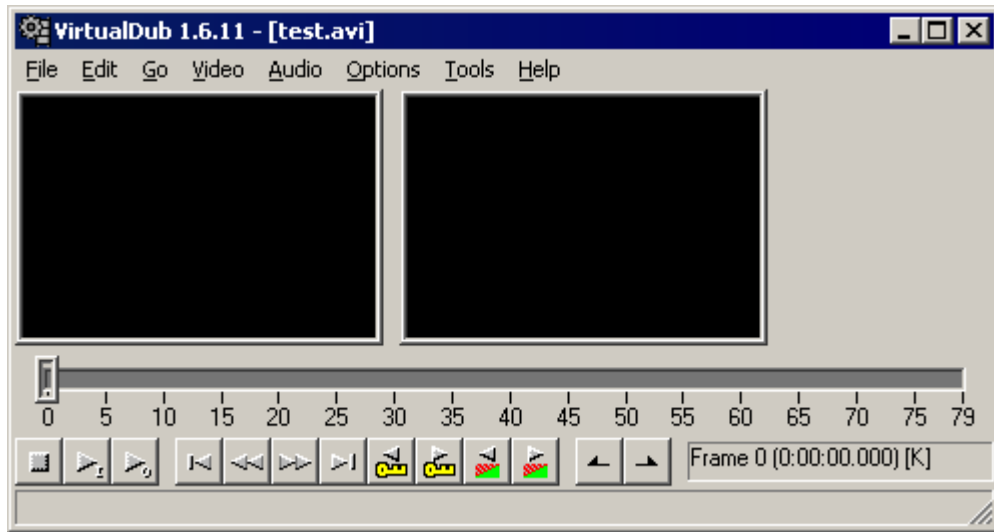


VirtualDub help

Welcome to the new helpfile for VirtualDub 1.5+.

The old help file used to have some general tips on how to create video, but I've come to the conclusion that such a help file is too much effort to maintain and that my efforts in this area are better spent explaining what the options do rather than how to use them. So, currently, this help file currently consists of comprehensive explanations of what various settings dialogs in VirtualDub do.

VirtualDub help - Processing: Main UI



Display panes

Two rectangular panes occupy most of VirtualDub's interface; the left one is the *input pane* and the right one is the *output pane*. As the current position is moved through the video, the panes will update with the original and filtered result of that frame. This allows convenient before-and-after comparisons of the video.

The positioning and size of the display panes can be modified in a number of ways; see [Display panes](#) for more information.

Position slider

Drag the slider below the panes to change the current position within the video. If enabled, the display panes will update to show the selected frame. Holding Shift while dragging forces the current position to be placed only on key frames. Using the right mouse button to drag the slider instead of the left causes the drag to occur at a much slower rate for more precise positioning.

Each position on the position slider corresponds to the *start* of each frame in the video timeline. As a result, there is one additional position at the end corresponding to the end of the video. For instance, with a 2280 frame video the slider may be positioned from 0 to 2280.

Playback controls ()

Starts and stops preview playback of the video. The play button with the small "I" only plays the input video; the one with the small "O" previews both the input and the filtered output video. Previewing the output video requires a lot more CPU power and may not be able to occur in real-time if the video frame is large or complex video filters are in use.

Frame step controls ()

Jumps to the beginning, previous frame, next frame, and end of the video, respectively. These actions can also be performed through the keyboard using Ctrl+ arrow, Left-arrow, Right-arrow, and Ctrl(arrow.

Key frame step controls ()

Jumps to the previous or next *key frame* in the video. A *key frame* is a frame that is stored fully-contained in the video file and doesn't depend on any other frames for decoding; these are the fastest seek points in the video and often of slightly higher quality as well. Stepping by key frame is often significantly faster and is handy when browsing through the video. In addition, the key frames represent appropriate cut points when [editing in Direct mode](#).

The key frame step controls can also be activated through the keyboard using Shift+ arrow and Shift(arrow).

When the video source is an MPEG-1 file, the key frame step controls jump between I-frames.

Scene step controls ()

Jumps to the previous or next scene in the video. Scene detection is done by image analysis heuristics, by looking for large picture changes that indicate possible cuts or fades in the video. The thresholds for determining a fade or a cut can be controlled in the Preferences dialog.

When scanning through a large amount of video, the display panes will begin to update at a slower rate. This is normal. The pane updates are reduced after a short period in order to reduce their CPU usage and speed up the scan.

Mark-in and mark-out controls ()

Sets the starting and ending points for the selection to the current position. The selection is used by the commands in the Edit menu to modify the portions of video to be rendered. Also, if a portion of video is selected when a save command is used, only that portion of video is processed.

The Home and End keys can also be used to select video.

Note Since the position slider moves to the *beginning* of frames, the mark-out point must be placed one frame *after* the last frame to be selected. This means that if 300 frames are to be deleted starting at frame 100, the selection should be started (mark-in) at frame 100 and ended (mark-out) at frame 100M = 400, not at frame 399. This is referred to as *end-point exclusive* selection.

Timestamp display ()

Displays the current frame number, timestamp, and frame type at the current position. Frame types are as follows:

- [K]: A key frame. (AVI)
- []: A delta frame — stored as a difference from the previous frame. (AVI)
- [D]: A drop or null frame, which repeats the previous frame. These are most often found in capture files. (AVI)
- [I]: An I-frame; similar to a key frame. (MPEG-1)
- [P]: A P-frame, or *forward predicted* frame. These are stored as a difference from an earlier frame. (MPEG-1)
- [B]: A B-frame, or *bidirectionally predicted* frame. These are stored as a difference from both an earlier frame and a future frame. (MPEG-1)
- [M]: A masked frame. These are frames that have been tagged in VirtualDub's timeline as not to be processed; instead, the previous frame should be used. This is most often used to bypass errors in the source. The end frame will also show

up as one of these.

The format of the timestamp display can be customized in Preferences.

VirtualDub help - Processing: Editing the source video

Although VirtualDub is not a full non-linear editing (NLE) application, it does have some limited functionality for editing source video. Unwanted portions of video can thus be trimmed off from a video before it is processed, saving disk space and time.

Selecting and editing portions of the timeline

Use the *Edit > Set selection start* and *Edit > Set selection end* commands to select a portion of video. This can also be done through the Home and End keys on the keyboard, or through the mark-in and mark-out buttons below the position slider. The current selection is then indicated by a sky-blue area on the position slider.

Once a portion of video has been selected, the *Delete* command (keyboard shortcut: Delete key) can be used to remove that video from the timeline. The *Cut*, *Copy*, and *Paste* commands can also be used to reorder video (they cannot be used to splice or combine video files together, however). The *Undo/Redo* commands can be used to reverse mistakes, and the *Reset timeline* command undoes all edits entirely, restoring the timeline to the entirety of the source video.

Editing the timeline only creates an edit list within VirtualDub for later use; it never modifies the source file in any way. In particular, deleting a section of the timeline does not delete anything from the source file, and the edited result must be saved to a new file. Editing a video file thus requires disk space to hold both the original and the edited clip.

As a shortcut, the selection mark-in/mark-out commands also modify the range selection that is exposed via the *Video > Select range...* menu command. Thus, selecting a portion of video prior to a save command causes only that selection to be rendered to disk. If this is undesired, clear the selection using the *Edit > Clear selection (Ctrl)* command.

Caveats when editing

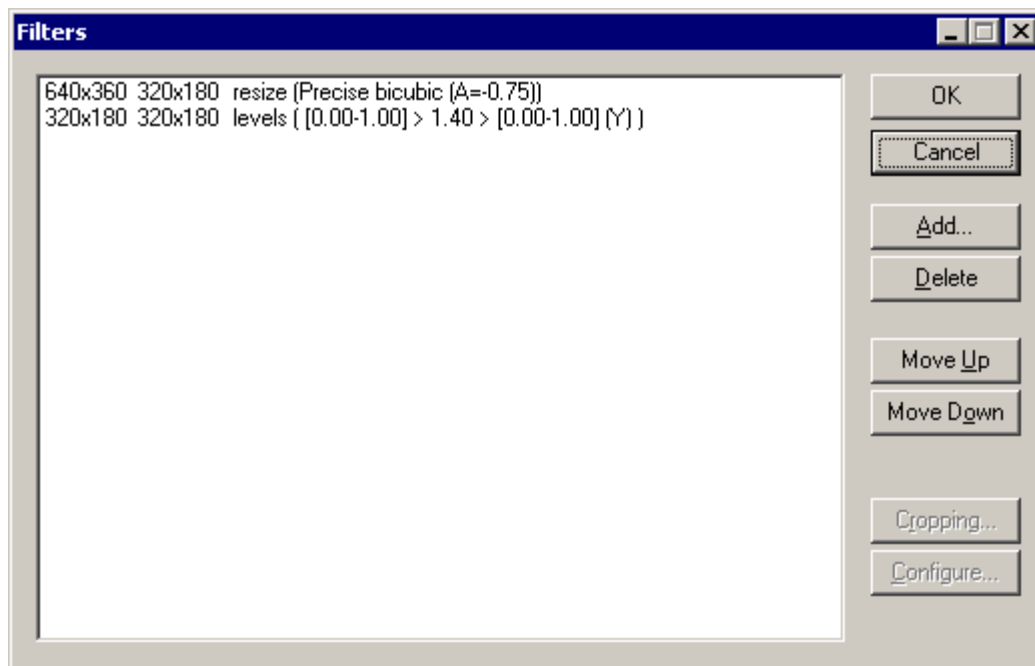
VirtualDub does not have support for transitions, so any edits will be abrupt unless the edit points are located at places in the source video that hide the seam. Silent fades to black are a good place to remove selections of video.

Editing can be performed in any video or audio mode, including Direct mode, which causes the render-to-disk to be extremely fast and without quality loss in most cases. However, there are serious limitations with editing in this mode that restrict where edits can be performed; also, advanced audio and video compression can impede VirtualDub's ability to edit cleanly. For more information, see [Direct mode](#).

VirtualDub help - Processing: Filtering video

The ability to apply *video filters* to a video is one of VirtualDub's more powerful features. This allows application of a number of algorithms to improve the quality of a video, including noise reduction, blurring, sharpening, brightness/contrast, and gamma correction.

Using video filters



The *Video > Filters...* menu command is the entry point into using the video filter system. Video filters in VirtualDub are arranged in a linear chain, such that the output of the first filter becomes the input of the second, the output of the second becomes the input of the third, and so on. Therefore, the order in which the video filters are listed is the order in which they are applied. By default, no filters are in this list, meaning that the video filter system is skipped entirely.

The video processing mode must be set to *Full processing mode* for the video filter system to be enabled. If the video processing mode is set to anything else, the *Filters...* menu option is grayed and the video filter chain is disabled.

For more information, see the documentation for the [video filters dialog](#).

Common video processing operations

VirtualDub contains a number of internal video filters that assist in a number of common video processing tasks. Consult the [video filter reference](#) for full details, but here are some useful filter techniques:

- The *resize* filter will change video to a different resolution (size), in pixels. It can also be used to letterbox a video by adding borders.
- *Levels* is useful to adjust brightness and contrast in a video, particularly when blacks are not truly black, or whites are not truly white.
- You can crop at the beginning of any filter by selecting it and then the Crop button. If you don't have any filters, though, just add *null transform* to give you a place to crop.
- *Temporal smoother* can be effective at denoising a video. You will need to adjust the threshold to set a balance between less noise and motion artifacts, however.
- Use *chroma smoother* to correct for blocky color produced by video codecs that do not interpolate color information properly.
- *Rotate* a video by 90° increments until it is upright, or use the slower but more powerful *rotate2* if the rotation is arbitrary.
- When video shows keystone from being displayed or shot off-center, try *perspective* to map it back parallel to the screen.
- Stamp a *logo* on video to mark it as yours.

The order in which filters are applied can make a significant difference in performance and quality. In terms of performance, a smaller video frame is faster to process, so when down,

try putting filters after the reduction, and when resizing up, prefer placing them before the enlargement. In terms of quality, blurring reduces noise, and sharpening amplifies it — so do your noise reduction early and your enhancement transforms late.

External video filters

VirtualDub exposes an application programming interface (API) that allows video filters to be written as plugins, extending the video filtering capabilities of the program. Such plugins can include effect plugins that render on top of the video, such as adding an animated logo or titles, or add noise reduction, motion compensation, and deinterlacing capabilities.

Video filters can be loaded on the spot by using the *Load...* button on the Add Filters dialog, but for filters that you use frequently, an easier way is to place them in the *plugins* folder inside the VirtualDub program directory. Any *.vdf* files in this folder are automatically loaded on startup, and the plugins within added to the available list. Note that VirtualDub does not currently support video filters written to a standard other than its own API.

Check the [VirtualDub website](#) for downloadable documentation and C header files for writing your own video filter.

VirtualDub help - Processing: Rendering and saving the processed output

Once the source video has been edited as necessary and appropriate processing parameters set, the video can be *rendered* to generate the final result. This can either be previewed live or saved to a file on disk.

Previewing the edited result

The *File > Preview output from start...* command begins rendering the timeline to the video display so that it can be previewed. Most video and audio processing operations are active in this mode, with the notable exception of audio and video compression, which are disabled. The result that is seen in the output display pane and heard from the system speakers is thus representative of the output of the video and audio filter systems, but may be of higher quality than what would be stored in compressed form.

Rendering filtered audio and video in real-time consumes a lot of CPU power and in many cases VirtualDub will have difficulty attaining full frame rate given a complex filter chain. When this occurs, the audio may lose sync and begin to stutter as the video frame rate drops below real-time, since all video frames are still displayed. The *Option > Drop video frames when behind* option can help here by allowing VirtualDub to process only a portion of the video frames in order to maintain real-time performance. This only affects preview and does not remove frames during any save-to-disk operation. Note that this may not be sufficient in extreme cases where the audio chain or the hard disk is unable to attain real-time either.

The *Sync to audio* option also affects preview by changing the way that VirtualDub synchronizes audio and video playback; it should normally be left on, but if there are problems with audio timing that prevent synchronized playback from occurring, disabling this option may allow preview to proceed. Like the option to drop frames, this option too only pertains to previewing and does not affect renders to disk.

Saving the processed result to disk

File > Save AVI... starts a render process to disk. A new AVI file is then generated containing the processed video and audio.

VirtualDub is normally able to write AVI files larger than 2GB using an extension to the AVI file format called the *OpenDML hierarchical index*. This is done in such a way that older applications that do not understand the hierarchical index can still open the first 2GB of the file. However, occasionally an application cannot open such AVI files at all. The *File > Save old format AVI* command disables VirtualDub's use of that extension so that only an original-format AVI is written. Note that this format does not support AVI files larger than 2GB, so care must be taken to appropriately trim or compress the video to fit below this threshold.

If only the audio is desired, the *File > Save WAV...* menu option produces an audio file on disk using the WAV format. All audio options are active, except for the interleaving interval, which does not apply since no video is being written. Video is not processed in this mode. Note that audio compression *is* active since WAV files can either be compressed or uncompressed, so be sure to disable audio compression if an uncompressed WAV file is desired.

Although VirtualDub can read MPEG-1 files, it is not currently able to write them, even in Direct mode.

Analysis passes

Some video filters and video codecs may require *analysis passes* in order to effectively filter or compress video. In the analysis passes, the video is scanned to determine difficult areas of motion or other features; knowledge of the *entire* video can then be used to optimize the final output. This is known as *multi-pass* processing.

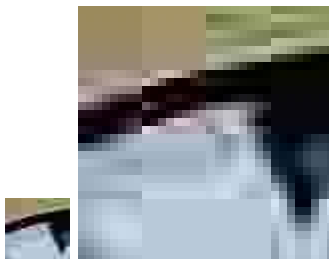
For various reasons, VirtualDub does not know that a multi-pass operation is required by a video filter or codec and cannot automate the process. However, the *File > Run video analysis pass* assists in running analysis passes by running the video pipeline without writing a dummy file to disk. The audio pipeline is disabled as well for additional speed.

VirtualDub help - Processing: Video artifacts

Artifacts, or undesirable errors, invariably creep into video. If you are working with analog video or with video compression, some artifacting is inevitable. Knowing different types of artifacts and their causes can help you determine which are due to shortcomings in hardware and software, deficiencies in your video process, or even software (or hardware!) bugs.

A few artifacts and their causes are listed below. This is not meant to be an exhaustive list of artifacts you may encounter, but it should cover the common ones.

Quilting (Compression artifacts)



Quilting results from high levels of compression on an image that cause warbling and edge artifacts to appear in the output. The term "quilting" refers to the fact that video compression is usually done in blocks of 8x8 or 16x16 pixels, so if the compression factor is set very high such that the blocks don't match well, the result looks like a quilt. The warbles around sharp edges are result of discarding detail from the image for higher compression. In other words,

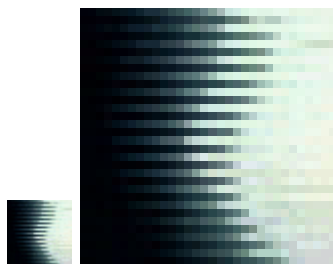
it's not that warbles were added, but that the detail which would sharpen the edge and flatten the area around it has been dropped.

Advanced video compression algorithms, particularly those based on the MPEG-4 video standard, have *post-processing filters* designed to reduce these artifacts, which can give the video a smeared and cartoony look that is less objectionable.

To reduce quilting artifacts:

- Use more advanced video compression formats.
- Compress less — use more bitrate and produce larger video files.
- Use multi-pass compression if available, to better distribute bits to where they are needed.

Combing (interlacing artifacts)



Analog video is delivered using a mechanism known as *interlacing* to reduce flicker. Instead of sending entire frames at 29.97 frames per second (25 for PAL/SECAM), the video is sent in halves called *fields*, at 59.94 fields per second (50 for PAL/SECAM). These fields are *interlaced* together such that a frame is composed of alternating lines from each field; the even lines make up the *even field* and the odd lines make up the *odd field*. The result is high resolution in static scenes and smoother motion in fast-moving scenes, with less flicker and without requiring more bandwidth.

The catch with interlacing is that you can't have both higher resolution and smoother motion at the same time, so artifacts appear whenever you try to extract both. The process of removing the interlacing and displaying the result non-interlaced is known as *deinterlacing*. Displaying each field by itself gives smoother motion at the cost of resolution and is known as *bob deinterlacing*. Pairing fields up and displaying them together as frames gives higher resolution in exchange for smeared motion and combing artifacts and is called *weave deinterlacing*. Both produce frames at field rate (59.94 or 50 fps). Trying to switch between the two on a per-frame or even per-area basis depending on the amount of local motion is *adaptive deinterlacing* and can produce an even higher quality image.

Most video capture devices do not attempt to deinterlace and simply pair fields together, which is similar to weave deinterlacing except that it gives half the field rate (29.97 or 25). If the video was produced originally from full frames at that rate, this has a 50/50 chance of producing whole frames instead of a combed mess. There is no requirement that this be the case, though, and since the alternating fields are evenly staggered in time they usually aren't. In that case there is no "correct" way to deinterlace the video, and different deinterlacing techniques must be tried to produce the best quality non-interlaced video.

Material derived from 24 fps film is a special case in that the video is sourced from whole frames and split into fields in a specific pattern; this is called *telecine*. In NTSC, this is done by slowing the video down very slightly and combining fields in a 3:2 pattern; VirtualDub's inverse telecine feature, accessible in the [video frame rate control dialog](#), can sometimes undo this pattern. In the case of PAL, the video is usually just sped up by 4% from 24 fps to 25 fps, so the most that is usually required is a single-field delay to pair up the fields correctly.

Banding (quantization artifacts)

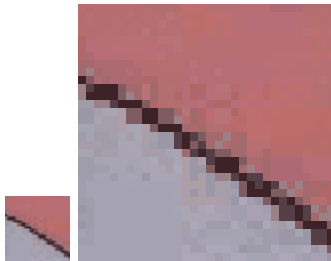


Banding occurs when the bit depth used to represent pixels in an image is too low; the result is stairstepping in the image as pixels are forced to hop between colors that are far enough apart to distinguish visually. This is most visible in large, shallow gradients, which become bands of solid color.

