, starting from 0.

`pts`

The presentation timestamp of the input frame, in time base units; the time base depends on the filter input pad, and is usually 1/*sample_rate*.

`pts_time`

The presentation timestamp of the input frame in seconds.

`pos`

position of the frame in the input stream, -1 if this information in unavailable and/or meaningless (for example in case of synthetic audio)

`fmt`

The sample format.

`chlayout`

The channel layout.

`rate`

The sample rate for the audio frame.

`nb_samples`

The number of samples (per channel) in the frame.

`checksum`

The Adler-32 checksum (printed in hexadecimal) of the audio data. For planar audio, the data is treated as if all the planes were concatenated.

`plane_checksums`

A list of Adler-32 checksums for each data plane.

## 6.16 astats

Display time domain statistical information about the audio channels. Statistics are calculated and displayed for each audio channel and, where applicable, an overall figure is also given.

It accepts the following option:

`length`

Short window length in seconds, used for peak and trough RMS measurement. Default is `0.05` (50 miliseconds). Allowed range is `[0.1 - 10]`.

A description of each shown parameter follows:

`DC offset`

Mean amplitude displacement from zero.

`Min level`

Minimal sample level.

`Max level`

Maximal sample level.

`Peak level dB`
`RMS level dB`

Standard peak and RMS level measured in dBFS.

`RMS peak dB`
`RMS trough dB`

Peak and trough values for RMS level measured over a short window.

`Crest factor`

Standard ratio of peak to RMS level (note: not in dB).

`Flat factor`

Flatness (i.e. consecutive samples with the same value) of the signal at its peak levels (i.e. either *Min level* or *Max level*).

`Peak count`

Number of occasions (not the number of samples) that the signal attained either *Min level* or *Max level*.

## 6.17 astreamsync

Forward two audio streams and control the order the buffers are forwarded.

The filter accepts the following options:

`expr, e`

Set the expression deciding which stream should be forwarded next: if the result is negative, the first stream is forwarded; if the result is positive or zero, the second stream is forwarded. It can use the following variables:

*b1 b2*

number of buffers forwarded so far on each stream

*s1 s2*

number of samples forwarded so far on each stream

*t1 t2*

current timestamp of each stream

The default value is `t1-t2`, which means to always forward the stream that has a smaller timestamp.

### 6.17.1 Examples

Stress-test `amerge` by randomly sending buffers on the wrong input, while avoiding too much of a desynchronization:

```
amovie=file.ogg [a] ; amovie=file.mp3 [b] ;
[a] [b] astreamsync=(2*random(1))-1+tanh(5*(t1-t2)) [a2] [b2] ;
[a2] [b2] amerge
```

## 6.18 asyncts

Synchronize audio data with timestamps by squeezing/stretching it and/or dropping samples/adding silence when needed.

This filter is not built by default, please use aresample to do squeezing/stretching.

It accepts the following parameters:

`compensate`

Enable stretching/squeezing the data to make it match the timestamps. Disabled by default. When disabled, time gaps are covered with silence.

`min_delta`

The minimum difference between timestamps and audio data (in seconds) to trigger adding/dropping samples. The default value is 0.1. If you get an imperfect sync with this filter, try setting this parameter to 0.

`max_comp`

The maximum compensation in samples per second. Only relevant with compensate=1. The default value is 500.

`first_pts`

Assume that the first PTS should be this value. The time base is 1 / sample rate. This allows for padding/trimming at the start of the stream. By default, no assumption is made about the first frame's expected PTS, so no padding or trimming is done. For example, this could be set to 0 to pad the beginning with silence if an audio stream starts after the video stream or to trim any samples with a negative PTS due to encoder delay.

## 6.19 atempo

Adjust audio tempo.

The filter accepts exactly one parameter, the audio tempo. If not specified then the filter will assume nominal 1.0 tempo. Tempo must be in the [0.5, 2.0] range.

### 6.19.1 Examples

- Slow down audio to 80% tempo:

```
atempo=0.8
```

- To speed up audio to 125% tempo:

```
atempo=1.25
```

## 6.20 atrim

Trim the input so that the output contains one continuous subpart of the input.

It accepts the following parameters:

`start`

Timestamp (in seconds) of the start of the section to keep. I.e. the audio sample with the timestamp *start* will be the first sample in the output.

`end`

Specify time of the first audio sample that will be dropped, i.e. the audio sample immediately preceding the one with the timestamp *end* will be the last sample in the output.

`start_pts`

Same as *start*, except this option sets the start timestamp in samples instead of seconds.

`end_pts`

Same as *end*, except this option sets the end timestamp in samples instead of seconds.

`duration`

The maximum duration of the output in seconds.

`start_sample`

The number of the first sample that should be output.

`end_sample`

The number of the first sample that should be dropped.

`start`, `end`, and `duration` are expressed as time duration specifications; see (ffmpeg-utils)the Time duration section in the ffmpeg-utils(1) manual.

Note that the first two sets of the start/end options and the `duration` option look at the frame timestamp, while the _sample options simply count the samples that pass through the filter. So start/end_pts and start/end_sample will give different results when the timestamps are wrong, inexact or do not start at zero. Also note that this filter does not modify the timestamps. If you wish to have the output timestamps start at zero, insert the asetpts filter after the atrim filter.

If multiple start or end options are set, this filter tries to be greedy and keep all samples that match at least one of the specified constraints. To keep only the part that matches all the constraints at once, chain multiple atrim filters.

The defaults are such that all the input is kept. So it is possible to set e.g. just the end values to keep everything before the specified time.

Examples:

- Drop everything except the second minute of input:

  ```
  ffmpeg -i INPUT -af atrim=60:120
  ```

- Keep only the first 1000 samples:

  ```
  ffmpeg -i INPUT -af atrim=end_sample=1000
  ```

## 6.21 bandpass

Apply a two-pole Butterworth band-pass filter with central frequency *frequency*, and (3dB-point) band-width width. The *csg* option selects a constant skirt gain (peak gain = Q) instead of the default: constant 0dB peak gain. The filter roll off at 6dB per octave (20dB per decade).

The filter accepts the following options:

`frequency, f`

Set the filter's central frequency. Default is `3000`.

`csg`

Constant skirt gain if set to 1. Defaults to 0.

`width_type`

Set method to specify band-width of filter.

`h`

Hz

`q`

Q-Factor

`o`

octave

`s`

slope

`width, w`

Specify the band-width of a filter in width_type units.

## 6.22 bandreject

Apply a two-pole Butterworth band-reject filter with central frequency *frequency*, and (3dB-point) band-width *width*. The filter roll off at 6dB per octave (20dB per decade).

The filter accepts the following options:

`frequency, f`

Set the filter's central frequency. Default is `3000`.

`width_type`

Set method to specify band-width of filter.

`h`

Hz

`q`

Q-Factor

`o`

octave

`s`

slope

`width, w`

Specify the band-width of a filter in width_type units.

## 6.23 bass

Boost or cut the bass (lower) frequencies of the audio using a two-pole shelving filter with a response similar to that of a standard hi-fi's tone-controls. This is also known as shelving equalisation (EQ).

The filter accepts the following options:

`gain, g`

Give the gain at 0 Hz. Its useful range is about -20 (for a large cut) to +20 (for a large boost). Beware of clipping when using a positive gain.

`frequency, f`

Set the filter's central frequency and so can be used to extend or reduce the frequency range to be boosted or cut. The default value is `100` Hz.

`width_type`

Set method to specify band-width of filter.

`h`

Hz

`q`

Q-Factor

`o`

octave

`s`

slope

`width, w`

Determine how steep is the filter’s shelf transition.

## 6.24 biquad

Apply a biquad IIR filter with the given coefficients. Where *b0*, *b1*, *b2* and *a0*, *a1*, *a2* are the numerator and denominator coefficients respectively.

## 6.25 bs2b

Bauer stereo to binaural transformation, which improves headphone listening of stereo audio records.

It accepts the following parameters:

`profile`

Pre-defined crossfeed level.

`default`

Default level (fcut=700, feed=50).

`cmoy`

Chu Moy circuit (fcut=700, feed=60).

`jmeier`

Jan Meier circuit (fcut=650, feed=95).

`fcut`

Cut frequency (in Hz).

`feed`

Feed level (in Hz).

## 6.26 channelmap

Remap input channels to new locations.

It accepts the following parameters:

`channel_layout`

The channel layout of the output stream.

`map`

Map channels from input to output. The argument is a '|'-separated list of mappings, each in the *`in_channel-out_channel`* or *in_channel* form. *in_channel* can be either the name of the input channel (e.g. FL for front left) or its index in the input channel layout. *out_channel* is the name of the output channel or its index in the output channel layout. If *out_channel* is not given then it is implicitly an index, starting with zero and increasing by one for each mapping.

If no mapping is present, the filter will implicitly map input channels to output channels, preserving indices.

For example, assuming a 5.1+downmix input MOV file,

```
ffmpeg -i in.mov -filter 'channelmap=map=DL-FL|DR-FR' out.wav
```

will create an output WAV file tagged as stereo from the downmix channels of the input.

To fix a 5.1 WAV improperly encoded in AAC's native channel order

```
ffmpeg -i in.wav -filter 'channelmap=1|2|0|5|3|4:channel_layout=5.1' out.wav
```

## 6.27 channelsplit

Split each channel from an input audio stream into a separate output stream.

It accepts the following parameters:

`channel_layout`

The channel layout of the input stream. The default is "stereo".

For example, assuming a stereo input MP3 file,

```
ffmpeg -i in.mp3 -filter_complex channelsplit out.mkv
```

will create an output Matroska file with two audio streams, one containing only the left channel and the other the right channel.

Split a 5.1 WAV file into per-channel files:

```
ffmpeg -i in.wav -filter_complex
'channelsplit=channel_layout=5.1[FL][FR][FC][LFE][SL][SR]'
-map '[FL]' front_left.wav -map '[FR]' front_right.wav -map '[FC]'
front_center.wav -map '[LFE]' lfe.wav -map '[SL]' side_left.wav -map '[SR]'
side_right.wav
```

## 6.28 compand

Compress or expand the audio's dynamic range.

It accepts the following parameters:

`attacks`
`decays`

A list of times in seconds for each channel over which the instantaneous level of the input signal is averaged to determine its volume. *attacks* refers to increase of volume and *decays* refers to decrease of volume. For most situations, the attack time (response to the audio getting louder) should be shorter than the decay time, because the human ear is more sensitive to sudden loud audio than sudden soft audio. A typical value for attack is 0.3 seconds and a typical value for decay is 0.8 seconds.

`points`

A list of points for the transfer function, specified in dB relative to the maximum possible signal amplitude. Each key points list must be defined using the following syntax: `x0/y0|x1/y1|x2/y2|....` or `x0/y0 x1/y1 x2/y2 ....`

The input values must be in strictly increasing order but the transfer function does not have to be monotonically rising. The point `0/0` is assumed but may be overridden (by `0/out-dBn`). Typical values for the transfer function are `-70/-70|-60/-20`.

`soft-knee`

Set the curve radius in dB for all joints. It defaults to 0.01.

`gain`

Set the additional gain in dB to be applied at all points on the transfer function. This allows for easy adjustment of the overall gain. It defaults to 0.

`volume`

Set an initial volume, in dB, to be assumed for each channel when filtering starts. This permits the user to supply a nominal level initially, so that, for example, a very large gain is not applied to initial signal levels before the companding has begun to operate. A typical value for audio which is initially quiet is -90 dB. It defaults to 0.

`delay`

Set a delay, in seconds. The input audio is analyzed immediately, but audio is delayed before being fed to the volume adjuster. Specifying a delay approximately equal to the attack/decay times allows the filter to effectively operate in predictive rather than reactive mode. It defaults to 0.

### 6.28.1 Examples

- Make music with both quiet and loud passages suitable for listening to in a noisy environment:

  ```
  compand=.3|.3:1|1:-90/-60|-60/-40|-40/-30|-20/-20:6:0:-90:0.2
  ```

- A noise gate for when the noise is at a lower level than the signal:

  ```
  compand=.1|.1:.2|.2:-900/-900|-50.1/-900|-50/-50:.01:0:-90:.1
  ```

- Here is another noise gate, this time for when the noise is at a higher level than the signal (making it, in some ways, similar to squelch):

  ```
  compand=.1|.1:.1|.1:-45.1/-45.1|-45/-900|0/-900:.01:45:-90:.1
  ```

## 6.29 earwax

Make audio easier to listen to on headphones.

This filter adds ‘cues’ to 44.1kHz stereo (i.e. audio CD format) audio so that when listened to on headphones the stereo image is moved from inside your head (standard for headphones) to outside and in front of the listener (standard for speakers).

Ported from SoX.

## 6.30 equalizer

Apply a two-pole peaking equalisation (EQ) filter. With this filter, the signal-level at and around a selected frequency can be increased or decreased, whilst (unlike bandpass and bandreject filters) that at all other frequencies is unchanged.

In order to produce complex equalisation curves, this filter can be given several times, each with a different central frequency.

The filter accepts the following options:

`frequency, f`

Set the filter’s central frequency in Hz.

`width_type`

Set method to specify band-width of filter.

`h`

Hz

`q`

Q-Factor

`o`

octave

`s`

slope

`width, w`

Specify the band-width of a filter in width_type units.

`gain, g`

Set the required gain or attenuation in dB. Beware of clipping when using a positive gain.

### 6.30.1 Examples

- Attenuate 10 dB at 1000 Hz, with a bandwidth of 200 Hz:

```
equalizer=f=1000:width_type=h:width=200:g=-10
```

- Apply 2 dB gain at 1000 Hz with Q 1 and attenuate 5 dB at 100 Hz with Q 2:

```
equalizer=f=1000:width_type=q:width=1:g=2,equalizer=f=100:width_type=q:width=2:g=-5
```

## 6.31 flanger

Apply a flanging effect to the audio.

The filter accepts the following options:

`delay`

Set base delay in milliseconds. Range from 0 to 30. Default value is 0.

`depth`

Set added swep delay in milliseconds. Range from 0 to 10. Default value is 2.

`regen`

Set percentage regeneneration (delayed signal feedback). Range from -95 to 95. Default value is 0.

`width`

Set percentage of delayed signal mixed with original. Range from 0 to 100. Default valu is 71.

`speed`

Set sweeps per second (Hz). Range from 0.1 to 10. Default value is 0.5.

`shape`

Set swept wave shape, can be *triangular* or *sinusoidal*. Default value is *sinusoidal*.

`phase`

Set swept wave percentage-shift for multi channel. Range from 0 to 100. Default value is 25.

`interp`

Set delay-line interpolation, *linear* or *quadratic*. Default is *linear*.

## 6.32 highpass

Apply a high-pass filter with 3dB point frequency. The filter can be either single-pole, or double-pole (the default). The filter roll off at 6dB per pole per octave (20dB per pole per decade).

The filter accepts the following options:

`frequency, f`

Set frequency in Hz. Default is 3000.

`poles, p`

Set number of poles. Default is 2.

`width_type`

Set method to specify band-width of filter.

`h`

Hz

`q`

Q-Factor

`o`

octave

`s`

slope

`width, w`

Specify the band-width of a filter in width_type units. Applies only to double-pole filter. The default is 0.707q and gives a Butterworth response.

## 6.33 join

Join multiple input streams into one multi-channel stream.

It accepts the following parameters:

`inputs`

The number of input streams. It defaults to 2.

`channel_layout`

The desired output channel layout. It defaults to stereo.

`map`

Map channels from inputs to output. The argument is a '|'-separated list of mappings, each in the *`input_idx.in_channel-out_channel`* form. *input_idx* is the 0-based index of the input stream. *in_channel* can be either the name of the input channel (e.g. FL for front left) or its index in the specified input stream. *out_channel* is the name of the output channel.

The filter will attempt to guess the mappings when they are not specified explicitly. It does so by first trying to find an unused matching input channel and if that fails it picks the first unused input channel.

Join 3 inputs (with properly set channel layouts):

```
ffmpeg -i INPUT1 -i INPUT2 -i INPUT3 -filter_complex join=inputs=3 OUTPUT
```

Build a 5.1 output from 6 single-channel streams:

```
ffmpeg -i fl -i fr -i fc -i sl -i sr -i lfe -filter_complex
'join=inputs=6:channel_layout=5.1:map=0.0-FL|1.0-FR|2.0-FC|3.0-SL|4.0-SR|5.0-LFE'
out
```

## 6.34 ladspa

Load a LADSPA (Linux Audio Developer's Simple Plugin API) plugin.

To enable compilation of this filter you need to configure FFmpeg with `--enable-ladspa`.

`file, f`

> Specifies the name of LADSPA plugin library to load. If the environment variable `LADSPA_PATH` is defined, the LADSPA plugin is searched in each one of the directories specified by the colon separated list in `LADSPA_PATH`, otherwise in the standard LADSPA paths, which are in this order: `HOME/.ladspa/lib/`, `/usr/local/lib/ladspa/`, `/usr/lib/ladspa/`.

`plugin, p`

> Specifies the plugin within the library. Some libraries contain only one plugin, but others contain many of them. If this is not set filter will list all available plugins within the specified library.

`controls, c`

> Set the '|' separated list of controls which are zero or more floating point values that determine the behavior of the loaded plugin (for example delay, threshold or gain). Controls need to be defined using the following syntax: c0=*value0*|c1=*value1*|c2=*value2*|..., where *valuei* is the value set on the *i*-th control. If `controls` is set to `help`, all available controls and their valid ranges are printed.

`sample_rate, s`

> Specify the sample rate, default to 44100. Only used if plugin have zero inputs.

`nb_samples, n`

> Set the number of samples per channel per each output frame, default is 1024. Only used if plugin have zero inputs.

`duration, d`

> Set the minimum duration of the sourced audio. See (ffmpeg-utils)the Time duration section in the ffmpeg-utils(1) manual for the accepted syntax. Note that the resulting duration may be greater than the specified duration, as the generated audio is always cut at the end of a complete frame. If not specified, or the expressed duration is negative, the audio is supposed to be generated forever. Only used if plugin have zero inputs.

### 6.34.1 Examples

- List all available plugins within amp (LADSPA example plugin) library:

  ```
  ladspa=file=amp
  ```

- List all available controls and their valid ranges for `vcf_notch` plugin from `VCF` library:

  ```
  ladspa=f=vcf:p=vcf_notch:c=help
  ```

- Simulate low quality audio equipment using `Computer Music Toolkit` (CMT) plugin library:

  ```
  ladspa=file=cmt:plugin=lofi:controls=c0=22|c1=12|c2=12
  ```

- Add reverberation to the audio using TAP-plugins (Tom’s Audio Processing plugins):

  ```
  ladspa=file=tap_reverb:tap_reverb
  ```

- Generate white noise, with 0.2 amplitude:

  ```
  ladspa=file=cmt:noise_source_white:c=c0=.2
  ```

- Generate 20 bpm clicks using plugin `C* Click - Metronome` from the `C* Audio Plugin Suite` (CAPS) library:

  ```
  ladspa=file=caps:Click:c=c1=20’
  ```

- Apply `C* Eq10X2 - Stereo 10-band equaliser` effect:

  ```
  ladspa=caps:Eq10X2:c=c0=-48|c9=-24|c3=12|c4=2
  ```

### 6.34.2 Commands

This filter supports the following commands:

`cN`

Modify the *N*-th control value.

If the specified value is not valid, it is ignored and prior one is kept.

## 6.35 lowpass

Apply a low-pass filter with 3dB point frequency. The filter can be either single-pole or double-pole (the default). The filter roll off at 6dB per pole per octave (20dB per pole per decade).

The filter accepts the following options:

`frequency, f`

Set frequency in Hz. Default is 500.

`poles, p`

Set number of poles. Default is 2.

`width_type`

Set method to specify band-width of filter.

`h`

Hz

`q`

Q-Factor

`o`

octave

`s`

slope

`width, w`

Specify the band-width of a filter in width_type units. Applies only to double-pole filter. The default is 0.707q and gives a Butterworth response.

## 6.36 pan

Mix channels with specific gain levels. The filter accepts the output channel layout followed by a set of channels definitions.

This filter is also designed to efficiently remap the channels of an audio stream.

The filter accepts parameters of the form: "*l*|*outdef*|*outdef*|..."

`l`

output channel layout or number of channels

`outdef`

output channel specification, of the form: "*out_name*=[*gain**]*in_name*[+[*gain**]*in_name*...]"

`out_name`

output channel to define, either a channel name (FL, FR, etc.) or a channel number (c0, c1, etc.)

`gain`

multiplicative coefficient for the channel, 1 leaving the volume unchanged

`in_name`

input channel to use, see out_name for details; it is not possible to mix named and numbered input channels

If the ‘=’ in a channel specification is replaced by ‘<’, then the gains for that specification will be renormalized so that the total is 1, thus avoiding clipping noise.

### 6.36.1 Mixing examples

For example, if you want to down-mix from stereo to mono, but with a bigger factor for the left channel:

```
pan=1c|c0=0.9*c0+0.1*c1
```

A customized down-mix to stereo that works automatically for 3-, 4-, 5- and 7-channels surround:

```
pan=stereo| FL < FL + 0.5*FC + 0.6*BL + 0.6*SL | FR < FR + 0.5*FC + 0.6*BR + 0.6*SR
```

Note that `ffmpeg` integrates a default down-mix (and up-mix) system that should be preferred (see "-ac" option) unless you have very specific needs.

### 6.36.2 Remapping examples

The channel remapping will be effective if, and only if:

- gain coefficients are zeroes or ones,
- only one input per channel output,

If all these conditions are satisfied, the filter will notify the user ("Pure channel mapping detected"), and use an optimized and lossless method to do the remapping.

For example, if you have a 5.1 source and want a stereo audio stream by dropping the extra channels:

```
pan="stereo| c0=FL | c1=FR"
```

Given the same source, you can also switch front left and front right channels and keep the input channel layout:

```
pan="5.1| c0=c1 | c1=c0 | c2=c2 | c3=c3 | c4=c4 | c5=c5"
```

If the input is a stereo audio stream, you can mute the front left channel (and still keep the stereo channel layout) with:

```
pan="stereo|c1=c1"
```

Still with a stereo audio stream input, you can copy the right channel in both front left and right:

```
pan="stereo| c0=FR | c1=FR"
```

## 6.37 replaygain

ReplayGain scanner filter. This filter takes an audio stream as an input and outputs it unchanged. At end of filtering it displays `track_gain` and `track_peak`.

## 6.38 resample

Convert the audio sample format, sample rate and channel layout. It is not meant to be used directly.

## 6.39 silencedetect

Detect silence in an audio stream.

This filter logs a message when it detects that the input audio volume is less or equal to a noise tolerance value for a duration greater or equal to the minimum detected noise duration.

The printed times and duration are expressed in seconds.

The filter accepts the following options:

`duration, d`

Set silence duration until notification (default is 2 seconds).

`noise, n`

Set noise tolerance. Can be specified in dB (in case "dB" is appended to the specified value) or amplitude ratio. Default is -60dB, or 0.001.

### 6.39.1 Examples

- Detect 5 seconds of silence with -50dB noise tolerance:

  ```
  silencedetect=n=-50dB:d=5
  ```

- Complete example with `ffmpeg` to detect silence with 0.0001 noise tolerance in `silence.mp3`:

  ```
  ffmpeg -i silence.mp3 -af silencedetect=noise=0.0001 -f null -
  ```

## 6.40 silenceremove

Remove silence from the beginning, middle or end of the audio.

The filter accepts the following options:

`start_periods`

This value is used to indicate if audio should be trimmed at beginning of the audio. A value of zero indicates no silence should be trimmed from the beginning. When specifying a non-zero value, it trims audio up until it finds non-silence. Normally, when trimming silence from beginning of audio the *start_periods* will be `1` but it can be increased to higher values to trim all audio up to specific count of non-silence periods. Default value is `0`.

`start_duration`

Specify the amount of time that non-silence must be detected before it stops trimming audio. By increasing the duration, bursts of noises can be treated as silence and trimmed off. Default value is `0`.

`start_threshold`

This indicates what sample value should be treated as silence. For digital audio, a value of `0` may be fine but for audio recorded from analog, you may wish to increase the value to account for background noise. Can be specified in dB (in case "dB" is appended to the specified value) or amplitude ratio. Default value is `0`.

`stop_periods`

Set the count for trimming silence from the end of audio. To remove silence from the middle of a file, specify a *stop_periods* that is negative. This value is then threated as a positive value and is used to indicate the effect should restart processing as specified by *start_periods*, making it suitable for removing periods of silence in the middle of the audio. Default value is `0`.

`stop_duration`

Specify a duration of silence that must exist before audio is not copied any more. By specifying a higher duration, silence that is wanted can be left in the audio. Default value is `0`.

`stop_threshold`

This is the same as `start_threshold` but for trimming silence from the end of audio. Can be specified in dB (in case "dB" is appended to the specified value) or amplitude ratio. Default value is `0`.

`leave_silence`

This indicate that *stop_duration* length of audio should be left intact at the beginning of each period of silence. For example, if you want to remove long pauses between words but do not want to remove the pauses completely. Default value is `0`.

### 6.40.1 Examples

- The following example shows how this filter can be used to start a recording that does not contain the delay at the start which usually occurs between pressing the record button and the start of the performance:

  ```
  silenceremove=1:5:0.02
  ```

## 6.41 treble

Boost or cut treble (upper) frequencies of the audio using a two-pole shelving filter with a response similar to that of a standard hi-fi's tone-controls. This is also known as shelving equalisation (EQ).

The filter accepts the following options:

`gain, g`

Give the gain at whichever is the lower of ~22 kHz and the Nyquist frequency. Its useful range is about -20 (for a large cut) to +20 (for a large boost). Beware of clipping when using a positive gain.

`frequency, f`

Set the filter's central frequency and so can be used to extend or reduce the frequency range to be boosted or cut. The default value is `3000` Hz.

`width_type`

Set method to specify band-width of filter.

`h`

Hz

`q`

Q-Factor

`o`

octave

`s`

slope

`width, w`

Determine how steep is the filter’s shelf transition.

## 6.42 volume

Adjust the input audio volume.

It accepts the following parameters:

`volume`

Set audio volume expression.

Output values are clipped to the maximum value.

The output audio volume is given by the relation:

*`output_volume = volume * input_volume`*

The default value for *volume* is "1.0".

`precision`

This parameter represents the mathematical precision.

It determines which input sample formats will be allowed, which affects the precision of the volume scaling.

`fixed`

8-bit fixed-point; this limits input sample format to U8, S16, and S32.

`float`

32-bit floating-point; this limits input sample format to FLT. (default)

`double`

64-bit floating-point; this limits input sample format to DBL.

`replaygain`

Choose the behaviour on encountering ReplayGain side data in input frames.

`drop`

Remove ReplayGain side data, ignoring its contents (the default).

`ignore`

Ignore ReplayGain side data, but leave it in the frame.

`track`

Prefer the track gain, if present.

`album`

Prefer the album gain, if present.

`replaygain_preamp`

Pre-amplification gain in dB to apply to the selected replaygain gain.

Default value for *replaygain_preamp* is 0.0.

`eval`

Set when the volume expression is evaluated.

It accepts the following values:

‘`once`’

only evaluate expression once during the filter initialization, or when the ‘`volume`’ command is sent

‘`frame`’

evaluate expression for each incoming frame

Default value is ‘`once`’.

The volume expression can contain the following parameters.

`n`

frame number (starting at zero)

`nb_channels`

number of channels

`nb_consumed_samples`

number of samples consumed by the filter

`nb_samples`

number of samples in the current frame

`pos`

original frame position in the file

`pts`

frame PTS

`sample_rate`

sample rate

`startpts`

PTS at start of stream

`startt`

time at start of stream

`t`

frame time

`tb`

timestamp timebase

`volume`

last set volume value

Note that when `eval` is set to ‘`once`’ only the *sample_rate* and *tb* variables are available, all other variables will evaluate to NAN.

### 6.42.1 Commands

This filter supports the following commands:

`volume`

Modify the volume expression. The command accepts the same syntax of the corresponding option.

If the specified expression is not valid, it is kept at its current value.

`replaygain_noclip`

Prevent clipping by limiting the gain applied.

Default value for *replaygain_noclip* is 1.

### 6.42.2 Examples

- Halve the input audio volume:

  ```
  volume=volume=0.5
  volume=volume=1/2
  volume=volume=-6.0206dB
  ```

  In all the above example the named key for `volume` can be omitted, for example like in:

  ```
  volume=0.5
  ```

- Increase input audio power by 6 decibels using fixed-point precision:

  ```
  volume=volume=6dB:precision=fixed
  ```

- Fade volume after time 10 with an annihilation period of 5 seconds:

  ```
  volume='if(lt(t,10),1,max(1-(t-10)/5,0))':eval=frame
  ```

## 6.43 volumedetect

Detect the volume of the input video.

The filter has no parameters. The input is not modified. Statistics about the volume will be printed in the log when the input stream end is reached.

In particular it will show the mean volume (root mean square), maximum volume (on a per-sample basis), and the beginning of a histogram of the registered volume values (from the maximum value to a cumulated 1/1000 of the samples).

All volumes are in decibels relative to the maximum PCM value.

### 6.43.1 Examples

Here is an excerpt of the output:

```
[Parsed_volumedetect_0  0xa23120] mean_volume: -27 dB
[Parsed_volumedetect_0  0xa23120] max_volume: -4 dB
[Parsed_volumedetect_0  0xa23120] histogram_4db: 6
[Parsed_volumedetect_0  0xa23120] histogram_5db: 62
[Parsed_volumedetect_0  0xa23120] histogram_6db: 286
[Parsed_volumedetect_0  0xa23120] histogram_7db: 1042
[Parsed_volumedetect_0  0xa23120] histogram_8db: 2551
[Parsed_volumedetect_0  0xa23120] histogram_9db: 4609
[Parsed_volumedetect_0  0xa23120] histogram_10db: 8409
```

It means that:

- The mean square energy is approximately -27 dB, or 10^-2.7.
- The largest sample is at -4 dB, or more precisely between -4 dB and -5 dB.
- There are 6 samples at -4 dB, 62 at -5 dB, 286 at -6 dB, etc.

In other words, raising the volume by +4 dB does not cause any clipping, raising it by +5 dB causes clipping for 6 samples, etc.

# 7 Audio Sources

Below is a description of the currently available audio sources.