When banding is observed, you should first check the display settings for your system to ensure that the display is set to 24-bit color or 32-bit color, which will produce the least banding. In particular, selecting a 15-bit or 16-bit display mode will introduce banding on screen that may not be present in the actual video.

Selecting 15-bit RGB or 16-bit RGB as a processing format will introduce noticeable banding into an image, so these formats should generally be avoided. It is still possible to see banding with YCbCr formats or 24/32-bit RGB if the gradient is very shallow and over a large area; using higher-precision formats or dithering can alleviate this. Note that VirtualDub does not currently support a format that has greater than 8 bits of precision per channel (256 levels).

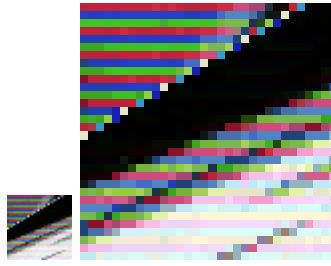
Scaling artifacts



Rough and slightly blocky edges may be indicative of a poor scaling algorithm being used to resize video. In particular, use of a *nearest neighbor* or *point sampling* algorithm can give a blocky look to video due to the lack of interpolation ("smooth" stretching) during the resize, which means that rows and columns are just duplicated or deleted instead of being blended to modify the video's size. The result is that thin creases appear in the image.

A bad resize operation is difficult to undo after the fact if you no longer have the source, but if you can redo the bad operation, try moving any scaling operations later in the process so they can be done using VirtualDub's high-quality *resize* filter, with a bilinear or bicubic filter. For instance, if you are attempting to capture analog video at a 480x360 resolution, try using 640x480 or 640x576 — something closer to the native resolution — and then scaling in post-processing. This will often take more CPU power and storage, however.

Rainbows (pitch/stride errors)



A regular slant to a decoded image with rainbows across scanlines is usually indicative of a buggy video codec that does not compute *pitch* or *stride* correctly. The technical reason for this is that padding at the end of each horizontal row is not being accounted for correctly, resulting in each row being progressively further off. This isn't that important to diagnose the problem, though.

The key to fixing the problem is that multiples of *four pixels* in width usually work around this bug, because in that case the row-end correction is unnecessary. If attempting to compress with a video codec gives striped results when using widths like 321, 639, etc. but 316, 320, 324, 636, 640, 644... work, then you are experiencing this issue.

VirtualDub help - Processing: The pipeline

VirtualDub's processing of audio and video during a render-to-disk operation is split into several pipeline stages. Some of these stages are enabled or disabled depending on the current audio/video mode selected.

Video pipeline

The video pipeline can be run in one of four different modes:

- *Direct stream copy*. In this mode, video frames are copied directly from input to output. No recompression takes place, and thus no quality loss can occur. This is the fastest possible mode for editing video in VirtualDub.

Because the video is not recompressed, video compression imposes restrictions on how the video can be edited.

- *Fast recompress*. Video is decompressed and then recompressed using the desired output codec. VirtualDub automatically chooses an intermediate video format to use between the codecs for quality and speed.

An output video codec must be chosen in this mode.

- *Slow recompress*. Video is decompressed and then recompressed using the desired output codec. This is similar to Fast Recompress except that the input and output formats can be chosen in the [Video color depth](#) dialog, and the two can be different, requiring a conversion in between.

If no output video codec is chosen, the video is written to disk uncompressed in the output format.

- *Full processing mode*. All pipeline stages and features are enabled.

Here's what the video pipeline looks like:

Direct

Recompress

Full

Frame sequencing

Read frame

Decompress frame

Inverse telecine

Convert to 32-bit RGB

Run video filters

Convert to target format

Compress frame

Write video

Here's what the various stages do:

Frame sequencing

Video frames are selected from sources and ordered. This is where any edits done to the timeline take place, along with the frame rate options, including rate adjustment, conversion, and decimation.

If Direct mode is selected, there are some restrictions as to how frames can be inserted or dropped. Any edits to the timeline that violate these restrictions are adjusted here to comply.

Read frame

Video frames are read from disk.

Decompress frame

Compressed video frames are run through a video codec to produce uncompressed video frames. The format is selected in the [Video color depth](#) dialog.

In Fast Recompress mode, the format is automatically selected based on compatibility between the input and output video codecs.

Inverse telecine

If inverse telecine (3:2 pulldown removal) is enabled in [Video frame rate control](#), fields are reordered and the video stream frame rate is reduced by 25% at this point.

Convert to 32-bit RGB

Video filters in VirtualDub currently only run in 32-bit RGB, so the video frames are converted to 32-bit RGB at this point.

Note In previous versions of VirtualDub, enabling full processing mode would always force a conversion to 32-bit RGB. This is no longer the case — if no video filters are used, this conversion step is omitted and the video is directly converted to the output format as in Slow Recompress mode.

Run video filters

All [video filters](#) are run at this point.

Convert to output format

The video frames are converted from their current format to the output format specified in the [Video color depth](#) dialog. If the formats are the same, no conversion takes place.

Conversions between YCbCr formats are done directly in YCbCr space without an RGB intermediate step. Chroma is subsampled or supersampled as necessary using bilinear filtering.

Compress frame

If a [video compression codec is selected](#), it is now used to compress the video frame.

Write video

The video frame is now written to disk.

Audio pipeline

The audio pipeline has three modes: Direct, Full without audio filters, and Full with audio filters. Enabling audio filters replaces other types of audio processing in the pipeline, thus the parallel path.

Direct	Full
Sequencing	
Read audio	
	Decompress audio
	Format conversion
	Filter graph
	Resampling
	Volume adjustment
	Compression
Write audio	

Sequencing

Any audio edits take place here. These are basically the same in time as the video edits.

Read audio

Source audio is read from disk.

Decompress audio

Audio is decompressed using an audio codec, if necessary, producing uncompressed PCM audio. This is usually in 16-bit mono or 16-bit stereo format.

Format conversion

Precision and channel changes requested under [audio conversion](#) now take place, including switching between 8-bit and 16-bit samples, as well as mixing down to mono or cloning channels to produce stereo.

Resampling

Changes in sampling rate requested in the [audio conversion dialog](#) now take place. If high quality mode is off, point sampling is used, otherwise linear interpolation is used.

If higher quality resampling is required, the resample audio filter should be used

instead, which uses a multi-tap windowed sinc filter.

Volume adjustment

If volume adjustment is enabled, the audio is now attenuated or amplified using a linear multiplication with clamping.

Compression

The audio is now recompressed using the selected output audio codec. If no audio codec is selected, the audio is simply written out using its current format.

Write audio

The finished audio is written to disk.

VirtualDub help - Processing: Direct mode

VirtualDub allows audio and video streams to be processed in *direct mode*. In this mode, data is simply copied from input and output. This has the advantage of much faster rendering and no quality loss, while still allowing a limited amount of editing.

Because of the way that audio and video compression works, there are some limitations imposed on the types and locations of edits that can be done in direct mode. However, because audio and video modes are independent, it is possible to have only one pipeline run in direct mode, and not incur the limitations that would be imposed by the other.

Limitations on editing compressed video streams

Video compression imposes severe restrictions on where edits can occur in the video stream in direct mode. Most compression occurs by removing redundant data between adjacent frames, which results in a *delta frame* that is dependant on the previous frame to be decoded properly. The result is that the previous frame can't be removed without making that delta frame undecodable. Frames which aren't dependant on the previous frame are known as *key frames* and serve as anchor points in the stream for seeking and editing purposes.

The rule that must be heeded when editing a direct mode stream in VirtualDub is that *a portion of video to be removed must end on a keyframe*.

Key frames are denoted by [K] next to the timestamp below the seek bar in VirtualDub; delta frames are denoted by [] instead. Because selections in VirtualDub are *endpoint exclusive* — meaning that the frame you end the selection on is not included in the selection — you want to *end the selection on a key frame*.

As an example, assume that you have a set of frames like this:

K K K

This cut is *kosher*:

K K K

This cut, however, is not, because it leaves a delta frame that is missing its predecessor:

K K K

When such a cut is made, VirtualDub automatically adjusts the cut ranges until the restrictions of delta frame compression are satisfied. Thus, the above cut would actually give the following:

K K K

The rules for such automatic corrections:

- VirtualDub will not let you write a video stream with dangling delta frames.
- Frames are always added back in, but never removed, duplicated or reordered.

Thus, if you make a mistake, you can always load in the edited file and *re-edit* in direct mode, making a larger cut that satisfies the rules.

A *null frame* or *drop frame*, which is a zero-byte frame that simply duplicates the previous frame, has special handling in VirtualDub's pipeline. These are denoted by [D] next to the timestamp indicator and are occasionally produced during video capture. Such frames are dependant upon the previous frame, but can still be removed without affecting decoding. Note that these frames occupy time in the stream, however, and so deleting them will remove the corresponding audio segment as well.

Video frame decimation/conversion in direct mode

The frame rate decimation and conversion modes resample a video stream by inserting or removing frames. This essentially involves micro-editing of the stream at the frame level and suffers from similar limitations with compressed streams. Here are the frame rate limitations when using direct mode:

- *Frame rate adjustment* simply tweaks the frame rate of the video stream and can be used without limitation.
- *Conversion to a higher rate* works by inserting zero-byte null frames into the output stream, and can also be used without limitation. (This means you can convert a 30fps stream to 120fps with no loss and with almost no size increase.)
- *Conversion to a lower rate* has to delete frames, but suffers from the limitation on dependant frame removal. If used on a compressed stream, the option is only able to remove frames immediately before a key frame, which means that sequences of delta frames are longer than a few frames, the video will stutter and audio sync will be affected. Conversion to a lower rate is thus only usable with a stream that has few or no delta frames.
- *Decimation* is equivalent to conversion to a 1/N frame rate and has the same issues.

As with edits, null frames also receive special treatment here, so if a video has been upsampled from 30fps to 120fps by inserting null frames, conversion can be used to discard the null frames and drop the stream back to 30fps.

Video streams that are direct-mode friendly

A video stream using a format that only uses key frames imposes no limitations on the location of cuts in direct mode. Such formats include:

- Any uncompressed RGB or paletted format
- Any uncompressed YCbCr format (UYVY, YUY2, YV12, I420, etc.)
- Video compression that only uses key frames, such as Huffiyuv, Motion JPEG, or DV.

These formats are thus very friendly to direct-mode editing and are good choices for capture or intermediate video files.

Limitations on direct-mode imposed by source format

MPEG-1 video streams cannot be copied in direct mode, because MPEG-1 video compression is incompatible with the AVI file format. Also, MPEG-1 audio streams are always decompressed to raw PCM regardless of the audio mode setting.

DV files that use interleaved storage (type-1 DV AVI) may have their audio streams slightly modified when processing the audio stream in direct mode, because VirtualDub has to resample the audio stream in some cases to force a consistent audio sample rate. This is not a problem for AVIs that have the DV data split into traditional audio and video streams (type-2).

Limitations on editing imposed by audio compression

Audio compression works by processing blocks of audio as individual units. In direct mode, VirtualDub copies these blocks as atomic units, so the length of time corresponding to the block sets the minimum granularity for edits, and thus the accuracy of edits that can be performed.^[1]

For some formats that simply translate samples 1:1, such as A-law and μ -law, the block size is one sample and no restrictions are necessary. Other formats, such as ADPCM, can have a block size as large as 2048 samples (0.18s at 11KHz). The [audio compression](#) dialog indicates the block size for each selectable format.

VirtualDub does *not* attempt to adjust edits to match audio granularity because the audio block size rarely corresponds to an integral number of video frames in time, which would require fractional edits. The difference between the ideal cut point and the cut point imposed by audio compression appears as sync error and thus editing a compressed audio stream should be avoided if possible.

Some compressed formats, particularly MPEG audio layer III, have additional decoding restrictions that are not described adequately in the audio format structure, such as dependencies on previous frames, or even specify a block size that is blatantly false (namely, one byte). Because VirtualDub is not able to detect or correct for such limitations, editing streams in such formats can result in audible decoding errors due to block fragments at the cut point and is not recommended.

[1] The size of the block is set, in bytes, by the `nBlockAlign` field in the `WAVEFORMATEX` structure that describes the audio format.

VirtualDub help - Processing: Display panes

The *display panes* are the rectangular displays in which the input and output video are shown when scrubbing over the timeline, previewing the output, or rendering to disk.

Adjusting the display panes

Right-click on a display pane to bring up a context menu.

Zoom	Aspect Ratio	Preferred filter
25%	✓ Free adjust	Point
33%	10:11 pixel (DV-NTSC)	Bilinear
50%	1:1 pixel (Square)	Bicubic
66%	59:54 pixel (DV-PAL)	✓ Any available
75%	40:33 pixel (DV-NTSC Wide)	
✓ 100%	15:11 pixel (SVCD-NTSC)	
150%	118:81 pixel (DV-PAL Wide)	
200%	59:36 pixel (SVCD-PAL)	
300%	20:11 pixel (SVCD-NTSC Wide)	
400%	59:27 pixel (SVCD-PAL Wide)	
Reset to exact size	4:3 frame (TV)	
	15:11 frame (DV)	
	16:9 frame (Wide)	

This menu allows the adjustment of the size, aspect ratio, and filtering mode of each display pane. All of these settings are purely for display purposes — none have any effect on the video written to disk. This means that a video pane can be stretched 2:1 vertically to display a field-split video at the correct aspect ratio without ill effects.

The size and aspect ratio can also be adjusted by dragging the bottom and right borders of the display pane. If the aspect ratio is anything other than *free adjust*, the pane dimensions are constrained to that aspect ratio during resizing; otherwise, both aspect ratio and size are adjusted during the drag.

Reset to exact size sets the size and aspect ratio of the display pane so that each pixel in the display pane corresponds to exactly one pixel in the video.

Preferred filter allows different stretch filters to be selected for aesthetic and performance reasons. *Point* makes it easiest to spot individual pixels; *bicubic* is generally the filter with the highest perceptual quality. The default display mode does not expose this option; one of the 3D display drivers must be enabled in *Options > Preferences > Display*. Enabling the OpenGL driver allows selection between point and bilinear, while the Direct3D9 driver also allows selection of *bicubic* if the 3D video hardware is powerful enough.

Display pane placement and enable/disable

By default, the input (source) pane is on the left, while the output (filtered) pane is on the right. This ordering can be swapped through the *Options > Swap input/output panes* menu option. A vertical formation can be used in place of the horizontal one by enabling *Options > Vertical display*.

The *Options > Display input video* and *Options > Display output video* menu commands temporarily disable updating of either of the display panes, which can be advantageous for performance. In particular, if the video filter chain contains filters which are expensive to run, disabling the output pane can significantly speed up navigation through the timeline.

Auto-fallback and auto-sparse

Whenever VirtualDub loses focus to another application, it automatically disables accelerated preview for its display panes, falling back to Windows GDI for video display, in order to release system resources. This results in a blockier display if VirtualDub needs to redraw itself. Also, display updates are slower when GDI is in use, although this is not ordinarily a problem unless a preview is running.

During a render-to-disk operation, VirtualDub intentionally sparses display updates so that

they do not occur faster than once per processed frame, or every half second, whichever is less frequent. This keeps the display updates from consuming a significant amount of CPU power. As a result, unless the video frame is very large and the processing rate relatively low, it does not usually make much difference in render times whether the panes are enabled or not.

Interlaced display

Selecting a field mode from *Options > Preview field mode* in the menu switches the display mode for preview from frames to fields. This causes each frame to be updated twice on screen, one field at a time, at twice the frame rate. Displaying a field at a time more closely mimics the way that analog video is actually displayed and delivers smoother motion for video that was originally delivered in analog format.

Field order A causes the top field (even scanlines) to be displayed first. *Field order B* displays the bottom field (odd scanlines) to be displayed first. It is more common for capture devices to use field order B. Also, for interlaced display to be effective, the video must be captured at full frame rate; capturing at less than full frame rate will result in an uneven sampling of fields in time.

No adaptive deinterlacing is attempted during interlaced display, so a thin comb of scanlines will be visible in scenes with fast motion. Also, each field is scaled independently, so some artifacts will be seen when stretching the display panes vertically, and bilinear filtering will not be effective along that axis.

Tearing

During times of smooth motion, it may be possible to see a momentary rip in the video, where the top and bottom portions of the video do not update together. This happens when VirtualDub attempts to update the displayed frame while the screen itself is updating that portion, and is known as *tearing*. This effect is purely a display effect and does not imply a problem with the source video or any rendered output. It also only happens on video that is moving or changing; it is not visible on a static frame.

Tearing can be mitigated or avoided under *Options > Preferences > Displays > Avoid tearing (enable vsync)*. This will cause VirtualDub to schedule display updates to be non-visible if possible. However, it only works with the DirectX (DirectDraw) and Direct3D display minidrivers, and can consume additional CPU when enabled. The anti-tearing code is automatically disabled during video capture and during renders to avoid consuming CPU during these lengthy operations.

VirtualDub help - Processing: Frameserver

The *frameserver* feature allows other programs to pull video directly from VirtualDub's rendering pipeline. This allows video to be filtered through VirtualDub and then processed by another application without the need to write an intermediate file to disk.

Starting the frameserver

The frameserver is built into VirtualDub and is accessible through the *File > Start frameserver* menu command. This command starts a frameserver in the system with the currently loaded video and filtering settings. After the frameserver is started, other applications can connect to it.

Not all features in VirtualDub work while the frameserver is active. The following processing features do work:

- Most video filters.
- Video frame rate adjustment.
- Video frame rate conversion.
- Range selection and the edit list.

The following features are disabled when the frameserver is active:

- Video filters that have a lag — most notably the *temporal smoother*. The filter will function, but there will be a delay in the video.
- Video compression.
- Inverse telecine (3:2 pulldown removal).
- Audio compression.
- Audio filters.
- Audio conversion.
- Audio volume adjustment.
- Audio interleaving (most are not applicable).

Video is always sent from the frameserver in 24-bit RGB format for maximum compatibility.

When the frameserver is started, VirtualDub will ask you for a name to use for the new frameserver, as well as to save a *signpost* file. The name is used to distinguish frameservers on the system; the signpost file tells the client driver which frameserver to connect to. The signpost file, which normally uses the `.vdx` filename extension, is used as input to the client application and contains the frameserver name.

Note Although the frameserver name contains the name of the computer, connections across the network are not currently supported.

Connecting a client application through the AVIFile driver

The best way to connect a client to VirtualDub's frameserver is through the *AVIFile driver*. AVIFile is an older programming API in Windows that allows programs to extract audio and video from media files, such as AVI. As it is extensible, VirtualDub ships with an AVIFile driver that extends such programs to read their data directly from the frameserver rather than from a file on disk.

Note It can often be difficult or impossible to tell whether an application uses the AVIFile APIs to read media files, and is thus compatible with this connection method; often the only way to tell is to try. However, this method is very similar to the connection method used by the *Avisynth* scriptable video processor, so applications which are Avisynth compatible will usually work with VirtualDub's frameserver as well.

To install the frameserver, use the `auxsetup` program that comes with VirtualDub. This will copy the driver to Windows' system folder and add entries to the Windows Registry to enable its use in AVIFile. This tool can also be used to remove the driver. On Windows NT/2000/XP, you must be running as a local administrator to install or uninstall the AVIFile driver.

Once the AVIFile driver is installed, AVIFile-based applications should be able to connect to the frameserver simply by opening the `.vdx` file that was created when the frameserver was started. VirtualDub is itself capable of doing this, so opening a second instance is a way to test the frameserver connection.

After an application is connected and has started pulling audio and video data, the activity totals on the frameserver dialog will increment as data is transferred.

Installing the AVIFile driver in proxy mode

Some applications use the AVIFile API are still not able to use the frameserver because they require the .avi filename extension or otherwise prevent non-AVI drivers from being used. In this case, enabling a special mode of the VirtualDub AVIFile driver called *proxy mode* can help. To do this, install the `proxyon.reg` file that comes with VirtualDub; this adds additional entries in the Windows Registry to remap the AVI entry as well. Use the `proxyoff.reg` file to undo this.