## 7.1 abuffer

Buffer audio frames, and make them available to the filter chain.

This source is mainly intended for a programmatic use, in particular through the interface defined in `libavfilter/asrc_abuffer.h`.

It accepts the following parameters:

`time_base`

> The timebase which will be used for timestamps of submitted frames. It must be either a floating-point number or in *numerator*/*denominator* form.

`sample_rate`

> The sample rate of the incoming audio buffers.

`sample_fmt`

> The sample format of the incoming audio buffers. Either a sample format name or its corresponging integer representation from the enum AVSampleFormat in `libavutil/samplefmt.h`

`channel_layout`

> The channel layout of the incoming audio buffers. Either a channel layout name from channel_layout_map in `libavutil/channel_layout.c` or its corresponding integer representation from the AV_CH_LAYOUT_* macros in `libavutil/channel_layout.h`

`channels`

> The number of channels of the incoming audio buffers. If both *channels* and *channel_layout* are specified, then they must be consistent.

### 7.1.1 Examples

```
abuffer=sample_rate=44100:sample_fmt=s16p:channel_layout=stereo
```

will instruct the source to accept planar 16bit signed stereo at 44100Hz. Since the sample format with name "s16p" corresponds to the number 6 and the "stereo" channel layout corresponds to the value 0x3, this is equivalent to:

```
abuffer=sample_rate=44100:sample_fmt=6:channel_layout=0x3
```

## 7.2 aevalsrc

Generate an audio signal specified by an expression.

This source accepts in input one or more expressions (one for each channel), which are evaluated and used to generate a corresponding audio signal.

This source accepts the following options:

`exprs`

> Set the '|'-separated expressions list for each separate channel. In case the `channel_layout` option is not specified, the selected channel layout depends on the number of provided expressions. Otherwise the last specified expression is applied to the remaining output channels.

`channel_layout, c`

> Set the channel layout. The number of channels in the specified layout must be equal to the number of specified expressions.

`duration, d`

> Set the minimum duration of the sourced audio. See (ffmpeg-utils)the Time duration section in the ffmpeg-utils(1) manual for the accepted syntax. Note that the resulting duration may be greater than the specified duration, as the generated audio is always cut at the end of a complete frame.
>
> If not specified, or the expressed duration is negative, the audio is supposed to be generated forever.

`nb_samples, n`

> Set the number of samples per channel per each output frame, default to 1024.

`sample_rate, s`

> Specify the sample rate, default to 44100.

Each expression in *exprs* can contain the following constants:

`n`

number of the evaluated sample, starting from 0

`t`

time of the evaluated sample expressed in seconds, starting from 0

`s`

sample rate

### 7.2.1 Examples

- Generate silence:

  ```
  aevalsrc=0
  ```

- Generate a sin signal with frequency of 440 Hz, set sample rate to 8000 Hz:

  ```
  aevalsrc="sin(440*2*PI*t):s=8000"
  ```

- Generate a two channels signal, specify the channel layout (Front Center + Back Center) explicitly:

  ```
  aevalsrc="sin(420*2*PI*t)|cos(430*2*PI*t):c=FC|BC"
  ```

- Generate white noise:

  ```
  aevalsrc="-2+random(0)"
  ```

- Generate an amplitude modulated signal:

  ```
  aevalsrc="sin(10*2*PI*t)*sin(880*2*PI*t)"
  ```

- Generate 2.5 Hz binaural beats on a 360 Hz carrier:

  ```
  aevalsrc="0.1*sin(2*PI*(360-2.5/2)*t) | 0.1*sin(2*PI*(360+2.5/2)*t)"
  ```

## 7.3 anullsrc

The null audio source, return unprocessed audio frames. It is mainly useful as a template and to be employed in analysis / debugging tools, or as the source for filters which ignore the input data (for example the sox synth filter).

This source accepts the following options:

`channel_layout, cl`

Specifies the channel layout, and can be either an integer or a string representing a channel layout. The default value of *channel_layout* is "stereo".

Check the channel_layout_map definition in `libavutil/channel_layout.c` for the mapping between strings and channel layout values.

`sample_rate, r`

Specifies the sample rate, and defaults to 44100.

`nb_samples, n`

Set the number of samples per requested frames.

### 7.3.1 Examples

- Set the sample rate to 48000 Hz and the channel layout to AV_CH_LAYOUT_MONO.

```
anullsrc=r=48000:cl=4
```

- Do the same operation with a more obvious syntax:

```
anullsrc=r=48000:cl=mono
```

All the parameters need to be explicitly defined.

## 7.4 flite

Synthesize a voice utterance using the libflite library.

To enable compilation of this filter you need to configure FFmpeg with `--enable-libflite`.

Note that the flite library is not thread-safe.

The filter accepts the following options:

`list_voices`

If set to 1, list the names of the available voices and exit immediately. Default value is 0.

`nb_samples, n`

Set the maximum number of samples per frame. Default value is 512.

`textfile`

Set the filename containing the text to speak.

`text`

Set the text to speak.

`voice, v`

Set the voice to use for the speech synthesis. Default value is `kal`. See also the *list_voices* option.

### 7.4.1 Examples

- Read from file `speech.txt`, and synthetize the text using the standard flite voice:

  ```
  flite=textfile=speech.txt
  ```

- Read the specified text selecting the `slt` voice:

  ```
  flite=text='So fare thee well, poor devil of a Sub-Sub, whose commentator I am':voice=slt
  ```

- Input text to ffmpeg:

  ```
  ffmpeg -f lavfi -i flite=text='So fare thee well, poor devil of a Sub-Sub, whose commentator I am':voice=slt
  ```

- Make `ffplay` speak the specified text, using `flite` and the `lavfi` device:

  ```
  ffplay -f lavfi flite=text='No more be grieved for which that thou hast done.'
  ```

For more information about libflite, check: http://www.speech.cs.cmu.edu/flite/

## 7.5 sine

Generate an audio signal made of a sine wave with amplitude 1/8.

The audio signal is bit-exact.

The filter accepts the following options:

`frequency, f`

Set the carrier frequency. Default is 440 Hz.

`beep_factor, b`

Enable a periodic beep every second with frequency *beep_factor* times the carrier frequency. Default is 0, meaning the beep is disabled.

`sample_rate, r`

Specify the sample rate, default is 44100.

`duration, d`

Specify the duration of the generated audio stream.

`samples_per_frame`

Set the number of samples per output frame, default is 1024.

### 7.5.1 Examples

- Generate a simple 440 Hz sine wave:

  ```
  sine
  ```

- Generate a 220 Hz sine wave with a 880 Hz beep each second, for 5 seconds:

  ```
  sine=220:4:d=5
  sine=f=220:b=4:d=5
  sine=frequency=220:beep_factor=4:duration=5
  ```

# 8 Audio Sinks

Below is a description of the currently available audio sinks.

## 8.1 abuffersink

Buffer audio frames, and make them available to the end of filter chain.

This sink is mainly intended for programmatic use, in particular through the interface defined in `libavfilter/buffersink.h` or the options system.

It accepts a pointer to an AVABufferSinkContext structure, which defines the incoming buffers' formats, to be passed as the opaque parameter to `avfilter_init_filter` for initialization.

## 8.2 anullsink

Null audio sink; do absolutely nothing with the input audio. It is mainly useful as a template and for use in analysis / debugging tools.

# 9 Video Filters

When you configure your FFmpeg build, you can disable any of the existing filters using `--disable-filters`. The configure output will show the video filters included in your build.

Below is a description of the currently available video filters.

## 9.1 alphaextract

Extract the alpha component from the input as a grayscale video. This is especially useful with the *alphamerge* filter.

## 9.2 alphamerge

Add or replace the alpha component of the primary input with the grayscale value of a second input. This is intended for use with *alphaextract* to allow the transmission or storage of frame sequences that have alpha in a format that doesn't support an alpha channel.

For example, to reconstruct full frames from a normal YUV-encoded video and a separate video created with *alphaextract*, you might use:

```
movie=in_alpha.mkv [alpha]; [in][alpha] alphamerge [out]
```

Since this filter is designed for reconstruction, it operates on frame sequences without considering timestamps, and terminates when either input reaches end of stream. This will cause problems if your encoding pipeline drops frames. If you're trying to apply an image as an overlay to a video stream, consider the *overlay* filter instead.

## 9.3 ass

Same as the subtitles filter, except that it doesn't require libavcodec and libavformat to work. On the other hand, it is limited to ASS (Advanced Substation Alpha) subtitles files.

This filter accepts the following option in addition to the common options from the subtitles filter:

`shaping`

Set the shaping engine

Available values are:

'`auto`'

The default libass shaping engine, which is the best available.

'`simple`'

Fast, font-agnostic shaper that can do only substitutions

'`complex`'

Slower shaper using OpenType for substitutions and positioning

The default is `auto`.

## 9.4 bbox

Compute the bounding box for the non-black pixels in the input frame luminance plane.

This filter computes the bounding box containing all the pixels with a luminance value greater than the minimum allowed value. The parameters describing the bounding box are printed on the filter log.

The filter accepts the following option:

`min_val`

> Set the minimal luminance value. Default is `16`.

## 9.5 blackdetect

Detect video intervals that are (almost) completely black. Can be useful to detect chapter transitions, commercials, or invalid recordings. Output lines contains the time for the start, end and duration of the detected black interval expressed in seconds.

In order to display the output lines, you need to set the loglevel at least to the AV_LOG_INFO value.

The filter accepts the following options:

`black_min_duration, d`

> Set the minimum detected black duration expressed in seconds. It must be a non-negative floating point number.
>
> Default value is 2.0.

`picture_black_ratio_th, pic_th`

> Set the threshold for considering a picture "black". Express the minimum value for the ratio:
>
> *`nb_black_pixels / nb_pixels`*
>
> for which a picture is considered black. Default value is 0.98.

`pixel_black_th, pix_th`

> Set the threshold for considering a pixel "black".
>
> The threshold expresses the maximum pixel luminance value for which a pixel is considered "black". The provided value is scaled according to the following equation:
>
> *`absolute_threshold = luminance_minimum_value + pixel_black_th * luminance_range_size`*
>
> *luminance_range_size* and *luminance_minimum_value* depend on the input video format, the range is [0-255] for YUV full-range formats and [16-235] for YUV non full-range formats.
>
> Default value is 0.10.

The following example sets the maximum pixel threshold to the minimum value, and detects only black intervals of 2 or more seconds:

```
blackdetect=d=2:pix_th=0.00
```

## 9.6 blackframe

Detect frames that are (almost) completely black. Can be useful to detect chapter transitions or commercials. Output lines consist of the frame number of the detected frame, the percentage of blackness, the position in the file if known or `-1` and the timestamp in seconds.

In order to display the output lines, you need to set the loglevel at least to the AV_LOG_INFO value.

It accepts the following parameters:

`amount`

> The percentage of the pixels that have to be below the threshold; it defaults to `98`.

`threshold, thresh`

> The threshold below which a pixel value is considered black; it defaults to `32`.

## 9.7 blend

Blend two video frames into each other.

It takes two input streams and outputs one stream, the first input is the "top" layer and second input is "bottom" layer. Output terminates when shortest input terminates.

A description of the accepted options follows.

`c0_mode`
`c1_mode`
`c2_mode`
`c3_mode`
`all_mode`

> Set blend mode for specific pixel component or all pixel components in case of *all_mode*. Default value is `normal`.
>
> Available values for component modes are:
>
> ‘`addition`’
> ‘`and`’
> ‘`average`’
> ‘`burn`’

‘darken’
‘difference’
‘divide’
‘dodge’
‘exclusion’
‘hardlight’
‘lighten’
‘multiply’
‘negation’
‘normal’
‘or’
‘overlay’
‘phoenix’
‘pinlight’
‘reflect’
‘screen’
‘softlight’
‘subtract’
‘vividlight’
‘xor’

c0_opacity
c1_opacity
c2_opacity
c3_opacity
all_opacity

Set blend opacity for specific pixel component or all pixel components in case of *all_opacity*. Only used in combination with pixel component blend modes.

c0_expr
c1_expr
c2_expr
c3_expr
all_expr

Set blend expression for specific pixel component or all pixel components in case of *all_expr*. Note that related mode options will be ignored if those are set.

The expressions can use the following variables:

N

The sequential number of the filtered frame, starting from `0`.

X
Y

the coordinates of the current sample

`W`
`H`

the width and height of currently filtered plane

`SW`
`SH`

Width and height scale depending on the currently filtered plane. It is the ratio between the corresponding luma plane number of pixels and the current plane ones. E.g. for YUV4:2:0 the values are `1,1` for the luma plane, and `0.5,0.5` for chroma planes.

`T`

Time of the current frame, expressed in seconds.

`TOP, A`

Value of pixel component at current location for first video frame (top layer).

`BOTTOM, B`

Value of pixel component at current location for second video frame (bottom layer).

`shortest`

Force termination when the shortest input terminates. Default is `0`.

`repeatlast`

Continue applying the last bottom frame after the end of the stream. A value of `0` disable the filter after the last frame of the bottom layer is reached. Default is `1`.

### 9.7.1 Examples

- Apply transition from bottom layer to top layer in first 10 seconds:

```
blend=all_expr='A*(if(gte(T,10),1,T/10))+B*(1-(if(gte(T,10),1,T/10)))'
```

- Apply 1x1 checkerboard effect:

```
blend=all_expr='if(eq(mod(X,2),mod(Y,2)),A,B)'
```

- Apply uncover left effect:

```
blend=all_expr='if(gte(N*SW+X,W),A,B)'
```

- Apply uncover down effect:

```
blend=all_expr='if(gte(Y-N*SH,0),A,B)'
```

- Apply uncover up-left effect:

```
blend=all_expr='if(gte(T*SH*40+Y,H)*gte((T*40*SW+X)*W/H,W),A,B)'
```

## 9.8 boxblur

Apply a boxblur algorithm to the input video.

It accepts the following parameters:

`luma_radius, lr`
`luma_power, lp`
`chroma_radius, cr`
`chroma_power, cp`
`alpha_radius, ar`
`alpha_power, ap`

A description of the accepted options follows.

`luma_radius, lr`
`chroma_radius, cr`
`alpha_radius, ar`

Set an expression for the box radius in pixels used for blurring the corresponding input plane.

The radius value must be a non-negative number, and must not be greater than the value of the expression `min(w,h)/2` for the luma and alpha planes, and of `min(cw,ch)/2` for the chroma planes.

Default value for `luma_radius` is "2". If not specified, `chroma_radius` and `alpha_radius` default to the corresponding value set for `luma_radius`.

The expressions can contain the following constants:

`w`
`h`

The input width and height in pixels.

`cw`
`ch`

The input chroma image width and height in pixels.

`hsub`
`vsub`

The horizontal and vertical chroma subsample values. For example, for the pixel format "yuv422p", *hsub* is 2 and *vsub* is 1.

`luma_power, lp`
`chroma_power, cp`
`alpha_power, ap`

Specify how many times the boxblur filter is applied to the corresponding plane.

Default value for `luma_power` is 2. If not specified, `chroma_power` and `alpha_power` default to the corresponding value set for `luma_power`.

A value of 0 will disable the effect.

### 9.8.1 Examples

- Apply a boxblur filter with the luma, chroma, and alpha radii set to 2:

```
boxblur=luma_radius=2:luma_power=1
boxblur=2:1
```

- Set the luma radius to 2, and alpha and chroma radius to 0:

```
boxblur=2:1:cr=0:ar=0
```

- Set the luma and chroma radii to a fraction of the video dimension:

```
boxblur=luma_radius=min(h\,w)/10:luma_power=1:chroma_radius=min(cw\,ch)/10:chroma_power=1
```

## 9.9 codecview

Visualize information exported by some codecs.

Some codecs can export information through frames using side-data or other means. For example, some MPEG based codecs export motion vectors through the *export_mvs* flag in the codec `flags2` option.

The filter accepts the following option:

`mv`

Set motion vectors to visualize.

Available flags for *mv* are:

‘`pf`’

forward predicted MVs of P-frames

‘`bf`’

forward predicted MVs of B-frames

‘bb’

backward predicted MVs of B-frames

### 9.9.1 Examples

- Visualizes multi-directionals MVs from P and B-Frames using ffplay:

```
ffplay -flags2 +export_mvs input.mpg -vf codecview=mv=pf+bf+bb
```

## 9.10 colorbalance

Modify intensity of primary colors (red, green and blue) of input frames.

The filter allows an input frame to be adjusted in the shadows, midtones or highlights regions for the red-cyan, green-magenta or blue-yellow balance.

A positive adjustment value shifts the balance towards the primary color, a negative value towards the complementary color.

The filter accepts the following options:

rs
gs
bs

Adjust red, green and blue shadows (darkest pixels).

rm
gm
bm

Adjust red, green and blue midtones (medium pixels).

rh
gh
bh

Adjust red, green and blue highlights (brightest pixels).

Allowed ranges for options are [-1.0, 1.0]. Defaults are 0.

### 9.10.1 Examples

- Add red color cast to shadows:

  ```
  colorbalance=rs=.3
  ```

## 9.11 colorchannelmixer

Adjust video input frames by re-mixing color channels.

This filter modifies a color channel by adding the values associated to the other channels of the same pixels. For example if the value to modify is red, the output value will be:

*red=red*rr + blue*rb + green*rg + alpha*ra*

The filter accepts the following options:

`rr`
`rg`
`rb`
`ra`

  Adjust contribution of input red, green, blue and alpha channels for output red channel. Default is `1` for *rr*, and `0` for *rg*, *rb* and *ra*.

`gr`
`gg`
`gb`
`ga`

  Adjust contribution of input red, green, blue and alpha channels for output green channel. Default is `1` for *gg*, and `0` for *gr*, *gb* and *ga*.

`br`
`bg`
`bb`
`ba`

  Adjust contribution of input red, green, blue and alpha channels for output blue channel. Default is `1` for *bb*, and `0` for *br*, *bg* and *ba*.

`ar`
`ag`
`ab`
`aa`

Adjust contribution of input red, green, blue and alpha channels for output alpha channel. Default is `1` for *aa*, and `0` for *ar*, *ag* and *ab*.

Allowed ranges for options are `[-2.0, 2.0]`.

### 9.11.1 Examples

- Convert source to grayscale:

  ```
  colorchannelmixer=.3:.4:.3:0:.3:.4:.3:0:.3:.4:.3
  ```

- Simulate sepia tones:

  ```
  colorchannelmixer=.393:.769:.189:0:.349:.686:.168:0:.272:.534:.131
  ```

## 9.12 colormatrix

Convert color matrix.

The filter accepts the following options:

`src`
`dst`

Specify the source and destination color matrix. Both values must be specified.

The accepted values are:

‘`bt709`’

BT.709

‘`bt601`’

BT.601

‘`smpte240m`’

SMPTE-240M

‘`fcc`’

FCC

For example to convert from BT.601 to SMPTE-240M, use the command:

```
colormatrix=bt601:smpte240m
```

## 9.13 copy

Copy the input source unchanged to the output. This is mainly useful for testing purposes.

## 9.14 crop

Crop the input video to given dimensions.

It accepts the following parameters:

`w, out_w`

The width of the output video. It defaults to `iw`. This expression is evaluated only once during the filter configuration.

`h, out_h`

The height of the output video. It defaults to `ih`. This expression is evaluated only once during the filter configuration.

`x`

The horizontal position, in the input video, of the left edge of the output video. It defaults to `(in_w-out_w)/2`. This expression is evaluated per-frame.

`y`

The vertical position, in the input video, of the top edge of the output video. It defaults to `(in_h-out_h)/2`. This expression is evaluated per-frame.

`keep_aspect`

If set to 1 will force the output display aspect ratio to be the same of the input, by changing the output sample aspect ratio. It defaults to 0.

The *out_w*, *out_h*, *x*, *y* parameters are expressions containing the following constants:

`x`
`y`

The computed values for *x* and *y*. They are evaluated for each new frame.

`in_w`
`in_h`

The input width and height.

`iw`
`ih`

These are the same as *in_w* and *in_h*.

`out_w`
`out_h`

The output (cropped) width and height.

`ow`
`oh`

These are the same as *out_w* and *out_h*.

`a`

same as *iw* / *ih*

`sar`

input sample aspect ratio

`dar`

input display aspect ratio, it is the same as (*iw* / *ih*) * *sar*

`hsub`
`vsub`

horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

`n`

The number of the input frame, starting from 0.

`pos`

the position in the file of the input frame, NAN if unknown

`t`

The timestamp expressed in seconds. It’s NAN if the input timestamp is unknown.

The expression for *out_w* may depend on the value of *out_h*, and the expression for *out_h* may depend on *out_w*, but they cannot depend on *x* and *y*, as *x* and *y* are evaluated after *out_w* and *out_h*.

The *x* and *y* parameters specify the expressions for the position of the top-left corner of the output (non-cropped) area. They are evaluated for each frame. If the evaluated value is not valid, it is approximated to the nearest valid value.

The expression for *x* may depend on *y*, and the expression for *y* may depend on *x*.

### 9.14.1 Examples

- Crop area with size 100x100 at position (12,34).

  ```
  crop=100:100:12:34
  ```

  Using named options, the example above becomes:

  ```
  crop=w=100:h=100:x=12:y=34
  ```

- Crop the central input area with size 100x100:

  ```
  crop=100:100
  ```

- Crop the central input area with size 2/3 of the input video:

  ```
  crop=2/3*in_w:2/3*in_h
  ```

- Crop the input video central square:

  ```
  crop=out_w=in_h
  crop=in_h
  ```

- Delimit the rectangle with the top-left corner placed at position 100:100 and the right-bottom corner corresponding to the right-bottom corner of the input image.

  ```
  crop=in_w-100:in_h-100:100:100
  ```

- Crop 10 pixels from the left and right borders, and 20 pixels from the top and bottom borders

  ```
  crop=in_w-2*10:in_h-2*20
  ```

- Keep only the bottom right quarter of the input image:

  ```
  crop=in_w/2:in_h/2:in_w/2:in_h/2
  ```

- Crop height for getting Greek harmony:

  ```
  crop=in_w:1/PHI*in_w
  ```

- Appply trembling effect:

  ```
  crop=in_w/2:in_h/2:(in_w-out_w)/2+((in_w-out_w)/2)*sin(n/10):(in_h-out_h)/2 +((in_h-out_h)/2)*sin(n/7)
  ```

- Apply erratic camera effect depending on timestamp:

```
crop=in_w/2:in_h/2:(in_w-out_w)/2+((in_w-out_w)/2)*sin(t*10):(in_h-out_h)/2 +((in_h-out_h)/2)*sin(t*13)"
```

- Set x depending on the value of y:

```
crop=in_w/2:in_h/2:y:10+10*sin(n/10)
```

## 9.15 cropdetect

Auto-detect the crop size.

It calculates the necessary cropping parameters and prints the recommended parameters via the logging system. The detected dimensions correspond to the non-black area of the input video.

It accepts the following parameters:

`limit`

Set higher black value threshold, which can be optionally specified from nothing (0) to everything (255). An intensity value greater to the set value is considered non-black. It defaults to 24.

`round`

The value which the width/height should be divisible by. It defaults to 16. The offset is automatically adjusted to center the video. Use 2 to get only even dimensions (needed for 4:2:2 video). 16 is best when encoding to most video codecs.

`reset_count, reset`

Set the counter that determines after how many frames cropdetect will reset the previously detected largest video area and start over to detect the current optimal crop area. Default value is 0.

This can be useful when channel logos distort the video area. 0 indicates ’never reset’, and returns the largest area encountered during playback.

## 9.16 curves

Apply color adjustments using curves.

This filter is similar to the Adobe Photoshop and GIMP curves tools. Each component (red, green and blue) has its values defined by *N* key points tied from each other using a smooth curve. The x-axis represents the pixel values from the input frame, and the y-axis the new pixel values to be set for the output frame.

By default, a component curve is defined by the two points *(0;0)* and *(1;1)*. This creates a straight line where each original pixel value is "adjusted" to its own value, which means no change to the image.

The filter allows you to redefine these two points and add some more. A new curve (using a natural cubic spline interpolation) will be define to pass smoothly through all these new coordinates. The new defined points needs to be strictly increasing over the x-axis, and their *x* and *y* values must be in the *[0;1]* interval.

If the computed curves happened to go outside the vector spaces, the values will be clipped accordingly.

If there is no key point defined in `x=0`, the filter will automatically insert a *(0;0)* point. In the same way, if there is no key point defined in `x=1`, the filter will automatically insert a *(1;1)* point.

The filter accepts the following options:

`preset`

> Select one of the available color presets. This option can be used in addition to the `r`, `g`, `b` parameters; in this case, the later options takes priority on the preset values. Available presets are:
>
> ‘`none`’
> ‘`color_negative`’
> ‘`cross_process`’
> ‘`darker`’
> ‘`increase_contrast`’
> ‘`lighter`’
> ‘`linear_contrast`’
> ‘`medium_contrast`’
> ‘`negative`’
> ‘`strong_contrast`’
> ‘`vintage`’
>
> Default is `none`.

`master, m`

> Set the master key points. These points will define a second pass mapping. It is sometimes called a "luminance" or "value" mapping. It can be used with `r`, `g`, `b` or `all` since it acts like a post-processing LUT.

`red, r`

> Set the key points for the red component.

`green, g`

> Set the key points for the green component.

`blue, b`

> Set the key points for the blue component.

`all`

> Set the key points for all components (not including master). Can be used in addition to the other key points component options. In this case, the unset component(s) will fallback on this `all` setting.

`psfile`

Specify a Photoshop curves file (`.asv`) to import the settings from.

To avoid some filtergraph syntax conflicts, each key points list need to be defined using the following syntax: `x0/y0 x1/y1 x2/y2 ....`

### 9.16.1 Examples

- Increase slightly the middle level of blue:

```
curves=blue='0.5/0.58'
```

- Vintage effect:

```
curves=r='0/0.11 .42/.51 1/0.95':g='0.50/0.48':b='0/0.22 .49/.44 1/0.8'
```

  Here we obtain the following coordinates for each components:

  *red*

  `(0;0.11) (0.42;0.51) (1;0.95)`

  *green*

  `(0;0) (0.50;0.48) (1;1)`

  *blue*

  `(0;0.22) (0.49;0.44) (1;0.80)`

- The previous example can also be achieved with the associated built-in preset:

```
curves=preset=vintage
```

- Or simply:

```
curves=vintage
```

- Use a Photoshop preset and redefine the points of the green component:

```
curves=psfile='MyCurvesPresets/purple.asv':green='0.45/0.53'
```

## 9.17 dctdnoiz

Denoise frames using 2D DCT (frequency domain filtering).

This filter is not designed for real time.

The filter accepts the following options:

`sigma, s`

Set the noise sigma constant.

This *sigma* defines a hard threshold of `3 * sigma`; every DCT coefficient (absolute value) below this threshold with be dropped.

If you need a more advanced filtering, see `expr`.

Default is `0`.

`overlap`

Set number overlapping pixels for each block. Since the filter can be slow, you may want to reduce this value, at the cost of a less effective filter and the risk of various artefacts.

If the overlapping value doesn't allow to process the whole input width or height, a warning will be displayed and according borders won't be denoised.

Default value is *blocksize*-1, which is the best possible setting.

`expr, e`

Set the coefficient factor expression.

For each coefficient of a DCT block, this expression will be evaluated as a multiplier value for the coefficient.

If this is option is set, the `sigma` option will be ignored.

The absolute value of the coefficient can be accessed through the *c* variable.

`n`

Set the *blocksize* using the number of bits. `1<<n` defines the *blocksize*, which is the width and height of the processed blocks.

The default value is *3* (8x8) and can be raised to *4* for a *blocksize* of 16x16. Note that changing this setting has huge consequences on the speed processing. Also, a larger block size does not necessarily means a better de-noising.

### 9.17.1 Examples

Apply a denoise with a `sigma` of `4.5`:

```
dctdnoiz=4.5
```

The same operation can be achieved using the expression system:

```
dctdnoiz=e='gte(c, 4.5*3)'
```

Violent denoise using a block size of `16x16`:

```
dctdnoiz=15:n=4
```

## 9.18 decimate

Drop duplicated frames at regular intervals.

The filter accepts the following options:

`cycle`

Set the number of frames from which one will be dropped. Setting this to *N* means one frame in every batch of *N* frames will be dropped. Default is `5`.

`dupthresh`

Set the threshold for duplicate detection. If the difference metric for a frame is less than or equal to this value, then it is declared as duplicate. Default is `1.1`

`scthresh`

Set scene change threshold. Default is `15`.

`blockx`
`blocky`

Set the size of the x and y-axis blocks used during metric calculations. Larger blocks give better noise suppression, but also give worse detection of small movements. Must be a power of two. Default is `32`.

`ppsrc`

Mark main input as a pre-processed input and activate clean source input stream. This allows the input to be pre-processed with various filters to help the metrics calculation while keeping the frame selection lossless. When set to `1`, the first stream is for the pre-processed input, and the second stream is the clean source from where the kept frames are chosen. Default is `0`.

`chroma`

Set whether or not chroma is considered in the metric calculations. Default is `1`.

## 9.19 dejudder

Remove judder produced by partially interlaced telecined content.

Judder can be introduced, for instance, by pullup filter. If the original source was partially telecined content then the output of `pullup,dejudder` will have a variable frame rate. May change the recorded frame rate of the container. Aside from that change, this filter will not affect constant frame rate video.

The option available in this filter is:

`cycle`

Specify the length of the window over which the judder repeats.

Accepts any integer greater than 1. Useful values are:

‘`4`’

If the original was telecined from 24 to 30 fps (Film to NTSC).

‘`5`’

If the original was telecined from 25 to 30 fps (PAL to NTSC).

‘`20`’

If a mixture of the two.

The default is ‘`4`’.

## 9.20 delogo

Suppress a TV station logo by a simple interpolation of the surrounding pixels. Just set a rectangle covering the logo and watch it disappear (and sometimes something even uglier appear - your mileage may vary).

It accepts the following parameters:

`x`
`y`

Specify the top left corner coordinates of the logo. They must be specified.

`w`
`h`

Specify the width and height of the logo to clear. They must be specified.

`band, t`

Specify the thickness of the fuzzy edge of the rectangle (added to *w* and *h*). The default value is 4.

`show`

When set to 1, a green rectangle is drawn on the screen to simplify finding the right *x*, *y*, *w*, and *h* parameters. The default value is 0.

The rectangle is drawn on the outermost pixels which will be (partly) replaced with interpolated values. The values of the next pixels immediately outside this rectangle in each direction will be used to compute the interpolated pixel values inside the rectangle.

### 9.20.1 Examples

- Set a rectangle covering the area with top left corner coordinates 0,0 and size 100x77, and a band of size 10:

  ```
  delogo=x=0:y=0:w=100:h=77:band=10
  ```

## 9.21 deshake

Attempt to fix small changes in horizontal and/or vertical shift. This filter helps remove camera shake from hand-holding a camera, bumping a tripod, moving on a vehicle, etc.

The filter accepts the following options:

`x`
`y`
`w`
`h`

Specify a rectangular area where to limit the search for motion vectors. If desired the search for motion vectors can be limited to a rectangular area of the frame defined by its top left corner, width and height. These parameters have the same meaning as the drawbox filter which can be used to visualise the position of the bounding box.

This is useful when simultaneous movement of subjects within the frame might be confused for camera motion by the motion vector search.

If any or all of *x*, *y*, *w* and *h* are set to -1 then the full frame is used. This allows later options to be set without specifying the bounding box for the motion vector search.

Default - search the whole frame.

`rx`
`ry`

Specify the maximum extent of movement in x and y directions in the range 0-64 pixels. Default 16.

`edge`

Specify how to generate pixels to fill blanks at the edge of the frame. Available values are:

‘`blank, 0`’

Fill zeroes at blank locations

‘`original, 1`’

Original image at blank locations

‘`clamp, 2`’

Extruded edge value at blank locations

‘`mirror, 3`’

Mirrored edge at blank locations

Default value is ‘`mirror`’.

`blocksize`

Specify the blocksize to use for motion search. Range 4-128 pixels, default 8.

`contrast`

Specify the contrast threshold for blocks. Only blocks with more than the specified contrast (difference between darkest and lightest pixels) will be considered. Range 1-255, default 125.

`search`

Specify the search strategy. Available values are:

‘`exhaustive, 0`’

Set exhaustive search

‘`less, 1`’

Set less exhaustive search.

Default value is ‘`exhaustive`’.

`filename`

If set then a detailed log of the motion search is written to the specified file.

`opencl`

If set to 1, specify using OpenCL capabilities, only available if FFmpeg was configured with `--enable-opencl`. Default value is 0.

## 9.22 drawbox

Draw a colored box on the input image.

It accepts the following parameters:

`x`
`y`

The expressions which specify the top left corner coordinates of the box. It defaults to 0.

`width, w`
`height, h`

The expressions which specify the width and height of the box; if 0 they are interpreted as the input width and height. It defaults to 0.

`color, c`

Specify the color of the box to write. For the general syntax of this option, check the "Color" section in the ffmpeg-utils manual. If the special value `invert` is used, the box edge color is the same as the video with inverted luma.

`thickness, t`

The expression which sets the thickness of the box edge. Default value is `3`.

See below for the list of accepted constants.

The parameters for *x*, *y*, *w* and *h* and *t* are expressions containing the following constants:

`dar`

The input display aspect ratio, it is the same as (*w* / *h*) * *sar*.

`hsub`
`vsub`

horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

`in_h, ih`
`in_w, iw`

The input width and height.

`sar`

The input sample aspect ratio.

`x`
`y`

The x and y offset coordinates where the box is drawn.

`w`
`h`

The width and height of the drawn box.

`t`

The thickness of the drawn box.

These constants allow the *x*, *y*, *w*, *h* and *t* expressions to refer to each other, so you may for example specify `y=x/dar` or `h=w/dar`.

### 9.22.1 Examples

- Draw a black box around the edge of the input image:

```
drawbox
```

- Draw a box with color red and an opacity of 50%:

```
drawbox=10:20:200:60:red@0.5
```

The previous example can be specified as:

```
drawbox=x=10:y=20:w=200:h=60:color=red@0.5
```

- Fill the box with pink color:

```
drawbox=x=10:y=10:w=100:h=100:color=pink@0.5:t=max
```

- Draw a 2-pixel red 2.40:1 mask:

```
drawbox=x=-t:y=0.5*(ih-iw/2.4)-t:w=iw+t*2:h=iw/2.4+t*2:t=2:c=red
```

## 9.23 drawgrid

Draw a grid on the input image.

It accepts the following parameters:

`x`
`y`

The expressions which specify the coordinates of some point of grid intersection (meant to configure offset). Both default to 0.

`width, w`
`height, h`

The expressions which specify the width and height of the grid cell, if 0 they are interpreted as the input width and height, respectively, minus `thickness`, so image gets framed. Default to 0.

`color, c`

Specify the color of the grid. For the general syntax of this option, check the "Color" section in the ffmpeg-utils manual. If the special value `invert` is used, the grid color is the same as the video with inverted luma.

`thickness, t`

The expression which sets the thickness of the grid line. Default value is `1`.

See below for the list of accepted constants.

The parameters for *x*, *y*, *w* and *h* and *t* are expressions containing the following constants:

`dar`

The input display aspect ratio, it is the same as (*w* / *h*) * *sar*.

`hsub`
`vsub`

horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

`in_h, ih`
`in_w, iw`

The input grid cell width and height.

`sar`

The input sample aspect ratio.

`x`
`y`

The x and y coordinates of some point of grid intersection (meant to configure offset).

`w`
`h`

The width and height of the drawn cell.

`t`

The thickness of the drawn cell.

These constants allow the *x*, *y*, *w*, *h* and *t* expressions to refer to each other, so you may for example specify `y=x/dar` or `h=w/dar`.

### 9.23.1 Examples

- Draw a grid with cell 100x100 pixels, thickness 2 pixels, with color red and an opacity of 50%:

  ```
  drawgrid=width=100:height=100:thickness=2:color=red@0.5
  ```

- Draw a white 3x3 grid with an opacity of 50%:

  ```
  drawgrid=w=iw/3:h=ih/3:t=2:c=white@0.5
  ```

## 9.24 drawtext

Draw a text string or text from a specified file on top of a video, using the libfreetype library.

To enable compilation of this filter, you need to configure FFmpeg with `--enable-libfreetype`. To enable default font fallback and the *font* option you need to configure FFmpeg with `--enable-libfontconfig`. To enable the *text_shaping* option, you need to configure FFmpeg with `--enable-libfribidi`.

### 9.24.1 Syntax

It accepts the following parameters:

`box`

Used to draw a box around text using the background color. The value must be either 1 (enable) or 0 (disable). The default value of *box* is 0.

`boxcolor`

The color to be used for drawing box around text. For the syntax of this option, check the "Color" section in the ffmpeg-utils manual.

The default value of *boxcolor* is "white".

`borderw`

Set the width of the border to be drawn around the text using *bordercolor*. The default value of *borderw* is 0.

`bordercolor`

Set the color to be used for drawing border around text. For the syntax of this option, check the "Color" section in the ffmpeg-utils manual.

The default value of *bordercolor* is "black".

`expansion`

Select how the *text* is expanded. Can be either `none`, `strftime` (deprecated) or `normal` (default). See the Text expansion section below for details.

`fix_bounds`

If true, check and fix text coords to avoid clipping.

`fontcolor`

The color to be used for drawing fonts. For the syntax of this option, check the "Color" section in the ffmpeg-utils manual.

The default value of *fontcolor* is "black".

`fontcolor_expr`

String which is expanded the same way as *text* to obtain dynamic *fontcolor* value. By default this option has empty value and is not processed. When this option is set, it overrides *fontcolor* option.

`font`

The font family to be used for drawing text. By default Sans.

`fontfile`

The font file to be used for drawing text. The path must be included. This parameter is mandatory if the fontconfig support is disabled.

`fontsize`

The font size to be used for drawing text. The default value of *fontsize* is 16.

`text_shaping`

If set to 1, attempt to shape the text (for example, reverse the order of right-to-left text and join Arabic characters) before drawing it. Otherwise, just draw the text exactly as given. By default 1 (if supported).

`ft_load_flags`

The flags to be used for loading the fonts.

The flags map the corresponding flags supported by libfreetype, and are a combination of the following values:

*default*
*no_scale*
*no_hinting*
*render*
*no_bitmap*
*vertical_layout*
*force_autohint*
*crop_bitmap*
*pedantic*
*ignore_global_advance_width*
*no_recurse*
*ignore_transform*
*monochrome*
*linear_design*
*no_autohint*

Default value is "default".

For more information consult the documentation for the FT_LOAD_* libfreetype flags.

`shadowcolor`

The color to be used for drawing a shadow behind the drawn text. For the syntax of this option, check the "Color" section in the ffmpeg-utils manual.

The default value of *shadowcolor* is "black".

`shadowx`
`shadowy`

The x and y offsets for the text shadow position with respect to the position of the text. They can be either positive or negative values. The default value for both is "0".

`start_number`

The starting frame number for the n/frame_num variable. The default value is "0".

`tabsize`

The size in number of spaces to use for rendering the tab. Default value is 4.

`timecode`

Set the initial timecode representation in "hh:mm:ss[:;.]ff" format. It can be used with or without text parameter. *timecode_rate* option must be specified.

`timecode_rate, rate, r`

Set the timecode frame rate (timecode only).

`text`

The text string to be drawn. The text must be a sequence of UTF-8 encoded characters. This parameter is mandatory if no file is specified with the parameter *textfile*.

`textfile`

A text file containing text to be drawn. The text must be a sequence of UTF-8 encoded characters.

This parameter is mandatory if no text string is specified with the parameter *text*.

If both *text* and *textfile* are specified, an error is thrown.

`reload`

If set to 1, the *textfile* will be reloaded before each frame. Be sure to update it atomically, or it may be read partially, or even fail.

`x`
`y`

The expressions which specify the offsets where text will be drawn within the video frame. They are relative to the top/left border of the output image.

The default value of *x* and *y* is "0".

See below for the list of accepted constants and functions.

The parameters for *x* and *y* are expressions containing the following constants and functions:

`dar`

input display aspect ratio, it is the same as (*w* / *h*) * *sar*

`hsub`
`vsub`

horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

`line_h, lh`

the height of each text line

`main_h, h, H`

the input height

`main_w, w, W`

the input width

`max_glyph_a, ascent`

the maximum distance from the baseline to the highest/upper grid coordinate used to place a glyph outline point, for all the rendered glyphs. It is a positive value, due to the grid's orientation with the Y axis upwards.

`max_glyph_d, descent`

the maximum distance from the baseline to the lowest grid coordinate used to place a glyph outline point, for all the rendered glyphs. This is a negative value, due to the grid's orientation, with the Y axis upwards.

`max_glyph_h`

maximum glyph height, that is the maximum height for all the glyphs contained in the rendered text, it is equivalent to *ascent* - *descent*.

`max_glyph_w`

maximum glyph width, that is the maximum width for all the glyphs contained in the rendered text

`n`

the number of input frame, starting from 0 `rand(min, max)`

return a random number included between *min* and *max*

`sar`

The input sample aspect ratio.

`t`

timestamp expressed in seconds, NAN if the input timestamp is unknown

`text_h, th`

the height of the rendered text

`text_w, tw`

the width of the rendered text

`x`
`y`

the x and y offset coordinates where the text is drawn.

These parameters allow the *x* and *y* expressions to refer each other, so you can for example specify `y=x/dar`.

### 9.24.2 Text expansion

If `expansion` is set to `strftime`, the filter recognizes strftime() sequences in the provided text and expands them accordingly. Check the documentation of strftime(). This feature is deprecated.

If `expansion` is set to `none`, the text is printed verbatim.

If `expansion` is set to `normal` (which is the default), the following expansion mechanism is used.

The backslash character '\', followed by any character, always expands to the second character.

Sequence of the form `%{...}` are expanded. The text between the braces is a function name, possibly followed by arguments separated by ':'. If the arguments contain special characters or delimiters (':' or '}'), they should be escaped.

Note that they probably must also be escaped as the value for the `text` option in the filter argument string and as the filter argument in the filtergraph description, and possibly also for the shell, that makes up to four levels of escaping; using a text file avoids these problems.

The following functions are available:

`expr, e`

> The expression evaluation result.
>
> It must take one argument specifying the expression to be evaluated, which accepts the same constants and functions as the *x* and *y* values. Note that not all constants should be used, for example the text size is not known when evaluating the expression, so the constants *text_w* and *text_h* will have an undefined value.

`expr_int_format, eif`

> Evaluate the expression's value and output as formatted integer.
>
> The first argument is the expression to be evaluated, just as for the *expr* function. The second argument specifies the output format. Allowed values are 'x', 'X', 'd' and 'u'. They are treated exactly as in the printf function. The third parameter is optional and sets the number of positions taken by the output. It can be used to add padding with zeros from the left.

`gmtime`

> The time at which the filter is running, expressed in UTC. It can accept an argument: a strftime() format string.

`localtime`

> The time at which the filter is running, expressed in the local time zone. It can accept an argument: a strftime() format string.

`metadata`

> Frame metadata. It must take one argument specifying metadata key.

`n, frame_num`

> The frame number, starting from 0.

`pict_type`

> A 1 character description of the current picture type.

`pts`

> The timestamp of the current frame. It can take up to two arguments.
>
> The first argument is the format of the timestamp; it defaults to `flt` for seconds as a decimal number with microsecond accuracy; `hms` stands for a formatted *[-]HH:MM:SS.mmm* timestamp with millisecond accuracy.

The second argument is an offset added to the timestamp.

### 9.24.3 Examples

- Draw "Test Text" with font FreeSerif, using the default values for the optional parameters.

  ```
  drawtext="fontfile=/usr/share/fonts/truetype/freefont/FreeSerif.ttf: text='Test Text'"
  ```

- Draw 'Test Text' with font FreeSerif of size 24 at position x=100 and y=50 (counting from the top-left corner of the screen), text is yellow with a red box around it. Both the text and the box have an opacity of 20%.

  ```
  drawtext="fontfile=/usr/share/fonts/truetype/freefont/FreeSerif.ttf: text='Test Text':\
            x=100: y=50: fontsize=24: fontcolor=yellow@0.2: box=1: boxcolor=red@0.2"
  ```

  Note that the double quotes are not necessary if spaces are not used within the parameter list.

- Show the text at the center of the video frame:

  ```
  drawtext="fontsize=30:fontfile=FreeSerif.ttf:text='hello world':x=(w-text_w)/2:y=(h-text_h-line_h)/2"
  ```

- Show a text line sliding from right to left in the last row of the video frame. The file `LONG_LINE` is assumed to contain a single line with no newlines.

  ```
  drawtext="fontsize=15:fontfile=FreeSerif.ttf:text=LONG_LINE:y=h-line_h:x=-50*t"
  ```

- Show the content of file `CREDITS` off the bottom of the frame and scroll up.

  ```
  drawtext="fontsize=20:fontfile=FreeSerif.ttf:textfile=CREDITS:y=h-20*t"
  ```

- Draw a single green letter "g", at the center of the input video. The glyph baseline is placed at half screen height.

  ```
  drawtext="fontsize=60:fontfile=FreeSerif.ttf:fontcolor=green:text=g:x=(w-max_glyph_w)/2:y=h/2-ascent"
  ```

- Show text for 1 second every 3 seconds:

  ```
  drawtext="fontfile=FreeSerif.ttf:fontcolor=white:x=100:y=x/dar:enable=lt(mod(t\,3)\,1):text='blink'"
  ```

- Use fontconfig to set the font. Note that the colons need to be escaped.

  ```
  drawtext='fontfile=Linux Libertine O-40\:style=Semibold:text=FFmpeg'
  ```

- Print the date of a real-time encoding (see strftime(3)):

  ```
  drawtext='fontfile=FreeSans.ttf:text=%{localtime\:%a %b %d %Y}'
  ```

- Show text fading in and out (appearing/disappearing):

  ```
  #!/bin/sh
  DS=1.0 # display start
  DE=10.0 # display end
  FID=1.5 # fade in duration
  FOD=5 # fade out duration
  ffplay -f lavfi "color,drawtext=text=TEST:fontsize=50:fontfile=FreeSerif.ttf:fontcolor_expr=ff0000%{eif\\\\: clip(255*(1*between(t\\, $DS + $FID\\, $DE - $FOD) + ((t - $DS)/$FID)*between(t\\, $DS\\, $DS + $FID) + (-(t - $DE)/$FOD)*between(t\\, $DE - $FOD\\, $DE) )\\, 0\\, 255) \\\\: x\\\\: 2 }"
  ```

For more information about libfreetype, check: http://www.freetype.org/.

For more information about fontconfig, check: http://freedesktop.org/software/fontconfig/fontconfig-user.html.

For more information about libfribidi, check: http://fribidi.org/.

## 9.25 edgedetect

Detect and draw edges. The filter uses the Canny Edge Detection algorithm.

The filter accepts the following options:

`low`
`high`

> Set low and high threshold values used by the Canny thresholding algorithm.
>
> The high threshold selects the "strong" edge pixels, which are then connected through 8-connectivity with the "weak" edge pixels selected by the low threshold.
>
> *low* and *high* threshold values must be chosen in the range [0,1], and *low* should be lesser or equal to *high*.
>
> Default value for *low* is `20/255`, and default value for *high* is `50/255`.

`mode`

> Define the drawing mode.
>
> ‘`wires`’
>
> > Draw white/gray wires on black background.
>
> ‘`colormix`’
>
> > Mix the colors to create a paint/cartoon effect.
>
> Default value is *wires*.

### 9.25.1 Examples

- Standard edge detection with custom values for the hysteresis thresholding:

  ```
  edgedetect=low=0.1:high=0.4
  ```

- Painting effect without thresholding:

```
edgedetect=mode=colormix:high=0
```

## 9.26 extractplanes

Extract color channel components from input video stream into separate grayscale video streams.

The filter accepts the following option:

`planes`

Set plane(s) to extract.

Available values for planes are:

'y'
'u'
'v'
'a'
'r'
'g'
'b'

Choosing planes not available in the input will result in an error. That means you cannot select `r`, `g`, `b` planes with `y`, `u`, `v` planes at same time.

### 9.26.1 Examples

- Extract luma, u and v color channel component from input video frame into 3 grayscale outputs:

```
ffmpeg -i video.avi -filter_complex 'extractplanes=y+u+v[y][u][v]' -map '[y]' y.avi -map '[u]' u.avi -map '[v]' v.avi
```

## 9.27 elbg

Apply a posterize effect using the ELBG (Enhanced LBG) algorithm.

For each input image, the filter will compute the optimal mapping from the input to the output given the codebook length, that is the number of distinct output colors.

This filter accepts the following options.

`codebook_length, l`

Set codebook length. The value must be a positive integer, and represents the number of distinct output colors. Default value is 256.

`nb_steps, n`

Set the maximum number of iterations to apply for computing the optimal mapping. The higher the value the better the result and the higher the computation time. Default value is 1.

`seed, s`

Set a random seed, must be an integer included between 0 and UINT32_MAX. If not specified, or if explicitly set to -1, the filter will try to use a good random seed on a best effort basis.

## 9.28 fade

Apply a fade-in/out effect to the input video.

It accepts the following parameters:

`type, t`

The effect type can be either "in" for a fade-in, or "out" for a fade-out effect. Default is `in`.

`start_frame, s`

Specify the number of the frame to start applying the fade effect at. Default is 0.

`nb_frames, n`

The number of frames that the fade effect lasts. At the end of the fade-in effect, the output video will have the same intensity as the input video. At the end of the fade-out transition, the output video will be filled with the selected `color`. Default is 25.

`alpha`

If set to 1, fade only alpha channel, if one exists on the input. Default value is 0.

`start_time, st`

Specify the timestamp (in seconds) of the frame to start to apply the fade effect. If both start_frame and start_time are specified, the fade will start at whichever comes last. Default is 0.

`duration, d`

The number of seconds for which the fade effect has to last. At the end of the fade-in effect the output video will have the same intensity as the input video, at the end of the fade-out transition the output video will be filled with the selected `color`. If both duration and nb_frames are specified, duration is used. Default is 0.

`color, c`

Specify the color of the fade. Default is "black".

### 9.28.1 Examples

- Fade in the first 30 frames of video:

  ```
  fade=in:0:30
  ```

  The command above is equivalent to:

  ```
  fade=t=in:s=0:n=30
  ```

- Fade out the last 45 frames of a 200-frame video:

  ```
  fade=out:155:45
  fade=type=out:start_frame=155:nb_frames=45
  ```

- Fade in the first 25 frames and fade out the last 25 frames of a 1000-frame video:

  ```
  fade=in:0:25, fade=out:975:25
  ```

- Make the first 5 frames yellow, then fade in from frame 5-24:

  ```
  fade=in:5:20:color=yellow
  ```

- Fade in alpha over first 25 frames of video:

  ```
  fade=in:0:25:alpha=1
  ```

- Make the first 5.5 seconds black, then fade in for 0.5 seconds:

  ```
  fade=t=in:st=5.5:d=0.5
  ```

## 9.29 field

Extract a single field from an interlaced image using stride arithmetic to avoid wasting CPU time. The output frames are marked as non-interlaced.

The filter accepts the following options:

`type`

  Specify whether to extract the top (if the value is `0` or `top`) or the bottom field (if the value is `1` or `bottom`).

## 9.30 fieldmatch

Field matching filter for inverse telecine. It is meant to reconstruct the progressive frames from a telecined stream. The filter does not drop duplicated frames, so to achieve a complete inverse telecine `fieldmatch` needs to be followed by a decimation filter such as decimate in the filtergraph.

The separation of the field matching and the decimation is notably motivated by the possibility of inserting a de-interlacing filter fallback between the two. If the source has mixed telecined and real interlaced content, `fieldmatch` will not be able to match fields for the interlaced parts. But these remaining combed frames will be marked as interlaced, and thus can be de-interlaced by a later filter such as yadif before decimation.

In addition to the various configuration options, `fieldmatch` can take an optional second stream, activated through the `ppsrc` option. If enabled, the frames reconstruction will be based on the fields and frames from this second stream. This allows the first input to be pre-processed in order to help the various algorithms of the filter, while keeping the output lossless (assuming the fields are matched properly). Typically, a field-aware denoiser, or brightness/contrast adjustments can help.

Note that this filter uses the same algorithms as TIVTC/TFM (AviSynth project) and VIVTC/VFM (VapourSynth project). The later is a light clone of TFM from which `fieldmatch` is based on. While the semantic and usage are very close, some behaviour and options names can differ.

The decimate filter currently only works for constant frame rate input. Do not use `fieldmatch` and decimate if your input has mixed telecined and progressive content with changing framerate.

The filter accepts the following options:

`order`

Specify the assumed field order of the input stream. Available values are:

‘`auto`’

Auto detect parity (use FFmpeg’s internal parity value).

‘`bff`’

Assume bottom field first.

‘`tff`’

Assume top field first.

Note that it is sometimes recommended not to trust the parity announced by the stream.

Default value is *auto*.

`mode`

Set the matching mode or strategy to use. `pc` mode is the safest in the sense that it won’t risk creating jerkiness due to duplicate frames when possible, but if there are bad edits or blended fields it will end up outputting combed frames when a good match might actually exist. On the other hand, `pcn_ub` mode is the most risky in terms of creating jerkiness, but will almost always find a good frame if there is one. The other values are all somewhere in between `pc` and `pcn_ub` in terms of risking jerkiness and creating duplicate frames versus finding good matches in sections with bad edits, orphaned fields, blended fields, etc.

More details about p/c/n/u/b are available in p/c/n/u/b meaning section.

Available values are:

‘`pc`’

    2-way matching (p/c)

‘`pc_n`’

    2-way matching, and trying 3rd match if still combed (p/c + n)

‘`pc_u`’

    2-way matching, and trying 3rd match (same order) if still combed (p/c + u)

‘`pc_n_ub`’

    2-way matching, trying 3rd match if still combed, and trying 4th/5th matches if still combed (p/c + n + u/b)

‘`pcn`’

    3-way matching (p/c/n)

‘`pcn_ub`’

    3-way matching, and trying 4th/5th matches if all 3 of the original matches are detected as combed (p/c/n + u/b)

The parenthesis at the end indicate the matches that would be used for that mode assuming `order`=*tff* (and `field` on *auto* or *top*).

In terms of speed `pc` mode is by far the fastest and `pcn_ub` is the slowest.

Default value is *pc_n*.

`ppsrc`

Mark the main input stream as a pre-processed input, and enable the secondary input stream as the clean source to pick the fields from. See the filter introduction for more details. It is similar to the `clip2` feature from VFM/TFM.

Default value is `0` (disabled).

`field`

Set the field to match from. It is recommended to set this to the same value as `order` unless you experience matching failures with that setting. In certain circumstances changing the field that is used to match from can have a large impact on matching performance. Available values are:

‘`auto`’

Automatic (same value as `order`).

‘`bottom`’

Match from the bottom field.

‘`top`’

Match from the top field.

Default value is *auto*.

`mchroma`

Set whether or not chroma is included during the match comparisons. In most cases it is recommended to leave this enabled. You should set this to `0` only if your clip has bad chroma problems such as heavy rainbowing or other artifacts. Setting this to `0` could also be used to speed things up at the cost of some accuracy.

Default value is `1`.

`y0`
`y1`

These define an exclusion band which excludes the lines between `y0` and `y1` from being included in the field matching decision. An exclusion band can be used to ignore subtitles, a logo, or other things that may interfere with the matching. `y0` sets the starting scan line and `y1` sets the ending line; all lines in between `y0` and `y1` (including `y0` and `y1`) will be ignored. Setting `y0` and `y1` to the same value will disable the feature. `y0` and `y1` defaults to `0`.

`scthresh`

Set the scene change detection threshold as a percentage of maximum change on the luma plane. Good values are in the `[8.0, 14.0]` range. Scene change detection is only relevant in case `combmatch`=*sc*. The range for `scthresh` is `[0.0, 100.0]`.

Default value is `12.0`.

`combmatch`

When `combatch` is not *none*, `fieldmatch` will take into account the combed scores of matches when deciding what match to use as the final match. Available values are:

‘`none`’

No final matching based on combed scores.

‘`sc`’

Combed scores are only used when a scene change is detected.

‘`full`’

Use combed scores all the time.

Default is *sc*.

`combdbg`

Force `fieldmatch` to calculate the combed metrics for certain matches and print them. This setting is known as `micout` in TFM/VFM vocabulary. Available values are:

‘`none`’

No forced calculation.

‘`pcn`’

Force p/c/n calculations.

‘`pcnub`’

Force p/c/n/u/b calculations.

Default value is *none*.

`cthresh`

This is the area combing threshold used for combed frame detection. This essentially controls how "strong" or "visible" combing must be to be detected. Larger values mean combing must be more visible and smaller values mean combing can be less visible or strong and still be detected. Valid settings are from `-1` (every pixel will be detected as combed) to `255` (no pixel will be detected as combed). This is basically a pixel difference value. A good range is `[8, 12]`.

Default value is `9`.

`chroma`

Sets whether or not chroma is considered in the combed frame decision. Only disable this if your source has chroma problems (rainbowing, etc.) that are causing problems for the combed frame detection with chroma enabled. Actually, using `chroma`=*0* is usually more reliable, except for the case where there is chroma only combing in the source.

Default value is `0`.

`blockx`
`blocky`

Respectively set the x-axis and y-axis size of the window used during combed frame detection. This has to do with the size of the area in which `combpel` pixels are required to be detected as combed for a frame to be declared combed. See the `combpel` parameter description for more info. Possible values are any number that is a power of 2 starting at 4 and going up to 512.

Default value is `16`.

`combpel`

The number of combed pixels inside any of the `blocky` by `blockx` size blocks on the frame for the frame to be detected as combed. While `cthresh` controls how "visible" the combing must be, this setting controls "how much" combing there must be in any localized area (a window defined by the `blockx` and `blocky` settings) on the frame. Minimum value is `0` and maximum is `blocky x blockx` (at which point no frames will ever be detected as combed). This setting is known as `MI` in TFM/VFM vocabulary.

Default value is `80`.

## 9.30.1 p/c/n/u/b meaning

### 9.30.1.1 p/c/n

We assume the following telecined stream:

```
Top fields:     1 2 2 3 4
Bottom fields:  1 2 3 4 4
```

The numbers correspond to the progressive frame the fields relate to. Here, the first two frames are progressive, the 3rd and 4th are combed, and so on.

When `fieldmatch` is configured to run a matching from bottom (`field`=*bottom*) this is how this input stream get transformed:

```
Input stream:
                T     1 2 2 3 4
                B     1 2 3 4 4   <-- matching reference

Matches:              c c n n c

Output stream:
                T     1 2 3 4 4
                B     1 2 3 4 4
```

As a result of the field matching, we can see that some frames get duplicated. To perform a complete inverse telecine, you need to rely on a decimation filter after this operation. See for instance the decimate filter.

The same operation now matching from top fields (`field`=*top*) looks like this:

```
Input stream:
                T     1 2 2 3 4   <-- matching reference
                B     1 2 3 4 4

Matches:              c c p p c

Output stream:
                T     1 2 2 3 4
                B     1 2 2 3 4
```

In these examples, we can see what *p*, *c* and *n* mean; basically, they refer to the frame and field of the opposite parity:

- *p* matches the field of the opposite parity in the previous frame
- *c* matches the field of the opposite parity in the current frame
- *n* matches the field of the opposite parity in the next frame

### 9.30.1.2 u/b

The *u* and *b* matching are a bit special in the sense that they match from the opposite parity flag. In the following examples, we assume that we are currently matching the 2nd frame (Top:2, bottom:2). According to the match, a 'x' is placed above and below each matched fields.

With bottom matching (`field`=*bottom*):

```
Match:           c         p           n          b          u

                 x       x               x        x          x
  Top          1 2 2     1 2 2       1 2 2      1 2 2      1 2 2
  Bottom       1 2 3     1 2 3       1 2 3      1 2 3      1 2 3
                 x         x           x        x              x

Output frames:
                 2          1          2          2          2
                 2          2          2          1          3
```

With top matching (`field`=*top*):

```
Match:           c         p           n          b          u

                 x         x           x        x              x
  Top          1 2 2     1 2 2       1 2 2      1 2 2      1 2 2
  Bottom       1 2 3     1 2 3       1 2 3      1 2 3      1 2 3
                 x       x               x        x          x

Output frames:
                 2          2          2          1          2
                 2          1          3          2          2
```

### 9.30.2 Examples

Simple IVTC of a top field first telecined stream:

```
fieldmatch=order=tff:combmatch=none, decimate
```

Advanced IVTC, with fallback on yadif for still combed frames:

```
fieldmatch=order=tff:combmatch=full, yadif=deint=interlaced, decimate
```

## 9.31 fieldorder

Transform the field order of the input video.