Note Proxy mode works by forcing all AVI files through VirtualDub's driver, which then attempts to proxy through all real AVI files to the regular Windows driver. Although attempts have been made to ensure this is as compatible as possible, there may occasionally be applications that do not work properly when proxy mode is enabled. It is recommended that proxy mode be enabled only when required to use the frameserver, and disabled in all other cases.

Once proxy mode has been enabled, renaming the .vdr signpost file to use the .avi filename extension will allow additional applications to function.

As an additional bonus, when proxy mode is enabled, the AVIFile driver will tunnel through any file that begins with the nine characters `#avisynth` through to the *Avisynth* scriptable video processing tool. The additional compatibility unlocked by proxy mode is thus also extended to Avisynth.

VirtualDub help - Video filter reference

2:1 reduction	Shrink video to half-size using a non-overlapping 2x2 kernel.
2:1 reduction (high quality)	Shrink video to half-size using an overlapping 3x3 kernel.
blur	Apply a radius-1 Gaussian blur to video.
blur more	Apply a radius-2 Gaussian blur to video.
box blur	Apply fast approximated blurs to video.
brightness/contrast	Perform fast linear luminance adjustments.
chroma smoother	Re-interpolate color information in an image without affecting brightness.
deinterlace	Split, unsplit, discard, and duplicate fields.
emboss	Apply psuedo-3D edge detection matrices.
field bob	Resamples alternating frames to remove jittering when splitting fields to frames.
field swap	Swaps even and odd fields to fix mistakes.
fill	Fills a rectangle in the video with a solid color.
flip horizontally	Flips video frame horizontally.
flip vertically	Flips video frame vertically.
general convolution	Apply an arbitrary 3x3 filtering matrix.
grayscale	Convert a color video to grayscale.
HSV adjust	Adjust hue, saturation, and value.
invert	Create or undo negatives.
levels	Apply non-linear ramp adjustments.
logo	Plop your very own bug into the frame.
motion blur	Blur video in time to create motion trails.
null transform	Does nothing, but can be used to crop.
resize	Convert video to a different size.
rotate	Rotate video by right angles.
rotate2	Rotate video by arbitrary angles.

sharpen	Make video crisper.
smoother	Adaptively blur a video while trying to preserve edges.
temporal smoother	Adaptively noise-reduce video across frames.
threshold	Convert video to black and white by comparing against a threshold.
TV	Blur or resample video in luma/chroma space.

2:1 reduction

- Shrink video to half-size using a non-overlapping 2x2 kernel.

This filter is equivalent to applying a 2x2 box filter (average 4 pixels) and then doing a point-sampled resize by half. It gives sharper results than *2:1 reduction (high quality)*, at the cost of some sparkling (aliasing). Where better quality is required, the *resize* filter should be used in *precise bicubic* or *Lanczos3* mode instead.

Note This filter is deprecated and will be removed in a later version.

2:1 reduction (high quality)

- Shrink video to half-size using an overlapping 3x3 kernel.

This filter is equivalent to applying a radius-1 Gaussian blur and then doing a point-sampled resize by half. It is marginally better than *2:1 reduction* in avoiding aliasing (sparkling), at the cost of a blurrier output. Where better quality is required, the *resize* filter should be used in *precise bicubic* or *Lanczos3* mode instead.

Note This filter is deprecated and will be removed in a later version.

blur

- Apply a radius-1 Gaussian blur to video.

blur more

- Apply a radius-2 Gaussian blur to video.

box blur

- Apply fast approximated blurs to video.

Box blur is so named because it uses "box" filters -- it averages blocks of pixels together. Two passes gives a triangle filter, three gives a quadratic, and four gives a cubic. The advantage of *box blur* is that it can do very large blurs very quickly, such as radius-50 cubic blur. The main disadvantage is that it cannot do small or intermediate (radius-5.2) blurs.

brightness/contrast

- Perform fast linear luminance adjustments.

Brightness/contrast provides quick and dirty adjustments -- one of the side effects that saturation is affected by contrast, and hues can shift if clamping at white or black occurs. Consider using *levels* or *HSV adjust* for non-subtle adjustments.

chroma smoother

- Re-interpolate color information in an image without affecting brightness.

Video compression algorithms often store color (chroma) information at a lower resolution than grayscale brightness information, since the eye is less sensitive to detail in color; however, some video codecs don't interpolate chroma well on decompression, resulting in blockiness around sharp color transitions. The *chroma smoother* filter attempts to apply a blur solely in the color channels to correct this situation.

Each chroma option in *chroma smoother*'s configuration corresponds to a blur that is tuned for a particular type of chroma subsampling; assuming that the video decoder did not interpolate chroma at all, choosing the correct setting will produce a result approximating bilinear interpolation. Of course, stronger or weaker options can be picked to taste.

deinterlace

- Split, unsplit, discard, and duplicate fields.

This filter applies quick-and-dirty field operations -- discarding one field is a brutal, but quick-and-dirty way to remove combing from the image. For better quality an adaptive deinterlacer should be used instead. The *fold* and *unfold* modes allow you to temporarily split a video into a double-width video, with even fields on the left and odd fields on the right. This essentially allows any filter to be applied on a field basis rather than a frame basis, avoiding blurring between fields which shows up as ghosting in interlaced output.

emboss

- Apply psuedo-3D edge detection matrices.

Emboss gives a 3D-like effect by "lighting" the video from a particular direction. This highlights edges in the image. It can also be used to amplify a video in order to look for low-level noise.

All forms of the *emboss* filter can also be done manually by the [general convolution](#) filter.

field bob

- Resamples alternating frames to remove jittering when splitting fields to frames.

field swap

- Swaps even and odd fields to fix mistakes.

fill

- Fills a rectangle in the video with a solid color.

If you need a translucent (alpha-blended) fill, use the *logo* filter with a solid color as the logo.

flip horizontally

- Flips video frame horizontally.

flip vertically

- Flips video frame vertically.

general convolution

- Apply an arbitrary 3x3 filtering matrix.

A *convolution filter* is a filter that uses a local area of pixels to compute a new pixel. It can be thought of as either taking a weighted sum of an area of source pixels, or adding a weighted area to the output scaled by the source pixel (the two are equivalent). Filters that can be performed using convolution include blurs, sharpens, and edge detectors.

The *general convolution* filter allows a 3x3 filter to be specified. Each value in the filter is specified as a value from a scale to 0-256, where 256 means a value of 1.0 (yes, this is a programmerism). Values can be specified outside this somewhat, including negative numbers, although large values like 10,000 are inadvisable. Each value then specifies either the amount of source to include in the result, or the amount of result to produce from the center source pixel (again, depending on your perspective).

The *clipping* option should almost always be enabled; if it is not, the result will be incorrect when the result of the filter is below black or above white.

Bias is a value to add to the result, again in the range 0-255. It is useful when both positive and negative values need to be visualized, such as error values. It acts much like a *brightness* adjustment.

Some hints regarding design of convolution filters:

- The sum of all of the values in the matrix is the *gain* of the filter, which determines the change in contrast produced by the filter. A sum of 256 results in unity gain, i.e. no change. Higher sums amplify the image and lower sums mute it. A zero sum completely removes the image, and a negative sum inverts it.
- Filter matrices that consist of all positive elements are blurs. For instance, here is a blur matrix:

```
28 28 28
28 32 28
28 28 28
```

- Matrices that aren't symmetrical — those that are lopsided in a direction — will have edge detection effects.
- A bright pixel in the image will take on the shape of the filter matrix.

grayscale

- Convert a color video to grayscale.

The formula used for conversion is $Y = 0.211R + 0.715G + 0.074B$.

HSV adjust

- Adjust hue, saturation, and value.

You can rotate hues, control the strength of color (saturation), or modify brightness (value)

using this filter. *HSV adjust* does not allow value adjustments to affect hue -- if a pixel becomes too bright or dark, its saturation is reduced accordingly to avoid clamping. This prevents orange, for instance, from becoming hot yellow when the image is amplified.

invert

- Create or undo negatives.

levels

- Apply non-linear ramp adjustments.

Levels does contrast, brightness, and gamma adjustments. It actually consists of three steps:

1. Scale input range up to full range. The black level and white levels are the first and third parameters on the top; anything lower than the black level becomes black, and anything above the white level becomes white.
2. Apply gamma correction (middle top parameter). You can either type in the gamma correction power, or you can move the slider, which controls which level in the source image becomes middle gray (50%). Black and white always stay black and white through this correction step.
3. Scale to output range (bottom parameters) -- the image levels are finally scaled down to the black and white levels indicated here.

You also have the option of working in luma (Y) space. This causes *levels* to work in brightness (luma) only, leaving color information alone. This preserves hue and saturation in the image, except for possible clipping at black and white.

Note *Levels* always works with full scale levels -- black and white are 0 and 255, not 16 and 235 as in YCbCr space.

logo

- Plop your very own bug into the frame.

Enable per-pixel blending using the alpha channel

An image can be augmented using an additional monochrome map called an *alpha channel*. This map contains a single value per pixel indicating how opaque or translucent that pixel should be; it allows logos to be created that have shaded areas. Most photographic image editing packages have options to edit an alpha, mask, or opacity layer on an image for this purpose. For use with this filter, it is best to save such images in TARGA (.tga) format with 32-bit pixels.

Use premultiplied alpha

Affects the interpretation of the alpha channel. Normally alpha channels simply control blending of an image, but when creating an image from scratch that has soft edges, creating such an alpha channel may be difficult. Premultiplied alpha refers to a method of blending where the image itself is expected to be pre-darkened where it is not fully opaque, to the point of being black where alpha indicates transparent areas. In a painting package, you may find this more convenient to create, as you can simply draw on top of "transparent black."

Use gray channel of secondary image for alpha channel

When it is inconvenient to create a single image with an alpha channel, enabling this option will allow the alpha to be supplied using the brightness of a second image. This image should have the same size as the main logo image, although it need not be of

the same format.

Opacity

Global translucency control; 0% will hide the logo, 100% shows the logo at full opacity, and intermediate values will fade it out somewhat. When per-pixel blending is enabled, this option will scale down all of the alpha values.

Justification, X/Y offset

Specifies the position of the image. If justification is something other than TL (top-left), the positioning will be relative to another point within the image, which is useful for consistently placing logos in the bottom-right corner when working with videos of different sizes.

motion blur

- Blur video in time to create motion trails.

null transform

- Does nothing, but can be used to crop.

resize

- Convert video to a different size.

When resampling video from one size to another, pixels in the new frame size don't necessarily map exactly to pixels in the source. A *resampling filter* is used to compute the intermediate pixels from the pixels in the source. VirtualDub's resize filter gives you a number of choices for the resampling filter:

Nearest neighbor (point sampling)

Choose the nearest source pixel. This results in the crispest video, but has sparkling and "chunkiness" problems. It is the fastest resampling mode and is useful for previews.

Bilinear (triangle interpolation filter)

Compute the desired pixel by linearly averaging the closest four source pixels. This gives a considerably better result than nearest neighbor, but results in a lot of blurring and gives diamond-shaped artifacts when enlarging. This is the resampling mode for most 3D texture mappers and interpolating video hardware overlays. High-ratio shrink operations (< ~60% or so) will give aliasing with this mode and for those the *precise bilinear* mode should be used instead.

Bicubic (cubic spline interpolation filter)

Compute the desired pixel by fitting cubic spline curves to the closest 16 source pixels. This gives a sharper result than the bilinear filter, although when enlarging it results in a slight halo (ringing) around edges. High-ratio shrink operations (< ~60% or so) will give aliasing with this mode and for those the *precise bicubic* mode should be used instead.

Precise bilinear (triangle decimation filter)

Compute the desired pixel by applying a triangle filter to the closest N source pixels, where N=4 for enlarging and N>4 for shrinking. This mode is the same as *bilinear* for enlargement but gives better results when shrinking.

Precise bicubic (cubic spline decimation filter)

Compute the desired pixel by applying a triangle filter to the closest N source pixels, where $N=16$ for enlarging and $N>16$ for shrinking. This mode is the same as *bicubic* for enlargement but gives better results when shrinking. Three different modes are given, $A=-1.0$, $A=-.75$, and $A=-0.6$. These vary the "stiffness" of the cubic spline and control the peaking of the filter, which perceptually alters the sharpness of the output. $A=-0.6$ gives the most consistent results mathematically, but the other modes may produce more visually pleasing results.

Lanczos3 (three-lobed decimation filter)

Compute the desired pixel by applying a three-lobed sinc filter to the closest N source pixels, where $N=64$ for enlarging and $N>64$ for shrinking. This produces slightly better results than the *precise bicubic* mode, at the expense of slower speed and more haloing (ringing). However, for a single pass the difference is very small and you should consider using *precise bicubic* instead.

The *resize* filter uses OpenGL conventions for pixel mapping, mapping the entire source texture onto the entire destination rect. Areas in the image are proportionally enlarged or shrunk with the picture frame: enlarging the frame 2x results in all features in the image being twice as large. This means that high enlargement ratios will result in some duplicated pixels around the border of the destination image (U/V clamping).

Note The *new width* and *new height* fields do not have to be integers — they can be fractional. VirtualDub will resize the video to that fractional size and then extend the borders to hit the next highest integer size in pixels. This means that exact aspect ratios can be maintained in the image content of a video even though the frame size is slightly off due to integral rounding.

rotate

- Rotate video by right angles.

rotate2

- Rotate video by arbitrary angles.

While *rotate2* can be used for right-angle rotations, that job should be left to *rotate*, which can do them faster and more accurately.

rotate2 is equivalent to a 1:1 texture-mapped quad, rotated around the center point of the quad by the designated angle, with linear or cubic spline interpolation filters applied to the texture mapping.

sharpen

- Make video crisper.

Sharpen cannot distinguish between artifacts, noise, and edges -- so sharpening edges with this filter will also make blocking and noise worse. Thus, there is a limit to how much video can be improved with sharpening, particularly highly-compressed or noisy video.

smoother

- Adaptively blur a video while trying to preserve edges.

temporal smoother

- Adaptively noise-reduce video across frames.

A *temporal* filter is one that works along time -- *temporal smoother* doesn't compare pixels within a frame, but across a 7-frame window. It is effective at reducing noise, but for best effect, it should be combined with a filter that works within a frame (spatial filter). Raising the power of the filter increases the amount of noise reduction but can also result in smearing of details when motion occurs as well as sparkling during transitions. It works best with a setting of 3-5.

threshold

- Convert video to black and white by comparing against a threshold.

TV

- Blur or resample video in luma/chroma space.

Due to the way that analog video encoding works, the color information in the video (chroma) typically has lower resolution and more noise than the brightness information (luma). This problem is especially severe with videotape. The *TV* filter can reduce chroma noise by averaging only the chroma, while leaving the luma alone. This can reduce color flickering in a video, but should be used carefully as too much chroma blurring will lead to bleeding.

VirtualDub help - Video display shader support

VirtualDub has a special video display driver called the *D3D FX driver* that allows the power of a 3D accelerator to be unleashed when displaying video. This is only for display purposes — the result can't be saved to disk — but it is useful for improving display quality as well as experimenting with different image processing algorithms. The power and ease of programming for modern GPUs makes it possible to prototype a shader for an algorithm in minutes that runs in real-time.

Requirements and enabling the driver

A 3D accelerator with hardware support for vertex and pixel shaders is required to use the Direct3D FX display driver in VirtualDub. A video card which supports at least pixel shader 2.0 is highly recommended, as pixel shader models 1.1-1.4 are very restricted in the amount of computation and texture fetches that can be performed in a single pass.

In addition, you must have `d3dx9_25.dll`, the D3DX DLL from the DirectX 9.0c April 2005 release, installed. This DLL does not come with VirtualDub and must be installed from the DirectX redistributable; currently it is *not* installed by the standard DirectX 9.0c install on Windows Update. As of September 19, 2005, the April 2005 D3DX dll redistributable is available as `directx_9c_Apr05sdk_redist.exe` at the following URL:

<http://www.microsoft.com/downloads/details.aspx?FamilyId=402111C4-6478-46C1-BBD9-1BF9FCCE52F4&displaylang=en>

Note that `d3dx9_25.dll` is a system DLL — it is intended to be installed only by the DirectX Setup installer into the `Windows\System32` directory.

To enable the FX driver, go to *Options > Preferences > Display*. DirectX support, Direct3D support, and the FX driver should all be enabled. The filename of an `.fx` file must be supplied

to use; if a full path is not given, the file is assumed to come from VirtualDub's program directory.

For detailed documentation on the `.fx` file format, consult the Microsoft DirectX 9.0c SDK.

Available surfaces and textures

A total of three textures and one surface are available for use:

- The output render target, sized to the output frame.
- The source texture, which holds the video frame.
- Two temporary render target textures, which are at least as large as the desktop.

Declaring variables with specific names will automatically cause them to be bound to the textures:

```
texture vd_srctexture;
texture vd temptexture;
texture vd temptexture2;
```

The output render target is at least X8R8G8B8. The temporary render targets are of format A8R8G8B8; in addition, they are guaranteed to be powers of two as long as the device does not support full non-power-of-two textures; in particular, the restrictions of the NONPOW2CONDITIONAL caps bit do not have to be followed when using them.

Technique selection and execution

A technique must be named either *point*, *bilinear*, or *bicubic* for it to be used. Each of these names maps to one of the filtering modes in the right-click context menu of the video pane; this allows up to three techniques to be selected from the `.fx` file. If a technique is not available, a nearby available technique from the three is used instead.

When a video frame is displayed, VirtualDub sequentially executes each of the passes in the file. If interlaced display mode is enabled, the technique is executed twice per frame, after each field is updated. This allows field-savvy shaders to do adaptive deinterlacing on the video input.

Each pass is executed with a quad (four vertex rectangular mesh). The components of the vertex declaration accessible from the vertex shader are (all are two component):

- *POSITION*: The four corners of the output viewport. These are already corrected for the Direct3D half-pixel offset so that they exactly encompass the screen.
- *TEXCOORD0*: The four corners of the source image subrect within the source texture.
- *TEXCOORD1*: Full-texture analogs of the first texcoord set — (0,0), (0,1), (1,0), and (1,1).

Thus, passing POSITION and TEXCOORD0 through is enough to do a straight blit using point or bilinear sampling.

Pass annotations

Passes within a rendering technique can be annotated to instruct VirtualDub to take certain actions prior to executing that pass.

```
float4 vd_clear = { red, green, blue, alpha };
```

Clears the render target to the given color. All channel values are normalized and range from 0-1.

```
string vd_target = "temp";
string vd_target = "temp2";
```

Selects the render target texture to use for rendering. If this is not specified, the output render target is used instead. The viewport is automatically set to the entire texture.

This is an example of a pass with annotations:

```
pass p0
<
    string vd_target = "temp";
    float4 vd_clear = { 0, 0, 0, 0 };
>
{
    VertexShader = compile vs_2_0 VS();
    PixelShader = compile ps_2_0 PS();
}
```

Exported variables

Declaring global variables with specific names will cause those variables to automatically be filled in by VirtualDub with data useful for rendering. These values are constant throughout the technique execution, as the source values do not change between passes.

Here are the supported variables:

```
float4 vd_vpsize;
```

Viewport size and inverse viewport size. This is the size of the output window, in pixels. `vd_vpsize.xy` gives the width and height of the viewport; `vd_vpsize.wz` gives 1/width and 1/height.

```
float4 vd_texsize;
```

Texture size and inverse texture size for the source texture. This is the size of the input source image *texture*, in texels (it is *not* the source image size). `vd_texsize.xy` gives the width and height of the texture; `vd_texsize.wz` gives 1/width and 1/height. This is useful for computing UV coordinates.

```
float4 vd_srcsize;
```

Size and inverse size of the source image. This is the size of the video frame, which is in the top-left subrect of the source texture. `vd_srcsize.xy` gives the size of the video frame, and `vd_srcsize.wz` gives 1/width and 1/height.

```
float4 vd_tempsize;
```

```
float4 vd_temp2size;
```

Texture size and inverse texture size for the primary and secondary render target textures, respectively.

```
float4 vd_vpcorrect;
```

Screen space mapping and correction for the viewport. `pos * vd_vpcorrect.xy` `vd_vpcorrect.wz` transforms `pos` from screen space coordinates, where (0,0) is the bottom-left corner and `vd_vpsize.xy` is the top-right corner, to the correct normalized display coordinates (NDC) for the output render target.

The annoying half-pixel shift imposed by Direct3D is taken care of in the translation — the screen space coordinates established by this transform use OpenGL-style pixel placement, with pixel centers at half-integers.

```
float4 vd_vpcorrect2;
```

This is the same as `vd_vpcorrect`, except that the coordinate mapping is inverted so that (0,0) is the top-left.

```
float4 vd_tvpcorrect;
float4 vd_tvpcorrect2;
float4 vd_t2vpcorrect;
float4 vd_t2vpcorrect2;
```

These are the analogous screen space mappings for the primary and secondary render target textures, respectively.

Texture shaders

D3DX texture shaders can be used to precalculate a texture prior to video display. These shaders are only run once when the `.fx` file is loaded, and the result then reused for each video frame. As they run on the CPU, they can use shader features not necessarily supported by the GPU, and are thus useful for computing lookup tables to work around the feeble calculation abilities of lower shader models.

VirtualDub uses a set of annotations on the texture object to control texture shader execution. These annotations are compatible with those used by the EffectEdit tool. The generated texture is always of the format A8R8G8B8.

```
float width;
float height;
```

Sets the width and height of the texture. These are not adjusted to device requirements, so it is safest to use powers of two.

```
string target;
```

The profile to use when compiling the texture shader. It defaults to `tx_1_0` if absent.

```
string function;
```

The name of the HLSL function to use for the texture shader.

Here is an example of a texture shader in use:

```
texture proceduraltex
<
    string function = "gen";
    int width = 16;
    int height = 256;
>;

float4 gen(float2 texcoord : POSITION,
          float2 texelSize : PSIZE) : color0
{
    return texcoord.xyxy;
}
```

Caveats

VirtualDub does not (cannot) do emulation of pixel shader models that are not supported by the GPU. This is particularly important for older video cards only supporting pixel shader 1.1-1.4, which are much more restrictive than pixel shader 2.0 or 3.0. Some restrictions to watch out for:

- ps_1_x does not have a way to normalize a vector or retrieve its length.
- ps_1_1 through ps_1_3 can't do dependant texture reads except through the special texture addressing operations, which can only be accessed from shader assembly. A dependant texture read is a texture read based on a calculated texture coordinate, i.e. not from a texture coordinate set.
- ps_1_4 can do dependant texture reads, but only one layer deep, and with limited precision.

This is not to say that interesting and useful display shaders can't be written in downlevel shader models, but it requires some careful thought and coding. ps_1_4 is pretty rare as it was only introduced in the ATI RADEON 8xxx series; ps_1_1 was introduced with the NVIDIA GeForce 3 and ps_1_2/ps_1_3 with the GeForce 4, however. All cards with given shader model support can run all downlevel shaders, however, so a GeForce FX 5800 can run ps_1_4 shaders even though its highest supported model is ps_2_a.

VirtualDub help - Video capture

Video capture is the process of taking video from an external hardware source, such as analog video, and digitizing it into a form that can be used on a computer. VirtualDub allows use of several classes of video capture hardware to do this conversion, depositing the result as an .avi file on disk.

What you need to use capture mode

In order to use VirtualDub's video capture mode, you need:

- A video capture device.
- An external video source.

Video capture mode cannot be used to capture streaming video over a network or graphics on your own local machine (unless you loop-back TV out into a video capture device).

VirtualDub help - Capture: Hardware setup

Here are some of the connectors you should have on your system:

- *Capture card coaxial input*: A round, threaded connector with a small hole in the middle that accepts modulated signals with audio (multiple channels).
- *Capture card composite input*: A smooth, non-threaded (RCA) connector with a large hole in the middle. This connector accepts a single video input without audio. The composite video cable, or the composite video portion of an octopus cable, usually has a yellow connector.
- *Capture card S-Video input*: A rounded socket with four pin holes and a small rectangular slot (DIN-4). This connector accepts a single video input without audio, but with brightness and color information separated. The result is higher video quality than composite input. S-Video cables usually have a black connector.
- *Capture card line input*: A mini-mono or mini-stereo socket about 1/8" in size, this is the audio input for the composite video and S-Video inputs.
- *Capture card line output*: Also a mini-phone socket about 1/8" in size, this is the audio output from the capture card.
- *Sound card line input*: A third mini-phone socket, this is the input to the sound card for recording. On sound cards with color-coded inputs, this should be blue in color.

If you have an all-in-one style capture device that does both video and audio, especially one over USB, then hooking it up is a no-brainer: just connect everything to the capture device. You don't have a choice anyway. If you have a "TV tuner" style device, though, then some of the capture work is also being done by the sound card, and you have to hook up a couple of cables to get everything working.

Capturing from cable (coaxial input)

In this scenario, you have a TV tuner type capture card and want to capture from a TV channel.

- Connect the round coaxial cable to the capture card.
- Connect the line-out from the capture card to the sound card line-in.

The TV tuner card accepts the cable input, selects and decodes the desired channel, captures the video, and splits off the audio for your sound card to capture.

Capturing from audio/video outputs

In this scenario, you have a TV tuner type capture card and want to capture from another device that has separate audio and video outputs, such as a VCR or video game console.

- Connect the composite video or S-Video output to the same video input on the capture card.
- Connect the audio output from the output device to the capture card line-in.
- Connect the line-out from the capture card to the sound card line-in.

The problem you will often encounter here is that the output device will have a pair of round RCA connectors, one for each of the left/right stereo channels (red and white), while the capture and sound cards will have 1/8" mini-stereo inputs. A cable with a male 1/8" on one end with stereo RCA male connectors on the other end, along with a pair of RCA female-female adaptors, will help you hook everything up here.

This assumes that you have a capture card that has integrated audio capture. If your capture card simply passes through the audio, it's better just to connect the audio output directly to the sound card:

- Connect the composite video or S-Video output to the same video input on the capture card.
- Connect the audio output from the output device directly to the sound card line-in.

This shortens the audio path for better quality, and you won't have to worry about whether the capture card's audio mixer is set to the right level or the correct input.

VirtualDub help - Capture: Driver types and driver selection

In order to use a video capture hardware device, you need a *capture driver* to interface VirtualDub to the hardware. This software should come with your hardware device and should be installed in Windows as part of the software package that came with it. If you are missing the driver software, check the vendor's website or contact the vendor for a replacement as otherwise you will not be able to use the hardware with VirtualDub.

All capture devices detected by VirtualDub are displayed at the end of the *Devices* menu. Selecting a driver entry causes the current capture driver to be stopped and the new capture driver to be started.

There are two types of video capture drivers in Windows, which behave differently in their capabilities. It is important to know the type of video capture driver you have installed as the distinction affects which features of VirtualDub's capture mode can be used. VirtualDub can use drivers that are written to either driver model.

Video for Windows (VFW) capture drivers

Video for Windows is the original video API for Windows, and is the older driver type of the two. These drivers can sometimes be found for versions of Windows as old as Windows 95, and it is even possible to use a capture driver written for Windows 3.1 (which is very much not recommended). These drivers have several limitations:

- Audio capture is always done through the sound card, or a driver that makes the capture device's audio path look like a sound card.
- VFW-model drivers expose few settings programmatically; settings such as audio control, channel selection, video standard, video image settings, etc. are only exposed through the driver's own dialogs. These are accessible through the *Video display*, *Video*

format, and *Video settings* commands in the Video menu. Unfortunately, this means that VirtualDub cannot save or restore these settings, or control them directly.

One advantage of VFW-model drivers is that they are often mature and well-debugged, and thus reliable. Another is that VFW drivers are supported by both the VFW and DirectShow capture APIs in Windows, making them usable across a wide variety of capture applications.

VFW drivers can be found for all versions of Windows, from Windows 95 to Windows XP. However, starting with Windows 2000 more manufacturers have started shipping WDM-model drivers instead. A few video capture devices have support for both.

A VFW-model driver can be identified by the (VFW) tag after the entry in the *Device* menu.

Windows Driver Model (WDM) capture drivers (DirectShow-based)

The newer type of video capture driver in Windows uses the Windows Driver Model (WDM), which was introduced in Windows 98 and 2000. The Microsoft DirectShow API is the primary API to use these drivers. Because the DirectShow API supports a larger variety of commands and settings than VFW, the functionality set of a WDM driver is significantly improved:

- WDM devices can expose integrated audio capture directly.
- Nearly all settings exposed in UI, such as video saturation, are also exposed programmatically for unattended control.
- If available, VirtualDub can directly control the TV tuner channel, as well as audio/video source selection.

DirectShow is a much more complex API than VFW, however, and WDM-model drivers historically have been a lot less stable than their VFW counterparts. It is not unusual to see problems such as capture applications that cannot be closed, because their program execution is stuck in the capture driver. WDM is the proscribed driver model going forward, however, so the situation should improve over time.

A WDM driver that requires use of DirectShow will have a (DirectShow) tag besides its entry in the *Device* menu.

The *Video display*, *Video format*, and *Video settings* dialogs will not normally be available when DirectShow is in use. The settings in those dialogs are usually available on the driver's *Capture pin* and *Capture filter* dialogs instead. The exception is if a Video for Windows (VFW) driver is being used through DirectShow; these drivers are indicated by (VFW > Directshow) next to their name in the *Device* menu. Using a driver in this manner piles the limitations of DirectShow on top of the limitations of VFW, so it is usually better to use those drivers directly with VFW instead.

The Microsoft WDM Image Capture (Win32) driver

If you have a Windows Driver Model (WDM) driver installed, you may also have an entry in the device list called Microsoft WDM Image Capture (Win32) (VFW). This entry comes from a Microsoft driver called VFWWDM32 and is a wrapper that allows WDM-model drivers to be used through the older Video for Windows (VFW) API. The WDM driver that is adapted can be selected through the *Video Source* driver dialog.

There are unfortunately some quirks in the way this adapter works, and some video capture devices will work erratically or not at all through this wrapper. Device settings not accessible through VFW will also still not be available when using it. If possible, use the capture device directly in DirectShow mode rather than using the VFWWDM32 driver.

Custom drivers

A few manufacturers had difficulty migrating their capture drivers from the relatively lenient Windows 95/98/ME versions to the newer, stricter versions of Windows based on the NT kernel. Instead of shipping a proper VFW or WDM driver — both of which are possible on NT/2000/XP — these manufacturers shipped a custom driver along with a custom, hardware-specific capture application. These capture devices are unfortunately not usable with VirtualDub because they use a proprietary programming interface (API).

The video emulation driver

The *Video emulation* capture driver is not a real capture driver installed in Windows, but rather an internal video capture minidriver in VirtualDub used for testing. It accepts a video file, selected using the `video source` menu command, and plays that video, pretending to be a live video source pushing audio and video data from that file. Although primarily useful for VirtualDub program development, it is also sometimes useful for diagnosing compatibility problems and determining if problems lie in VirtualDub or in a video capture driver.

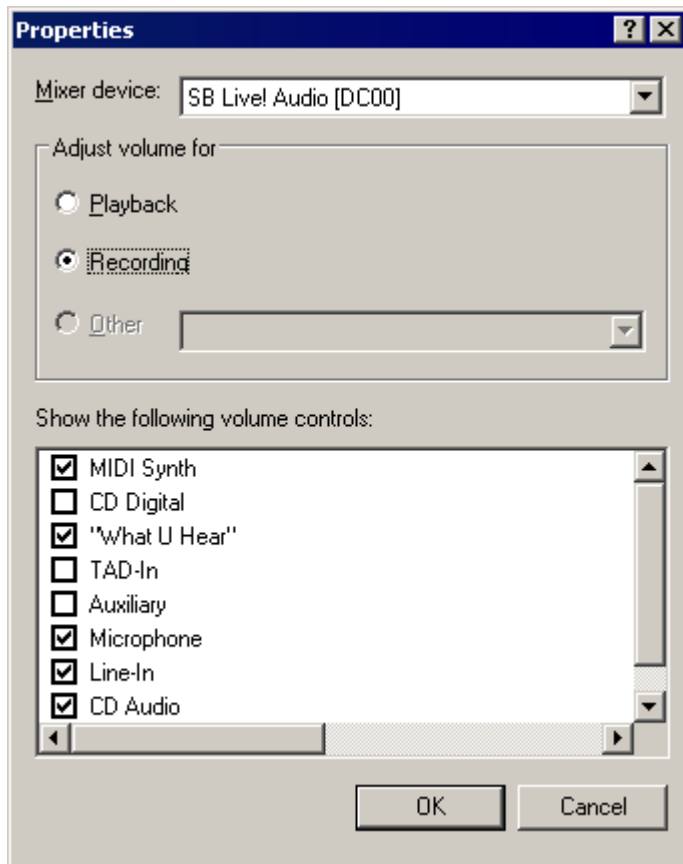
Note that additional load is placed on the hard disk and on the CPU by the video decoding, so this driver isn't necessarily useful for performance testing.

VirtualDub help - Capture: Audio setup

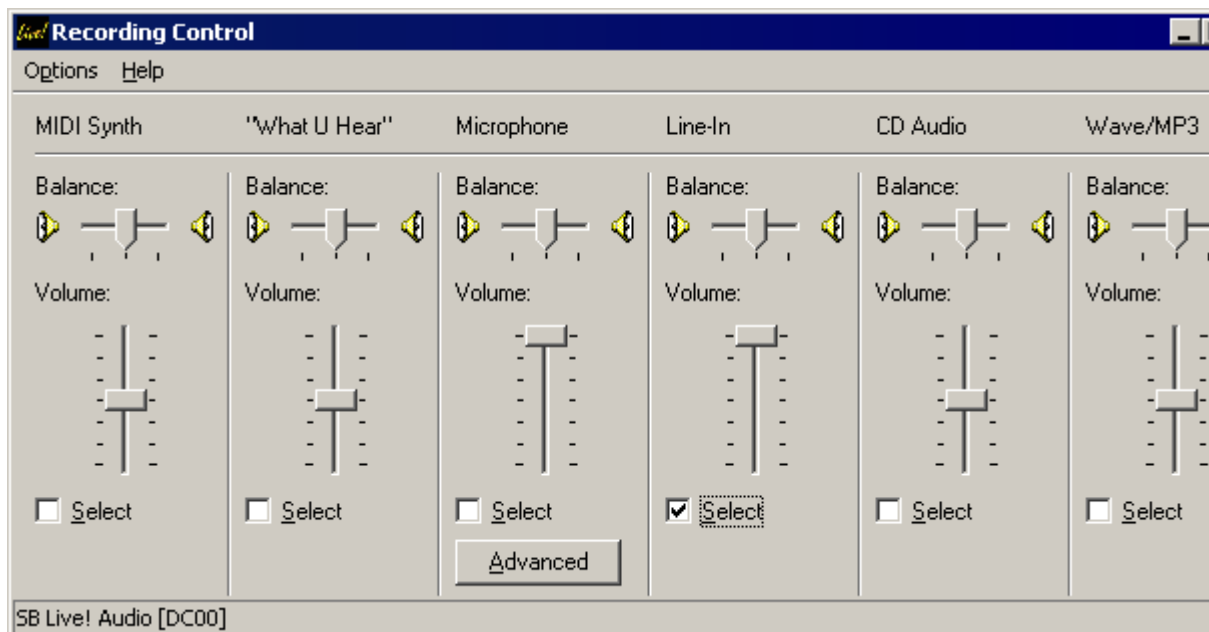
Using Windows Volume Control to adjust recording levels

If your video capture device relies on the installed sound card to capture audio, you will need to adjust the recording levels on the sound card through the Volume Control tool in Windows. Double-click on the speaker icon in the notification area of the taskbar, or in Windows XP, launch it through *Start > Programs > Accessories > Entertainment > Volume Control*. You can also launch it from VirtualDub using the *Audio > Windows mixer...* menu command.

Once Volume Control is open, select *Options > Properties* from the menu:



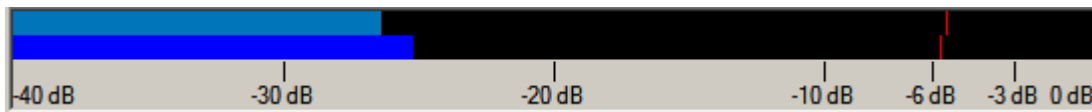
The display on your system will differ slightly. Change the current control set from Playback to Recording, and make sure the appropriate input is checked in the list of controls to show, depending on which sound card input the capture device's audio output is plugged into (usually Line-In or Aux-In).



Check that the correct inputs are selected for recording. If not, first check if it can be changed in VirtualDub using the *Audio > Audio input* submenu; the Volume Control should automatically update as VirtualDub changes inputs. Otherwise, the correct input should be manually selected. Now use the volume slider for the input to adjust the volume of the captured audio.

Checking levels with the volume meter

To check volume levels, display the volume meter in VirtualDub by selecting *Audio > Volume meter* from the menu, or press the *V* key.

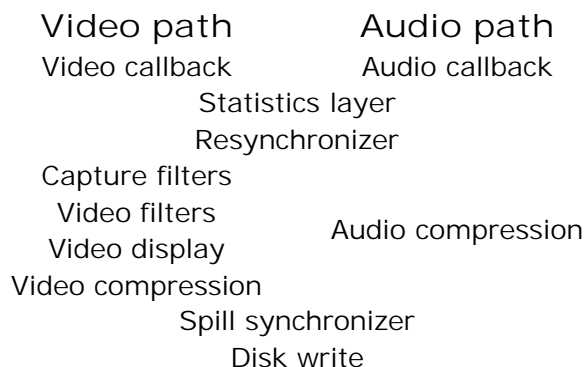


The volume meter samples incoming audio from the recording input and displays the current and long-term peaks in blue and red, respectively. It measures sound power in decibels (dB); -6 dB is one-quarter maximum power, or one-half maximum amplitude. Lower peak values mean quieter audio.

Audio should be adjusted so that it is reasonably loud without clipping (bars go all the way to the right end) — once this happens audio is distorted. It is usually safer for the audio to be on the quieter side, although if it is really quiet quality will be lost due to limited precision, even after amplification in post-processing.

VirtualDub help - Capture: Pipeline

Like during a render-to-disk, captured audio and video flows through a series of processing stages before written to disk. This is what VirtualDub's capture pipeline looks like:



Video/audio callback

This is the entry point at which the capture API (VFW/DirectShow) notifies VirtualDub that audio or video data has been captured.

Statistics layer

Most of the statistics in the information sidebar are collected at this point. Additional information not available from the capture layer, such as the timestamp of audio capture, is added at this point.

Resynchronizer

Audio/video resynchronization occurs at this point, both adjusting the video timing and resampling audio. The exact operations performed here are controlled by the settings in the [Timing dialog](#).

Capture video filters

Capture-specific video filters such as field swap, 2:1 vertical reduction, and level compression occur here. Unlike the regular video filters, capture filters run directly in the capture format, either YCbCr or RGB — if the capture filter can't run in the current format, an error is displayed instead of a conversion being performed.

Video filters

Any standard video filters execute now; the video is converted to 32-bit RGB before entering the video filter chain. If the video filter chain is not enabled, the conversion to 32-bit RGB is skipped.

Video display

If the display mode is set to Preview, the video stream is tapped off at this point for display purposes.

Video/audio compression

The selected video and audio compression codecs now apply data compression.

Spill synchronizer

If spill mode is enabled for multi-file capture, the spill synchronization code now determines which file the audio and video streams write into, to ensure that each file is cleanly cut with the same audio and video durations.

Video/audio write, disk write

Audio and video data is buffered and eventually written to disk.

VirtualDub help - Capture: Filtering

VirtualDub allows the incoming video to be filtered before it is compressed and written to disk. This requires a lot of CPU power to do reliably, but correcting, shrinking, and cleaning up the video before it is compressed can result in higher quality captures without requiring an additional post-process pass.

All capture filters are accessible through the *Video* menu. If the video display mode is set to Preview *and* preview acceleration is enabled, the post-filtered result can be seen on the preview display.