It accepts the following parameters:

`order`

The output field order. Valid values are *tff* for top field first or *bff* for bottom field first.

The default value is ‘`tff`’.

The transformation is done by shifting the picture content up or down by one line, and filling the remaining line with appropriate picture content. This method is consistent with most broadcast field order converters.

If the input video is not flagged as being interlaced, or it is already flagged as being of the required output field order, then this filter does not alter the incoming video.

It is very useful when converting to or from PAL DV material, which is bottom field first.

For example:

```
ffmpeg -i in.vob -vf "fieldorder=bff" out.dv
```

## 9.32 fifo

Buffer input images and send them when they are requested.

It is mainly useful when auto-inserted by the libavfilter framework.

It does not take parameters.

## 9.33 format

Convert the input video to one of the specified pixel formats. Libavfilter will try to pick one that is suitable as input to the next filter.

It accepts the following parameters:

`pix_fmts`

A ’|’-separated list of pixel format names, such as "pix_fmts=yuv420p|monow|rgb24".

### 9.33.1 Examples

- Convert the input video to the *yuv420p* format

  ```
  format=pix_fmts=yuv420p
  ```

  Convert the input video to any of the formats in the list

  ```
  format=pix_fmts=yuv420p|yuv444p|yuv410p
  ```

## 9.34 fps

Convert the video to specified constant frame rate by duplicating or dropping frames as necessary.

It accepts the following parameters:

`fps`

The desired output frame rate. The default is `25`.

`round`

Rounding method.

Possible values are:

`zero`

zero round towards 0

`inf`

round away from 0

`down`

round towards -infinity

`up`

round towards +infinity

`near`

round to nearest

The default is `near`.

`start_time`

Assume the first PTS should be the given value, in seconds. This allows for padding/trimming at the start of stream. By default, no assumption is made about the first frame's expected PTS, so no padding or trimming is done. For example, this could be set to 0 to pad the beginning with duplicates of the first frame if a video stream starts after the audio stream or to trim any frames with a negative PTS.

Alternatively, the options can be specified as a flat string: *fps*[:*round*].

See also the setpts filter.

### 9.34.1 Examples

- A typical usage in order to set the fps to 25:

  ```
  fps=fps=25
  ```

- Sets the fps to 24, using abbreviation and rounding method to round to nearest:

  ```
  fps=fps=film:round=near
  ```

## 9.35 framepack

Pack two different video streams into a stereoscopic video, setting proper metadata on supported codecs. The two views should have the same size and framerate and processing will stop when the shorter video ends. Please note that you may conveniently adjust view properties with the scale and fps filters.

It accepts the following parameters:

`format`

The desired packing format. Supported values are:

`sbs`

The views are next to each other (default).

`tab`

The views are on top of each other.

`lines`

The views are packed by line.

`columns`

The views are packed by column.

`frameseq`

The views are temporally interleaved.

Some examples:

```
# Convert left and right views into a frame-sequential video
ffmpeg -i LEFT -i RIGHT -filter_complex framepack=frameseq OUTPUT

# Convert views into a side-by-side video with the same output resolution as the input
ffmpeg -i LEFT -i RIGHT -filter_complex [0:v]scale=w=iw/2[left],[1:v]scale=w=iw/2[right],[left][right]framepack=sbs OUTPUT
```

## 9.36 framestep

Select one frame every N-th frame.

This filter accepts the following option:

`step`

Select frame after every `step` frames. Allowed values are positive integers higher than 0. Default value is `1`.

## 9.37 frei0r

Apply a frei0r effect to the input video.

To enable the compilation of this filter, you need to install the frei0r header and configure FFmpeg with `--enable-frei0r`.

It accepts the following parameters:

`filter_name`

The name of the frei0r effect to load. If the environment variable `FREI0R_PATH` is defined, the frei0r effect is searched for in each of the directories specified by the colon-separated list in `FREIOR_PATH`. Otherwise, the standard frei0r paths are searched, in this order: `HOME/.frei0r-1/lib/`, `/usr/local/lib/frei0r-1/`, `/usr/lib/frei0r-1/`.

`filter_params`

A ’|’-separated list of parameters to pass to the frei0r effect.

A frei0r effect parameter can be a boolean (its value is either "y" or "n"), a double, a color (specified as *R*/*G*/*B*, where *R*, *G*, and *B* are floating point numbers between 0.0 and 1.0, inclusive) or by a color description specified in the "Color" section in the ffmpeg-utils manual), a position (specified as *X*/*Y*, where *X* and *Y* are floating point numbers) and/or a string.

The number and types of parameters depend on the loaded effect. If an effect parameter is not specified, the default value is set.

### 9.37.1 Examples

- Apply the distort0r effect, setting the first two double parameters:

```
frei0r=filter_name=distort0r:filter_params=0.5|0.01
```

- Apply the colordistance effect, taking a color as the first parameter:

```
frei0r=colordistance:0.2/0.3/0.4
frei0r=colordistance:violet
frei0r=colordistance:0x112233
```

- Apply the perspective effect, specifying the top left and top right image positions:

```
frei0r=perspective:0.2/0.2|0.8/0.2
```

For more information, see http://frei0r.dyne.org

## 9.38 geq

The filter accepts the following options:

`lum_expr, lum`

Set the luminance expression.

`cb_expr, cb`

Set the chrominance blue expression.

`cr_expr, cr`

Set the chrominance red expression.

`alpha_expr, a`

Set the alpha expression.

`red_expr, r`

Set the red expression.

`green_expr, g`

Set the green expression.

`blue_expr, b`

Set the blue expression.

The colorspace is selected according to the specified options. If one of the `lum_expr`, `cb_expr`, or `cr_expr` options is specified, the filter will automatically select a YCbCr colorspace. If one of the `red_expr`, `green_expr`, or `blue_expr` options is specified, it will select an RGB colorspace.

If one of the chrominance expression is not defined, it falls back on the other one. If no alpha expression is specified it will evaluate to opaque value. If none of chrominance expressions are specified, they will evaluate to the luminance expression.

The expressions can use the following variables and functions:

`N`

The sequential number of the filtered frame, starting from `0`.

`X`
`Y`

The coordinates of the current sample.

`W`
`H`

The width and height of the image.

`SW`
`SH`

Width and height scale depending on the currently filtered plane. It is the ratio between the corresponding luma plane number of pixels and the current plane ones. E.g. for YUV4:2:0 the values are `1,1` for the luma plane, and `0.5,0.5` for chroma planes.

`T`

Time of the current frame, expressed in seconds.

`p(x, y)`

Return the value of the pixel at location (*x*,*y*) of the current plane.

`lum(x, y)`

Return the value of the pixel at location (*x*,*y*) of the luminance plane.

`cb(x, y)`

Return the value of the pixel at location (*x*,*y*) of the blue-difference chroma plane. Return 0 if there is no such plane.

`cr(x, y)`

Return the value of the pixel at location (*x*,*y*) of the red-difference chroma plane. Return 0 if there is no such plane.

`r(x, y)`
`g(x, y)`
`b(x, y)`

Return the value of the pixel at location (*x*,*y*) of the red/green/blue component. Return 0 if there is no such component.

`alpha(x, y)`

Return the value of the pixel at location (*x*,*y*) of the alpha plane. Return 0 if there is no such plane.

For functions, if *x* and *y* are outside the area, the value will be automatically clipped to the closer edge.

### 9.38.1 Examples

- Flip the image horizontally:

```
geq=p(W-X\,Y)
```

- Generate a bidimensional sine wave, with angle `PI/3` and a wavelength of 100 pixels:

  ```
  geq=128 + 100*sin(2*(PI/100)*(cos(PI/3)*(X-50*T) + sin(PI/3)*Y)):128:128
  ```

- Generate a fancy enigmatic moving light:

  ```
  nullsrc=s=256x256,geq=random(1)/hypot(X-cos(N*0.07)*W/2-W/2\,Y-sin(N*0.09)*H/2-H/2)^2*1000000*sin(N*0.02):128:128
  ```

- Generate a quick emboss effect:

  ```
  format=gray,geq=lum_expr='(p(X,Y)+(256-p(X-4,Y-4)))/2'
  ```

- Modify RGB components depending on pixel position:

  ```
  geq=r='X/W*r(X,Y)':g='(1-X/W)*g(X,Y)':b='(H-Y)/H*b(X,Y)'
  ```

- Create a radial gradient that is the same size as the input (also see the vignette filter):

  ```
  geq=lum=255*gauss((X/W-0.5)*3)*gauss((Y/H-0.5)*3)/gauss(0)/gauss(0),format=gray
  ```

- Create a linear gradient to use as a mask for another filter, then compose with overlay. In this example the video will gradually become more blurry from the top to the bottom of the y-axis as defined by the linear gradient:

  ```
  ffmpeg -i input.mp4 -filter_complex "geq=lum=255*(Y/H),format=gray[grad];[0:v]boxblur=4[blur];[blur][grad]alphamerge[alpha];[0:v][alpha]overlay" output.mp4
  ```

## 9.39 gradfun

Fix the banding artifacts that are sometimes introduced into nearly flat regions by truncation to 8bit color depth. Interpolate the gradients that should go where the bands are, and dither them.

It is designed for playback only. Do not use it prior to lossy compression, because compression tends to lose the dither and bring back the bands.

It accepts the following parameters:

`strength`

> The maximum amount by which the filter will change any one pixel. This is also the threshold for detecting nearly flat regions. Acceptable values range from .51 to 64; the default value is 1.2. Out-of-range values will be clipped to the valid range.

`radius`

> The neighborhood to fit the gradient to. A larger radius makes for smoother gradients, but also prevents the filter from modifying the pixels near detailed regions. Acceptable values are 8-32; the default value is 16. Out-of-range values will be clipped to the valid range.

Alternatively, the options can be specified as a flat string: *strength*[:*radius*]

### 9.39.1 Examples

- Apply the filter with a `3.5` strength and radius of `8`:

  ```
  gradfun=3.5:8
  ```

- Specify radius, omitting the strength (which will fall-back to the default value):

  ```
  gradfun=radius=8
  ```

## 9.40 haldclut

Apply a Hald CLUT to a video stream.

First input is the video stream to process, and second one is the Hald CLUT. The Hald CLUT input can be a simple picture or a complete video stream.

The filter accepts the following options:

`shortest`

Force termination when the shortest input terminates. Default is `0`.

`repeatlast`

Continue applying the last CLUT after the end of the stream. A value of `0` disable the filter after the last frame of the CLUT is reached. Default is `1`.

`haldclut` also has the same interpolation options as lut3d (both filters share the same internals).

More information about the Hald CLUT can be found on Eskil Steenberg's website (Hald CLUT author) at http://www.quelsolaar.com/technology/clut.html.

### 9.40.1 Workflow examples

#### 9.40.1.1 Hald CLUT video stream

Generate an identity Hald CLUT stream altered with various effects:

```
ffmpeg -f lavfi -i haldclutsrc=8 -vf "hue=H=2*PI*t:s=sin(2*PI*t)+1, curves=cross_process" -t 10 -c:v ffv1 clut.nut
```

Note: make sure you use a lossless codec.

Then use it with `haldclut` to apply it on some random stream:

```
ffmpeg -f lavfi -i mandelbrot -i clut.nut -filter_complex '[0][1] haldclut' -t 20 mandelclut.mkv
```

The Hald CLUT will be applied to the 10 first seconds (duration of `clut.nut`), then the latest picture of that CLUT stream will be applied to the remaining frames of the `mandelbrot` stream.

#### 9.40.1.2 Hald CLUT with preview

A Hald CLUT is supposed to be a squared image of `Level*Level*Level` by `Level*Level*Level` pixels. For a given Hald CLUT, FFmpeg will select the biggest possible square starting at the top left of the picture. The remaining padding pixels (bottom or right) will be ignored. This area can be used to add a preview of the Hald CLUT.

Typically, the following generated Hald CLUT will be supported by the `haldclut` filter:

```
ffmpeg -f lavfi -i haldclutsrc=8 -vf "
   pad=iw+320 [padded_clut];
   smptebars=s=320x256, split [a][b];
   [padded_clut][a] overlay=W-320:h, curves=color_negative [main];
   [main][b] overlay=W-320" -frames:v 1 clut.png
```

It contains the original and a preview of the effect of the CLUT: SMPTE color bars are displayed on the right-top, and below the same color bars processed by the color changes.

Then, the effect of this Hald CLUT can be visualized with:

```
ffplay input.mkv -vf "movie=clut.png, [in] haldclut"
```

## 9.41 hflip

Flip the input video horizontally.

For example, to horizontally flip the input video with `ffmpeg`:

```
ffmpeg -i in.avi -vf "hflip" out.avi
```

## 9.42 histeq

This filter applies a global color histogram equalization on a per-frame basis.

It can be used to correct video that has a compressed range of pixel intensities. The filter redistributes the pixel intensities to equalize their distribution across the intensity range. It may be viewed as an "automatically adjusting contrast filter". This filter is useful only for correcting degraded or poorly captured source video.

The filter accepts the following options:

`strength`

: Determine the amount of equalization to be applied. As the strength is reduced, the distribution of pixel intensities more-and-more approaches that of the input frame. The value must be a float number in the range [0,1] and defaults to 0.200.

`intensity`

Set the maximum intensity that can generated and scale the output values appropriately. The strength should be set as desired and then the intensity can be limited if needed to avoid washing-out. The value must be a float number in the range [0,1] and defaults to 0.210.

`antibanding`

Set the antibanding level. If enabled the filter will randomly vary the luminance of output pixels by a small amount to avoid banding of the histogram. Possible values are `none`, `weak` or `strong`. It defaults to `none`.

## 9.43 histogram

Compute and draw a color distribution histogram for the input video.

The computed histogram is a representation of the color component distribution in an image.

The filter accepts the following options:

`mode`

Set histogram mode.

It accepts the following values:

'`levels`'

Standard histogram that displays the color components distribution in an image. Displays color graph for each color component. Shows distribution of the Y, U, V, A or R, G, B components, depending on input format, in the current frame. Below each graph a color component scale meter is shown.

'`color`'

Displays chroma values (U/V color placement) in a two dimensional graph (which is called a vectorscope). The brighter a pixel in the vectorscope, the more pixels of the input frame correspond to that pixel (i.e., more pixels have this chroma value). The V component is displayed on the horizontal (X) axis, with the leftmost side being V = 0 and the rightmost side being V = 255. The U component is displayed on the vertical (Y) axis, with the top representing U = 0 and the bottom representing U = 255.

The position of a white pixel in the graph corresponds to the chroma value of a pixel of the input clip. The graph can therefore be used to read the hue (color flavor) and the saturation (the dominance of the hue in the color). As the hue of a color changes, it moves around the square. At the center of the square the saturation is zero, which means that the corresponding pixel has no color. If the amount of a specific color is increased (while leaving the other colors unchanged) the saturation increases, and the indicator moves towards the edge of the square.

‘color2’

Chroma values in vectorscope, similar as `color` but actual chroma values are displayed.

‘waveform’

Per row/column color component graph. In row mode, the graph on the left side represents color component value 0 and the right side represents value = 255. In column mode, the top side represents color component value = 0 and bottom side represents value = 255.

Default value is `levels`.

`level_height`

Set height of level in `levels`. Default value is `200`. Allowed range is [50, 2048].

`scale_height`

Set height of color scale in `levels`. Default value is `12`. Allowed range is [0, 40].

`step`

Set step for `waveform` mode. Smaller values are useful to find out how many values of the same luminance are distributed across input rows/columns. Default value is `10`. Allowed range is [1, 255].

`waveform_mode`

Set mode for `waveform`. Can be either `row`, or `column`. Default is `row`.

`waveform_mirror`

Set mirroring mode for `waveform`. `0` means unmirrored, `1` means mirrored. In mirrored mode, higher values will be represented on the left side for `row` mode and at the top for `column` mode. Default is `0` (unmirrored).

`display_mode`

Set display mode for `waveform` and `levels`. It accepts the following values:

‘parade’

Display separate graph for the color components side by side in `row` waveform mode or one below the other in `column` waveform mode for `waveform` histogram mode. For `levels` histogram mode, per color component graphs are placed below each other.

Using this display mode in `waveform` histogram mode makes it easy to spot color casts in the highlights and shadows of an image, by comparing the contours of the top and the bottom graphs of each waveform. Since whites, grays, and blacks are characterized by exactly equal amounts of red, green, and blue, neutral areas of the picture should display three waveforms of roughly equal width/height. If not, the correction is easy to perform by making level adjustments the three waveforms.

'overlay'

Presents information identical to that in the parade, except that the graphs representing color components are superimposed directly over one another.

This display mode in waveform histogram mode makes it easier to spot relative differences or similarities in overlapping areas of the color components that are supposed to be identical, such as neutral whites, grays, or blacks.

Default is parade.

levels_mode

Set mode for levels. Can be either linear, or logarithmic. Default is linear.

### 9.43.1 Examples

- Calculate and draw histogram:

```
ffplay -i input -vf histogram
```

## 9.44 hqdn3d

This is a high precision/quality 3d denoise filter. It aims to reduce image noise, producing smooth images and making still images really still. It should enhance compressibility.

It accepts the following optional parameters:

luma_spatial

A non-negative floating point number which specifies spatial luma strength. It defaults to 4.0.

chroma_spatial

A non-negative floating point number which specifies spatial chroma strength. It defaults to 3.0**luma_spatial*/4.0.

luma_tmp

A floating point number which specifies luma temporal strength. It defaults to 6.0**luma_spatial*/4.0.

chroma_tmp

A floating point number which specifies chroma temporal strength. It defaults to *luma_tmp***chroma_spatial*/*luma_spatial*.

## 9.45 hqx

Apply a high-quality magnification filter designed for pixel art. This filter was originally created by Maxim Stepin.

It accepts the following option:

`n`

Set the scaling dimension: `2` for `hq2x`, `3` for `hq3x` and `4` for `hq4x`. Default is `3`.

## 9.46 hue

Modify the hue and/or the saturation of the input.

It accepts the following parameters:

`h`

Specify the hue angle as a number of degrees. It accepts an expression, and defaults to "0".

`s`

Specify the saturation in the [-10,10] range. It accepts an expression and defaults to "1".

`H`

Specify the hue angle as a number of radians. It accepts an expression, and defaults to "0".

`b`

Specify the brightness in the [-10,10] range. It accepts an expression and defaults to "0".

`h` and `H` are mutually exclusive, and can’t be specified at the same time.

The `b`, `h`, `H` and `s` option values are expressions containing the following constants:

`n`

frame count of the input frame starting from 0

`pts`

presentation timestamp of the input frame expressed in time base units

`r`

frame rate of the input video, NAN if the input frame rate is unknown

`t`

timestamp expressed in seconds, NAN if the input timestamp is unknown

`tb`

time base of the input video

### 9.46.1 Examples

- Set the hue to 90 degrees and the saturation to 1.0:

  ```
  hue=h=90:s=1
  ```

- Same command but expressing the hue in radians:

  ```
  hue=H=PI/2:s=1
  ```

- Rotate hue and make the saturation swing between 0 and 2 over a period of 1 second:

  ```
  hue="H=2*PI*t: s=sin(2*PI*t)+1"
  ```

- Apply a 3 seconds saturation fade-in effect starting at 0:

  ```
  hue="s=min(t/3\,1)"
  ```

  The general fade-in expression can be written as:

  ```
  hue="s=min(0\, max((t-START)/DURATION\, 1))"
  ```

- Apply a 3 seconds saturation fade-out effect starting at 5 seconds:

  ```
  hue="s=max(0\, min(1\, (8-t)/3))"
  ```

  The general fade-out expression can be written as:

  ```
  hue="s=max(0\, min(1\, (START+DURATION-t)/DURATION))"
  ```

### 9.46.2 Commands

This filter supports the following commands:

`b`
`s`
`h`
`H`

Modify the hue and/or the saturation and/or brightness of the input video. The command accepts the same syntax of the corresponding option.

If the specified expression is not valid, it is kept at its current value.

## 9.47 idet

Detect video interlacing type.

This filter tries to detect if the input frames as interlaced, progressive, top or bottom field first. It will also try and detect fields that are repeated between adjacent frames (a sign of telecine).

Single frame detection considers only immediately adjacent frames when classifying each frame. Multiple frame detection incorporates the classification history of previous frames.

The filter will log these metadata values:

`single.current_frame`

Detected type of current frame using single-frame detection. One of: “tff” (top field first), “bff” (bottom field first), “progressive”, or “undetermined”

`single.tff`

Cumulative number of frames detected as top field first using single-frame detection.

`multiple.tff`

Cumulative number of frames detected as top field first using multiple-frame detection.

`single.bff`

Cumulative number of frames detected as bottom field first using single-frame detection.

`multiple.current_frame`

Detected type of current frame using multiple-frame detection. One of: “tff” (top field first), “bff” (bottom field first), “progressive”, or “undetermined”

`multiple.bff`

Cumulative number of frames detected as bottom field first using multiple-frame detection.

`single.progressive`

Cumulative number of frames detected as progressive using single-frame detection.

`multiple.progressive`

Cumulative number of frames detected as progressive using multiple-frame detection.

`single.undetermined`

Cumulative number of frames that could not be classified using single-frame detection.

`multiple.undetermined`

Cumulative number of frames that could not be classified using multiple-frame detection.

`repeated.current_frame`

Which field in the current frame is repeated from the last. One of “neither”, “top”, or “bottom”.

`repeated.neither`

Cumulative number of frames with no repeated field.

`repeated.top`

Cumulative number of frames with the top field repeated from the previous frame’s top field.

`repeated.bottom`

Cumulative number of frames with the bottom field repeated from the previous frame’s bottom field.

The filter accepts the following options:

`intl_thres`

Set interlacing threshold.

`prog_thres`

Set progressive threshold.

`repeat_thres`

Threshold for repeated field detection.

`half_life`

Number of frames after which a given frame’s contribution to the statistics is halved (i.e., it contributes only 0.5 to it’s classification). The default of 0 means that all frames seen are given full weight of 1.0 forever.

## 9.48 il

Deinterleave or interleave fields.

This filter allows one to process interlaced images fields without deinterlacing them. Deinterleaving splits the input frame into 2 fields (so called half pictures). Odd lines are moved to the top half of the output image, even lines to the bottom half. You can process (filter) them independently and then re-interleave them.

The filter accepts the following options:

`luma_mode, l`
`chroma_mode, c`
`alpha_mode, a`

Available values for *luma_mode*, *chroma_mode* and *alpha_mode* are:

'`none`'

Do nothing.

'`deinterleave, d`'

Deinterleave fields, placing one above the other.

'`interleave, i`'

Interleave fields. Reverse the effect of deinterleaving.

Default value is `none`.

`luma_swap, ls`
`chroma_swap, cs`
`alpha_swap, as`

Swap luma/chroma/alpha fields. Exchange even & odd lines. Default value is `0`.

## 9.49 interlace

Simple interlacing filter from progressive contents. This interleaves upper (or lower) lines from odd frames with lower (or upper) lines from even frames, halving the frame rate and preserving image height.

```
   Original        Original             New Frame
   Frame 'j'      Frame 'j+1'             (tff)
  ==========      ===========       ==================
    Line 0  -------------------->    Frame 'j' Line 0
    Line 1          Line 1  ---->   Frame 'j+1' Line 1
    Line 2 --------------------->    Frame 'j' Line 2
    Line 3          Line 3  ---->   Frame 'j+1' Line 3
     ...             ...                   ...
New Frame + 1 will be generated by Frame 'j+2' and Frame 'j+3' and so on
```

It accepts the following optional parameters:

`scan`

This determines whether the interlaced frame is taken from the even (tff - default) or odd (bff) lines of the progressive frame.

`lowpass`

Enable (default) or disable the vertical lowpass filter to avoid twitter interlacing and reduce moire patterns.

## 9.50 kerndeint

Deinterlace input video by applying Donald Graft’s adaptive kernel deinterling. Work on interlaced parts of a video to produce progressive frames.

The description of the accepted parameters follows.

`thresh`

Set the threshold which affects the filter’s tolerance when determining if a pixel line must be processed. It must be an integer in the range [0,255] and defaults to 10. A value of 0 will result in applying the process on every pixels.

`map`

Paint pixels exceeding the threshold value to white if set to 1. Default is 0.

`order`

Set the fields order. Swap fields if set to 1, leave fields alone if 0. Default is 0.

`sharp`

Enable additional sharpening if set to 1. Default is 0.

`twoway`

Enable twoway sharpening if set to 1. Default is 0.

### 9.50.1 Examples

- Apply default values:

```
kerndeint=thresh=10:map=0:order=0:sharp=0:twoway=0
```

- Enable additional sharpening:

```
kerndeint=sharp=1
```

- Paint processed pixels in white:

```
kerndeint=map=1
```

## 9.51 lenscorrection

Correct radial lens distortion

This filter can be used to correct for radial distortion as can result from the use of wide angle lenses, and thereby re-rectify the image. To find the right parameters one can use tools available for example as part of opencv or simply trial-and-error. To use opencv use the calibration sample (under samples/cpp) from the opencv sources and extract the k1 and k2 coefficients from the resulting matrix.

Note that effectively the same filter is available in the open-source tools Krita and Digikam from the KDE project.

In contrast to the vignette filter, which can also be used to compensate lens errors, this filter corrects the distortion of the image, whereas vignette corrects the brightness distribution, so you may want to use both filters together in certain cases, though you will have to take care of ordering, i.e. whether vignetting should be applied before or after lens correction.

### 9.51.1 Options

The filter accepts the following options:

`cx`

Relative x-coordinate of the focal point of the image, and thereby the center of the distrortion. This value has a range [0,1] and is expressed as fractions of the image width.

`cy`

Relative y-coordinate of the focal point of the image, and thereby the center of the distrortion. This value has a range [0,1] and is expressed as fractions of the image height.

`k1`

Coefficient of the quadratic correction term. 0.5 means no correction.

`k2`

Coefficient of the double quadratic correction term. 0.5 means no correction.

The formula that generates the correction is:

$r\_src = r\_tgt * (1 + k1 * (r\_tgt / r\_0)^2 + k2 * (r\_tgt / r\_0)^4)$

where *r_0* is halve of the image diagonal and *r_src* and *r_tgt* are the distances from the focal point in the source and target images, respectively.

## 9.52 lut3d

Apply a 3D LUT to an input video.

The filter accepts the following options:

`file`

Set the 3D LUT file name.

Currently supported formats:

‘`3dl`’

AfterEffects

‘`cube`’

Iridas

‘`dat`’

DaVinci

‘`m3d`’

Pandora

`interp`

Select interpolation mode.

Available values are:

‘`nearest`’

Use values from the nearest defined point.

‘`trilinear`’

Interpolate values using the 8 points defining a cube.

‘`tetrahedral`’

Interpolate values using a tetrahedron.

## 9.53 lut, lutrgb, lutyuv

Compute a look-up table for binding each pixel component input value to an output value, and apply it to the input video.

*lutyuv* applies a lookup table to a YUV input video, *lutrgb* to an RGB input video.

These filters accept the following parameters:

`c0`

set first pixel component expression

`c1`

set second pixel component expression

`c2`

set third pixel component expression

`c3`

set fourth pixel component expression, corresponds to the alpha component

`r`

set red component expression

`g`

set green component expression

`b`

set blue component expression

`a`

alpha component expression

`y`

set Y/luminance component expression

`u`

set U/Cb component expression

`v`

set V/Cr component expression

Each of them specifies the expression to use for computing the lookup table for the corresponding pixel component values.

The exact component associated to each of the *c** options depends on the format in input.

The *lut* filter requires either YUV or RGB pixel formats in input, *lutrgb* requires RGB pixel formats in input, and *lutyuv* requires YUV.

The expressions can contain the following constants and functions:

`w`
`h`

The input width and height.

`val`

The input value for the pixel component.

`clipval`

The input value, clipped to the *minval*-*maxval* range.

`maxval`

The maximum value for the pixel component.

`minval`

The minimum value for the pixel component.

`negval`

The negated value for the pixel component value, clipped to the *minval*-*maxval* range; it corresponds to the expression "maxval-clipval+minval".

`clip(val)`

The computed value in *val*, clipped to the *minval*-*maxval* range.

`gammaval(gamma)`

The computed gamma correction value of the pixel component value, clipped to the *minval*-*maxval* range. It corresponds to the expression "pow((clipval-minval)/(maxval-minval)\,*gamma*)*(maxval-minval)+minval"

All expressions default to "val".

### 9.53.1 Examples

- Negate input video:

  ```
  lutrgb="r=maxval+minval-val:g=maxval+minval-val:b=maxval+minval-val"
  lutyuv="y=maxval+minval-val:u=maxval+minval-val:v=maxval+minval-val"
  ```

  The above is the same as:

  ```
  lutrgb="r=negval:g=negval:b=negval"
  lutyuv="y=negval:u=negval:v=negval"
  ```

- Negate luminance:

  ```
  lutyuv=y=negval
  ```

- Remove chroma components, turning the video into a graytone image:

  ```
  lutyuv="u=128:v=128"
  ```

- Apply a luma burning effect:

  ```
  lutyuv="y=2*val"
  ```

- Remove green and blue components:

  ```
  lutrgb="g=0:b=0"
  ```

- Set a constant alpha channel value on input:

  ```
  format=rgba,lutrgb=a="maxval-minval/2"
  ```

- Correct luminance gamma by a factor of 0.5:

  ```
  lutyuv=y=gammaval(0.5)
  ```

- Discard least significant bits of luma:

  ```
  lutyuv=y='bitand(val, 128+64+32)'
  ```

## 9.54 mergeplanes

Merge color channel components from several video streams.

The filter accepts up to 4 input streams, and merge selected input planes to the output video.

This filter accepts the following options:

`mapping`

Set input to output plane mapping. Default is `0`.