Capture format compatibility

Unlike the normal filter system, the capture filter system does not attempt to automatically convert video filters to accommodate filter stages. Care must be taken to choose a video format which is compatible with all of the filters you wish to use:

Format	Crop	Swap fields	Noise reduction	Vertical reduction	Extend luma	Filter chain
15-bit RGB	Yes	Yes				Yes
16-bit RGB	Yes	Yes				Yes
24-bit RGB	Yes	Yes	Yes	Yes		Yes
32-bit RGB	Yes	Yes	Yes	Yes		Yes
UYVY	Yes	Yes	Yes	Yes	Yes	Yes
YUY2	Yes	Yes	Yes	Yes	Yes	Yes
YV16	Yes	Yes		Yes	Yes	Yes
YV12	Yes			Yes	Yes	Yes
I420	Yes			Yes	Yes	Yes
IYUV	Yes			Yes	Yes	Yes

YVU9	Yes		Yes	Yes	Yes
Y41P	Yes	Yes	Yes		Yes
Y8	Yes	Yes	Yes		Yes

If a capture video filter is not compatible with the input format, it is automatically disabled. Note that compressed video formats are not supported at all — if the capture device outputs compressed data, no video filtering is possible at all.

Detailed description of capture filters

Cropping

Discards borders of the video, resulting in a smaller output video. If the video is letterboxed, applying cropping can improve performance as well as reduce output file sizes. It is nearly free if vertical reduction or RGB filtering is enabled.

The video format imposes restrictions on the alignment of the cropping boundaries. RGB formats can be cropped to pixel accuracy; YUY2/UYVY can only be cropped to the nearest even pixel boundary horizontally, and YV12/I420 to the nearest 2x2 pixel block.

Swap fields

Swaps even and odd scanlines within the image to correct for a video capture device that is assembling fields in reverse order. This affliction looks like a "raspy" effect in the image. Field swapping is a relatively fast operation.

Noise reduction

Applies a quick, first-order recursive filter to reduce the amount of noise within the image. This is a primitive noise reduction filter, but can be effective, particularly for light noise or videos with low motion. The amount of noise reduction is adjustable; higher thresholds produce more noise reduction at the cost of higher motion artifacts.

Vertical reduction

Reduces the vertical height of the captured video by one-half using either bilinear or bicubic resampling. This is useful if the additional vertical resolution is not required, but better image quality is desired than would be obtained by using the capture device's scaler, which often handles such a request by simply throwing away one of the fields (every other scanline). This option consumes a decent amount of CPU power and should be used with care; however, it can also significantly speed up subsequent video compression due to the smaller output image.

Note that vertical reduction causes fields to be blended together, which can cause ghosting artifacts due to motion within the image.

Extend luma black point

Compresses the luminance range of an image so that "superblack" levels from 0-15 are scaled to within the valid luminance range of 16-235, out of 0-255. These levels are ordinarily shown as red at the lower end of the histogram scale and would be clamped to black without this option. This is only useful if the capture format uses the YCbCr color space, as these values are already lost if RGB is used.

The transform is a straightforward linear mapping of the luminance values from [0,

235] to [16, 235], if *extend luma white point* is disabled, or [0, 255] to [16, 235] if that option is also enabled.

Extend luma white point

Compresses the luminance range of an image so that "superwhite" levels from 0-15 are scaled to within the valid luminance range of 16-235, out of 0-255. These levels are ordinarily shown as red at the upper end of the histogram scale and would be clamped to white without this option. This is only useful if the capture format uses the YCbCr color space, as these values are already lost if RGB is used. However, it can allow for recovery of very bright colors that would ordinarily white-out or become garish yellows due to signal overamplification.

The transform is a straightforward linear mapping of the luminance values from [16, 255] to [16, 235], if *extend luma black point* is disabled, or [0, 255] to [16, 235] if that option is also enabled.

Filters / Enable RGB Filtering

Pushes video through the normal VirtualDub video processing filter chain. Nearly all video filters can be used here; however, the *temporal smoother* should be avoided as it has a frame lag, which is not supported in capture mode.

Using normal video filters will force the video to be converted to 32-bit RGB, and consumes a significant amount of CPU power.

VirtualDub help - Capture: Information panel

Frames captured	122
Total time	0:05
Time left	6:22
Total file size	80.5MB
Disk space free	5.97GB
CPU usage	25%
Spill status	writing #1
Video	
Size	79.8MB
Average rate	24.02701 fps
Data rate	16219KB/s
Compression ratio	0.7:1
Avg frame size	691200
Frames dropped	0
Frames inserted	0
Resample	1.00174x
Audio	
Size	789KB
Average rate	48058.47Hz
Relative rate	48041.11Hz
Data rate	162KB/s
Compression ratio	1.0:1
Resample	+0.005 s.t.
Sync	
VT adjust	+0 ms
Relative latency	-706 ms
Current latency	-3 ms

The information panel shows current disk, video, and audio status during a video capture. It is toggled through the *Option > Show information panel* menu command. Not all entries are always shown; the subset that is displayed can be changed in Preferences.

Frames captured

The total number of video frames captured.

Total time

The amount of time the capture has been running, in days:hours:minutes:seconds.

Time left

The estimated amount of time the capture can continue, based on available disk space, video frame rate, and current compression ratios.

Total file size

The total amount of data written to disk in the current capture session.

Disk space free

How much disk space is left on the capture drive(s).

CPU usage

Estimated CPU utilization during the video capture. This includes CPU usage by processes other than VirtualDub.

Note On a system with multiple logical CPUs (SMP, dual core, or hyperthreading), this value may rise above 100%, up to 100% times the number of CPUs. Video capture is mostly single-threaded, so approaching 100% in these situations may still indicate CPU overload.

Spill status

Current status of a multi-segment capture operation. Normally this will indicate which segment number is currently being written; it will also indicate when the capture engine is in the process of spilling over from one segment to the next. If the spill takes a long time or never completes, this can prevent VirtualDub from switching files and may be indicative of a bad audio/video timing problem.

Video: Size

Total amount of video data written to disk.

Video: Average rate

The overall rate at which video frames are arriving from the video capture device. The more this diverges from nominal, the more likely sync and frame drop problems are to appear.

Video: Data rate

The overall rate at which video data is being written to disk, in bytes/kilobytes/megabytes per second. When video compression is in use, this statistic refers to compressed video data.

Video: Compression

The overall video compression ratio. Ratios greater than 1.0:1 indicate shrinkage in video size, whereas less than 1.0:1 means enlargement (i.e. the "compression" is making the video bigger). Lossless algorithms typically range from 1:1 to 3:1, whereas lossy compression can go much higher.

Video: Avg frame size

The average size, in bytes, of each video frame written to disk. When video compression is in use, this statistic refers to compressed video frames.

Video: Frames dropped

This refers to the number of aberrations in the video stream which caused VirtualDub to drop a frame in the video stream due to them being crowded too close together (too fast). Fewer is better, although it is normal for frame drops to occur where there are disruptions in the video stream, such as the start of a new recording on a tape.

Video: Frames inserted

This refers to the number of aberrations in the video stream which caused VirtualDub to insert a placeholder frame into the video stream due to there being too few frames in that area (too slow). Fewer is better, although it is normal for frame inserts to occur where there are disruptions in the video stream, such as the start of a new recording on a tape.

Video: Resampling factor

When video timing correction is enabled, this indicates the factor by which "video time" is being accelerated or slowed to match the expected output rate. This is only active if video timing correction is enabled. Unlike audio resampling, video resampling only affects the assignment of frame numbers to incoming video frames; a number other than 1x does not mean that video frames are being interpolated.

Audio: Size

Total amount of audio data written to disk.

Audio: Average rate

The average rate of the raw PCM data in the audio stream, relative to real time. This is the estimated actual frequency of the incoming audio data. Small discrepancies in this value from expected are normal, as all clocks have some error; however, large discrepancies in this value from the specified sampling rate may indicate that the sound card has an audio clock with poor accuracy. Audio resampling can be used to stretch the audio to compensate.

Audio: Relative rate

The average rate of the raw PCM data in the audio stream, relative to the *corrected* video stream. This is thus the frequency of the audio if the video stream were perfectly timed, as it is assumed in the on-disk video. Discrepancies between this value and the ideal frequency indicate overall sync error. Small errors are expected due to measurement issues.

Audio: Data rate

Average bandwidth of audio data written to disk. When audio compression is active, this refers to the compressed result.

Audio: Compression

The overall audio compression ratio; larger ratios mean smaller audio on disk.

Audio: Resample

Stretch factor applied to the audio stream, in semitones, in order to correct for speed errors relative to the video stream. This is only active if the timing setting is "sync audio to video." A semitone is a step between minor notes on the musical scale, such as between the notes C and C#; a factor of +/-12.000 is a full octave (half or double speed). Positive values indicate the audio is being sped up (higher pitch), negative values indicate slowing (lower pitch), and zero means no change to the audio speed.

This value is an instantaneous measurement, not an average, so a varying adjustment means varying speeds in the written audio track.

An adjustment within about 0.030 semitones is not usually noticeable, and slow, gradual drifts or oscillations in this value are normal. However, factors above +/-0.100 semitones and rapid changes in this value indicate warbling in the resampler's output, which may indicate timing problems.

Sync: Video timing adjust

Amount of adjustment, in milliseconds, applied to the video stream caused by detected differences from audio timing. This is only pertinent if the timing setting is "sync video to audio." Positive values mean the video stream is being sped up; negative values mean it is being slowed down. It is normal for this value to increment or decrement occasionally over the course of a video capture session.

Sync: Relative latency

Estimated difference in arrival time, in milliseconds, between the audio and video streams. This is the difference between when VirtualDub sees a video frame and the audio that corresponds to that frame, not necessarily a sync error in the output. Positive values indicate the audio is arriving later, whereas negative values indicate it is arriving earlier. This value will typically be in to 10-100ms range for "TV tuner" style devices and possibly as high as +/-300ms for devices with integrated compression.

An abnormally large value here, particularly in the range of seconds or more, likely indicates a timing problem.

Sync: Current error

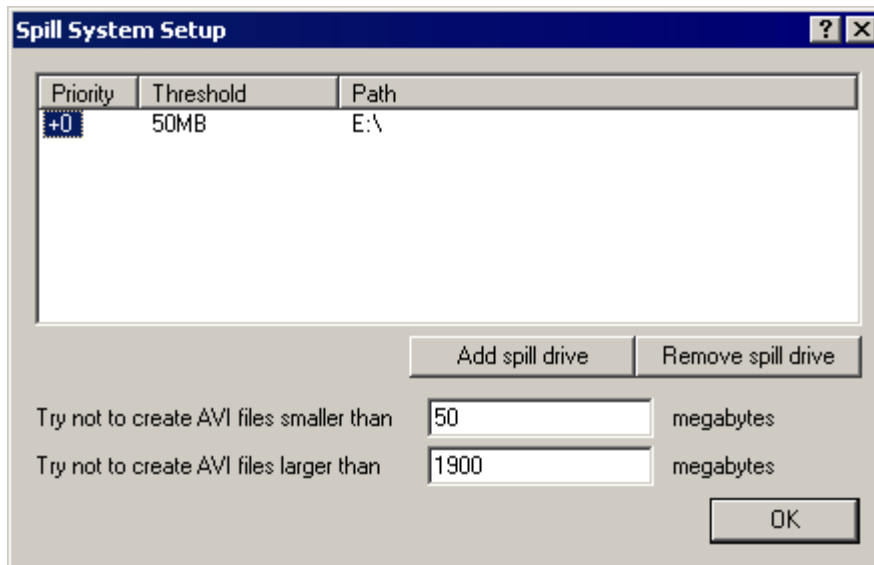
Estimated sync error between the audio and video streams; zero means that the audio and video streams are in sync. This is only calculated if the sync mode is "sync audio to video." The audio resync controller continually adjusts the audio resampling rate in an attempt to drive this error as close to zero as possible.

VirtualDub help - Capture: Multi-segment mode

Spill mode allows a video capture operation to be split across multiple files. This allows file size limits to be bypassed and also permits use of multiple partitions.

Selecting spill drives for capture

Select the *Capture > Capture drives* to set up the drive array for spill mode:



Add/remove spill drive

Add a new spill drive, or remove the currently selected drive. When adding a new drive, click on the threshold or path portion of the entry to edit it.

Capture drives can be on the same physical hard drive, but should not share the same logical hard drive (volume). It is a bad idea to add two entries for the same volume, as the two will have the same free space totals. For instance, C:\ and C:\Capture should not both be added to the list unless a different partition is mounted at C:\Capture.

Threshold

The threshold controls the minimum amount of disk space, in megabytes, that must be free on the drive for VirtualDub to use it. For instance, if the threshold is set to 50MB and the minimum capture file size is 100MB, VirtualDub will not start a new file on that drive unless there is at least 50MB + 100MB = 150MB free space, and will stop writing to a file being generated there once free space drops below 50MB.

Try not to create AVI files smaller than... MB

Sets the minimum file size that should be creatable above a spill drive's threshold before a file should be started on it. This prevents VirtualDub from switching to a spill drive that only has 14MB above its threshold; not only is this annoying, but it can result in back-to-back file switches, which can cause problems.

Try not to create AVI files larger than... MB

Sets the maximum file size that VirtualDub should aim for before switching to a new file. The maximum value for this file is 2048MB, because VirtualDub disables its large AVI file support when writing spill files. It should be set some amount below that to provide a buffer to account for latency during the video capture, which is why the default is 1900MB.

Using multi-segment capture

Enable *Capture > Enable multisegment capture* to allow use of spill drives. The capture filename will then be used to generate the prefix for sequentially numbered files, so that a base name of capture.avi produces capture.00.avi, capture.01.avi, capture.02.avi, etc. Then begin capture as usual to start capturing to a series of sequential files.

Note Make sure the capture path is on one of the spill drives, or VirtualDub will switch to one of the drives shortly after the start of the capture, leaving a ~50MB runt as the first file.

Caveats with multi-segment capture

As when editing, audio and video compression can impact the ability of VirtualDub to produce clean cuts between segments. In the interest of speed, VirtualDub is a bit sloppy when writing capture segments. Issues include:

- If video compression with delta frames is being used, a segment can start on a delta frame. This is OK if VirtualDub is loading the capture segment, but can be a problem if you are trying to load individual segments into a program that doesn't expect raw stream slices, because the delta frame is undecodable without the previous segment.
- Audio compression can cause a segment's audio to be slightly desynchronized from the video, subject to the block size of the audio compression. This isn't a problem when joining all of the segments together, though.
- There may be one additional segment that has a video stream with no video frames in it; this can occur when writing out the final audio during a capture, more audio has been captured than video, and the audio spills over into a new segment but the video doesn't.

If you will attempt to load individual segments or load the files into a different program, it is recommended that you use video compression that does not use delta frames, and disable audio compression.

Reading multi-segment capture files

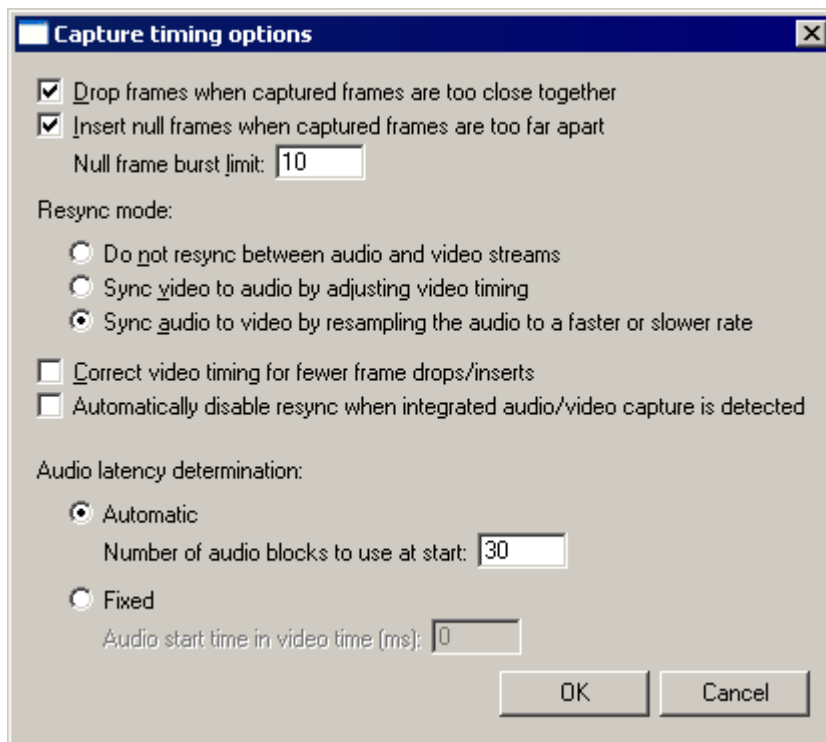
Each capture segment written by VirtualDub contains a record in it that indicates the location of the next segment. This makes it possible for VirtualDub to follow the sequence of capture files without knowing the list of spill drives, even if the sequence hops between drives. The algorithm for following the chain is as follows:

- Try the same directory as the previous segment.
- If that doesn't work, try the directory indicated by the segment hint in the AVI file.
- If that still doesn't work, ask the user to locate the next segment.

It is thus not a good idea to mix segments from different captures, as VirtualDub may get confused if it sees a `capture.06.avi` that is actually from a different sequence than `capture.05.avi` in the same location.

The format of the segment hint is a chunk with the FOURCC `segm` within the AVI header block. It consists of a single byte which is 00 if no further segments are present, or 01 if more segments should be loaded, followed by the filename of the next segment, null terminated. (Regrettably, this is 8-bit ANSI encoded, not 16-bit Unicode.) The full filename is encoded, although only the path is used, and the filename should still follow the *basename.nn.avi* convention.

VirtualDub help - Capture: Timing dialog



The *Capture timing* dialog allows control of the way VirtualDub matches video to audio during a capture operation. Video frames can arrive from the video capture driver with non-regular timestamps due to timing accuracy issues and interference with background tasks in the system; timing disturbances can also occur due to irregularities in the source video signal. During the video capture process, the varying timestamps have to be matched to regular frame slots in the output video stream.

Drop frames when captured frames are too close together

If enabled, VirtualDub discards frames that are spaced too close together for the output frame rate. For instance, three frames with timestamps 10ms apart (100 fps) cannot fit into a 29.97 fps stream. If this option is disabled, all captured frames are written to the output stream.

Unless the audio resampler is enabled (see below), disabling this option can result in desynchronization of the audio and video streams.

This option is enabled by default.

Insert null frames when captured frames are too far apart

If enabled, VirtualDub inserts dummy frames into the output video whenever captured frames are spaced too far apart for the output frame rate. For example, two captured frames spaced 66ms apart would have a dummy frame between them in a 29.97 fps stream. This dummy frame is identifiable in the video stream as a zero-byte frame and displays the same image as the previous non-dummy frame.

Unless the audio resampler is enabled (see below), disabling this option can result in desynchronization of the audio and video streams.

This option is enabled by default.

Null frame burst limit

Controls the maximum number of dummy frames in a row that VirtualDub will insert when captured frames are too far apart. This limits the amount of damage that occurs if timestamps in the video stream are wildly incorrect for a moment, such as a full day ahead.

The default for this option is 10.

Resync mode: Do not resync between audio and video streams

In this mode, auto-resync is disabled: VirtualDub will not monitor the audio and video streams for timing discrepancies. This means that the quality of synchronization in the output video is dependant on the accuracy of the audio and video clocks; for example, if the sound device is recording slightly slower than the requested sampling rate, or the video capture device reports timestamps that are 0.1% too fast, the output video will be accordingly desynchronized.

It is highly recommended that you enable auto-resync, as the audio and video clocks are never exactly identical unless both are being captured on the same device.

Resync mode: Sync video to audio by adjusting video timing

VirtualDub will speed up or slow down the video clock to match the audio clock. This will affect the number of frame drops and inserts in the video stream. If the audio clock is more inaccurate this will result in more drops/inserts; if the video clock is the worse one, this can sometimes *reduce* the number of drops and inserts.

The amount of adjustment to the video clock is reported on the information side bar as the *VT adjust* field.