The mappings is specified as a bitmap. It should be specified as a hexadecimal number in the form 0xAa[Bb[Cc[Dd]]]. 'Aa' describes the mapping for the first plane of the output stream. 'A' sets the number of the input stream to use (from 0 to 3), and 'a' the plane number of the corresponding input to use (from 0 to 3). The rest of the mappings is similar, 'Bb' describes the mapping for the output stream second plane, 'Cc' describes the mapping for the output stream third plane and 'Dd' describes the mapping for the output stream fourth plane.

`format`

Set output pixel format. Default is `yuva444p`.

### 9.54.1 Examples

- Merge three gray video streams of same width and height into single video stream:

```
[a0][a1][a2]mergeplanes=0x001020:yuv444p
```

- Merge 1st yuv444p stream and 2nd gray video stream into yuva444p video stream:

```
[a0][a1]mergeplanes=0x00010210:yuva444p
```

- Swap Y and A plane in yuva444p stream:

```
format=yuva444p,mergeplanes=0x03010200:yuva444p
```

- Swap U and V plane in yuv420p stream:

```
format=yuv420p,mergeplanes=0x000201:yuv420p
```

- Cast a rgb24 clip to yuv444p:

```
format=rgb24,mergeplanes=0x000102:yuv444p
```

## 9.55 mcdeint

Apply motion-compensation deinterlacing.

It needs one field per frame as input and must thus be used together with yadif=1/3 or equivalent.

This filter accepts the following options:

`mode`

Set the deinterlacing mode.

It accepts one of the following values:

‘`fast`’
‘`medium`’
‘`slow`’

use iterative motion estimation

‘`extra_slow`’

like ‘`slow`’, but use multiple reference frames.

Default value is ‘`fast`’.

`parity`

Set the picture field parity assumed for the input video. It must be one of the following values:

‘`0, tff`’

assume top field first

‘`1, bff`’

assume bottom field first

Default value is ‘`bff`’.

`qp`

Set per-block quantization parameter (QP) used by the internal encoder.

Higher values should result in a smoother motion vector field but less optimal individual vectors. Default value is 1.

## 9.56 mp

Apply an MPlayer filter to the input video.

This filter provides a wrapper around some of the filters of MPlayer/MEncoder.

This wrapper is considered experimental. Some of the wrapped filters may not work properly and we may drop support for them, as they will be implemented natively into FFmpeg. Thus you should avoid depending on them when writing portable scripts.

The filter accepts the parameters: *filter_name*[:=]*filter_params*

*filter_name* is the name of a supported MPlayer filter, *filter_params* is a string containing the parameters accepted by the named filter.

The list of the currently supported filters follows:

*eq2*
*eq*
*fspp*
*ilpack*
*pp7*
*softpulldown*
*uspp*

The parameter syntax and behavior for the listed filters are the same of the corresponding MPlayer filters. For detailed instructions check the "VIDEO FILTERS" section in the MPlayer manual.

### 9.56.1 Examples

- Adjust gamma, brightness, contrast:

  ```
  mp=eq2=1.0:2:0.5
  ```

See also mplayer(1), http://www.mplayerhq.hu/.

## 9.57 mpdecimate

Drop frames that do not differ greatly from the previous frame in order to reduce frame rate.

The main use of this filter is for very-low-bitrate encoding (e.g. streaming over dialup modem), but it could in theory be used for fixing movies that were inverse-telecined incorrectly.

A description of the accepted options follows.

`max`

> Set the maximum number of consecutive frames which can be dropped (if positive), or the minimum interval between dropped frames (if negative). If the value is 0, the frame is dropped unregarding the number of previous sequentially dropped frames.
>
> Default value is 0.

`hi`
`lo`
`frac`

Set the dropping threshold values.

Values for `hi` and `lo` are for 8x8 pixel blocks and represent actual pixel value differences, so a threshold of 64 corresponds to 1 unit of difference for each pixel, or the same spread out differently over the block.

A frame is a candidate for dropping if no 8x8 blocks differ by more than a threshold of `hi`, and if no more than `frac` blocks (1 meaning the whole image) differ by more than a threshold of `lo`.

Default value for `hi` is 64*12, default value for `lo` is 64*5, and default value for `frac` is 0.33.

## 9.58 negate

Negate input video.

It accepts an integer in input; if non-zero it negates the alpha component (if available). The default value in input is 0.

## 9.59 noformat

Force libavfilter not to use any of the specified pixel formats for the input to the next filter.

It accepts the following parameters:

`pix_fmts`

A '|'-separated list of pixel format names, such as apix_fmts=yuv420p|monow|rgb24".

### 9.59.1 Examples

- Force libavfilter to use a format different from *yuv420p* for the input to the vflip filter:

```
noformat=pix_fmts=yuv420p,vflip
```

- Convert the input video to any of the formats not contained in the list:

```
noformat=yuv420p|yuv444p|yuv410p
```

## 9.60 noise

Add noise on video input frame.

The filter accepts the following options:

`all_seed`
`c0_seed`
`c1_seed`

`c2_seed`
`c3_seed`

Set noise seed for specific pixel component or all pixel components in case of *all_seed*. Default value is `123457`.

`all_strength, alls`
`c0_strength, c0s`
`c1_strength, c1s`
`c2_strength, c2s`
`c3_strength, c3s`

Set noise strength for specific pixel component or all pixel components in case *all_strength*. Default value is `0`. Allowed range is [0, 100].

`all_flags, allf`
`c0_flags, c0f`
`c1_flags, c1f`
`c2_flags, c2f`
`c3_flags, c3f`

Set pixel component flags or set flags for all components if *all_flags*. Available values for component flags are:

‘`a`’

averaged temporal noise (smoother)

‘`p`’

mix random noise with a (semi)regular pattern

‘`t`’

temporal noise (noise pattern changes between frames)

‘`u`’

uniform noise (gaussian otherwise)

### 9.60.1 Examples

Add temporal and uniform noise to input video:

```
noise=alls=20:allf=t+u
```

## 9.61 null

Pass the video source unchanged to the output.

## 9.62 ocv

Apply a video transform using libopencv.

To enable this filter, install the libopencv library and headers and configure FFmpeg with `--enable-libopencv`.

It accepts the following parameters:

`filter_name`

  The name of the libopencv filter to apply.

`filter_params`

  The parameters to pass to the libopencv filter. If not specified, the default values are assumed.

Refer to the official libopencv documentation for more precise information: http://docs.opencv.org/master/modules/imgproc/doc/filtering.html

Several libopencv filters are supported; see the following subsections.

### 9.62.1 dilate

Dilate an image by using a specific structuring element. It corresponds to the libopencv function `cvDilate`.

It accepts the parameters: *struct_el*|*nb_iterations*.

*struct_el* represents a structuring element, and has the syntax: *cols*x*rows*+*anchor_x*x*anchor_y*/*shape*

*cols* and *rows* represent the number of columns and rows of the structuring element, *anchor_x* and *anchor_y* the anchor point, and *shape* the shape for the structuring element. *shape* must be "rect", "cross", "ellipse", or "custom".

If the value for *shape* is "custom", it must be followed by a string of the form "=*filename*". The file with name *filename* is assumed to represent a binary image, with each printable character corresponding to a bright pixel. When a custom *shape* is used, *cols* and *rows* are ignored, the number or columns and rows of the read file are assumed instead.

The default value for *struct_el* is "3x3+0x0/rect".

*nb_iterations* specifies the number of times the transform is applied to the image, and defaults to 1.

Some examples:

```
# Use the default values
ocv=dilate

# Dilate using a structuring element with a 5x5 cross, iterating two times
ocv=filter_name=dilate:filter_params=5x5+2x2/cross|2

# Read the shape from the file diamond.shape, iterating two times.
# The file diamond.shape may contain a pattern of characters like this
#   *
#  ***
# *****
#  ***
#   *
# The specified columns and rows are ignored
# but the anchor point coordinates are not
ocv=dilate:0x0+2x2/custom=diamond.shape|2
```

### 9.62.2 erode

Erode an image by using a specific structuring element. It corresponds to the libopencv function `cvErode`.

It accepts the parameters: *struct_el*:*nb_iterations*, with the same syntax and semantics as the dilate filter.

### 9.62.3 smooth

Smooth the input video.

The filter takes the following parameters: *type*|*param1*|*param2*|*param3*|*param4*.

*type* is the type of smooth filter to apply, and must be one of the following values: "blur", "blur_no_scale", "median", "gaussian", or "bilateral". The default value is "gaussian".

The meaning of *param1*, *param2*, *param3*, and *param4* depend on the smooth type. *param1* and *param2* accept integer positive values or 0. *param3* and *param4* accept floating point values.

The default value for *param1* is 3. The default value for the other parameters is 0.

These parameters correspond to the parameters assigned to the libopencv function `cvSmooth`.

## 9.63 overlay

Overlay one video on top of another.

It takes two inputs and has one output. The first input is the "main" video on which the second input is overlayed.

It accepts the following parameters:

A description of the accepted options follows.

`x`
`y`

Set the expression for the x and y coordinates of the overlayed video on the main video. Default value is "0" for both expressions. In case the expression is invalid, it is set to a huge value (meaning that the overlay will not be displayed within the output visible area).

`eof_action`

The action to take when EOF is encountered on the secondary input; it accepts one of the following values:

`repeat`

Repeat the last frame (the default).

`endall`

End both streams.

`pass`

Pass the main input through.

`eval`

Set when the expressions for `x`, and `y` are evaluated.

It accepts the following values:

‘`init`’

only evaluate expressions once during the filter initialization or when a command is processed

‘`frame`’

evaluate expressions for each incoming frame

Default value is ‘`frame`’.

`shortest`

If set to 1, force the output to terminate when the shortest input terminates. Default value is 0.

`format`

Set the format for the output video.

It accepts the following values:

'`yuv420`'

force YUV420 output

'`yuv422`'

force YUV422 output

'`yuv444`'

force YUV444 output

'`rgb`'

force RGB output

Default value is '`yuv420`'.

`rgb` *(deprecated)*

If set to 1, force the filter to accept inputs in the RGB color space. Default value is 0. This option is deprecated, use `format` instead.

`repeatlast`

If set to 1, force the filter to draw the last overlay frame over the main input until the end of the stream. A value of 0 disables this behavior. Default value is 1.

The `x`, and `y` expressions can contain the following parameters.

`main_w, W`
`main_h, H`

The main input width and height.

`overlay_w, w`
`overlay_h, h`

The overlay input width and height.

`x`
`y`

The computed values for *x* and *y*. They are evaluated for each new frame.

`hsub`
`vsub`

horizontal and vertical chroma subsample values of the output format. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

`n`

the number of input frame, starting from 0

`pos`

the position in the file of the input frame, NAN if unknown

`t`

The timestamp, expressed in seconds. It’s NAN if the input timestamp is unknown.

Note that the *n*, *pos*, *t* variables are available only when evaluation is done *per frame*, and will evaluate to NAN when `eval` is set to ‘`init`’.

Be aware that frames are taken from each input video in timestamp order, hence, if their initial timestamps differ, it is a good idea to pass the two inputs through a *setpts=PTS-STARTPTS* filter to have them begin in the same zero timestamp, as the example for the *movie* filter does.

You can chain together more overlays but you should test the efficiency of such approach.

### 9.63.1 Commands

This filter supports the following commands:

`x`
`y`

Modify the x and y of the overlay input. The command accepts the same syntax of the corresponding option.

If the specified expression is not valid, it is kept at its current value.

### 9.63.2 Examples

- Draw the overlay at 10 pixels from the bottom right corner of the main video:

  ```
  overlay=main_w-overlay_w-10:main_h-overlay_h-10
  ```

Using named options the example above becomes:

```
overlay=x=main_w-overlay_w-10:y=main_h-overlay_h-10
```

- Insert a transparent PNG logo in the bottom left corner of the input, using the `ffmpeg` tool with the `-filter_complex` option:

  ```
  ffmpeg -i input -i logo -filter_complex 'overlay=10:main_h-overlay_h-10' output
  ```

- Insert 2 different transparent PNG logos (second logo on bottom right corner) using the `ffmpeg` tool:

  ```
  ffmpeg -i input -i logo1 -i logo2 -filter_complex 'overlay=x=10:y=H-h-10,overlay=x=W-w-10:y=H-h-10' output
  ```

- Add a transparent color layer on top of the main video; `WxH` must specify the size of the main input to the overlay filter:

  ```
  color=color=red@.3:size=WxH [over]; [in][over] overlay [out]
  ```

- Play an original video and a filtered version (here with the deshake filter) side by side using the `ffplay` tool:

  ```
  ffplay input.avi -vf 'split[a][b]; [a]pad=iw*2:ih[src]; [b]deshake[filt]; [src][filt]overlay=w'
  ```

  The above command is the same as:

  ```
  ffplay input.avi -vf 'split[b], pad=iw*2[src], [b]deshake, [src]overlay=w'
  ```

- Make a sliding overlay appearing from the left to the right top part of the screen starting since time 2:

  ```
  overlay=x='if(gte(t,2), -w+(t-2)*20, NAN)':y=0
  ```

- Compose output by putting two input videos side to side:

  ```
  ffmpeg -i left.avi -i right.avi -filter_complex "
  nullsrc=size=200x100 [background];
  [0:v] setpts=PTS-STARTPTS, scale=100x100 [left];
  [1:v] setpts=PTS-STARTPTS, scale=100x100 [right];
  [background][left]       overlay=shortest=1       [background+left];
  [background+left][right] overlay=shortest=1:x=100 [left+right]
  "
  ```

- Mask 10-20 seconds of a video by applying the delogo filter to a section

  ```
  ffmpeg -i test.avi -codec:v:0 wmv2 -ar 11025 -b:v 9000k
  -vf '[in]split[split_main][split_delogo];[split_delogo]trim=start=360:end=371,delogo=0:0:640:480[delogoed];[split_main][delogoed]overlay=eof_action=pass[out]'
  masked.avi
  ```

- Chain several overlays in cascade:

  ```
  nullsrc=s=200x200 [bg];
  testsrc=s=100x100, split=4 [in0][in1][in2][in3];
  [in0] lutrgb=r=0, [bg]   overlay=0:0     [mid0];
  [in1] lutrgb=g=0, [mid0] overlay=100:0   [mid1];
  [in2] lutrgb=b=0, [mid1] overlay=0:100   [mid2];
  [in3] null,       [mid2] overlay=100:100 [out0]
  ```

## 9.64 owdenoise

Apply Overcomplete Wavelet denoiser.

The filter accepts the following options:

`depth`

Set depth.

Larger depth values will denoise lower frequency components more, but slow down filtering.

Must be an int in the range 8-16, default is `8`.

`luma_strength, ls`

Set luma strength.

Must be a double value in the range 0-1000, default is `1.0`.

`chroma_strength, cs`

Set chroma strength.

Must be a double value in the range 0-1000, default is `1.0`.

## 9.65 pad

Add paddings to the input image, and place the original input at the provided *x*, *y* coordinates.

It accepts the following parameters:

`width, w`
`height, h`

Specify an expression for the size of the output image with the paddings added. If the value for *width* or *height* is 0, the corresponding input size is used for the output.

The *width* expression can reference the value set by the *height* expression, and vice versa.

The default value of *width* and *height* is 0.

`x`
`y`

Specify the offsets to place the input image at within the padded area, with respect to the top/left border of the output image.

The *x* expression can reference the value set by the *y* expression, and vice versa.

The default value of *x* and *y* is 0.

`color`

Specify the color of the padded area. For the syntax of this option, check the "Color" section in the ffmpeg-utils manual.

The default value of *color* is "black".

The value for the *width*, *height*, *x*, and *y* options are expressions containing the following constants:

`in_w`
`in_h`

The input video width and height.

`iw`
`ih`

These are the same as *in_w* and *in_h*.

`out_w`
`out_h`

The output width and height (the size of the padded area), as specified by the *width* and *height* expressions.

`ow`
`oh`

These are the same as *out_w* and *out_h*.

`x`
`y`

The x and y offsets as specified by the *x* and *y* expressions, or NAN if not yet specified.

`a`

same as *iw* / *ih*

`sar`

input sample aspect ratio

`dar`

input display aspect ratio, it is the same as (*iw* / *ih*) * *sar*

`hsub`
`vsub`

The horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

### 9.65.1 Examples

- Add paddings with the color "violet" to the input video. The output video size is 640x480, and the top-left corner of the input video is placed at column 0, row 40

  ```
  pad=640:480:0:40:violet
  ```

  The example above is equivalent to the following command:

  ```
  pad=width=640:height=480:x=0:y=40:color=violet
  ```

- Pad the input to get an output with dimensions increased by 3/2, and put the input video at the center of the padded area:

  ```
  pad="3/2*iw:3/2*ih:(ow-iw)/2:(oh-ih)/2"
  ```

- Pad the input to get a squared output with size equal to the maximum value between the input width and height, and put the input video at the center of the padded area:

  ```
  pad="max(iw\,ih):ow:(ow-iw)/2:(oh-ih)/2"
  ```

- Pad the input to get a final w/h ratio of 16:9:

  ```
  pad="ih*16/9:ih:(ow-iw)/2:(oh-ih)/2"
  ```

- In case of anamorphic video, in order to set the output display aspect correctly, it is necessary to use *sar* in the expression, according to the relation:

  ```
  (ih * X / ih) * sar = output_dar
  X = output_dar / sar
  ```

  Thus the previous example needs to be modified to:

  ```
  pad="ih*16/9/sar:ih:(ow-iw)/2:(oh-ih)/2"
  ```

- Double the output size and put the input video in the bottom-right corner of the output padded area:

  ```
  pad="2*iw:2*ih:ow-iw:oh-ih"
  ```

## 9.66 perspective

Correct perspective of video not recorded perpendicular to the screen.

A description of the accepted parameters follows.

`x0`
`y0`
`x1`
`y1`
`x2`
`y2`
`x3`
`y3`

Set coordinates expression for top left, top right, bottom left and bottom right corners. Default values are `0:0:W:0:0:H:W:H` with which perspective will remain unchanged. If the `sense` option is set to `source`, then the specified points will be sent to the corners of the destination. If the `sense` option is set to `destination`, then the corners of the source will be sent to the specified coordinates.

The expressions can use the following variables:

`W`
`H`

the width and height of video frame.

`interpolation`

Set interpolation for perspective correction.

It accepts the following values:

'`linear`'
'`cubic`'

Default value is '`linear`'.

`sense`

Set interpretation of coordinate options.

It accepts the following values:

'`0, source`'

Send point in the source specified by the given coordinates to the corners of the destination.

'`1, destination`'

Send the corners of the source to the point in the destination specified by the given coordinates.

Default value is '`source`'.

## 9.67 phase

Delay interlaced video by one field time so that the field order changes.

The intended use is to fix PAL movies that have been captured with the opposite field order to the film-to-video transfer.

A description of the accepted parameters follows.

`mode`

Set phase mode.

It accepts the following values:

'`t`'

Capture field order top-first, transfer bottom-first. Filter will delay the bottom field.

'`b`'

Capture field order bottom-first, transfer top-first. Filter will delay the top field.

'`p`'

Capture and transfer with the same field order. This mode only exists for the documentation of the other options to refer to, but if you actually select it, the filter will faithfully do nothing.

'`a`'

Capture field order determined automatically by field flags, transfer opposite. Filter selects among '`t`' and '`b`' modes on a frame by frame basis using field flags. If no field information is available, then this works just like '`u`'.

'`u`'

Capture unknown or varying, transfer opposite. Filter selects among '`t`' and '`b`' on a frame by frame basis by analyzing the images and selecting the alternative that produces best match between the fields.

‘T’

Capture top-first, transfer unknown or varying. Filter selects among ‘t’ and ‘p’ using image analysis.

‘B’

Capture bottom-first, transfer unknown or varying. Filter selects among ‘b’ and ‘p’ using image analysis.

‘A’

Capture determined by field flags, transfer unknown or varying. Filter selects among ‘t’, ‘b’ and ‘p’ using field flags and image analysis. If no field information is available, then this works just like ‘U’. This is the default mode.

‘U’

Both capture and transfer unknown or varying. Filter selects among ‘t’, ‘b’ and ‘p’ using image analysis only.

## 9.68 pixdesctest

Pixel format descriptor test filter, mainly useful for internal testing. The output video should be equal to the input video.

For example:

```
format=monow, pixdesctest
```

can be used to test the monowhite pixel format descriptor definition.

## 9.69 pp

Enable the specified chain of postprocessing subfilters using libpostproc. This library should be automatically selected with a GPL build (`--enable-gpl`). Subfilters must be separated by ’/’ and can be disabled by prepending a ’-’. Each subfilter and some options have a short and a long name that can be used interchangeably, i.e. dr/dering are the same.

The filters accept the following options:

`subfilters`

Set postprocessing subfilters string.

All subfilters share common options to determine their scope:

`a/autoq`

Honor the quality commands for this subfilter.

`c/chrom`

Do chrominance filtering, too (default).

`y/nochrom`

Do luminance filtering only (no chrominance).

`n/noluma`

Do chrominance filtering only (no luminance).

These options can be appended after the subfilter name, separated by a ’|’.

Available subfilters are:

`hb/hdeblock[|difference[|flatness]]`

Horizontal deblocking filter

`difference`

Difference factor where higher values mean more deblocking (default: `32`).

`flatness`

Flatness threshold where lower values mean more deblocking (default: `39`).

`vb/vdeblock[|difference[|flatness]]`

Vertical deblocking filter

`difference`

Difference factor where higher values mean more deblocking (default: `32`).

`flatness`

Flatness threshold where lower values mean more deblocking (default: `39`).

`ha/hadeblock[|difference[|flatness]]`

Accurate horizontal deblocking filter

`difference`

Difference factor where higher values mean more deblocking (default: `32`).

`flatness`

Flatness threshold where lower values mean more deblocking (default: `39`).

`va/vadeblock[|difference[|flatness]]`

Accurate vertical deblocking filter

`difference`

Difference factor where higher values mean more deblocking (default: `32`).

`flatness`

Flatness threshold where lower values mean more deblocking (default: `39`).

The horizontal and vertical deblocking filters share the difference and flatness values so you cannot set different horizontal and vertical thresholds.

`h1/x1hdeblock`

Experimental horizontal deblocking filter

`v1/x1vdeblock`

Experimental vertical deblocking filter

`dr/dering`

Deringing filter

`tn/tmpnoise[|threshold1[|threshold2[|threshold3]]], temporal noise reducer`

`threshold1`

larger -> stronger filtering

`threshold2`

larger -> stronger filtering

`threshold3`

larger -> stronger filtering

`al/autolevels[:f/fullyrange], automatic brightness / contrast correction` `f/fullyrange`

Stretch luminance to `0-255`.

`lb/linblenddeint`

Linear blend deinterlacing filter that deinterlaces the given block by filtering all lines with a `(1 2 1)` filter.

`li/linipoldeint`

Linear interpolating deinterlacing filter that deinterlaces the given block by linearly interpolating every second line.

`ci/cubicipoldeint`

Cubic interpolating deinterlacing filter deinterlaces the given block by cubically interpolating every second line.

`md/mediandeint`

Median deinterlacing filter that deinterlaces the given block by applying a median filter to every second line.

`fd/ffmpegdeint`

FFmpeg deinterlacing filter that deinterlaces the given block by filtering every second line with a `(-1 4 2 4 -1)` filter.

`l5/lowpass5`

Vertically applied FIR lowpass deinterlacing filter that deinterlaces the given block by filtering all lines with a `(-1 2 6 2 -1)` filter.

`fq/forceQuant[|quantizer]`

Overrides the quantizer table from the input with the constant quantizer you specify.

`quantizer`

Quantizer to use

`de/default`

Default pp filter combination (`hb|a,vb|a,dr|a`)

`fa/fast`

Fast pp filter combination (`h1|a,v1|a,dr|a`)

`ac`

High quality pp filter combination (`ha|a|128|7,va|a,dr|a`)

### 9.69.1 Examples

- Apply horizontal and vertical deblocking, deringing and automatic brightness/contrast:

  ```
  pp=hb/vb/dr/al
  ```

- Apply default filters without brightness/contrast correction:

  ```
  pp=de/-al
  ```

- Apply default filters and temporal denoiser:

  ```
  pp=default/tmpnoise|1|2|3
  ```

- Apply deblocking on luminance only, and switch vertical deblocking on or off automatically depending on available CPU time:

  ```
  pp=hb|y/vb|a
  ```

## 9.70 psnr

Obtain the average, maximum and minimum PSNR (Peak Signal to Noise Ratio) between two input videos.

This filter takes in input two input videos, the first input is considered the "main" source and is passed unchanged to the output. The second input is used as a "reference" video for computing the PSNR.

Both video inputs must have the same resolution and pixel format for this filter to work correctly. Also it assumes that both inputs have the same number of frames, which are compared one by one.

The obtained average PSNR is printed through the logging system.

The filter stores the accumulated MSE (mean squared error) of each frame, and at the end of the processing it is averaged across all frames equally, and the following formula is applied to obtain the PSNR:

```
PSNR = 10*log10(MAX^2/MSE)
```

Where MAX is the average of the maximum values of each component of the image.

The description of the accepted parameters follows.

`stats_file, f`

If specified the filter will use the named file to save the PSNR of each individual frame.

The file printed if *stats_file* is selected, contains a sequence of key/value pairs of the form *key*:*value* for each compared couple of frames.

A description of each shown parameter follows:

`n`

sequential number of the input frame, starting from 1

`mse_avg`

Mean Square Error pixel-by-pixel average difference of the compared frames, averaged over all the image components.

`mse_y, mse_u, mse_v, mse_r, mse_g, mse_g, mse_a`

Mean Square Error pixel-by-pixel average difference of the compared frames for the component specified by the suffix.

`psnr_y, psnr_u, psnr_v, psnr_r, psnr_g, psnr_b, psnr_a`

Peak Signal to Noise ratio of the compared frames for the component specified by the suffix.

For example:

```
movie=ref_movie.mpg, setpts=PTS-STARTPTS [main];
[main][ref] psnr="stats_file=stats.log" [out]
```

On this example the input file being processed is compared with the reference file `ref_movie.mpg`. The PSNR of each individual frame is stored in `stats.log`.

## 9.71 pullup

Pulldown reversal (inverse telecine) filter, capable of handling mixed hard-telecine, 24000/1001 fps progressive, and 30000/1001 fps progressive content.

The pullup filter is designed to take advantage of future context in making its decisions. This filter is stateless in the sense that it does not lock onto a pattern to follow, but it instead looks forward to the following fields in order to identify matches and rebuild progressive frames.

To produce content with an even framerate, insert the fps filter after pullup, use `fps=24000/1001` if the input frame rate is 29.97fps, `fps=24` for 30fps and the (rare) telecined 25fps input.

The filter accepts the following options:

`jl`
`jr`
`jt`
`jb`

These options set the amount of "junk" to ignore at the left, right, top, and bottom of the image, respectively. Left and right are in units of 8 pixels, while top and bottom are in units of 2 lines. The default is 8 pixels on each side.

`sb`

Set the strict breaks. Setting this option to 1 will reduce the chances of filter generating an occasional mismatched frame, but it may also cause an excessive number of frames to be dropped during high motion sequences. Conversely, setting it to `-1` will make filter match fields more easily. This may help processing of video where there is slight blurring between the fields, but may also cause there to be interlaced frames in the output. Default value is `0`.

`mp`

Set the metric plane to use. It accepts the following values:

‘`l`’

Use luma plane.

‘`u`’

Use chroma blue plane.

‘`v`’

Use chroma red plane.

This option may be set to use chroma plane instead of the default luma plane for doing filter’s computations. This may improve accuracy on very clean source material, but more likely will decrease accuracy, especially if there is chroma noise (rainbow effect) or any grayscale video. The main purpose of setting `mp` to a chroma plane is to reduce CPU load and make pullup usable in realtime on slow machines.

For best results (without duplicated frames in the output file) it is necessary to change the output frame rate. For example, to inverse telecine NTSC input:

```
ffmpeg -i input -vf pullup -r 24000/1001 ...
```

## 9.72 removelogo

Suppress a TV station logo, using an image file to determine which pixels comprise the logo. It works by filling in the pixels that comprise the logo with neighboring pixels.

The filter accepts the following options:

`filename, f`

> Set the filter bitmap file, which can be any image format supported by libavformat. The width and height of the image file must match those of the video stream being processed.

Pixels in the provided bitmap image with a value of zero are not considered part of the logo, non-zero pixels are considered part of the logo. If you use white (255) for the logo and black (0) for the rest, you will be safe. For making the filter bitmap, it is recommended to take a screen capture of a black frame with the logo visible, and then using a threshold filter followed by the erode filter once or twice.

If needed, little splotches can be fixed manually. Remember that if logo pixels are not covered, the filter quality will be much reduced. Marking too many pixels as part of the logo does not hurt as much, but it will increase the amount of blurring needed to cover over the image and will destroy more information than necessary, and extra pixels will slow things down on a large logo.

## 9.73 rotate

Rotate video by an arbitrary angle expressed in radians.

The filter accepts the following options:

A description of the optional parameters follows.

`angle, a`

> Set an expression for the angle by which to rotate the input video clockwise, expressed as a number of radians. A negative value will result in a counter-clockwise rotation. By default it is set to "0".
>
> This expression is evaluated for each frame.

`out_w, ow`

> Set the output width expression, default value is "iw". This expression is evaluated just once during configuration.

`out_h, oh`

> Set the output height expression, default value is "ih". This expression is evaluated just once during configuration.

`bilinear`

Enable bilinear interpolation if set to 1, a value of 0 disables it. Default value is 1.

`fillcolor, c`

Set the color used to fill the output area not covered by the rotated image. For the generalsyntax of this option, check the "Color" section in the ffmpeg-utils manual. If the special value "none" is selected then no background is printed (useful for example if the background is never shown).

Default value is "black".

The expressions for the angle and the output size can contain the following constants and functions:

`n`

sequential number of the input frame, starting from 0. It is always NAN before the first frame is filtered.

`t`

time in seconds of the input frame, it is set to 0 when the filter is configured. It is always NAN before the first frame is filtered.

`hsub`
`vsub`

horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

`in_w, iw`
`in_h, ih`

the input video width and height

`out_w, ow`
`out_h, oh`

the output width and height, that is the size of the padded area as specified by the *width* and *height* expressions

`rotw(a)`
`roth(a)`

the minimal width/height required for completely containing the input video rotated by *a* radians.

These are only available when computing the `out_w` and `out_h` expressions.