Resync mode: Sync audio to video by resampling the audio to a faster or slower rate

The audio will be stretched or compressed to match the video clock. The video stream is not affected, but the audio stream will have a slightly higher or lower pitch and speed. As the discrepancy in clocks is typically very small, this change is not normally noticeable. This mode is only usable if the audio device is capturing in an uncompressed (PCM) format; if a compressed format is in use and this mode is selected, the *sync video to audio* mode is used instead. Note that this only applies to the raw audio capture format; any audio codec can still be used.

The amount of adjustment to the audio stream is noted in the information side bar as the *Resample* field.

This mode is selected by default.

The behavior of the audio resampler was significantly changed for 1.6.12.

Correct video timing rate errors

If enabled, VirtualDub will attempt to adjust video timing to compensate for frames arriving slightly faster or slower than expected. This can reduce the number of frame drops or inserts incurred during video capture, at the expense of making the video clip play slightly faster or slower than real-time. This option is meant to be used with the *sync audio to video* resync mode, or when audio is not being captured.

Video timing correction is automatically disabled if the resync mode is set to *sync video to audio* or if the resync mode is automatically forced to that mode because *sync audio to video* can't be used due to compressed audio. It is also automatically disabled if

integrated A/V capture is detected and resync disabling is enabled (see below).

The behavior of the video timing corrector was significantly changed for 1.6.12.

Automatically disable resync when integrated audio/video capture is detected

Capture devices that have both audio and video capture integrated usually use a shared clock to avoid sync errors between the streams. When enabled, this option causes VirtualDub to automatically disable auto-resync when such a situation is detected.

Note that VirtualDub must see both audio and video devices on the same capture driver for this option to take effect. If you are using a capture device which exposes its audio capture as a separate sound driver in Windows, VirtualDub will not see it as integrated.

VirtualDub help - Capture: Performance

Capturing live video is a real-time operation and places high demands on your system. Here are some steps you can take to improve video capture performance.

Shut down background tasks and applications

Interruptions by background programs can interfere with video capture and cause dropped frames. Applications you should watch out for, and temporarily disable, include:

- virus scanners
- disk defragmenters
- search indexers, especially the Microsoft Indexing Service (formerly Microsoft Office Fast Find Indexer)
- task schedulers
- on-screen tickers and status readers — particularly anything that flashes or scrolls on-screen

CPU usage is a problem here, but competition for the hard disk is usually a much worse problem: any attempts to access the disk by another application will cause the disk to seek back and forth, which seriously cuts available write bandwidth.

Absolutely avoid using the CD-ROM drive during video capture, as the access traffic during the spin-up of the CD-ROM drive can cause the hard drive to go off-line for more than a second.

It is not recommended to use other applications during video capture. Even if the other applications are light in disk and CPU usage, they may cause momentary hiccups that result in dropped frames or timing anomalies during the capture. The less that is going on in the system, the more accurate VirtualDub's timing statistics are and the better it can keep audio and video streams in synchronization.

Note When in capture mode, VirtualDub temporarily sets its process priority to High and disables both the screensaver and power saving mode on the display device. It isn't necessary to change these manually.

Keep the disk clean

Hard drives reach peak write performance when writing sequentially on disk — the more they have to seek around to different regions, the lower the available bandwidth. When files are scattered throughout a disk, free space is broken into a lot of small chunks, which is called

fragmentation. This means that it is important to have large areas of contiguous free space on a drive. Here are some tips to improve disk write performance:

- Run Disk Defragmenter and check that free space is not overly fragmented on the target drive. The fragmentation of existing files doesn't matter, just the free space. It's OK if the free space is split into a few dozen chunks, as a seek every minute isn't a problem. If it's really swiss-cheesed, though, consider defragmenting.
- Have extra free space on the drive. VirtualDub allocates large blocks of space a time to give Windows a chance to find clear areas on the drive; however, this becomes more difficult as the drive gets full and Windows scavenges for the last free space on the disk. Fragmentation becomes much worse once you start filling the last 5-10% of a drive.
- Use a different partition or disk than the one that holds Windows system files, as that partition typically has a large number of small, volatile files and fragments very quickly.

Use appropriate video compression and video formats

Uncompressed video capture dumps a *lot* of data onto the disk — 720x480 in 16-bit YUY2 at 29.97 fps produces approximately 20 megabytes per second. Modern computers have much more CPU power and thus you can reduce the strain on the disk by storing the video in a more efficient format.

Start by choosing an efficient raw video format for your capture device to produce:

- *32-bit RGB*. Avoid this format, as it wastes 33% of the space used.
- *24-bit RGB*. This is the baseline, most compatible format. Start here.
- *15/16-bit RGB*. Avoid, as it introduces serious banding (quantization) artifacts.
- *16-bit YCbCr (UYVY or YUY2)*. Try this format. It is closer to the format produced internally by most hardware video decoders and used internally by many video codecs, but it is 33% smaller than 24-bit RGB. Using this format will often significantly improve performance.

Apply video compression on top of this to further reduce data bandwidth. Because the raw video capture will likely need some post-editing to be useful, avoid formats that overly degrade video or are difficult to edit.

- The *Huffyuv* video codec by Ben Rudiak-Gould is an excellent capture codec to use, as it is lossless, typically achieves around a 2:1 compression ratio, and can work directly with YCbCr video. It is very fast and should work in real-time on a 500MHz or faster CPU.
- *Motion-JPEG (MJPEG)* codecs are also an excellent choice. They are lossy and thus will reduce video quality very slightly, but the compression ratio is significantly better than lossless codecs like Huffyuv, at least 4:1 with little or no perceptible quality loss. The Motion JPEG format is also field-savvy and will store interlaced video without screwing up the fields.
- *Digital Video (DV)* is another format to consider. Like Motion JPEG, DV is also a slightly-lossy format that is friendly to interlaced video. It does take a bit more CPU to compress and decompress, however. Also, unlike Motion JPEG, DV is always constant in data rate — 3.6MB/sec — so it is easy to predict how much disk space is required for a given amount of time.
- *MPEG-4* and other high-compression video formats should be avoided, as they require significant CPU power to compress and may not be able to keep up with the incoming video at adequate quality. Also, their extremely long delta frame chains can make the resulting video difficult or impossible to edit.

Note that if you have a capture device that has hardware video compression, your options here are likely very limited. In that case, browse the capture driver's configuration dialogs, usually *Video > Video Source* or *Video > Capture Filter*, and select a format with relatively

light compression.

Disable audio compression

Uncompressed audio requires much less bandwidth than uncompressed video and reasonable audio compression usually requires a lot of CPU power. This is particularly true of modern audio compression formats such as MPEG audio layer III (MP3). It is highly recommended that you *not* use audio compression during video capture, as it can consume a lot of CPU and make video capturing less reliable.

VirtualDub help - Capture: Troubleshooting

Fireworks on entering capture mode or selecting a particular capture driver

VirtualDub tries to change settings in such a way that if the settings cause the driver to blow up, that the modified settings aren't saved — but occasionally the failure occurs later, in the form of a hang, crash, blue-screen, etc. The result is that you can't use the capture device any more, because as soon as you try entering capture mode VirtualDub auto-selects the last capture driver, and then restores the saved settings, and... well, you get the point.

Fortunately, there is an escape hatch.

Holding down a Shift key when entering capture mode will prevent VirtualDub from automatically selecting the last used capture driver. The easiest way to do this is to hold Shift when selecting *Capture AVI...* from the menu. Similarly, holding down Shift when selecting a capture driver will prevent any saved settings that were recorded for that driver from being restored. The errant settings can then be changed and re-saved to correct the problem.

Note that this doesn't help for any settings that the capture driver itself saves and that VirtualDub doesn't know about. In that case, you can either attempt to reinstall the driver, or find the location in the Registry where it saves those settings, and try to change them.

System-wide hang, blue-screen, or instant reboot when using the "overlay" display mode

The "overlay" display mode on most capture devices causes the capture hardware to stream the video image directly into the display memory of the video card. Sometimes the capture device and video card don't cooperate well and the result is a lockup or blowup when overlay display mode is chosen. In most cases it is better to choose the Preview mode instead, as then VirtualDub can directly control the video display, but sometimes it is possible to switch a Video for Windows capture driver's method of overlay display to a more compatible mode.

If this is occurring, check if the capture driver has an options screen. In Windows XP, navigate as follows:

- Start Menu, Settings, Control Panel, Sounds and Audio Devices
- Hardware tab, Legacy Video Capture Devices entry, Properties button
- Select the Properties tab, find the video capture driver, and then the Properties button.

If the *Settings...* button is not disabled (grayed out), select it and browse through the driver's configuration dialog. Look for an option called "overlay mode" or "use DrawDib"; this will switch the driver to a slower, but more compatible method of display.

No color, rainbow coloring, or simply bad color

First, check that you are using the correct video input and that the capture device is

configured to use the right one. On some devices the composite video input is simply the S-Video input with an adapter, so it is possible to set the video input to S-Video and still see a grayscale version of a composite video input.

If you have both S-Video as well as an "external" or "camera" input, sometimes the external/camera input works better, inexplicably.

Check the video standard in use. Attempting to capture an NTSC stream as PAL, PAL as SECAM, etc. will result in scrambled color. The video standard is normally controlled in *Video settings* for VFW drivers and *Capture filter* for WDM drivers, although the location may vary.

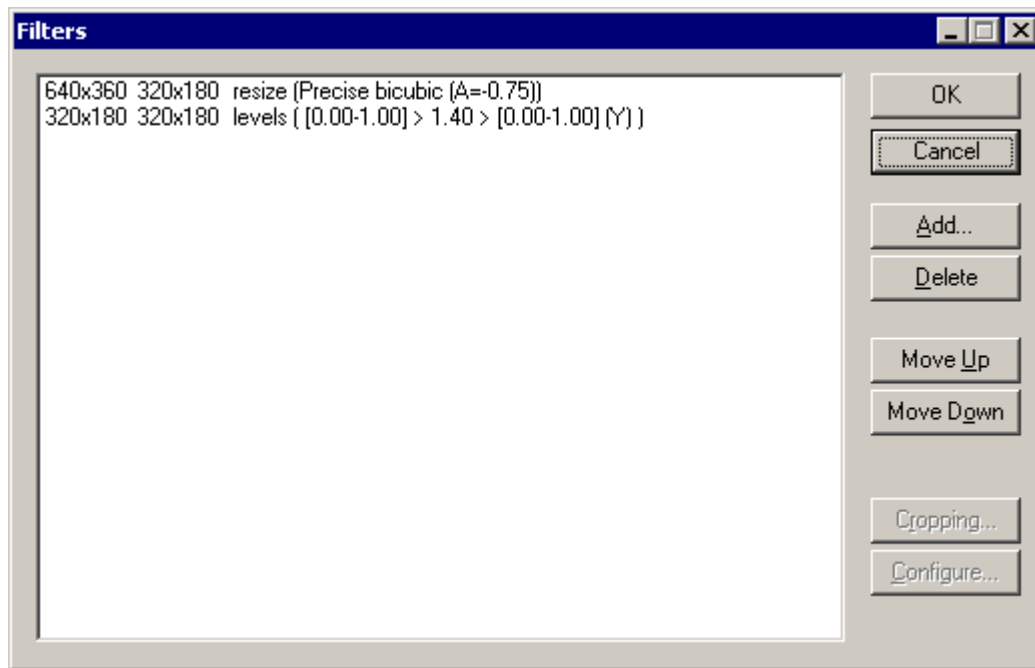
Video capture mysteriously stops

Check *Capture > Stop Conditions* and make sure no conditions have been inadvertently enabled.

Interruptions in video sources, causing a loss of video signal, can occasionally cause this problem.

On Windows NT-based platforms, locking the workstation with Ctrl+Alt+Del can cause a capture operation to stop. In general, it is best to avoid doing anything that might change desktops or the video display mode.

VirtualDub help - Dialogs: Video filters



- [Video filter reference](#)

Video filters transform video frames sequentially, such as blurring the image, resizing the image frame, or applying noise reduction. VirtualDub's video filter system takes chains of filters, which are set up through the Video Filters dialog.

Frame sizes

Each video filter in the list is shown with the size of the video frame entering the filter and the video frame size produced by that filter. Video filter chains may be created without a video loaded, and when the video is changed, the input and output frame sizes for each frame will change accordingly.

Note If no video is loaded, VirtualDub will compute frame sizes as if a 320x240 input were present.

Filter name and parameters

Each filter entry has the name of the filter beside the frame sizes, as well as a parameter list for the filter. The parameter list is filter-specific and is only shown to aid in scanning the filter chain.

Cropping

Cropping can be applied at the beginning of any filter by selecting the entry and selecting the *Cropping...* button. Once cropping borders are set, the cropped frame is reflected in the input frame size for that filter entry.

Note The post-crop width and height should never be zero or negative.

Filter order

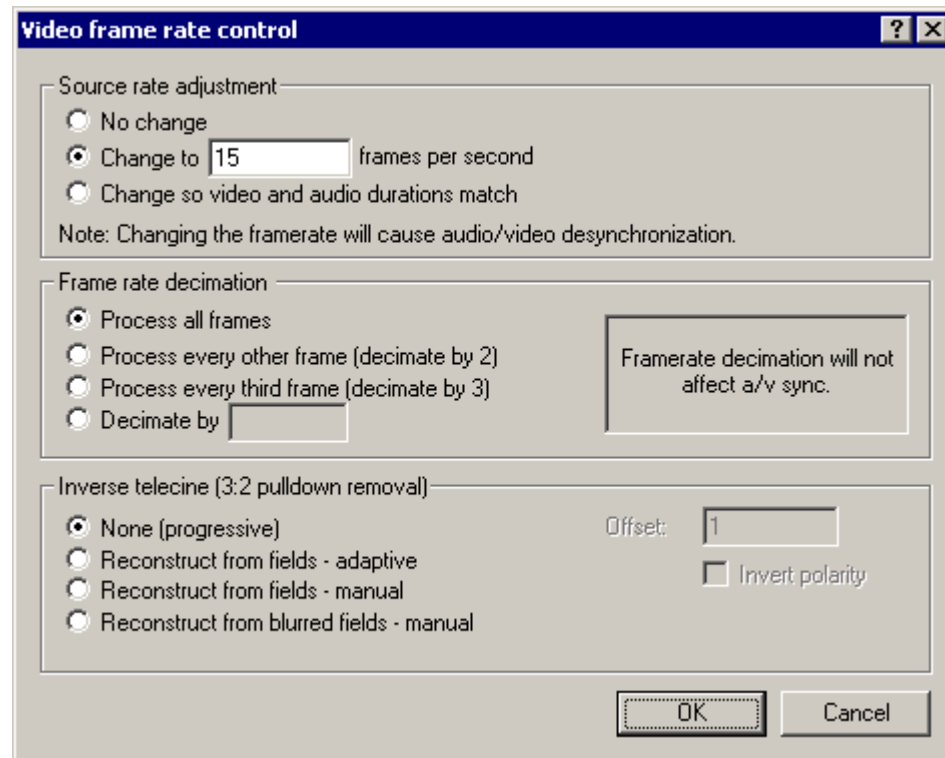
Filters are executed from top to bottom; use the *Move Up* and *Move Down* buttons move the currently selected filter up or down one slot. Frame sizes are automatically recomputed as filters are moved.

Loading filters

The *Add Filters* sub-dialog has a *Load...* button for loading third-party video filters. These video filters need to be written specifically for VirtualDub and have a *.vdf* file extension.

Note Video filters loaded from this dialog are loaded only for the current session. If you want filters loaded all the time, move them under *plugins* in VirtualDub's program directory.

VirtualDub help - Dialogs: Video frame rate control



The *Video frame rate control* dialog allows you to alter the frame rate of video, reduce the number of frames, or remove 3:2 pulldown.

Adjusting frame rate

If the source video has the wrong frame rate or doesn't natively have a frame rate (image sequence), you can specify one in the source correction portion of the dialog. You can either type in a rate, or have VirtualDub automatically choose a "same length" setting, such that the video and audio tracks end at the same time.

Note No frames are added or deleted by this setting, so if the video is synchronized with the audio beforehand it won't be after you change the frame rate. Similarly, if the video isn't synchronized, you may be able to fix it with this setting.

Note Manually entered frame rates are rounded to the nearest microsecond period. In particular, you cannot enter an exact NTSC fraction (30000/1001).

Frame rate decimation

Decimation pulls frames from a source at regular intervals. This is useful for producing "thumbnail" videos with small sizes, and for dropping the rate of a video without introducing jerkiness due to uneven frame rate. The decimation interval must be a positive integer.

Inverse telecine (3:2 pulldown)

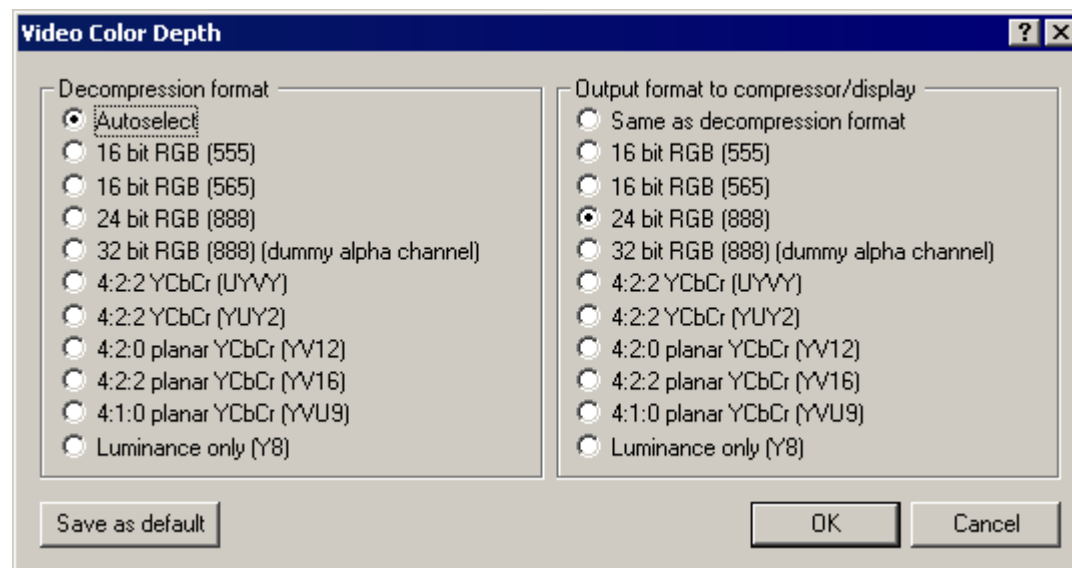
3:2 pulldown, or telecine, is the process by which film-rate material (24 fps) is converted to NTSC rate (29.97 fps). This is done by splitting the film frames into fields, and then "pulling" fields down in an alternating 3,2,3,2... pattern. This produces five output frames for every four input frames, in a characteristic pattern of three progressive frames, followed by two interlaced frames. The resultant 30 fps stream is then slowed down slightly to the target 29.97 fps.

Inverse telecine (IVTC) attempts to recover the original 24 fps stream by analyzing the input frames and removing the duplicate fields. In *adaptive mode*, VirtualDub attempts to guess the position of the 3:2 pattern, while in *manual mode*, you specify the offset and polarity. In both cases, the input frame rate is dropped by 20%, changing a 29.97 fps input to 23.976 fps.

The third mode, *reconstruct from blurred fields*, handles the case where a video has been telecined, and then the fields blurred together, usually by shrinking the video size. In this case the original frames cannot be recovered by matching fields, but can be recovered through simple frame algebra. The reconstructed frames will have more noise and possibly ghosting where clipping at black or white has occurred, but if successful the result is a smooth progressive video.

Note All of VirtualDub's IVTC modes assume a regular 3-2 pattern. More complex telecine patterns are possible which cannot be removed through this system.

VirtualDub help - Dialogs: Video color depth



Selects the desired image formats for video decompression and video compression.

Decompression format

Selects a color format to target when decompressing the input video.

For a compressed source, this format is requested directly from the video codec. "Autoselect" will choose a common, safe RGB format, such as 24-bit, 32-bit, or 15-bit RGB.