### 9.73.1 Examples

- Rotate the input by PI/6 radians clockwise:

  ```
  rotate=PI/6
  ```

- Rotate the input by PI/6 radians counter-clockwise:

  ```
  rotate=-PI/6
  ```

- Rotate the input by 45 degrees clockwise:

  ```
  rotate=45*PI/180
  ```

- Apply a constant rotation with period T, starting from an angle of PI/3:

  ```
  rotate=PI/3+2*PI*t/T
  ```

- Make the input video rotation oscillating with a period of T seconds and an amplitude of A radians:

  ```
  rotate=A*sin(2*PI/T*t)
  ```

- Rotate the video, output size is chosen so that the whole rotating input video is always completely contained in the output:

  ```
  rotate='2*PI*t:ow=hypot(iw,ih):oh=ow'
  ```

- Rotate the video, reduce the output size so that no background is ever shown:

  ```
  rotate=2*PI*t:ow='min(iw,ih)/sqrt(2)':oh=ow:c=none
  ```

### 9.73.2 Commands

The filter supports the following commands:

`a, angle`

Set the angle expression. The command accepts the same syntax of the corresponding option.

If the specified expression is not valid, it is kept at its current value.

## 9.74 sab

Apply Shape Adaptive Blur.

The filter accepts the following options:

`luma_radius, lr`

Set luma blur filter strength, must be a value in range 0.1-4.0, default value is 1.0. A greater value will result in a more blurred image, and in slower processing.

`luma_pre_filter_radius, lpfr`

Set luma pre-filter radius, must be a value in the 0.1-2.0 range, default value is 1.0.

`luma_strength, ls`

Set luma maximum difference between pixels to still be considered, must be a value in the 0.1-100.0 range, default value is 1.0.

`chroma_radius, cr`

Set chroma blur filter strength, must be a value in range 0.1-4.0. A greater value will result in a more blurred image, and in slower processing.

`chroma_pre_filter_radius, cpfr`

Set chroma pre-filter radius, must be a value in the 0.1-2.0 range.

`chroma_strength, cs`

Set chroma maximum difference between pixels to still be considered, must be a value in the 0.1-100.0 range.

Each chroma option value, if not explicitly specified, is set to the corresponding luma option value.

## 9.75 scale

Scale (resize) the input video, using the libswscale library.

The scale filter forces the output display aspect ratio to be the same of the input, by changing the output sample aspect ratio.

If the input image format is different from the format requested by the next filter, the scale filter will convert the input to the requested format.

### 9.75.1 Options

The filter accepts the following options, or any of the options supported by the libswscale scaler.

See (ffmpeg-scaler)the ffmpeg-scaler manual for the complete list of scaler options.

`width, w`
`height, h`

Set the output video dimension expression. Default value is the input dimension.

If the value is 0, the input width is used for the output.

If one of the values is -1, the scale filter will use a value that maintains the aspect ratio of the input image, calculated from the other specified dimension. If both of them are -1, the input size is used

If one of the values is -n with n > 1, the scale filter will also use a value that maintains the aspect ratio of the input image, calculated from the other specified dimension. After that it will, however, make sure that the calculated dimension is divisible by n and adjust the value if necessary.

See below for the list of accepted constants for use in the dimension expression.

`interl`

Set the interlacing mode. It accepts the following values:

‘`1`’

Force interlaced aware scaling.

‘`0`’

Do not apply interlaced scaling.

‘`-1`’

Select interlaced aware scaling depending on whether the source frames are flagged as interlaced or not.

Default value is ‘`0`’.

`flags`

Set libswscale scaling flags. See (ffmpeg-scaler)the ffmpeg-scaler manual for the complete list of values. If not explicitly specified the filter applies the default flags.

`size, s`

Set the video size. For the syntax of this option, check the "Video size" section in the ffmpeg-utils manual.

`in_color_matrix`
`out_color_matrix`

Set in/output YCbCr color space type.

This allows the autodetected value to be overridden as well as allows forcing a specific value used for the output and encoder.

If not specified, the color space type depends on the pixel format.

Possible values:

'`auto`'

Choose automatically.

'`bt709`'

Format conforming to International Telecommunication Union (ITU) Recommendation BT.709.

'`fcc`'

Set color space conforming to the United States Federal Communications Commission (FCC) Code of Federal Regulations (CFR) Title 47 (2003) 73.682 (a).

'`bt601`'

Set color space conforming to:

- ITU Radiocommunication Sector (ITU-R) Recommendation BT.601
- ITU-R Rec. BT.470-6 (1998) Systems B, B1, and G
- Society of Motion Picture and Television Engineers (SMPTE) ST 170:2004

'`smpte240m`'

Set color space conforming to SMPTE ST 240:1999.

`in_range`
`out_range`

Set in/output YCbCr sample range.

This allows the autodetected value to be overridden as well as allows forcing a specific value used for the output and encoder. If not specified, the range depends on the pixel format. Possible values:

'`auto`'

Choose automatically.

'`jpeg/full/pc`'

Set full range (0-255 in case of 8-bit luma).

'`mpeg/tv`'

Set "MPEG" range (16-235 in case of 8-bit luma).

`force_original_aspect_ratio`

Enable decreasing or increasing output video width or height if necessary to keep the original aspect ratio. Possible values:

‘`disable`’

Scale the video as specified and disable this feature.

‘`decrease`’

The output video dimensions will automatically be decreased if needed.

‘`increase`’

The output video dimensions will automatically be increased if needed.

One useful instance of this option is that when you know a specific device’s maximum allowed resolution, you can use this to limit the output video to that, while retaining the aspect ratio. For example, device A allows 1280x720 playback, and your video is 1920x800. Using this option (set it to decrease) and specifying 1280x720 to the command line makes the output 1280x533.

Please note that this is a different thing than specifying `-1` for `w` or `h`, you still need to specify the output resolution for this option to work.

The values of the `w` and `h` options are expressions containing the following constants:

*in_w*
*in_h*

The input width and height

*iw*
*ih*

These are the same as *in_w* and *in_h*.

*out_w*
*out_h*

The output (scaled) width and height

*ow*
*oh*

These are the same as *out_w* and *out_h*

*a*

The same as *iw* / *ih*

*sar*

input sample aspect ratio

*dar*

The input display aspect ratio. Calculated from `(iw / ih) * sar`.

*hsub*
*vsub*

horizontal and vertical input chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

*ohsub*
*ovsub*

horizontal and vertical output chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

### 9.75.2 Examples

- Scale the input video to a size of 200x100

  ```
  scale=w=200:h=100
  ```

  This is equivalent to:

  ```
  scale=200:100
  ```

  or:

  ```
  scale=200x100
  ```

- Specify a size abbreviation for the output size:

  ```
  scale=qcif
  ```

  which can also be written as:

  ```
  scale=size=qcif
  ```

- Scale the input to 2x:

  ```
  scale=w=2*iw:h=2*ih
  ```

- The above is the same as:

```
scale=2*in_w:2*in_h
```

- Scale the input to 2x with forced interlaced scaling:

```
scale=2*iw:2*ih:interl=1
```

- Scale the input to half size:

```
scale=w=iw/2:h=ih/2
```

- Increase the width, and set the height to the same size:

```
scale=3/2*iw:ow
```

- Seek Greek harmony:

```
scale=iw:1/PHI*iw
scale=ih*PHI:ih
```

- Increase the height, and set the width to 3/2 of the height:

```
scale=w=3/2*oh:h=3/5*ih
```

- Increase the size, making the size a multiple of the chroma subsample values:

```
scale="trunc(3/2*iw/hsub)*hsub:trunc(3/2*ih/vsub)*vsub"
```

- Increase the width to a maximum of 500 pixels, keeping the same aspect ratio as the input:

```
scale=w='min(500\, iw*3/2):h=-1'
```

## 9.76 separatefields

The `separatefields` takes a frame-based video input and splits each frame into its components fields, producing a new half height clip with twice the frame rate and twice the frame count.

This filter use field-dominance information in frame to decide which of each pair of fields to place first in the output. If it gets it wrong use setfield filter before `separatefields` filter.

## 9.77 setdar, setsar

The `setdar` filter sets the Display Aspect Ratio for the filter output video.

This is done by changing the specified Sample (aka Pixel) Aspect Ratio, according to the following equation:

*`DAR = HORIZONTAL_RESOLUTION / VERTICAL_RESOLUTION * SAR`*

Keep in mind that the `setdar` filter does not modify the pixel dimensions of the video frame. Also, the display aspect ratio set by this filter may be changed by later filters in the filterchain, e.g. in case of scaling or if another "setdar" or a "setsar" filter is applied.

The `setsar` filter sets the Sample (aka Pixel) Aspect Ratio for the filter output video.

Note that as a consequence of the application of this filter, the output display aspect ratio will change according to the equation above.

Keep in mind that the sample aspect ratio set by the `setsar` filter may be changed by later filters in the filterchain, e.g. if another "setsar" or a "setdar" filter is applied.

It accepts the following parameters:

`r, ratio, dar (setdar only), sar (setsar only)`

> Set the aspect ratio used by the filter.
>
> The parameter can be a floating point number string, an expression, or a string of the form *num*:*den*, where *num* and *den* are the numerator and denominator of the aspect ratio. If the parameter is not specified, it is assumed the value "0". In case the form "*num*:*den*" is used, the `:` character should be escaped.

`max`

> Set the maximum integer value to use for expressing numerator and denominator when reducing the expressed aspect ratio to a rational. Default value is `100`.

The parameter *sar* is an expression containing the following constants:

`E, PI, PHI`

> These are approximated values for the mathematical constants e (Euler’s number), pi (Greek pi), and phi (the golden ratio).

`w, h`

> The input width and height.

`a`

> These are the same as *w* / *h*.

`sar`

> The input sample aspect ratio.

`dar`

> The input display aspect ratio. It is the same as (*w* / *h*) * *sar*.

`hsub, vsub`

Horizontal and vertical chroma subsample values. For example, for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

### 9.77.1 Examples

- To change the display aspect ratio to 16:9, specify one of the following:

```
setdar=dar=1.77777
setdar=dar=16/9
setdar=dar=1.77777
```

- To change the sample aspect ratio to 10:11, specify:

```
setsar=sar=10/11
```

- To set a display aspect ratio of 16:9, and specify a maximum integer value of 1000 in the aspect ratio reduction, use the command:

```
setdar=ratio=16/9:max=1000
```

## 9.78 setfield

Force field for the output video frame.

The `setfield` filter marks the interlace type field for the output frames. It does not change the input frame, but only sets the corresponding property, which affects how the frame is treated by following filters (e.g. `fieldorder` or `yadif`).

The filter accepts the following options:

`mode`

Available values are:

‘`auto`’

Keep the same field property.

‘`bff`’

Mark the frame as bottom-field-first.

‘`tff`’

Mark the frame as top-field-first.

‘`prog`’

Mark the frame as progressive.

## 9.79 showinfo

Show a line containing various information for each input video frame. The input video is not modified.

The shown line contains a sequence of key/value pairs of the form *key*:*value*.

The following values are shown in the output:

`n`

The (sequential) number of the input frame, starting from 0.

`pts`

The Presentation TimeStamp of the input frame, expressed as a number of time base units. The time base unit depends on the filter input pad.

`pts_time`

The Presentation TimeStamp of the input frame, expressed as a number of seconds.

`pos`

The position of the frame in the input stream, or -1 if this information is unavailable and/or meaningless (for example in case of synthetic video).

`fmt`

The pixel format name.

`sar`

The sample aspect ratio of the input frame, expressed in the form *num*/*den*.

`s`

The size of the input frame. For the syntax of this option, check the "Video size" section in the ffmpeg-utils manual.

`i`

The type of interlaced mode ("P" for "progressive", "T" for top field first, "B" for bottom field first).

`iskey`

This is 1 if the frame is a key frame, 0 otherwise.

`type`

The picture type of the input frame ("I" for an I-frame, "P" for a P-frame, "B" for a B-frame, or "?" for an unknown type). Also refer to the documentation of the `AVPictureType` enum and of the `av_get_picture_type_char` function defined in `libavutil/avutil.h`.

`checksum`

The Adler-32 checksum (printed in hexadecimal) of all the planes of the input frame.

`plane_checksum`

The Adler-32 checksum (printed in hexadecimal) of each plane of the input frame, expressed in the form "[*c0 c1 c2 c3*]".

## 9.80 shuffleplanes

Reorder and/or duplicate video planes.

It accepts the following parameters:

`map0`

The index of the input plane to be used as the first output plane.

`map1`

The index of the input plane to be used as the second output plane.

`map2`

The index of the input plane to be used as the third output plane.

`map3`

The index of the input plane to be used as the fourth output plane.

The first plane has the index 0. The default is to keep the input unchanged.

Swap the second and third planes of the input:

```
ffmpeg -i INPUT -vf shuffleplanes=0:2:1:3 OUTPUT
```

## 9.81 signalstats

Evaluate various visual metrics that assist in determining issues associated with the digitization of analog video media.

By default the filter will log these metadata values:

`YMIN`

Display the minimal Y value contained within the input frame. Expressed in range of [0-255].

`YLOW`

Display the Y value at the 10% percentile within the input frame. Expressed in range of [0-255].

`YAVG`

Display the average Y value within the input frame. Expressed in range of [0-255].

`YHIGH`

Display the Y value at the 90% percentile within the input frame. Expressed in range of [0-255].

`YMAX`

Display the maximum Y value contained within the input frame. Expressed in range of [0-255].

`UMIN`

Display the minimal U value contained within the input frame. Expressed in range of [0-255].

`ULOW`

Display the U value at the 10% percentile within the input frame. Expressed in range of [0-255].

`UAVG`

Display the average U value within the input frame. Expressed in range of [0-255].

`UHIGH`

Display the U value at the 90% percentile within the input frame. Expressed in range of [0-255].

`UMAX`

Display the maximum U value contained within the input frame. Expressed in range of [0-255].

`VMIN`

Display the minimal V value contained within the input frame. Expressed in range of [0-255].

VLOW

Display the V value at the 10% percentile within the input frame. Expressed in range of [0-255].

VAVG

Display the average V value within the input frame. Expressed in range of [0-255].

VHIGH

Display the V value at the 90% percentile within the input frame. Expressed in range of [0-255].

VMAX

Display the maximum V value contained within the input frame. Expressed in range of [0-255].

SATMIN

Display the minimal saturation value contained within the input frame. Expressed in range of [0-~181.02].

SATLOW

Display the saturation value at the 10% percentile within the input frame. Expressed in range of [0-~181.02].

SATAVG

Display the average saturation value within the input frame. Expressed in range of [0-~181.02].

SATHIGH

Display the saturation value at the 90% percentile within the input frame. Expressed in range of [0-~181.02].

SATMAX

Display the maximum saturation value contained within the input frame. Expressed in range of [0-~181.02].

HUEMED

Display the median value for hue within the input frame. Expressed in range of [0-360].

HUEAVG

Display the average value for hue within the input frame. Expressed in range of [0-360].

`YDIF`

Display the average of sample value difference between all values of the Y plane in the current frame and corresponding values of the previous input frame. Expressed in range of [0-255].

`UDIF`

Display the average of sample value difference between all values of the U plane in the current frame and corresponding values of the previous input frame. Expressed in range of [0-255].

`VDIF`

Display the average of sample value difference between all values of the V plane in the current frame and corresponding values of the previous input frame. Expressed in range of [0-255].

The filter accepts the following options:

`stat`
`out`

`stat` specify an additional form of image analysis. `out` output video with the specified type of pixel highlighted.

Both options accept the following values:

'`tout`'

Identify *temporal outliers* pixels. A *temporal outlier* is a pixel unlike the neighboring pixels of the same field. Examples of temporal outliers include the results of video dropouts, head clogs, or tape tracking issues.

'`vrep`'

Identify *vertical line repetition*. Vertical line repetition includes similar rows of pixels within a frame. In born-digital video vertical line repetition is common, but this pattern is uncommon in video digitized from an analog source. When it occurs in video that results from the digitization of an analog source it can indicate concealment from a dropout compensator.

'`brng`'

Identify pixels that fall outside of legal broadcast range.

`color, c`

Set the highlight color for the `out` option. The default color is yellow.

### 9.81.1 Examples

- Output data of various video metrics:

```
ffprobe -f lavfi movie=example.mov,signalstats="stat=tout+vrep+brng" -show_frames
```

- Output specific data about the minimum and maximum values of the Y plane per frame:

```
ffprobe -f lavfi movie=example.mov,signalstats -show_entries frame_tags=lavfi.signalstats.YMAX,lavfi.signalstats.YMIN
```

- Playback video while highlighting pixels that are outside of broadcast range in red.

```
ffplay example.mov -vf signalstats="out=brng:color=red"
```

- Playback video with signalstats metadata drawn over the frame.

```
ffplay example.mov -vf signalstats=stat=brng+vrep+tout,drawtext=fontfile=FreeSerif.ttf:textfile=signalstat_drawtext.txt
```

  The contents of signalstat_drawtext.txt used in the command are:

```
time %{pts:hms}
Y (%{metadata:lavfi.signalstats.YMIN}-%{metadata:lavfi.signalstats.YMAX})
U (%{metadata:lavfi.signalstats.UMIN}-%{metadata:lavfi.signalstats.UMAX})
V (%{metadata:lavfi.signalstats.VMIN}-%{metadata:lavfi.signalstats.VMAX})
saturation maximum: %{metadata:lavfi.signalstats.SATMAX}
```

## 9.82 smartblur

Blur the input video without impacting the outlines.

It accepts the following options:

`luma_radius, lr`

Set the luma radius. The option value must be a float number in the range [0.1,5.0] that specifies the variance of the gaussian filter used to blur the image (slower if larger). Default value is 1.0.

`luma_strength, ls`

Set the luma strength. The option value must be a float number in the range [-1.0,1.0] that configures the blurring. A value included in [0.0,1.0] will blur the image whereas a value included in [-1.0,0.0] will sharpen the image. Default value is 1.0.

`luma_threshold, lt`

Set the luma threshold used as a coefficient to determine whether a pixel should be blurred or not. The option value must be an integer in the range [-30,30]. A value of 0 will filter all the image, a value included in [0,30] will filter flat areas and a value included in [-30,0] will filter edges. Default value is 0.

chroma_radius, cr

Set the chroma radius. The option value must be a float number in the range [0.1,5.0] that specifies the variance of the gaussian filter used to blur the image (slower if larger). Default value is 1.0.

chroma_strength, cs

Set the chroma strength. The option value must be a float number in the range [-1.0,1.0] that configures the blurring. A value included in [0.0,1.0] will blur the image whereas a value included in [-1.0,0.0] will sharpen the image. Default value is 1.0.

chroma_threshold, ct

Set the chroma threshold used as a coefficient to determine whether a pixel should be blurred or not. The option value must be an integer in the range [-30,30]. A value of 0 will filter all the image, a value included in [0,30] will filter flat areas and a value included in [-30,0] will filter edges. Default value is 0.

If a chroma option is not explicitly set, the corresponding luma value is set.

## 9.83 stereo3d

Convert between different stereoscopic image formats.

The filters accept the following options:

in

Set stereoscopic image format of input.

Available values for input image formats are:

'sbsl'

side by side parallel (left eye left, right eye right)

'sbsr'

side by side crosseye (right eye left, left eye right)

'sbs2l'

side by side parallel with half width resolution (left eye left, right eye right)

'sbs2r'

side by side crosseye with half width resolution (right eye left, left eye right) ‘abl’

above-below (left eye above, right eye below)

‘abr’

above-below (right eye above, left eye below)

‘ab2l’

above-below with half height resolution (left eye above, right eye below)

‘ab2r’

above-below with half height resolution (right eye above, left eye below)

‘al’

alternating frames (left eye first, right eye second)

‘ar’

alternating frames (right eye first, left eye second)

Default value is ‘sbsl’.

out

Set stereoscopic image format of output.

Available values for output image formats are all the input formats as well as:

‘arbg’

anaglyph red/blue gray (red filter on left eye, blue filter on right eye)

‘argg’

anaglyph red/green gray (red filter on left eye, green filter on right eye)

‘arcg’

anaglyph red/cyan gray (red filter on left eye, cyan filter on right eye)

‘arch’

anaglyph red/cyan half colored (red filter on left eye, cyan filter on right eye)

‘arcc’

anaglyph red/cyan color (red filter on left eye, cyan filter on right eye)

‘`arcd`’

anaglyph red/cyan color optimized with the least squares projection of dubois (red filter on left eye, cyan filter on right eye)

‘`agmg`’

anaglyph green/magenta gray (green filter on left eye, magenta filter on right eye)

‘`agmh`’

anaglyph green/magenta half colored (green filter on left eye, magenta filter on right eye)

‘`agmc`’

anaglyph green/magenta colored (green filter on left eye, magenta filter on right eye)

‘`agmd`’

anaglyph green/magenta color optimized with the least squares projection of dubois (green filter on left eye, magenta filter on right eye)

‘`aybg`’

anaglyph yellow/blue gray (yellow filter on left eye, blue filter on right eye)

‘`aybh`’

anaglyph yellow/blue half colored (yellow filter on left eye, blue filter on right eye)

‘`aybc`’

anaglyph yellow/blue colored (yellow filter on left eye, blue filter on right eye)

‘`aybd`’

anaglyph yellow/blue color optimized with the least squares projection of dubois (yellow filter on left eye, blue filter on right eye)

‘`irl`’

interleaved rows (left eye has top row, right eye starts on next row)

‘`irr`’

interleaved rows (right eye has top row, left eye starts on next row) 'ml'

mono output (left eye only)

'mr'

mono output (right eye only)

Default value is 'arcd'.

### 9.83.1 Examples

- Convert input video from side by side parallel to anaglyph yellow/blue dubois:

```
stereo3d=sbsl:aybd
```

- Convert input video from above bellow (left eye above, right eye below) to side by side crosseye.

```
stereo3d=abl:sbsr
```

## 9.84 spp

Apply a simple postprocessing filter that compresses and decompresses the image at several (or - in the case of quality level 6 - all) shifts and average the results.

The filter accepts the following options:

quality

Set quality. This option defines the number of levels for averaging. It accepts an integer in the range 0-6. If set to 0, the filter will have no effect. A value of 6 means the higher quality. For each increment of that value the speed drops by a factor of approximately 2. Default value is 3.

qp

Force a constant quantization parameter. If not set, the filter will use the QP from the video stream (if available).

mode

Set thresholding mode. Available modes are:

'hard'

Set hard thresholding (default).

'soft'

Set soft thresholding (better de-ringing effect, but likely blurrier).

`use_bframe_qp`

Enable the use of the QP from the B-Frames if set to `1`. Using this option may cause flicker since the B-Frames have often larger QP. Default is `0` (not enabled).

## 9.85 subtitles

Draw subtitles on top of input video using the libass library.

To enable compilation of this filter you need to configure FFmpeg with `--enable-libass`. This filter also requires a build with libavcodec and libavformat to convert the passed subtitles file to ASS (Advanced Substation Alpha) subtitles format.

The filter accepts the following options:

`filename, f`

Set the filename of the subtitle file to read. It must be specified.

`original_size`

Specify the size of the original video, the video for which the ASS file was composed. For the syntax of this option, check the "Video size" section in the ffmpeg-utils manual. Due to a misdesign in ASS aspect ratio arithmetic, this is necessary to correctly scale the fonts if the aspect ratio has been changed.

`charenc`

Set subtitles input character encoding. `subtitles` filter only. Only useful if not UTF-8.

`stream_index, si`

Set subtitles stream index. `subtitles` filter only.

If the first key is not specified, it is assumed that the first value specifies the `filename`.

For example, to render the file `sub.srt` on top of the input video, use the command:

```
subtitles=sub.srt
```

which is equivalent to:

```
subtitles=filename=sub.srt
```

To render the default subtitles stream from file `video.mkv`, use:

```
subtitles=video.mkv
```

To render the second subtitles stream from that file, use:

```
subtitles=video.mkv:si=1
```

## 9.86 super2xsai

Scale the input by 2x and smooth using the Super2xSaI (Scale and Interpolate) pixel art scaling algorithm.

Useful for enlarging pixel art images without reducing sharpness.

## 9.87 swapuv

Swap U & V plane.

## 9.88 telecine

Apply telecine process to the video.

This filter accepts the following options:

`first_field`

: ‘`top, t`’

  top field first

  ‘`bottom, b`’

  bottom field first The default value is `top`.

`pattern`

: A string of numbers representing the pulldown pattern you wish to apply. The default value is `23`.

```
Some typical patterns:

NTSC output (30i):
27.5p: 32222
24p: 23 (classic)
24p: 2332 (preferred)
20p: 33
18p: 334
16p: 3444

PAL output (25i):
27.5p: 12222
24p: 222222222223 ("Euro pulldown")
16.67p: 33
16p: 33333334
```

## 9.89 thumbnail

Select the most representative frame in a given sequence of consecutive frames.

The filter accepts the following options:

`n`

> Set the frames batch size to analyze; in a set of *n* frames, the filter will pick one of them, and then handle the next batch of *n* frames until the end. Default is `100`.

Since the filter keeps track of the whole frames sequence, a bigger *n* value will result in a higher memory usage, so a high value is not recommended.

### 9.89.1 Examples

- Extract one picture each 50 frames:

  ```
  thumbnail=50
  ```

- Complete example of a thumbnail creation with `ffmpeg`:

  ```
  ffmpeg -i in.avi -vf thumbnail,scale=300:200 -frames:v 1 out.png
  ```

## 9.90 tile

Tile several successive frames together.

The filter accepts the following options:

`layout`

> Set the grid size (i.e. the number of lines and columns). For the syntax of this option, check the "Video size" section in the ffmpeg-utils manual.

`nb_frames`

> Set the maximum number of frames to render in the given area. It must be less than or equal to *w*x*h*. The default value is `0`, meaning all the area will be used.

`margin`

> Set the outer border margin in pixels.

`padding`

> Set the inner border thickness (i.e. the number of pixels between frames). For more advanced padding options (such as having different values for the edges), refer to the pad video filter.

`color`

Specify the color of the unused areaFor the syntax of this option, check the "Color" section in the ffmpeg-utils manual. The default value of *color* is "black".

### 9.90.1 Examples

- Produce 8x8 PNG tiles of all keyframes (`-skip_frame nokey`) in a movie:

  ```
  ffmpeg -skip_frame nokey -i file.avi -vf 'scale=128:72,tile=8x8' -an -vsync 0 keyframes%03d.png
  ```

  The `-vsync 0` is necessary to prevent `ffmpeg` from duplicating each output frame to accommodate the originally detected frame rate.

- Display `5` pictures in an area of `3x2` frames, with `7` pixels between them, and `2` pixels of initial margin, using mixed flat and named options:

  ```
  tile=3x2:nb_frames=5:padding=7:margin=2
  ```

## 9.91 tinterlace

Perform various types of temporal field interlacing.

Frames are counted starting from 1, so the first input frame is considered odd.

The filter accepts the following options:

`mode`

Specify the mode of the interlacing. This option can also be specified as a value alone. See below for a list of values for this option.

Available values are:

'`merge, 0`'

Move odd frames into the upper field, even into the lower field, generating a double height frame at half frame rate.

'`drop_odd, 1`'

Only output even frames, odd frames are dropped, generating a frame with unchanged height at half frame rate.

'`drop_even, 2`'

Only output odd frames, even frames are dropped, generating a frame with unchanged height at half frame rate.

'`pad, 3`'

Expand each frame to full height, but pad alternate lines with black, generating a frame with double height at the same input frame rate.

'`interleave_top, 4`'

Interleave the upper field from odd frames with the lower field from even frames, generating a frame with unchanged height at half frame rate.

'`interleave_bottom, 5`'

Interleave the lower field from odd frames with the upper field from even frames, generating a frame with unchanged height at half frame rate.

'`interlacex2, 6`'

Double frame rate with unchanged height. Frames are inserted each containing the second temporal field from the previous input frame and the first temporal field from the next input frame. This mode relies on the top_field_first flag. Useful for interlaced video displays with no field synchronisation.

Numeric values are deprecated but are accepted for backward compatibility reasons.

Default mode is `merge`.

`flags`

Specify flags influencing the filter process.

Available value for *flags* is:

`low_pass_filter, vlfp`

Enable vertical low-pass filtering in the filter. Vertical low-pass filtering is required when creating an interlaced destination from a progressive source which contains high-frequency vertical detail. Filtering will reduce interlace 'twitter' and Moire patterning.

Vertical low-pass filtering can only be enabled for `mode` *interleave_top* and *interleave_bottom*.

## 9.92 transpose

Transpose rows with columns in the input video and optionally flip it.

It accepts the following parameters:

`dir`

Specify the transposition direction.

Can assume the following values:

'`0, 4, cclock_flip`'

Rotate by 90 degrees counterclockwise and vertically flip (default), that is:

```
L.R     L.l
. . ->  . .
l.r     R.r
```

'`1, 5, clock`'

Rotate by 90 degrees clockwise, that is:

```
L.R     l.L
. . ->  . .
l.r     r.R
```

'`2, 6, cclock`'

Rotate by 90 degrees counterclockwise, that is:

```
L.R     R.r
. . ->  . .
l.r     L.l
```

'`3, 7, clock_flip`'

Rotate by 90 degrees clockwise and vertically flip, that is:

```
L.R     r.R
. . ->  . .
l.r     l.L
```

For values between 4-7, the transposition is only done if the input video geometry is portrait and not landscape. These values are deprecated, the `passthrough` option should be used instead.

Numerical values are deprecated, and should be dropped in favor of symbolic constants.

`passthrough`

Do not apply the transposition if the input geometry matches the one specified by the specified value. It accepts the following values:

'`none`'

Always apply transposition.

'`portrait`'

Preserve portrait geometry (when *height* >= *width*).

‘`landscape`’

Preserve landscape geometry (when *width* >= *height*).

Default value is `none`.

For example to rotate by 90 degrees clockwise and preserve portrait layout:

```
transpose=dir=1:passthrough=portrait
```

The command above can also be specified as:

```
transpose=1:portrait
```

## 9.93 trim

Trim the input so that the output contains one continuous subpart of the input.

It accepts the following parameters:

`start`

Specify the time of the start of the kept section, i.e. the frame with the timestamp *start* will be the first frame in the output.

`end`

Specify the time of the first frame that will be dropped, i.e. the frame immediately preceding the one with the timestamp *end* will be the last frame in the output.

`start_pts`

This is the same as *start*, except this option sets the start timestamp in timebase units instead of seconds.

`end_pts`

This is the same as *end*, except this option sets the end timestamp in timebase units instead of seconds.

`duration`

The maximum duration of the output in seconds.

`start_frame`

The number of the first frame that should be passed to the output.

`end_frame`

The number of the first frame that should be dropped.

`start`, `end`, and `duration` are expressed as time duration specifications; see (ffmpeg-utils)the Time duration section in the ffmpeg-utils(1) manual for the accepted syntax.

Note that the first two sets of the start/end options and the `duration` option look at the frame timestamp, while the _frame variants simply count the frames that pass through the filter. Also note that this filter does not modify the timestamps. If you wish for the output timestamps to start at zero, insert a setpts filter after the trim filter.

If multiple start or end options are set, this filter tries to be greedy and keep all the frames that match at least one of the specified constraints. To keep only the part that matches all the constraints at once, chain multiple trim filters.

The defaults are such that all the input is kept. So it is possible to set e.g. just the end values to keep everything before the specified time.

Examples:

- Drop everything except the second minute of input:

```
ffmpeg -i INPUT -vf trim=60:120
```

- Keep only the first second:

```
ffmpeg -i INPUT -vf trim=duration=1
```

## 9.94 unsharp

Sharpen or blur the input video.

It accepts the following parameters:

`luma_msize_x, lx`

Set the luma matrix horizontal size. It must be an odd integer between 3 and 63. The default value is 5.

`luma_msize_y, ly`

Set the luma matrix vertical size. It must be an odd integer between 3 and 63. The default value is 5.

`luma_amount, la`

Set the luma effect strength. It must be a floating point number, reasonable values lay between -1.5 and 1.5.

Negative values will blur the input video, while positive values will sharpen it, a value of zero will disable the effect.

Default value is 1.0.

`chroma_msize_x, cx`

Set the chroma matrix horizontal size. It must be an odd integer between 3 and 63. The default value is 5.

`chroma_msize_y, cy`

Set the chroma matrix vertical size. It must be an odd integer between 3 and 63. The default value is 5.

`chroma_amount, ca`

Set the chroma effect strength. It must be a floating point number, reasonable values lay between -1.5 and 1.5.

Negative values will blur the input video, while positive values will sharpen it, a value of zero will disable the effect.

Default value is 0.0.

`opencl`

If set to 1, specify using OpenCL capabilities, only available if FFmpeg was configured with `--enable-opencl`. Default value is 0.

All parameters are optional and default to the equivalent of the string '5:5:1.0:5:5:0.0'.

### 9.94.1 Examples

- Apply strong luma sharpen effect:

```
unsharp=luma_msize_x=7:luma_msize_y=7:luma_amount=2.5
```

- Apply a strong blur of both luma and chroma parameters:

```
unsharp=7:7:-2:7:7:-2
```

### 9.95 vidstabdetect

Analyze video stabilization/deshaking. Perform pass 1 of 2, see vidstabtransform for pass 2.

This filter generates a file with relative translation and rotation transform information about subsequent frames, which is then used by the vidstabtransform filter.

To enable compilation of this filter you need to configure FFmpeg with `--enable-libvidstab`.

This filter accepts the following options:

`result`

> Set the path to the file used to write the transforms information. Default value is `transforms.trf`.

`shakiness`

> Set how shaky the video is and how quick the camera is. It accepts an integer in the range 1-10, a value of 1 means little shakiness, a value of 10 means strong shakiness. Default value is 5.

`accuracy`

> Set the accuracy of the detection process. It must be a value in the range 1-15. A value of 1 means low accuracy, a value of 15 means high accuracy. Default value is 15.

`stepsize`

> Set stepsize of the search process. The region around minimum is scanned with 1 pixel resolution. Default value is 6.

`mincontrast`

> Set minimum contrast. Below this value a local measurement field is discarded. Must be a floating point value in the range 0-1. Default value is 0.3.

`tripod`

> Set reference frame number for tripod mode.
>
> If enabled, the motion of the frames is compared to a reference frame in the filtered stream, identified by the specified number. The idea is to compensate all movements in a more-or-less static scene and keep the camera view absolutely still.
>
> If set to 0, it is disabled. The frames are counted starting from 1.

`show`

Show fields and transforms in the resulting frames. It accepts an integer in the range 0-2. Default value is 0, which disables any visualization.

### 9.95.1 Examples

- Use default values:

```
vidstabdetect
```

- Analyze strongly shaky movie and put the results in file `mytransforms.trf`:

```
vidstabdetect=shakiness=10:accuracy=15:result="mytransforms.trf"
```

- Visualize the result of internal transformations in the resulting video:

```
vidstabdetect=show=1
```

- Analyze a video with medium shakiness using `ffmpeg`:

```
ffmpeg -i input -vf vidstabdetect=shakiness=5:show=1 dummy.avi
```

## 9.96 vidstabtransform

Video stabilization/deshaking: pass 2 of 2, see vidstabdetect for pass 1.

Read a file with transform information for each frame and apply/compensate them. Together with the vidstabdetect filter this can be used to deshake videos. See also http://public.hronopik.de/vid.stab. It is important to also use the unsharp filter, see below.

To enable compilation of this filter you need to configure FFmpeg with `--enable-libvidstab`.

### 9.96.1 Options

`input`

Set path to the file used to read the transforms. Default value is `transforms.trf`.

`smoothing`

Set the number of frames (value*2 + 1) used for lowpass filtering the camera movements. Default value is 10.

For example a number of 10 means that 21 frames are used (10 in the past and 10 in the future) to smoothen the motion in the video. A larger value leads to a smoother video, but limits the acceleration of the camera (pan/tilt movements). 0 is a special case where a static camera is simulated.

`optalgo`

Set the camera path optimization algorithm.

Accepted values are:

'`gauss`'

gaussian kernel low-pass filter on camera motion (default)

'`avg`'

averaging on transformations

`maxshift`

Set maximal number of pixels to translate frames. Default value is -1, meaning no limit.

`maxangle`

Set maximal angle in radians (degree*PI/180) to rotate frames. Default value is -1, meaning no limit.

`crop`

Specify how to deal with borders that may be visible due to movement compensation.

Available values are:

'`keep`'

keep image information from previous frame (default)

'`black`'

fill the border black

`invert`

Invert transforms if set to 1. Default value is 0.

`relative`

Consider transforms as relative to previous frame if set to 1, absolute if set to 0. Default value is 0.

`zoom`

Set percentage to zoom. A positive value will result in a zoom-in effect, a negative value in a zoom-out effect. Default value is 0 (no zoom).

`optzoom`

Set optimal zooming to avoid borders.

Accepted values are:

‘`0`’

disabled

‘`1`’

optimal static zoom value is determined (only very strong movements will lead to visible borders) (default)

‘`2`’

optimal adaptive zoom value is determined (no borders will be visible), see `zoomspeed`

Note that the value given at zoom is added to the one calculated here.

`zoomspeed`

Set percent to zoom maximally each frame (enabled when `optzoom` is set to 2). Range is from 0 to 5, default value is 0.25.

`interpol`

Specify type of interpolation.

Available values are:

‘`no`’

no interpolation

‘`linear`’

linear only horizontal

‘`bilinear`’

linear in both directions (default)

‘`bicubic`’

cubic in both directions (slow)

`tripod`

Enable virtual tripod mode if set to 1, which is equivalent to `relative=0:smoothing=0`. Default value is 0.

Use also `tripod` option of vidstabdetect.

`debug`

Increase log verbosity if set to 1. Also the detected global motions are written to the temporary file `global_motions.trf`. Default value is 0.

### 9.96.2 Examples

- Use `ffmpeg` for a typical stabilization with default values:

```
ffmpeg -i inp.mpeg -vf vidstabtransform,unsharp=5:5:0.8:3:3:0.4 inp_stabilized.mpeg
```

  Note the use of the unsharp filter which is always recommended.

- Zoom in a bit more and load transform data from a given file:

```
vidstabtransform=zoom=5:input="mytransforms.trf"
```

- Smoothen the video even more:

```
vidstabtransform=smoothing=30
```

## 9.97 vflip

Flip the input video vertically.

For example, to vertically flip a video with `ffmpeg`:

```
ffmpeg -i in.avi -vf "vflip" out.avi
```

## 9.98 vignette

Make or reverse a natural vignetting effect.

The filter accepts the following options:

`angle, a`

Set lens angle expression as a number of radians.

The value is clipped in the `[0,PI/2]` range.

Default value: `"PI/5"`

`x0`
`y0`

Set center coordinates expressions. Respectively `"w/2"` and `"h/2"` by default.

`mode`

Set forward/backward mode.

Available modes are:

'`forward`'

The larger the distance from the central point, the darker the image becomes.

'`backward`'

The larger the distance from the central point, the brighter the image becomes. This can be used to reverse a vignette effect, though there is no automatic detection to extract the lens `angle` and other settings (yet). It can also be used to create a burning effect.

Default value is '`forward`'.

`eval`

Set evaluation mode for the expressions (`angle`, `x0`, `y0`).

It accepts the following values:

'`init`'

Evaluate expressions only once during the filter initialization.

'`frame`'

Evaluate expressions for each incoming frame. This is way slower than the '`init`' mode since it requires all the scalers to be re-computed, but it allows advanced dynamic expressions.

Default value is '`init`'.

`dither`

Set dithering to reduce the circular banding effects. Default is `1` (enabled).

`aspect`

Set vignette aspect. This setting allows one to adjust the shape of the vignette. Setting this value to the SAR of the input will make a rectangular vignetting following the dimensions of the video.

Default is `1/1`.

### 9.98.1 Expressions

The `alpha`, `x0` and `y0` expressions can contain the following parameters.

`w`
`h`

input width and height

`n`

the number of input frame, starting from 0

`pts`

the PTS (Presentation TimeStamp) time of the filtered video frame, expressed in *TB* units, NAN if undefined

`r`

frame rate of the input video, NAN if the input frame rate is unknown

`t`

the PTS (Presentation TimeStamp) of the filtered video frame, expressed in seconds, NAN if undefined

`tb`

time base of the input video

### 9.98.2 Examples

- Apply simple strong vignetting effect:

```
vignette=PI/4
```

- Make a flickering vignetting:

```
vignette='PI/4+random(1)*PI/50':eval=frame
```

## 9.99 w3fdif

Deinterlace the input video ("w3fdif" stands for "Weston 3 Field Deinterlacing Filter").

Based on the process described by Martin Weston for BBC R&D, and implemented based on the de-interlace algorithm written by Jim Easterbrook for BBC R&D, the Weston 3 field deinterlacing filter uses filter coefficients calculated by BBC R&D.

There are two sets of filter coefficients, so called "simple": and "complex". Which set of filter coefficients is used can be set by passing an optional parameter:

`filter`

Set the interlacing filter coefficients. Accepts one of the following values:

‘`simple`’

Simple filter coefficient set.

‘`complex`’

More-complex filter coefficient set.

Default value is ‘`complex`’.

`deint`

Specify which frames to deinterlace. Accept one of the following values:

‘`all`’

Deinterlace all frames,

‘`interlaced`’

Only deinterlace frames marked as interlaced.

Default value is ‘`all`’.

## 9.100 xbr

Apply the xBR high-quality magnification filter which is designed for pixel art. It follows a set of edge-detection rules, see http://www.libretro.com/forums/viewtopic.php?f=6&t=134.

It accepts the following option:

`n`

Set the scaling dimension: `2` for `2xBR`, `3` for `3xBR` and `4` for `4xBR`. Default is `3`.

## 9.101 yadif

Deinterlace the input video ("yadif" means "yet another deinterlacing filter").

It accepts the following parameters:

`mode`

The interlacing mode to adopt. It accepts one of the following values:

`0, send_frame`

Output one frame for each frame.

`1, send_field`

Output one frame for each field.

`2, send_frame_nospatial`

Like `send_frame`, but it skips the spatial interlacing check.

`3, send_field_nospatial`

Like `send_field`, but it skips the spatial interlacing check.

The default value is `send_frame`.

`parity`

The picture field parity assumed for the input interlaced video. It accepts one of the following values:

`0, tff`

Assume the top field is first.

`1, bff`

Assume the bottom field is first.

`-1, auto`

Enable automatic detection of field parity.

The default value is `auto`. If the interlacing is unknown or the decoder does not export this information, top field first will be assumed.

`deint`

Specify which frames to deinterlace. Accept one of the following values:

`0, all`

Deinterlace all frames.

`1, interlaced`

Only deinterlace frames marked as interlaced.

The default value is `all`.

## 9.102 zoompan

Apply Zoom & Pan effect.

This filter accepts the following options:

`zoom, z`

Set the zoom expression. Default is 1.

`x`
`y`

Set the x and y expression. Default is 0.

`d`

Set the duration expression in number of frames. This sets for how many number of frames effect will last for single input image.

`s`

Set the output image size, default is ’hd720’.

Each expression can contain the following constants:

`in_w, iw`

Input width.

`in_h, ih`

Input height.

`out_w, ow`

Output width.

`out_h, oh`

Output height.

`in`

Input frame count.

`on`

Output frame count.

`x`
`y`

Last calculated ’x’ and ’y’ position from ’x’ and ’y’ expression for current input frame.

`px`
`py`

’x’ and ’y’ of last output frame of previous input frame or 0 when there was not yet such frame (first input frame).

`zoom`

Last calculated zoom from ’z’ expression for current input frame.

`pzoom`

Last calculated zoom of last output frame of previous input frame.

`duration`

Number of output frames for current input frame. Calculated from ’d’ expression for each input frame.

`pduration`

number of output frames created for previous input frame

`a`

Rational number: input width / input height

`sar`

sample aspect ratio

`dar`

display aspect ratio

### 9.102.1 Examples

- Zoom-in up to 1.5 and pan at same time to some spot near center of picture:

```
zoompan=z='min(zoom+0.0015,1.5)':d=700:x='if(gte(zoom,1.5),x,x+1/a)':y='if(gte(zoom,1.5),y,y+1)':s=640x360
```

# 10 Video Sources

Below is a description of the currently available video sources.

## 10.1 buffer

Buffer video frames, and make them available to the filter chain.

This source is mainly intended for a programmatic use, in particular through the interface defined in `libavfilter/vsrc_buffer.h`.

It accepts the following parameters:

`video_size`

Specify the size (width and height) of the buffered video frames. For the syntax of this option, check the "Video size" section in the ffmpeg-utils manual.

`width`

The input video width.

`height`

The input video height.

`pix_fmt`

A string representing the pixel format of the buffered video frames. It may be a number corresponding to a pixel format, or a pixel format name.

`time_base`

Specify the timebase assumed by the timestamps of the buffered frames.

`frame_rate`

Specify the frame rate expected for the video stream.

`pixel_aspect, sar`

The sample (pixel) aspect ratio of the input video.

`sws_param`

Specify the optional parameters to be used for the scale filter which is automatically inserted when an input change is detected in the input size or format.

For example:

```
buffer=width=320:height=240:pix_fmt=yuv410p:time_base=1/24:sar=1
```

will instruct the source to accept video frames with size 320x240 and with format "yuv410p", assuming 1/24 as the timestamps timebase and square pixels (1:1 sample aspect ratio). Since the pixel format with name "yuv410p" corresponds to the number 6 (check the enum AVPixelFormat definition in `libavutil/pixfmt.h`), this example corresponds to:

```
buffer=size=320x240:pixfmt=6:time_base=1/24:pixel_aspect=1/1
```

Alternatively, the options can be specified as a flat string, but this syntax is deprecated:

*width*:*height*:*pix_fmt*:*time_base.num*:*time_base.den*:*pixel_aspect.num*:*pixel_aspect.den*[:*sws_param*]

## 10.2 cellauto

Create a pattern generated by an elementary cellular automaton.

The initial state of the cellular automaton can be defined through the `filename`, and `pattern` options. If such options are not specified an initial state is created randomly.

At each new frame a new row in the video is filled with the result of the cellular automaton next generation. The behavior when the whole frame is filled is defined by the `scroll` option.

This source accepts the following options:

`filename, f`

Read the initial cellular automaton state, i.e. the starting row, from the specified file. In the file, each non-whitespace character is considered an alive cell, a newline will terminate the row, and further characters in the file will be ignored.

`pattern, p`

Read the initial cellular automaton state, i.e. the starting row, from the specified string.

Each non-whitespace character in the string is considered an alive cell, a newline will terminate the row, and further characters in the string will be ignored.

`rate, r`

Set the video rate, that is the number of frames generated per second. Default is 25.

`random_fill_ratio, ratio`

Set the random fill ratio for the initial cellular automaton row. It is a floating point number value ranging from 0 to 1, defaults to 1/PHI.

This option is ignored when a file or a pattern is specified.

`random_seed, seed`

Set the seed for filling randomly the initial row, must be an integer included between 0 and UINT32_MAX. If not specified, or if explicitly set to -1, the filter will try to use a good random seed on a best effort basis.

`rule`

Set the cellular automaton rule, it is a number ranging from 0 to 255. Default value is 110.

`size, s`

Set the size of the output video. For the syntax of this option, check the "Video size" section in the ffmpeg-utils manual.

If `filename` or `pattern` is specified, the size is set by default to the width of the specified initial state row, and the height is set to *width* * PHI.

If `size` is set, it must contain the width of the specified pattern string, and the specified pattern will be centered in the larger row.

If a filename or a pattern string is not specified, the size value defaults to "320x518" (used for a randomly generated initial state).

`scroll`

If set to 1, scroll the output upward when all the rows in the output have been already filled. If set to 0, the new generated row will be written over the top row just after the bottom row is filled. Defaults to 1.

`start_full, full`

If set to 1, completely fill the output with generated rows before outputting the first frame. This is the default behavior, for disabling set the value to 0.

`stitch`

If set to 1, stitch the left and right row edges together. This is the default behavior, for disabling set the value to 0.

### 10.2.1 Examples

- Read the initial state from `pattern`, and specify an output of size 200x400.

  ```
  cellauto=f=pattern:s=200x400
  ```

- Generate a random initial row with a width of 200 cells, with a fill ratio of 2/3:

  ```
  cellauto=ratio=2/3:s=200x200
  ```

- Create a pattern generated by rule 18 starting by a single alive cell centered on an initial row with width 100:

  ```
  cellauto=p=@:s=100x400:full=0:rule=18
  ```

- Specify a more elaborated initial pattern:

  ```
  cellauto=p='@@ @ @@':s=100x400:full=0:rule=18
  ```

## 10.3 mandelbrot

Generate a Mandelbrot set fractal, and progressively zoom towards the point specified with *start_x* and *start_y*.

This source accepts the following options:

`end_pts`

Set the terminal pts value. Default value is 400.

`end_scale`

Set the terminal scale value. Must be a floating point value. Default value is 0.3.

`inner`

Set the inner coloring mode, that is the algorithm used to draw the Mandelbrot fractal internal region.

It shall assume one of the following values:

`black`

Set black mode.

`convergence`

Show time until convergence.

`mincol`

Set color based on point closest to the origin of the iterations.

`period`

Set period mode.

Default value is *mincol*.

`bailout`

Set the bailout value. Default value is 10.0.

`maxiter`

Set the maximum of iterations performed by the rendering algorithm. Default value is 7189.

`outer`

Set outer coloring mode. It shall assume one of following values:

`iteration_count`

Set iteration cound mode.

`normalized_iteration_count`

set normalized iteration count mode.

Default value is *normalized_iteration_count*.

`rate, r`

Set frame rate, expressed as number of frames per second. Default value is "25".

`size, s`

Set frame size. For the syntax of this option, check the "Video size" section in the ffmpeg-utils manual. Default value is "640x480".

`start_scale`

Set the initial scale value. Default value is 3.0.

`start_x`

Set the initial x position. Must be a floating point value between -100 and 100. Default value is -0.743643887037158704752191506114774.

`start_y`

Set the initial y position. Must be a floating point value between -100 and 100. Default value is -0.131825904205311970493132056385139.

## 10.4 mptestsrc

Generate various test patterns, as generated by the MPlayer test filter.

The size of the generated video is fixed, and is 256x256. This source is useful in particular for testing encoding features.

This source accepts the following options:

`rate, r`

Specify the frame rate of the sourced video, as the number of frames generated per second. It has to be a string in the format *frame_rate_num*/*frame_rate_den*, an integer number, a floating point number or a valid video frame rate abbreviation. The default value is "25".

`duration, d`

Set the duration of the sourced video. See (ffmpeg-utils)the Time duration section in the ffmpeg-utils(1) manual for the accepted syntax.

If not specified, or the expressed duration is negative, the video is supposed to be generated forever.

`test, t`

Set the number or the name of the test to perform. Supported tests are:

```
dc_luma
dc_chroma
freq_luma
freq_chroma
amp_luma
amp_chroma
cbp
mv
ring1
ring2
all
```

Default value is "all", which will cycle through the list of all tests.

Some examples:

```
mptestsrc=t=dc_luma
```

will generate a "dc_luma" test pattern.

## 10.5 frei0r_src

Provide a frei0r source.

To enable compilation of this filter you need to install the frei0r header and configure FFmpeg with `--enable-frei0r`.

This source accepts the following parameters:

`size`

> The size of the video to generate. For the syntax of this option, check the "Video size" section in the ffmpeg-utils manual.

`framerate`

> The framerate of the generated video. It may be a string of the form *num*/*den* or a frame rate abbreviation.

`filter_name`

> The name to the frei0r source to load. For more information regarding frei0r and how to set the parameters, read the frei0r section in the video filters documentation.

`filter_params`

> A ’|’-separated list of parameters to pass to the frei0r source.

For example, to generate a frei0r partik0l source with size 200x200 and frame rate 10 which is overlayed on the overlay filter main input:

```
frei0r_src=size=200x200:framerate=10:filter_name=partik0l:filter_params=1234 [overlay]; [in][overlay] overlay
```

## 10.6 life

Generate a life pattern.

This source is based on a generalization of John Conway’s life game.

The sourced input represents a life grid, each pixel represents a cell which can be in one of two possible states, alive or dead. Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent.

At each interaction the grid evolves according to the adopted rule, which specifies the number of neighbor alive cells which will make a cell stay alive or born. The `rule` option allows one to specify the rule to adopt.

This source accepts the following options:

`filename, f`

> Set the file from which to read the initial grid state. In the file, each non-whitespace character is considered an alive cell, and newline is used to delimit the end of each row.
>
> If this option is not specified, the initial grid is generated randomly.

`rate, r`

> Set the video rate, that is the number of frames generated per second. Default is 25.

`random_fill_ratio, ratio`

> Set the random fill ratio for the initial random grid. It is a floating point number value ranging from 0 to 1, defaults to 1/PHI. It is ignored when a file is specified.

`random_seed, seed`

> Set the seed for filling the initial random grid, must be an integer included between 0 and UINT32_MAX. If not specified, or if explicitly set to -1, the filter will try to use a good random seed on a best effort basis.

`rule`

> Set the life rule.
>
> A rule can be specified with a code of the kind "S*NS*/B*NB*", where *NS* and *NB* are sequences of numbers in the range 0-8, *NS* specifies the number of alive neighbor cells which make a live cell stay alive, and *NB* the number of alive neighbor cells which make a dead cell to become alive (i.e. to "born"). "s" and "b" can be used in place of "S" and "B", respectively.
>
> Alternatively a rule can be specified by an 18-bits integer. The 9 high order bits are used to encode the next cell state if it is alive for each number of neighbor alive cells, the low order bits specify the rule for "borning" new cells. Higher order bits encode for an higher number of neighbor cells. For example the number 6153 = `(12<<9)+9` specifies a stay alive rule of 12 and a born rule of 9, which corresponds to "S23/B03".
>
> Default value is "S23/B3", which is the original Conway’s game of life rule, and will keep a cell alive if it has 2 or 3 neighbor alive cells, and will born a new cell if there are three alive cells around a dead cell.

`size, s`

Set the size of the output video. For the syntax of this option, check the "Video size" section in the ffmpeg-utils manual.

If `filename` is specified, the size is set by default to the same size of the input file. If `size` is set, it must contain the size specified in the input file, and the initial grid defined in that file is centered in the larger resulting area.

If a filename is not specified, the size value defaults to "320x240" (used for a randomly generated initial grid).

`stitch`

If set to 1, stitch the left and right grid edges together, and the top and bottom edges also. Defaults to 1.

`mold`

Set cell mold speed. If set, a dead cell will go from `death_color` to `mold_color` with a step of `mold`. `mold` can have a value from 0 to 255.

`life_color`

Set the color of living (or new born) cells.

`death_color`

Set the color of dead cells. If `mold` is set, this is the first color used to represent a dead cell.

`mold_color`

Set mold color, for definitely dead and moldy cells.

For the syntax of these 3 color options, check the "Color" section in the ffmpeg-utils manual.

### 10.6.1 Examples

- Read a grid from `pattern`, and center it on a grid of size 300x300 pixels:

```
life=f=pattern:s=300x300
```

- Generate a random grid of size 200x200, with a fill ratio of 2/3:

```
life=ratio=2/3:s=200x200
```

- Specify a custom rule for evolving a randomly generated grid:

```
life=rule=S14/B34
```

- Full example with slow death effect (mold) using `ffplay`:

```
ffplay -f lavfi life=s=300x200:mold=10:r=60:ratio=0.1:death_color=#C83232:life_color=#00ff00,scale=1200:800:flags=16
```

## 10.7 color, haldclutsrc, nullsrc, rgbtestsrc, smptebars, smptehdbars, testsrc

The `color` source provides an uniformly colored input.

The `haldclutsrc` source provides an identity Hald CLUT. See also haldclut filter.

The `nullsrc` source returns unprocessed video frames. It is mainly useful to be employed in analysis / debugging tools, or as the source for filters which ignore the input data.

The `rgbtestsrc` source generates an RGB test pattern useful for detecting RGB vs BGR issues. You should see a red, green and blue stripe from top to bottom.

The `smptebars` source generates a color bars pattern, based on the SMPTE Engineering Guideline EG 1-1990.

The `smptehdbars` source generates a color bars pattern, based on the SMPTE RP 219-2002.

The `testsrc` source generates a test video pattern, showing a color pattern, a scrolling gradient and a timestamp. This is mainly intended for testing purposes.

The sources accept the following parameters:

`color, c`

> Specify the color of the source, only available in the `color` source. For the syntax of this option, check the "Color" section in the ffmpeg-utils manual.

`level`

> Specify the level of the Hald CLUT, only available in the `haldclutsrc` source. A level of `N` generates a picture of `N*N*N` by `N*N*N` pixels to be used as identity matrix for 3D lookup tables. Each component is coded on a `1/(N*N)` scale.

`size, s`

> Specify the size of the sourced video. For the syntax of this option, check the "Video size" section in the ffmpeg-utils manual. The default value is "320x240".
>
> This option is not available with the `haldclutsrc` filter.

`rate, r`

Specify the frame rate of the sourced video, as the number of frames generated per second. It has to be a string in the format *frame_rate_num*/*frame_rate_den*, an integer number, a floating point number or a valid video frame rate abbreviation. The default value is "25".

`sar`

Set the sample aspect ratio of the sourced video.

`duration, d`

Set the duration of the sourced video. See (ffmpeg-utils)the Time duration section in the ffmpeg-utils(1) manual for the accepted syntax.

If not specified, or the expressed duration is negative, the video is supposed to be generated forever.

`decimals, n`

Set the number of decimals to show in the timestamp, only available in the `testsrc` source.

The displayed timestamp value will correspond to the original timestamp value multiplied by the power of 10 of the specified value. Default value is 0.

For example the following:

```
testsrc=duration=5.3:size=qcif:rate=10
```

will generate a video with a duration of 5.3 seconds, with size 176x144 and a frame rate of 10 frames per second.

The following graph description will generate a red source with an opacity of 0.2, with size "qcif" and a frame rate of 10 frames per second.

```
color=c=red@0.2:s=qcif:r=10
```

If the input content is to be ignored, `nullsrc` can be used. The following command generates noise in the luminance plane by employing the `geq` filter:

```
nullsrc=s=256x256, geq=random(1)*255:128:128
```

### 10.7.1 Commands

The `color` source supports the following commands:

`c, color`

Set the color of the created image. Accepts the same syntax of the corresponding `color` option.

# 11 Video Sinks

Below is a description of the currently available video sinks.

## 11.1 buffersink

Buffer video frames, and make them available to the end of the filter graph.

This sink is mainly intended for programmatic use, in particular through the interface defined in `libavfilter/buffersink.h` or the options system.

It accepts a pointer to an AVBufferSinkContext structure, which defines the incoming buffers' formats, to be passed as the opaque parameter to `avfilter_init_filter` for initialization.

## 11.2 nullsink

Null video sink: do absolutely nothing with the input video. It is mainly useful as a template and for use in analysis / debugging tools.

# 12 Multimedia Filters

Below is a description of the currently available multimedia filters.

## 12.1 avectorscope

Convert input audio to a video output, representing the audio vector scope.

The filter is used to measure the difference between channels of stereo audio stream. A monoaural signal, consisting of identical left and right signal, results in straight vertical line. Any stereo separation is visible as a deviation from this line, creating a Lissajous figure. If the straight (or deviation from it) but horizontal line appears this indicates that the left and right channels are out of phase.

The filter accepts the following options:

`mode, m`

Set the vectorscope mode.

Available values are:

'`lissajous`'

Lissajous rotated by 45 degrees.

'`lissajous_xy`'

Same as above but not rotated.

Default value is ‘`lissajous`’.

`size, s`

Set the video size for the output. For the syntax of this option, check the "Video size" section in the ffmpeg-utils manual. Default value is `400x400`.

`rate, r`

Set the output frame rate. Default value is `25`.

`rc`
`gc`
`bc`

Specify the red, green and blue contrast. Default values are `40`, `160` and `80`. Allowed range is `[0, 255]`.

`rf`
`gf`
`bf`

Specify the red, green and blue fade. Default values are `15`, `10` and `5`. Allowed range is `[0, 255]`.

`zoom`

Set the zoom factor. Default value is `1`. Allowed range is `[1, 10]`.

### 12.1.1 Examples

- Complete example using `ffplay`:

```
ffplay -f lavfi 'amovie=input.mp3, asplit [a][out1];
             [a] avectorscope=zoom=1.3:rc=2:gc=200:bc=10:rf=1:gf=8:bf=7 [out0]'
```

## 12.2 concat

Concatenate audio and video streams, joining them together one after the other.