For an uncompressed source (or Avisynth script), the desired format is produced

through image conversion if it is not the same as the source. YCbCr-to-YCbCr conversions do not round-trip through RGB. "Autoselect" chooses the source format so that no conversion is necessary.

If the video codec cannot produce the format you have selected, VirtualDub will automatically attempt to degrade the video format to a similar format that is supported. For instance, if 16-bit RGB fails, 24-bit RGB will be tried. The fallback code avoids color space conversions if possible; if a YCbCr format fails, other YCbCr formats will be tested before resorting to RGB.

Output format

Selects a color format for output. During preview, this is the format sent to the display panes. When video compression is active, this selects the format received by the video compressor.

The output format is produced by conversion from the input format. When video filters are in use, the input format is converted to 32-bit RGB, processed, and the result converted to the output format. If video filters are not being used, the input format is converted directly to the output format.

Note As is standard in GDI bitmaps, the 32-bit format has a dummy alpha channel, used only for padding. VirtualDub cannot currently target a 32-bit format with meaningful alpha.

Note The size of the output format doesn't necessarily relate to how well the output compresses through a video codec. For instance, converting from 24-bit RGB to 16-bit RGB can hinder compression, because it increases quantization noise (banding) in the image.

Format descriptions

16-bit RGB (555)

This is a format with five bits per channel (32 levels) for each of red, green, and blue. It is a relatively low-precision format that is prone to some banding artifacts but is common on older video display hardware, as it is faster and consumes less memory than 24-bit RGB. This format is sometimes known as HiColor.

16-bit RGB (565)

This is a slightly improved version of 16-bit RGB (555), as it uses the unused bit to improve precision in green.

24-bit RGB (888)

This is a format with eight bits per channel (256 levels) for each of red, green, and blue. It is common for photographic images and is often known as TrueColor. This is the safest and most reliable format to use for video interchange.

32-bit RGB (8888) (dummy alpha channel)

This is similar to 24-bit RGB, except that it has an additional unused eight bits per pixel. Picture quality is identical to 24-bit, but the additional padding makes the pixels a more convenient size for computation. This format is sometimes slightly faster than 24-bit for processing, but should be avoided for storage as it wastes one-third more space.

4:2:2 YCbCr (UYVY)

This is a format which uses the YCbCr color space (luma, chroma-blue, chroma-red), which is closer to the way color images are perceived by the human brain. It averages

only 16 bits per pixel with similar perceptual quality to 24-bit RGB by only storing color information at half horizontal resolution, producing slight color bleeding but only taking two-thirds as much space.

This format, as do all other YCbCr formats listed below, encodes luminance (Y) with a range of [16, 235] and chroma (Cb/Cr, or U/V), with a range of [16, 240].

UYVY is accepted directly by many video codecs. Since many video codecs internally use color spaces similar to YCbCr, using this format with video codecs can speed up rendering.

4:2:2 YCbCr (YUY2)

This is the same as 4:2:2 YCbCr (UYVY), except for a shuffling of data bytes. It has the same quality and performance advantages as UYVY.

4:2:0 planar YCbCr (YV12)

This is a YCbCr format with an average of 12 bits per pixel and half resolution both horizontally and vertically in color information. It thus takes 25% less space than UYVY or YUY2, but with some vertical bleeding. "Planar" refers to the organization of data in the format, where luma, chroma-blue, and chroma-red are stored separately.

Some codecs accept this format directly for further increases in performance over the 4:2:2 formats. However, it should be avoided for interlaced video, where the loss of vertical color resolution can cause motion artifacts between fields.

Note Although 4:2:0 YCbCr formats exist that accommodate interlacing, the YV12 four-character code (FOURCC) denotes a specific 4:2:0 encoding that is non-interlaced.

4:2:2 planar YCbCr (YV16)

This is a YCbCr format with an average of 16 bits per pixel and half horizontal resolution in color information.

The YV16 format is rare and not well supported by video codecs and playback applications, but is supported here for completeness.

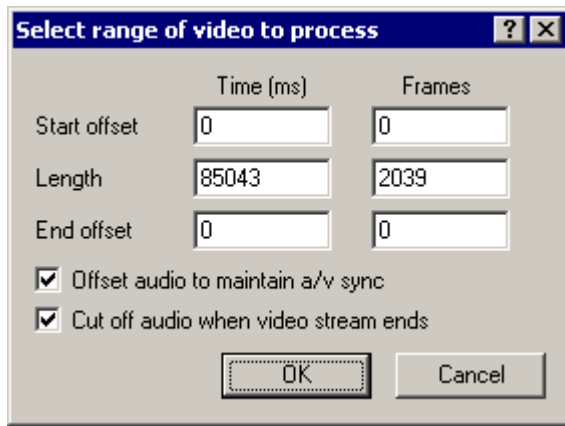
4:1:0 planar YCbCr (YVU9)

This is a YCbCr format with an average of 9 bits per pixel and quarter resolution both horizontally and vertically in color information. YVU9 thus takes 43% less space than UYVY/YUY2 and 62% less space than 24-bit RGB, but at the cost of significant color bleeding.

Luminance only (Y8)

Y8 is a monochrome format, and thus selecting it will cause video to be converted to grayscale. However, it only requires half of the space of UYVY or YUY2 at 8 bits per pixel.

Y8 uses the same luma scale as YCbCr, 16-240, so there is a very slight loss of luma precision compared to 24-bit RGB. However, conversion between Y8 and YCbCr formats is lossless in luma and extremely fast.



Selects a subset of the input to use.

Input range

Controls how much video at the beginning and ends of the source is skipped during processing, so that only a portion in the middle is used. All offsets can be set in either frames or milliseconds (ms).

Note Offsets persist even if you load a new file.

Note The end offset is the number of milliseconds from the end, not from the beginning. It is usually easier to set start/end points from the position control (*HOME/END*).

Note You can only select a single range with this dialog; use the *delete frames* feature to extract multiple segments in a single pass.

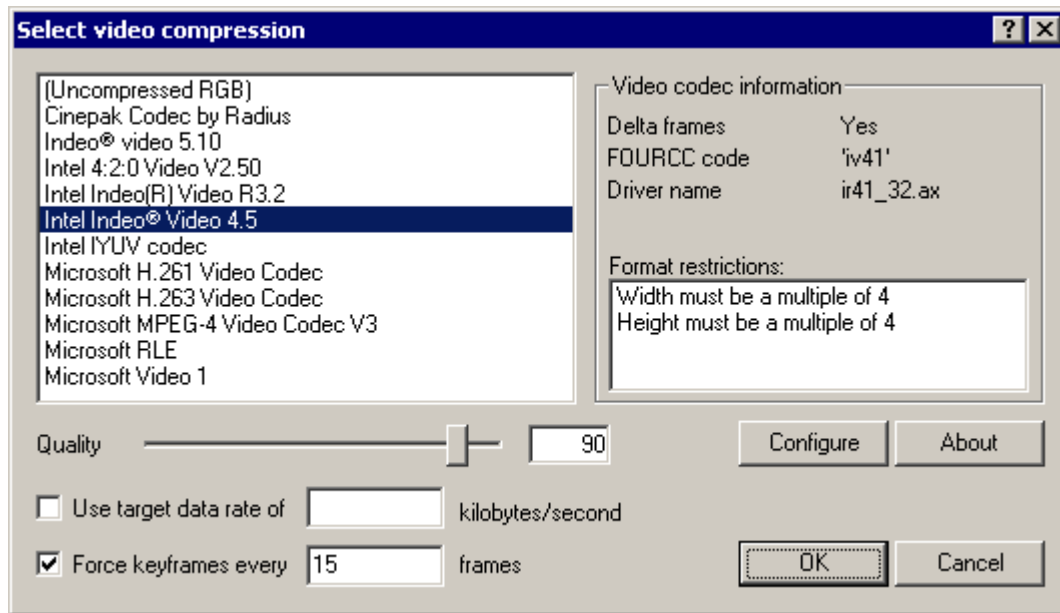
Offset audio to maintain a/v sync

Controls whether the start offset applies to the audio stream. If unchecked, the start offset deletes frames only from the video stream, advancing it that many frames ahead of the audio stream. This will alter audio/video synchronization accordingly.

Cut off audio when video stream ends

If the audio stream is longer than the video stream, this controls whether the extra audio is used or not. Generally, players continue to show the last video frame while the remaining audio plays.

VirtualDub help - Dialogs: Video compression



VirtualDub can use video codecs installed in Windows to compress video.

Note Video codecs are external drivers created by third parties and are not part of VirtualDub. VirtualDub does not install codecs by itself and does not provide any video compression technology of its own.

Selecting a video codec

Currently installed video codecs are listed in the top left pane; select a codec to use or choose "uncompressed RGB" to disable video compression. Once a codec is selected, diagnostic information appears on the right side. Every codec has a unique *fourcc*, or four character code, associated with it.

Note Not all codecs are usable -- sometimes programs install versions of a codec designed only to decompress video, not to compress it. Also, sometimes the codecs are keyed to a particular program due to licensing concerns, and thus won't work in VirtualDub.

Note For performance reasons, many codecs split the video frame into equal-size tiles and thus require the frame width and height to be specific multiples of the tile size. Requiring the width to be a multiple of four pixels is very common. VirtualDub attempts to detect such limitations and displays any such information in the right pane. In some rare cases with broken codecs this check fails, and the result may be a "bad format" error when you attempt to begin compression.

Configuring the codec

Some codecs accept *quality*, *data rate*, and *key frame interval* parameters. The higher the quality, generally the slower the codec runs and the larger the output, but the better the video looks after compression. By setting a data rate, you request that the codec attempt to produce output at a fixed size-per-time ratio, set in kilobytes/second. And by setting a key frame interval, you are enforcing that seek points are placed at regular intervals in the video in order to reduce wait time when seeking. Nearly all codecs support additional parameters, which can be set by pressing the *Configure* button. The dialog that appears is codec-specific; please consult the documentation for each codec for further details.

Note Codecs are designed for a particular range of frame sizes and compression ratios and perform inefficiently outside that range. In particular, for most codecs, setting Quality to 100 is likely to produce ridiculous file sizes and may

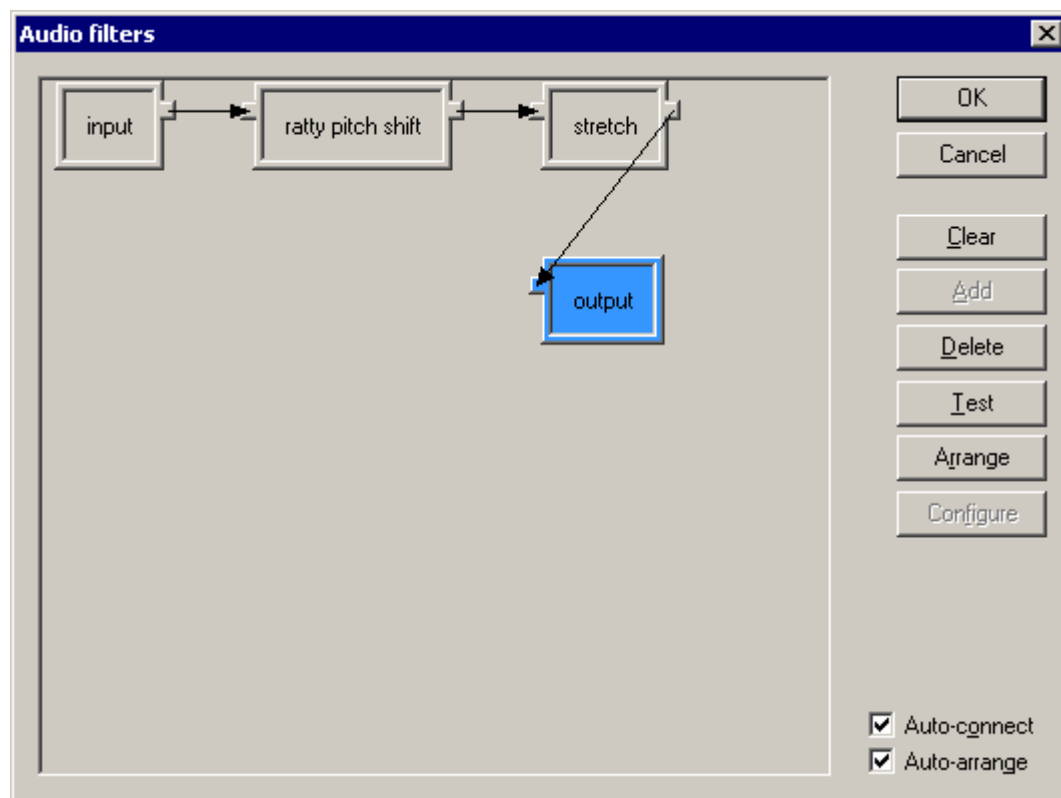
not even produce perfect-looking video. Small file size and perfect-quality video are mutually exclusive. If you need perfect quality and are willing to put up with large file sizes, lossless codecs exist that are created specifically for this purpose.

Note If the codec allows you to enter a key frame interval but you don't enter one, you may get a video with a single key frame at the beginning. This means that every time you seek backwards in the video the player will have to decompress the entire video from the start up to that point, which is a tad slow. Key frames generally have lower compression ratios, so the interval determines a tradeoff between seek latency and compression efficacy.

Note Data rates specified here only pertain to video. Make sure to allow for audio and for file structure overhead.

Note Many modern codecs do adaptive key frame placement and more complex data rate regulation, and place all their settings in their configuration dialog. Although the data rate entry in VirtualDub's configuration dialog is specified in units of kilobytes per second, or 1024 bytes/sec, codec dialogs generally use kilobits per second (Kbits/sec), or 1000 bits/sec (125 bytes/sec).

VirtualDub help - Dialogs: Audio filters



- [Audio filter reference](#)

VirtualDub's advanced audio filter mode supports a *filter graph model*, where you can chain filters in complex, branching configurations. Filters are represented by rectangular *nodes* on the graph; each node may have *input pins* on the left and *output pins* on the right.

Adding and removing filter nodes

To add filters, click the *Add...* button. This displays the *Add filter* subdialog, which

displays a list of available filters. In that subdialog, click a filter entry and then *Add* to add the filter to the graph, or alternatively, double-click the filter entry. The subdialog stays up until you are done adding filters and close it. Filter nodes added by mistake can be removed by clicking on them and pressing the *Delete* key or button.

Connecting filter nodes

To connect filters together, drag from an input pin of one node to the output pin of another. (You cannot connect two input pins or two output pins.) An arrow will be displayed showing the connection. As with nodes, connections may be selected and deleted. Since this process is rather tedious, by default the *auto-connect* option is enabled. Auto-connect ties the first unused output pin of the currently selected filter to the first input pin of the newly added filter, and then selects the new filter. Simple chains can then be established just by adding the filters sequentially.

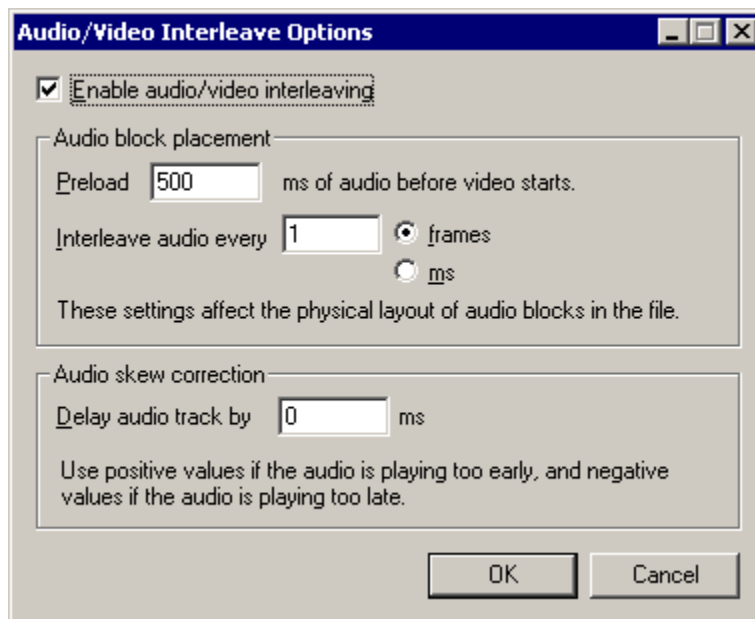
Filter graph restrictions

The following rules apply to audio filter graphs:

- There must be exactly one "source" node.
- There must be exactly one "output" node.
- All pins must be connected.
- No cycles are allowed, although parallel branching is permitted.
- Parallel paths must run at the same speed.

The last rule means you can't take a stream, slow it down with "stretch", and then mix it back with itself.

VirtualDub help - Dialogs: Audio interleaving



Although video clips have simultaneous audio and video streams that play at the same time, the files that contain them are a single stream. *Interleaving* fakes having two streams by slicing the audio and video streams into chunks and mixing them together in chunks by time. A player reading an interleaved file receives a little bit of audio, then a bit of video, and then more audio, etc., buffering them in memory for a short time before playing the two together. VirtualDub's *audio interleaving* dialog controls how interleaving is performed.

Note Audio interleaving has absolutely *zero* impact on when audio plays -- regardless of how audio and video frames are scattered in an AVI file, they still maintain the same timing relationship, because the index block sequences them properly.

Note Interleaving audio and video is not necessary unless the file is stored on a medium that streams much more efficiently than it handles random access, such as a CD-ROM drive or transmission over the Internet. High-speed devices such as hard disks can handle playback from non-interleaved files without problem, which is a consideration as interleaving isn't free -- it costs a small amount of overhead in the file size.

Enable audio/video interleaving

If this option is selected, audio blocks are interleaved between video frames throughout the file. If not, all of the audio is placed at the end.

Preload

Players require a certain amount of audio to be buffered ahead of time before starting video playback. The *preload* option controls how much audio is placed at the beginning of the file before the first video frame, in milliseconds. Usual values for this are 500ms (half second) or 1000ms (full second). If this setting is improperly adjusted, the file may not play optimally on slow computers or slow devices, although with modern video players this is less of a concern.

Interleave interval

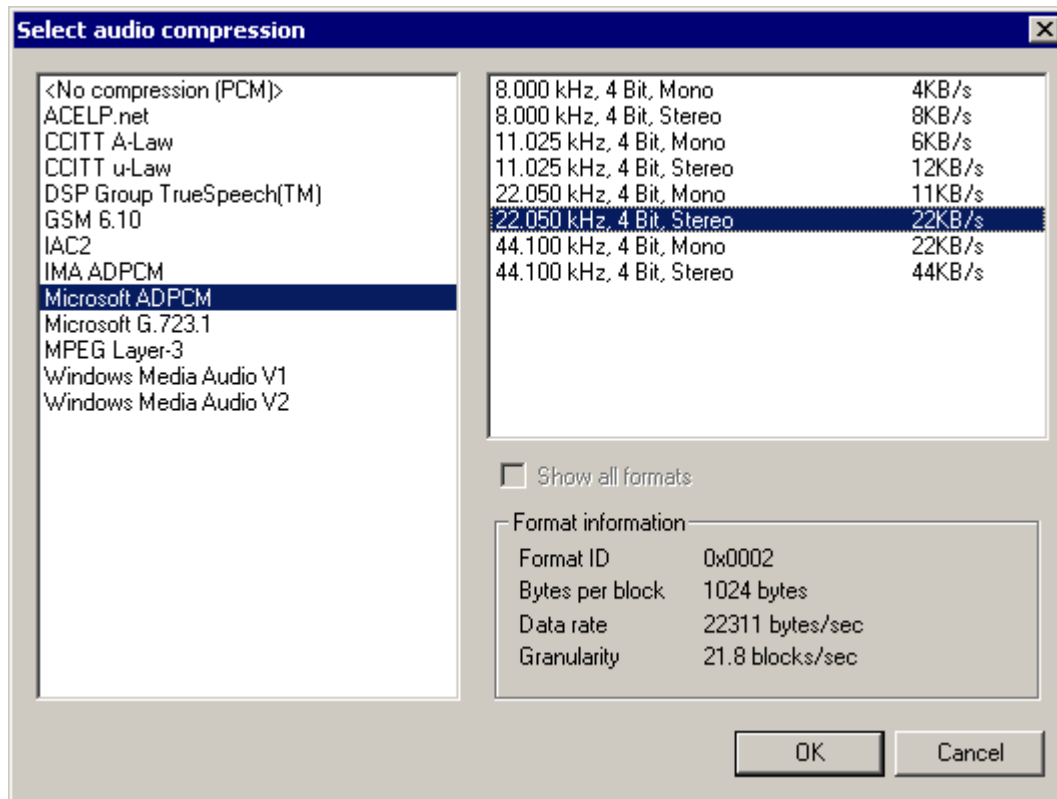
Controls how often audio blocks are scattered throughout the file -- the shorter the interval, the less buffering required by the player, but the more overhead in the output file. If the interleaving interval is too large, players will have to seek back and forth in the file, which has the worst of both worlds since the interleaving won't allow the player to do large, contiguous reads in the file. Usual values for this option are either interleave once per frame (1 frame) or every half second (500ms).

Audio skew correction

This isn't really an interleaving option, and is the only option here that actually does affect audio sync -- but it is here for lack of a better place. Audio delay basically shifts the audio track back and forth, either by dropping samples from the beginning or adding samples at the start (zero for PCM, duplicated first sample for compressed audio). Use this option to shift an audio track into place if it seems a bit off.

Note Audio compression may impede use of delays. Some formats, particularly, ADPCM, use large block sizes that will limit the granularity of adjustments. The MP3 format used in AVI, on the other hand, has a hack in its format that prevents applications from seeing the "true" block size of the format. The result is generally junk produced at the beginning of the MP3 stream, which may not produce the desired delay depending on the player. An audio editor specifically designed to edit MPEG audio streams is required in these cases.

VirtualDub help - Dialogs: Audio compression



VirtualDub can use audio codecs installed in Windows to compress video.

Note Audio codecs are external drivers created by third parties and are not part of VirtualDub. VirtualDub does not install codecs by itself and does not provide any audio compression technology of its own.

Selecting a audio codec

Currently installed audio codecs are listed on the left. Select *No compression (PCM)* to disable audio compression, or choose a codec. Once a codec is chosen, the available compression formats produced by the codec are shown on the right. By default, VirtualDub only displays those formats that can be produced by the codec from the current audio source and filtering settings. Uncheck *show all formats* to display all formats allowed by the codec, but note that attempting a conversion unsupported by the codec will generally result in an error.

Note Not all codecs are usable -- sometimes programs install versions of a codec designed only to decompress audio, not to compress it. Also, sometimes the codecs are keyed to a particular program due to licensing concerns, and thus won't work in VirtualDub. In particular, codecs installed by some versions of Windows Media are not licensed for general use and won't work except for in an ASF file under Windows Media Player.

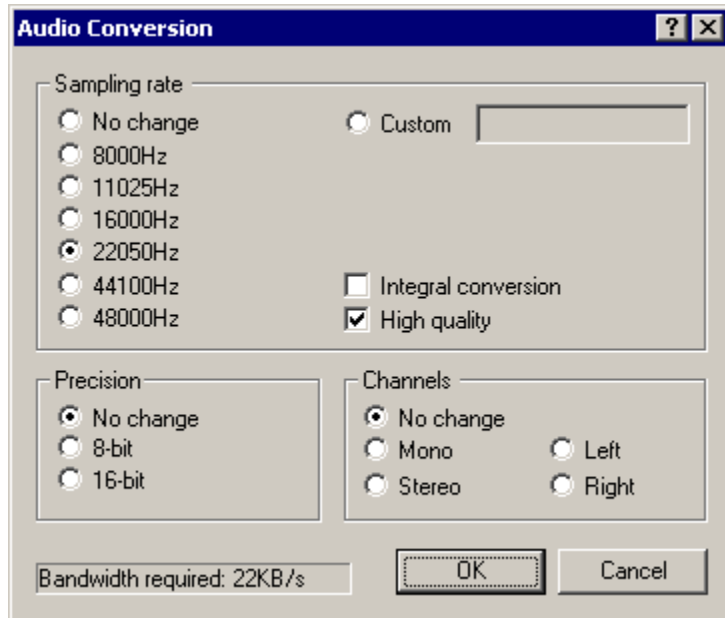
Note Installing two codecs that compress and decompress the same format may confuse VirtualDub. This includes having more than one MP3 codec, or having both Windows Media Audio and "DivX Audio" installed.

Note There are four common versions of the *MPEG Layer-3* codec that appear the same, except for the compression formats supported:

- L3CODECX.ACM - no compression supported
- L3CODECA.ACM (Advanced) - up to 56Kbps
- L3CODECP.ACM (Professional) - up to 128Kbps
- L3CODECP.ACM (Radium warez version) - up to 256Kbps

All four of these codecs are very similar and have the same ID in the Windows Audio Compression Manager, so two programs that come with different versions sometimes overwrite each others' codecs when installing. If you used to have certain MP3 compression options and no longer have them, it's possible a program you recently installed has done this.

VirtualDub help - Dialogs: Audio conversion



The audio conversion dialog allows for conversion between different PCM audio formats, including changes in sampling frequency, sample precision, and channels.

Note Audio conversion is only permitted if advanced filtering is off -- if advanced audio filtering is on, use the *resample* filter.

Sampling rate conversions

Allows changes in the sampling rate of the audio stream, without changing the pitch of the audio itself (resampling). You can select one of the common sampling rates, or *custom* to resample to any other rate. Higher sampling rates represent higher frequencies better and give "brighter" audio, but consume more space.

Integral conversion

Forces the conversion to operate by integral multiplies (one-half, one-third, twice, three times, etc). Use this if you want to convert to a integral factor of the original rate that's close but not exactly the same as one of the listed values, i.e. 22047.5Hz. If the frequency you select is exactly what you want then this option should be unchecked.

High quality

Switches the resampling filter from point sampling to a triangle filter for better quality. This reduces aliasing, which sounds like noise or halo in the output.

Note The sampling rate converter controlled by this dialog is rather antiquated and not very high-quality by today's standards. The *resample* filter introduced in VirtualDub 1.5 uses a 65-point windowed-sinc filter and produces better quality than the triangle filter allowed here. It is slated to replace this resampler as the

primary resampler in a later version.

Precision

Controls the accuracy of audio samples -- 8-bit samples only take half as much space as 16-bit, but can't represent soft sounds as accurately and have a higher noise floor.

Note 8-bit should really only be used if you are outputting uncompressed PCM to another program that needs it, or aren't changing existing 8-bit PCM. Otherwise, instinctually choose 16-bit. Modern audio codecs work in different audio representations, such as subbands or cosines, and thus don't get better compression with 8-bit source than with 16-bit.

Channels

Allows conversion of stereo audio to mono or mono to stereo. For stereo-to-mono conversions, the two channels are summed at half amplitude $((1+x)/2)$. For mono-to-stereo conversions, the single channel is replicated into the left and right outputs.

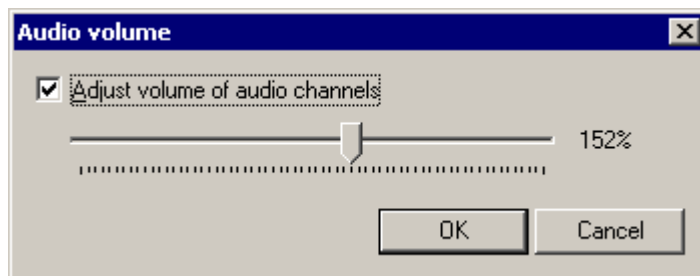
The *left* and *right* options convert a stereo stream to mono by discarding the unwanted channel. This is particularly useful for some dual-language or karaoke produced, where two mono streams are stored as the left and right channels of a stereo track.

Note For simple audio encodings such as PCM or ADPCM, dropping a stereo stream to mono halves the size of the audio stream. More complex encodings such as MP3, however, can encode a stereo stream as "center" and "side" tracks or volume differentials (mid/side and intensity modes of joint stereo). Such methods take advantage of similarities between the stereo channels and can make stereo relatively cheap to encode over mono.

Bandwidth required

Shows how much space would be required per second of audio without compression (uncompressed PCM), in kilobytes per second (KB/s). CD-quality audio, for instance, requires 172KB/sec (176,400 bytes per second).

VirtualDub help - Dialogs: Audio volume



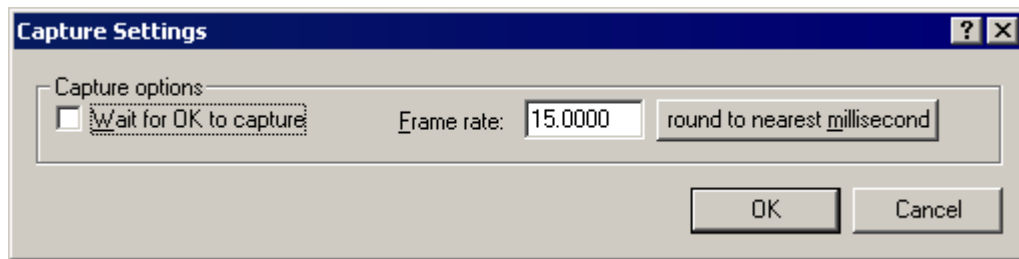
Permits volume adjustments to audio, to make the audio louder or softer.

Note VirtualDub hard clips when amplifying audio, so boosting the volume too much will result in distortion.

Note It is easier to make soft audio louder than to make loud audio softer -- if the source audio was so loud that it clipped during recording, introducing distortion, reducing volume after the fact won't remove the distortion. On the other hand, amplifying soft audio also amplifies quantization noise, which is particularly bad with 8-bit source. Try not to adjust volume more than you need to; it is better to record audio at the

right volume in the first place.

VirtualDub help - Dialogs: Capture settings



Controls basic video capture settings.

Wait for OK to capture

If checked, a dialog is displayed after capture is initialized. This minimizes click-to-capture delay.

Frame rate

Controls how fast video frames are captured.

Note Frame rates are rounded to the nearest microsecond period.

Round to nearest millisecond

Some capture drivers only allow frame rates corresponding to integer periods in milliseconds. With such a driver, for instance, the two closest rates to 15 fps are 14.925 fps (67 ms) and 15.152 fps (66 ms). This button allows you to see the actual rate that would occur.

Obsolete settings

Many settings that used to be in this dialog have been obsoleted or moved:

Capture audio

Now an option on the Audio menu.

Abort hotkey and button

The abort hotkey is now always Esc. Also, it is program-local (VirtualDub must have focus for it to work).

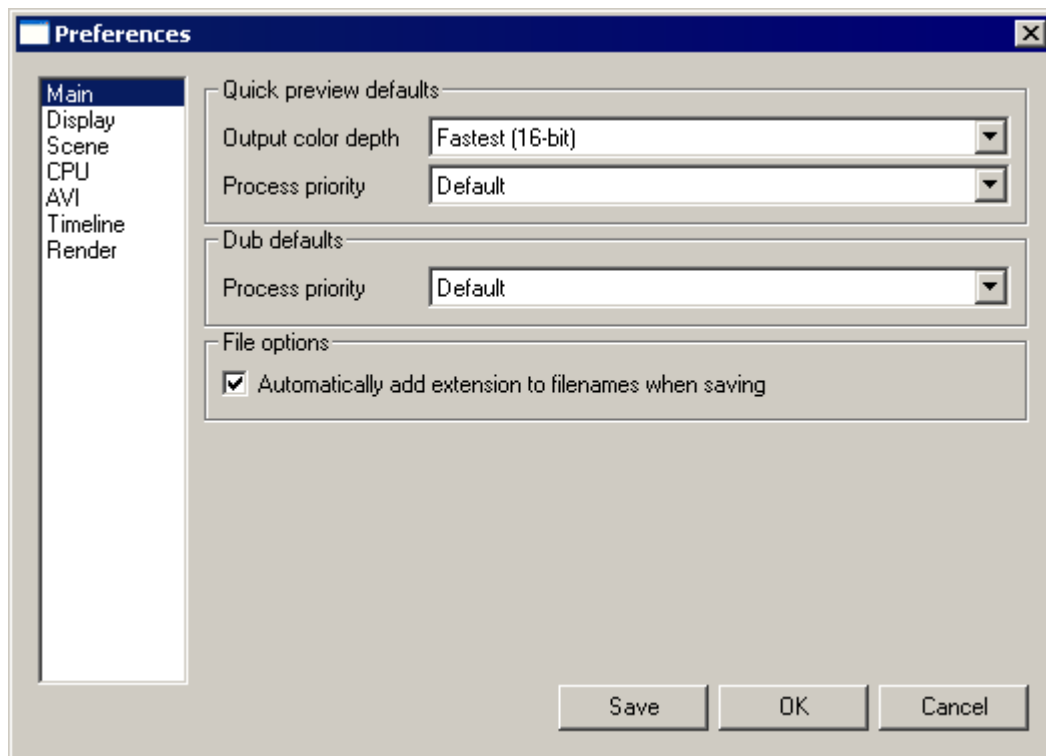
Audio and video buffer limits, audio buffer size

Set automatically.

Drop % limit, maximum index entries, lock video stream to audio

These options applied only to AVICap (compatibility mode) capture, which is no longer supported.

VirtualDub help - Dialogs: Preferences



Sets application preferences.

Main tab:

Output color depth

Selects the default color precision for video display -- 24-bit looks better, but 16-bit is generally faster. This does not affect file output.

Process priority

Force VirtualDub to be at a higher or lower priority than other applications when processing files. VirtualDub will always use CPU time that goes unused by other programs, but you can use this setting to force it to use more or less CPU when other applications need it as well.

Automatically add extension to filenames when saving

Controls whether file extensions (.avi) are automatically added when you type a filename without one in a save dialog.

Display tab (none of these options affect file output):

Enable 16-bit dithering

If enabled, 24-bit images are dithered when displaying in 16-bit to reduce banding, at the cost of a little speed.

Use DirectX for display panes

Enables accelerated video display for the display panes. This usually results in much better display performance as well as a better looking, smoother (interpolated) image. Disabling this option forces use of the safest but slowest display mechanism, which uses Windows GDI to render video images.

Use DirectX when Terminal Services is active

Enables accelerated video display over a remote session using Windows Terminal Services or Remote Desktop. Due to apparent bugs in the Terminal Server

implementation, allowing DirectX acceleration in this mode can cause video to always display in the upper-left corner of the desktop, so by default VirtualDub disables acceleration when it detects a remote session. This is often faster anyway.

Use Direct3D

Enables use of the Direct3D 9.0c accelerated 3D graphics API to display video. This enables use of the filtering options in the display panes to select between point sampling and bilinear filtering, and if the 3D device is powerful enough, bicubic filtering.

Use effect file

Enables use of a D3DX effect file (.fx) to specify custom Direct3D vertex and pixel shaders to display video. This requires an additional DirectX system DLL to be installed and knowledge of the effect file format to create the appropriate effects. Consult the [video shader reference](#) for details.

Use OpenGL

Enables use of the OpenGL accelerated 3D graphics API to display video. This enables use of the filtering options in the display panes to select between point sampling and bilinear filtering. Note that use of OpenGL when an accelerated 3D device is not present may result in substantial performance degradation.

Scene tab:

Interframe (cut) and intraframe (fade) thresholds

Controls the sensitivity of the scene forward/backward buttons on the position bar. A "scene change" is detected whenever there is a significant change in the image (cut), or scene details drop below a threshold (fade).

CPU tab:

Use default optimizations, or force specific optimizations

By default, VirtualDub automatically detects your CPU and chooses appropriately optimized code paths. If you experience problems due to incorrect detection, you can force specific codepaths on or off in this tab. Note that enabling an optimization not supported by your CPU will result in a crash or incorrect execution.

Note Video and audio codecs do their own CPU-specific dispatch, and are not controllable by these settings. If you experience problems with a codec that is not detecting your CPU correctly, you must contact the codec manufacturer for help.

AVI tab:

Restrict legacy AVI support to 1 gigabyte

Extended AVI files (AVI 2.0 or OpenDML AVI) are composed of two parts: a legacy AVI portion, and extended AVI blocks. This option drops the limit for the legacy portion from 2GB to 1GB, for applications that cannot handle AVI files between 1-2GB, and may be helpful if you have running applications that scan AVI files and choke on large ones. It has no effect on programs that handle extended AVI files.

Do not correct MPEG layer III audio streams

Some MP3 codecs do not generate MP3 streams at exactly the same rate as specified in their audio format when 44.1KHz sampling rates are used -- this causes a small discrepancy in audio sync of around 0.5%. By default, VirtualDub recomputes the data rate and automatically corrects the audio header when MP3 compression is active. This option allows you to disable correction if it is causing problems, such as an MP3 stream

being generated in a format that VirtualDub cannot parse correctly.

Directly decode YCbCr (YUV) sources

By default, VirtualDub will directly decode AVI video streams with uncompressed YCbCr formats that it recognizes, including UYVY, YUY2, YV16, YV12, I420, IYUV, Y41P, YVU9, Y8, and Y800, even if an external video codec would otherwise be used. This can result in better quality as VirtualDub uses bilinear interpolation when upsampling chroma. Disabling this option will disable YCbCr decode support and allow use of video codecs to handle such data.

Align large uncompressed frames to sector boundary

Some self-contained, professional-level hardware playback devices attain higher performance when working with AVIs that have video frames aligned to sector boundaries on the disk rather than the simple two-byte alignment required by AVI. Enabling this option causes sector alignment to be used, at the cost of slightly larger AVI files. The additional alignment is only in effect when a recognized uncompressed video format is used and the size of the video frame exceeds the specified threshold size.

Timeline page:

Timeline format

This option allows customization of the timestamp indicator on VirtualDub's main UI. Formatting specifiers — a percent sign followed by a letter — cause various pieces of information to be substituted into the given string. The permitted formatting specifiers are listed below the editable string.

Render options:

Confirm when abort button is pressed

When enabled, attempts to abort a preview or save operation will cause a confirmation dialog to be displayed before actually aborting.

VirtualDub help

So, you managed to crash VirtualDub...

VirtualDub attempts to diagnose the immediate cause for your crash in the initial crash dialog that is displayed; this is usually helpful in determining the cause of your crash. While reading the context dump, please note the following:

- The analysis is not always correct. Sometimes a failure causes the CPU to go off into nowhere and the analyzer reports misleading information.
- VirtualDub reports the immediate cause of the crash, but that may not always indicate the true cause, if the failure is delayed.
- Third-party drivers can cause VirtualDub to crash. This includes capture drivers, video codecs, and audio codecs.

VirtualDub produces a crash dump file (`crashinfo.txt`) that can be of help to the program author when diagnosing a crash, as noted in the crash dialog. However, before submitting a crash dump, please attempt the following diagnosis steps:

- *Upgrade to the latest version of VirtualDub if one is available.* Newer versions almost always fix bugs. The *Help / Change Log...* menu option will tell you what fixes have been applied.
- *Try to reproduce it.* Crashes that only happen once aren't fixable.
- *Remove third-party drivers.* If you are using audio and video codecs, try using no codec at all and see if it makes a difference. Third-party codecs are the #1 cause of crashes within VirtualDub and it is very helpful if you can identify the codec as a trigger, so that the root cause of the problem is more easily found. Also, please attempt to update to the latest version of any drivers, as the bug you are experiencing may already have been fixed.

Note If VirtualDub's context info tells you that an instruction not supported by your CPU was executed in a particular codec, while that codec was being used, 99.9% of the time the cause is the codec not properly checking your CPU before executing optimized code. VirtualDub does not have control over this and you must consult the codec manufacturer for technical support in this case. Make sure your CPU meets the stated minimum requirements for that codec.

- *Determine if a particular setting is implicated.* If a particular filter is needed to trigger the crash, I need to know that.
- *Check your input.* Many crashes in codecs are caused by AVI files being damaged during transmission over the Internet. Suspect this has occurred if you are consistently getting crashes on a particular video frame.
- *Un-overclock and check system temperature.* A rare, but nasty, case of crashes is an overclocked CPU, or worse, an overheated CPU (frozen fans are not unheard of). If you are overclocking, please attempt to downclock to the correct speeds for your CPU and motherboard before reporting a crash.