The filter works on segments of synchronized video and audio streams. All segments must have the same number of streams of each type, and that will also be the number of streams at output.

The filter accepts the following options:

`n`

Set the number of segments. Default is 2.

`v`

Set the number of output video streams, that is also the number of video streams in each segment. Default is 1.

`a`

Set the number of output audio streams, that is also the number of audio streams in each segment. Default is 0.

`unsafe`

Activate unsafe mode: do not fail if segments have a different format.

The filter has *v*+*a* outputs: first *v* video outputs, then *a* audio outputs.

There are *n*x(*v*+*a*) inputs: first the inputs for the first segment, in the same order as the outputs, then the inputs for the second segment, etc.

Related streams do not always have exactly the same duration, for various reasons including codec frame size or sloppy authoring. For that reason, related synchronized streams (e.g. a video and its audio track) should be concatenated at once. The concat filter will use the duration of the longest stream in each segment (except the last one), and if necessary pad shorter audio streams with silence.

For this filter to work correctly, all segments must start at timestamp 0.

All corresponding streams must have the same parameters in all segments; the filtering system will automatically select a common pixel format for video streams, and a common sample format, sample rate and channel layout for audio streams, but other settings, such as resolution, must be converted explicitly by the user.

Different frame rates are acceptable but will result in variable frame rate at output; be sure to configure the output file to handle it.

### 12.2.1 Examples

- Concatenate an opening, an episode and an ending, all in bilingual version (video in stream 0, audio in streams 1 and 2):

```
ffmpeg -i opening.mkv -i episode.mkv -i ending.mkv -filter_complex \
  '[0:0] [0:1] [0:2] [1:0] [1:1] [1:2] [2:0] [2:1] [2:2]
   concat=n=3:v=1:a=2 [v] [a1] [a2]' \
  -map '[v]' -map '[a1]' -map '[a2]' output.mkv
```

- Concatenate two parts, handling audio and video separately, using the (a)movie sources, and adjusting the resolution:

```
movie=part1.mp4, scale=512:288 [v1] ; amovie=part1.mp4 [a1] ;
movie=part2.mp4, scale=512:288 [v2] ; amovie=part2.mp4 [a2] ;
[v1] [v2] concat [outv] ; [a1] [a2] concat=v=0:a=1 [outa]
```

Note that a desync will happen at the stitch if the audio and video streams do not have exactly the same duration in the first file.

## 12.3 ebur128

EBU R128 scanner filter. This filter takes an audio stream as input and outputs it unchanged. By default, it logs a message at a frequency of 10Hz with the Momentary loudness (identified by `M`), Short-term loudness (`S`), Integrated loudness (`I`) and Loudness Range (`LRA`).

The filter also has a video output (see the *video* option) with a real time graph to observe the loudness evolution. The graphic contains the logged message mentioned above, so it is not printed anymore when this option is set, unless the verbose logging is set. The main graphing area contains the short-term loudness (3 seconds of analysis), and the gauge on the right is for the momentary loudness (400 milliseconds).

More information about the Loudness Recommendation EBU R128 on http://tech.ebu.ch/loudness.

The filter accepts the following options:

`video`

Activate the video output. The audio stream is passed unchanged whether this option is set or no. The video stream will be the first output stream if activated. Default is `0`.

`size`

Set the video size. This option is for video only. For the syntax of this option, check the "Video size" section in the ffmpeg-utils manual. Default and minimum resolution is `640x480`.

`meter`

Set the EBU scale meter. Default is `9`. Common values are `9` and `18`, respectively for EBU scale meter +9 and EBU scale meter +18. Any other integer value between this range is allowed.

`metadata`

Set metadata injection. If set to `1`, the audio input will be segmented into 100ms output frames, each of them containing various loudness information in metadata. All the metadata keys are prefixed with `lavfi.r128.`.

Default is `0`.

`framelog`

Force the frame logging level.

Available values are:

'`info`'

information logging level

'`verbose`'

verbose logging level

By default, the logging level is set to *info*. If the `video` or the `metadata` options are set, it switches to *verbose*.

`peak`

Set peak mode(s).

Available modes can be cumulated (the option is a `flag` type). Possible values are:

'`none`'

Disable any peak mode (default).

'`sample`'

Enable sample-peak mode.

Simple peak mode looking for the higher sample value. It logs a message for sample-peak (identified by `SPK`).

'`true`'

Enable true-peak mode.

If enabled, the peak lookup is done on an over-sampled version of the input stream for better peak accuracy. It logs a message for true-peak. (identified by `TPK`) and true-peak per frame (identified by `FTPK`). This mode requires a build with `libswresample`.

### 12.3.1 Examples

- Real-time graph using `ffplay`, with a EBU scale meter +18:

  ```
  ffplay -f lavfi -i "amovie=input.mp3,ebur128=video=1:meter=18 [out0][out1]"
  ```

- Run an analysis with `ffmpeg`:

```
ffmpeg -nostats -i input.mp3 -filter_complex ebur128 -f null -
```

## 12.4 interleave, ainterleave

Temporally interleave frames from several inputs.

`interleave` works with video inputs, `ainterleave` with audio.

These filters read frames from several inputs and send the oldest queued frame to the output.

Input streams must have a well defined, monotonically increasing frame timestamp values.

In order to submit one frame to output, these filters need to enqueue at least one frame for each input, so they cannot work in case one input is not yet terminated and will not receive incoming frames.

For example consider the case when one input is a `select` filter which always drop input frames. The `interleave` filter will keep reading from that input, but it will never be able to send new frames to output until the input will send an end-of-stream signal.

Also, depending on inputs synchronization, the filters will drop frames in case one input receives more frames than the other ones, and the queue is already filled.

These filters accept the following options:

`nb_inputs, n`

Set the number of different inputs, it is 2 by default.

### 12.4.1 Examples

- Interleave frames belonging to different streams using `ffmpeg`:

```
ffmpeg -i bambi.avi -i pr0n.mkv -filter_complex "[0:v][1:v] interleave" out.avi
```

- Add flickering blur effect:

```
select='if(gt(random(0), 0.2), 1, 2)':n=2 [tmp], boxblur=2:2, [tmp] interleave
```

## 12.5 perms, aperms

Set read/write permissions for the output frames.

These filters are mainly aimed at developers to test direct path in the following filter in the filtergraph.

The filters accept the following options:

`mode`

Select the permissions mode.

It accepts the following values:

'`none`'

Do nothing. This is the default.

'`ro`'

Set all the output frames read-only.

'`rw`'

Set all the output frames directly writable.

'`toggle`'

Make the frame read-only if writable, and writable if read-only.

'`random`'

Set each output frame read-only or writable randomly.

`seed`

Set the seed for the *random* mode, must be an integer included between `0` and `UINT32_MAX`. If not specified, or if explicitly set to `-1`, the filter will try to use a good random seed on a best effort basis.

Note: in case of auto-inserted filter between the permission filter and the following one, the permission might not be received as expected in that following filter. Inserting a format or aformat filter before the perms/aperms filter can avoid this problem.

## 12.6 select, aselect

Select frames to pass in output.

This filter accepts the following options:

`expr, e`

Set expression, which is evaluated for each input frame.

If the expression is evaluated to zero, the frame is discarded.

If the evaluation result is negative or NaN, the frame is sent to the first output; otherwise it is sent to the output with index `ceil(val)-1`, assuming that the input index starts from 0.

For example a value of `1.2` corresponds to the output with index `ceil(1.2)-1 = 2-1 = 1`, that is the second output.

`outputs, n`

Set the number of outputs. The output to which to send the selected frame is based on the result of the evaluation. Default value is 1.

The expression can contain the following constants:

`n`

The (sequential) number of the filtered frame, starting from 0.

`selected_n`

The (sequential) number of the selected frame, starting from 0.

`prev_selected_n`

The sequential number of the last selected frame. It's NAN if undefined.

`TB`

The timebase of the input timestamps.

`pts`

The PTS (Presentation TimeStamp) of the filtered video frame, expressed in *TB* units. It's NAN if undefined.

`t`

The PTS of the filtered video frame, expressed in seconds. It's NAN if undefined.

`prev_pts`

The PTS of the previously filtered video frame. It's NAN if undefined.

`prev_selected_pts`

The PTS of the last previously filtered video frame. It's NAN if undefined.

`prev_selected_t`

The PTS of the last previously selected video frame. It's NAN if undefined.

`start_pts`

The PTS of the first video frame in the video. It’s NAN if undefined.

`start_t`

The time of the first video frame in the video. It’s NAN if undefined.

`pict_type` *(video only)*

The type of the filtered frame. It can assume one of the following values:

`I`
`P`
`B`
`S`
`SI`
`SP`
`BI`

`interlace_type` *(video only)*

The frame interlace type. It can assume one of the following values:

`PROGRESSIVE`

The frame is progressive (not interlaced).

`TOPFIRST`

The frame is top-field-first.

`BOTTOMFIRST`

The frame is bottom-field-first.

`consumed_sample_n` *(audio only)*

the number of selected samples before the current frame

`samples_n` *(audio only)*

the number of samples in the current frame

`sample_rate` *(audio only)*

the input sample rate

`key`

This is 1 if the filtered frame is a key-frame, 0 otherwise.

`pos`

the position in the file of the filtered frame, -1 if the information is not available (e.g. for synthetic video)

`scene` *(video only)*

value between 0 and 1 to indicate a new scene; a low value reflects a low probability for the current frame to introduce a new scene, while a higher value means the current frame is more likely to be one (see the example below)

The default value of the select expression is "1".

### 12.6.1 Examples

- Select all frames in input:

  ```
  select
  ```

  The example above is the same as:

  ```
  select=1
  ```

- Skip all frames:

  ```
  select=0
  ```

- Select only I-frames:

  ```
  select='eq(pict_type\,I)'
  ```

- Select one frame every 100:

  ```
  select='not(mod(n\,100))'
  ```

- Select only frames contained in the 10-20 time interval:

  ```
  select=between(t\,10\,20)
  ```

- Select only I frames contained in the 10-20 time interval:

  ```
  select=between(t\,10\,20)*eq(pict_type\,I)
  ```

- Select frames with a minimum distance of 10 seconds:

  ```
  select='isnan(prev_selected_t)+gte(t-prev_selected_t\,10)'
  ```

- Use aselect to select only audio frames with samples number > 100:

  ```
  aselect='gt(samples_n\,100)'
  ```

- Create a mosaic of the first scenes:

```
ffmpeg -i video.avi -vf select='gt(scene\,0.4)',scale=160:120,tile -frames:v 1 preview.png
```

Comparing *scene* against a value between 0.3 and 0.5 is generally a sane choice.

- Send even and odd frames to separate outputs, and compose them:

```
select=n=2:e='mod(n, 2)+1' [odd][even]; [odd] pad=h=2*ih [tmp]; [tmp][even] overlay=y=h
```

## 12.7 sendcmd, asendcmd

Send commands to filters in the filtergraph.

These filters read commands to be sent to other filters in the filtergraph.

`sendcmd` must be inserted between two video filters, `asendcmd` must be inserted between two audio filters, but apart from that they act the same way.

The specification of commands can be provided in the filter arguments with the *commands* option, or in a file specified by the *filename* option.

These filters accept the following options:

`commands, c`

Set the commands to be read and sent to the other filters.

`filename, f`

Set the filename of the commands to be read and sent to the other filters.

### 12.7.1 Commands syntax

A commands description consists of a sequence of interval specifications, comprising a list of commands to be executed when a particular event related to that interval occurs. The occurring event is typically the current frame time entering or leaving a given time interval.

An interval is specified by the following syntax:

```
START[-END] COMMANDS;
```

The time interval is specified by the *START* and *END* times. *END* is optional and defaults to the maximum time.

The current frame time is considered within the specified interval if it is included in the interval [*START*, *END*), that is when the time is greater or equal to *START* and is lesser than *END*.

*COMMANDS* consists of a sequence of one or more command specifications, separated by ",", relating to that interval. The syntax of a command specification is given by:

`[FLAGS] TARGET COMMAND ARG`

*FLAGS* is optional and specifies the type of events relating to the time interval which enable sending the specified command, and must be a non-null sequence of identifier flags separated by "+" or "|" and enclosed between "[" and "]".

The following flags are recognized:

`enter`

> The command is sent when the current frame timestamp enters the specified interval. In other words, the command is sent when the previous frame timestamp was not in the given interval, and the current is.

`leave`

> The command is sent when the current frame timestamp leaves the specified interval. In other words, the command is sent when the previous frame timestamp was in the given interval, and the current is not.

If *FLAGS* is not specified, a default value of `[enter]` is assumed.

*TARGET* specifies the target of the command, usually the name of the filter class or a specific filter instance name.

*COMMAND* specifies the name of the command for the target filter.

*ARG* is optional and specifies the optional list of argument for the given *COMMAND*.

Between one interval specification and another, whitespaces, or sequences of characters starting with # until the end of line, are ignored and can be used to annotate comments.

A simplified BNF description of the commands specification syntax follows:

```
COMMAND_FLAG  ::= "enter" | "leave"
COMMAND_FLAGS ::= COMMAND_FLAG [(+|"|")COMMAND_FLAG]
COMMAND       ::= ["[" COMMAND_FLAGS "]"] TARGET COMMAND [ARG]
COMMANDS      ::= COMMAND [,COMMANDS]
INTERVAL      ::= START[-END] COMMANDS
INTERVALS     ::= INTERVAL[;INTERVALS]
```

## 12.7.2 Examples

- Specify audio tempo change at second 4:

  ```
  asendcmd=c='4.0 atempo tempo 1.5',atempo
  ```

- Specify a list of drawtext and hue commands in a file.

```
# show text in the interval 5-10
5.0-10.0 [enter] drawtext reinit 'fontfile=FreeSerif.ttf:text=hello world',
         [leave] drawtext reinit 'fontfile=FreeSerif.ttf:text=';

# desaturate the image in the interval 15-20
15.0-20.0 [enter] hue s 0,
          [enter] drawtext reinit 'fontfile=FreeSerif.ttf:text=nocolor',
          [leave] hue s 1,
          [leave] drawtext reinit 'fontfile=FreeSerif.ttf:text=color';

# apply an exponential saturation fade-out effect, starting from time 25
25 [enter] hue s exp(25-t)
```

A filtergraph allowing to read and process the above command list stored in a file `test.cmd`, can be specified with:

```
sendcmd=f=test.cmd,drawtext=fontfile=FreeSerif.ttf:text='',hue
```

## 12.8 setpts, asetpts

Change the PTS (presentation timestamp) of the input frames.

`setpts` works on video frames, `asetpts` on audio frames.

This filter accepts the following options:

`expr`

The expression which is evaluated for each frame to construct its timestamp.

The expression is evaluated through the eval API and can contain the following constants:

`FRAME_RATE`

frame rate, only defined for constant frame-rate video

`PTS`

The presentation timestamp in input

`N`

The count of the input frame for video or the number of consumed samples, not including the current frame for audio, starting from 0.

`NB_CONSUMED_SAMPLES`

The number of consumed samples, not including the current frame (only audio)

`NB_SAMPLES, S`

The number of samples in the current frame (only audio)

`SAMPLE_RATE, SR`

The audio sample rate.

`STARTPTS`

The PTS of the first frame.

`STARTT`

the time in seconds of the first frame

`INTERLACED`

State whether the current frame is interlaced.

`T`

the time in seconds of the current frame

`POS`

original position in the file of the frame, or undefined if undefined for the current frame

`PREV_INPTS`

The previous input PTS.

`PREV_INT`

previous input time in seconds

`PREV_OUTPTS`

The previous output PTS.

`PREV_OUTT`

previous output time in seconds

`RTCTIME`

The wallclock (RTC) time in microseconds.. This is deprecated, use time(0) instead.

`RTCSTART`

The wallclock (RTC) time at the start of the movie in microseconds.

`TB`

The timebase of the input timestamps.

### 12.8.1 Examples

- Start counting PTS from zero

```
setpts=PTS-STARTPTS
```

- Apply fast motion effect:

```
setpts=0.5*PTS
```

- Apply slow motion effect:

```
setpts=2.0*PTS
```

- Set fixed rate of 25 frames per second:

```
setpts=N/(25*TB)
```

- Set fixed rate 25 fps with some jitter:

```
setpts='1/(25*TB) * (N + 0.05 * sin(N*2*PI/25))'
```

- Apply an offset of 10 seconds to the input PTS:

```
setpts=PTS+10/TB
```

- Generate timestamps from a "live source" and rebase onto the current timebase:

```
setpts='(RTCTIME - RTCSTART) / (TB * 1000000)'
```

- Generate timestamps by counting samples:

```
asetpts=N/SR/TB
```

## 12.9 settb, asettb

Set the timebase to use for the output frames timestamps. It is mainly useful for testing timebase configuration.

It accepts the following parameters:

`expr, tb`

The expression which is evaluated into the output timebase.

The value for `tb` is an arithmetic expression representing a rational. The expression can contain the constants "AVTB" (the default timebase), "intb" (the input timebase) and "sr" (the sample rate, audio only). Default value is "intb".

### 12.9.1 Examples

- Set the timebase to 1/25:

  ```
  settb=expr=1/25
  ```

- Set the timebase to 1/10:

  ```
  settb=expr=0.1
  ```

- Set the timebase to 1001/1000:

  ```
  settb=1+0.001
  ```

- Set the timebase to 2*intb:

  ```
  settb=2*intb
  ```

- Set the default timebase value:

  ```
  settb=AVTB
  ```

## 12.10 showcqt

Convert input audio to a video output representing frequency spectrum logarithmically (using constant Q transform with Brown-Puckette algorithm), with musical tone scale, from E0 to D#10 (10 octaves).

The filter accepts the following options:

`volume`

Specify transform volume (multiplier) expression. The expression can contain variables:

`frequency, freq, f`

the frequency where transform is evaluated

`timeclamp, tc`

value of timeclamp option

and functions:

`a_weighting(f)`

A-weighting of equal loudness

`b_weighting(f)`

B-weighting of equal loudness

`c_weighting(f)`

C-weighting of equal loudness

Default value is `16`.

`tlength`

Specify transform length expression. The expression can contain variables:

`frequency, freq, f`

the frequency where transform is evaluated

`timeclamp, tc`

value of timeclamp option

Default value is `384/f*tc/(384/f+tc)`.

`timeclamp`

Specify the transform timeclamp. At low frequency, there is trade-off between accuracy in time domain and frequency domain. If timeclamp is lower, event in time domain is represented more accurately (such as fast bass drum), otherwise event in frequency domain is represented more accurately (such as bass guitar). Acceptable value is [0.1, 1.0]. Default value is `0.17`.

`coeffclamp`

Specify the transform coeffclamp. If coeffclamp is lower, transform is more accurate, otherwise transform is faster. Acceptable value is [0.1, 10.0]. Default value is `1.0`.

`gamma`

Specify gamma. Lower gamma makes the spectrum more contrast, higher gamma makes the spectrum having more range. Acceptable value is [1.0, 7.0]. Default value is `3.0`.

`fontfile`

Specify font file for use with freetype. If not specified, use embedded font.

`fontcolor`

Specify font color expression. This is arithmetic expression that should return integer value 0xRRGGBB. The expression can contain variables:

`frequency, freq, f`

the frequency where transform is evaluated

`timeclamp, tc`

value of timeclamp option

and functions:

`midi(f)`

midi number of frequency f, some midi numbers: E0(16), C1(24), C2(36), A4(69)

`r(x), g(x), b(x)`

red, green, and blue value of intensity x

Default value is `st(0, (midi(f)-59.5)/12); st(1, if(between(ld(0),0,1), 0.5-0.5*cos(2*PI*ld(0)), 0)); r(1-ld(1)) + b(ld(1))`

`fullhd`

If set to 1 (the default), the video size is 1920x1080 (full HD), if set to 0, the video size is 960x540. Use this option to make CPU usage lower.

`fps`

Specify video fps. Default value is `25`.

`count`

Specify number of transform per frame, so there are fps*count transforms per second. Note that audio data rate must be divisible by fps*count. Default value is `6`.

### 12.10.1 Examples

- Playing audio while showing the spectrum:

```
ffplay -f lavfi 'amovie=a.mp3, asplit [a][out1]; [a] showcqt [out0]'
```

- Same as above, but with frame rate 30 fps:

```
ffplay -f lavfi 'amovie=a.mp3, asplit [a][out1]; [a] showcqt=fps=30:count=5 [out0]'
```

- Playing at 960x540 and lower CPU usage:

```
ffplay -f lavfi 'amovie=a.mp3, asplit [a][out1]; [a] showcqt=fullhd=0:count=3 [out0]'
```

- A1 and its harmonics: A1, A2, (near)E3, A3:

```
ffplay -f lavfi 'aevalsrc=0.1*sin(2*PI*55*t)+0.1*sin(4*PI*55*t)+0.1*sin(6*PI*55*t)+0.1*sin(8*PI*55*t),
                 asplit[a][out1]; [a] showcqt [out0]'
```

- Same as above, but with more accuracy in frequency domain (and slower):

```
ffplay -f lavfi 'aevalsrc=0.1*sin(2*PI*55*t)+0.1*sin(4*PI*55*t)+0.1*sin(6*PI*55*t)+0.1*sin(8*PI*55*t),
                 asplit[a][out1]; [a] showcqt=timeclamp=0.5 [out0]'
```

- B-weighting of equal loudness

```
volume=16*b_weighting(f)
```

- Lower Q factor

```
tlength=100/f*tc/(100/f+tc)
```

- Custom fontcolor, C-note is colored green, others are colored blue

```
fontcolor='if(mod(floor(midi(f)+0.5),12), 0x0000FF, g(1))'
```

## 12.11 showspectrum

Convert input audio to a video output, representing the audio frequency spectrum.

The filter accepts the following options:

`size, s`

Specify the video size for the output. For the syntax of this option, check the "Video size" section in the ffmpeg-utils manual. Default value is `640x512`.

`slide`

Specify how the spectrum should slide along the window.

It accepts the following values:

'`replace`'

the samples start again on the left when they reach the right

'`scroll`'

the samples scroll from right to left

'`fullframe`'

frames are only produced when the samples reach the right

Default value is `replace`.

`mode`

Specify display mode.

It accepts the following values:

‘`combined`’

all channels are displayed in the same row

‘`separate`’

all channels are displayed in separate rows

Default value is ‘`combined`’.

`color`

Specify display color mode.

It accepts the following values:

‘`channel`’

each channel is displayed in a separate color

‘`intensity`’

each channel is is displayed using the same color scheme

Default value is ‘`channel`’.

`scale`

Specify scale used for calculating intensity color values.

It accepts the following values:

‘`lin`’

linear

‘`sqrt`’

square root, default

‘cbrt’

cubic root

‘log’

logarithmic

Default value is ‘sqrt’.

saturation

Set saturation modifier for displayed colors. Negative values provide alternative color scheme. 0 is no saturation at all. Saturation must be in [-10.0, 10.0] range. Default value is 1.

win_func

Set window function.

It accepts the following values:

‘none’

No samples pre-processing (do not expect this to be faster)

‘hann’

Hann window

‘hamming’

Hamming window

‘blackman’

Blackman window

Default value is hann.

The usage is very similar to the showwaves filter; see the examples in that section.

### 12.11.1 Examples

- Large window with logarithmic color scaling:

```
showspectrum=s=1280x480:scale=log
```

- Complete example for a colored and sliding spectrum per channel using ffplay:

```
ffplay -f lavfi 'amovie=input.mp3, asplit [a][out1];
             [a] showspectrum=mode=separate:color=intensity:slide=1:scale=cbrt [out0]'
```

## 12.12 showwaves

Convert input audio to a video output, representing the samples waves.

The filter accepts the following options:

`size, s`

Specify the video size for the output. For the syntax of this option, check the "Video size" section in the ffmpeg-utils manual. Default value is "600x240".

`mode`

Set display mode.

Available values are:

'`point`'

Draw a point for each sample.

'`line`'

Draw a vertical line for each sample.

'`p2p`'

Draw a point for each sample and a line between them.

'`cline`'

Draw a centered vertical line for each sample.

Default value is `point`.

`n`

Set the number of samples which are printed on the same column. A larger value will decrease the frame rate. Must be a positive integer. This option can be set only if the value for *rate* is not explicitly specified.

`rate, r`

Set the (approximate) output frame rate. This is done by setting the option *n*. Default value is "25".

`split_channels`

Set if channels should be drawn separately or overlap. Default value is 0.

### 12.12.1 Examples

- Output the input file audio and the corresponding video representation at the same time:

```
amovie=a.mp3,asplit[out0],showwaves[out1]
```

- Create a synthetic signal and show it with showwaves, forcing a frame rate of 30 frames per second:

```
aevalsrc=sin(1*2*PI*t)*sin(880*2*PI*t):cos(2*PI*200*t),asplit[out0],showwaves=r=30[out1]
```

## 12.13 split, asplit

Split input into several identical outputs.

`asplit` works with audio input, `split` with video.

The filter accepts a single parameter which specifies the number of outputs. If unspecified, it defaults to 2.

### 12.13.1 Examples

- Create two separate outputs from the same input:

```
[in] split [out0][out1]
```

- To create 3 or more outputs, you need to specify the number of outputs, like in:

```
[in] asplit=3 [out0][out1][out2]
```

- Create two separate outputs from the same input, one cropped and one padded:

```
[in] split [splitout1][splitout2];
[splitout1] crop=100:100:0:0    [cropout];
[splitout2] pad=200:200:100:100 [padout];
```

- Create 5 copies of the input audio with `ffmpeg`:

```
ffmpeg -i INPUT -filter_complex asplit=5 OUTPUT
```

## 12.14 zmq, azmq

Receive commands sent through a libzmq client, and forward them to filters in the filtergraph.

`zmq` and `azmq` work as a pass-through filters. `zmq` must be inserted between two video filters, `azmq` between two audio filters.

To enable these filters you need to install the libzmq library and headers and configure FFmpeg with `--enable-libzmq`.

For more information about libzmq see: http://www.zeromq.org/

The `zmq` and `azmq` filters work as a libzmq server, which receives messages sent through a network interface defined by the `bind_address` option.

The received message must be in the form:

```
TARGET COMMAND [ARG]
```

*TARGET* specifies the target of the command, usually the name of the filter class or a specific filter instance name.

*COMMAND* specifies the name of the command for the target filter.

*ARG* is optional and specifies the optional argument list for the given *COMMAND*.

Upon reception, the message is processed and the corresponding command is injected into the filtergraph. Depending on the result, the filter will send a reply to the client, adopting the format:

```
ERROR_CODE ERROR_REASON
MESSAGE
```

*MESSAGE* is optional.

### 12.14.1 Examples

Look at `tools/zmqsend` for an example of a zmq client which can be used to send commands processed by these filters.

Consider the following filtergraph generated by `ffplay`

```
ffplay -dumpgraph 1 -f lavfi "
color=s=100x100:c=red  [l];
color=s=100x100:c=blue [r];
nullsrc=s=200x100, zmq [bg];
[bg][l]   overlay      [bg+l];
[bg+l][r] overlay=x=100 "
```

To change the color of the left side of the video, the following command can be used:

```
echo Parsed_color_0 c yellow | tools/zmqsend
```

To change the right side:

```
echo Parsed_color_1 c pink | tools/zmqsend
```

# 13 Multimedia Sources

Below is a description of the currently available multimedia sources.

## 13.1 amovie

This is the same as movie source, except it selects an audio stream by default.

## 13.2 movie

Read audio and/or video stream(s) from a movie container.

It accepts the following parameters:

`filename`

> The name of the resource to read (not necessarily a file; it can also be a device or a stream accessed through some protocol).

`format_name, f`

> Specifies the format assumed for the movie to read, and can be either the name of a container or an input device. If not specified, the format is guessed from *movie_name* or by probing.

`seek_point, sp`

> Specifies the seek point in seconds. The frames will be output starting from this seek point. The parameter is evaluated with `av_strtod`, so the numerical value may be suffixed by an IS postfix. The default value is "0".

`streams, s`

> Specifies the streams to read. Several streams can be specified, separated by "+". The source will then have as many outputs, in the same order. The syntax is explained in the “Stream specifiers” section in the ffmpeg manual. Two special names, "dv" and "da" specify respectively the default (best suited) video and audio stream. Default is "dv", or "da" if the filter is called as "amovie".

`stream_index, si`

> Specifies the index of the video stream to read. If the value is -1, the most suitable video stream will be automatically selected. The default value is "-1". Deprecated. If the filter is called "amovie", it will select audio instead of video.

`loop`

> Specifies how many times to read the stream in sequence. If the value is less than 1, the stream will be read again and again. Default value is "1".

Note that when the movie is looped the source timestamps are not changed, so it will generate non monotonically increasing timestamps.

It allows overlaying a second video on top of the main input of a filtergraph, as shown in this graph:

```
input -----------> deltapts0 --> overlay --> output
                                    ^
                                    |
movie --> scale--> deltapts1 -------+
```

### 13.2.1 Examples

- Skip 3.2 seconds from the start of the AVI file in.avi, and overlay it on top of the input labelled "in":

  ```
  movie=in.avi:seek_point=3.2, scale=180:-1, setpts=PTS-STARTPTS [over];
  [in] setpts=PTS-STARTPTS [main];
  [main][over] overlay=16:16 [out]
  ```

- Read from a video4linux2 device, and overlay it on top of the input labelled "in":

  ```
  movie=/dev/video0:f=video4linux2, scale=180:-1, setpts=PTS-STARTPTS [over];
  [in] setpts=PTS-STARTPTS [main];
  [main][over] overlay=16:16 [out]
  ```

- Read the first video stream and the audio stream with id 0x81 from dvd.vob; the video is connected to the pad named "video" and the audio is connected to the pad named "audio":

  ```
  movie=dvd.vob:s=v:0+#0x81 [video] [audio]
  ```

# 14 See Also

ffmpeg, ffplay, ffprobe, ffserver, libavfilter

# 15 Authors

The FFmpeg developers.

For details about the authorship, see the Git history of the project (git://source.ffmpeg.org/ffmpeg), e.g. by typing the command `git log` in the FFmpeg source directory, or browsing the online repository at http://source.ffmpeg.org.

Maintainers for the specific components are listed in the file `MAINTAINERS` in the source code tree.

This document was generated on *January 10, 2015* using *makeinfo*.