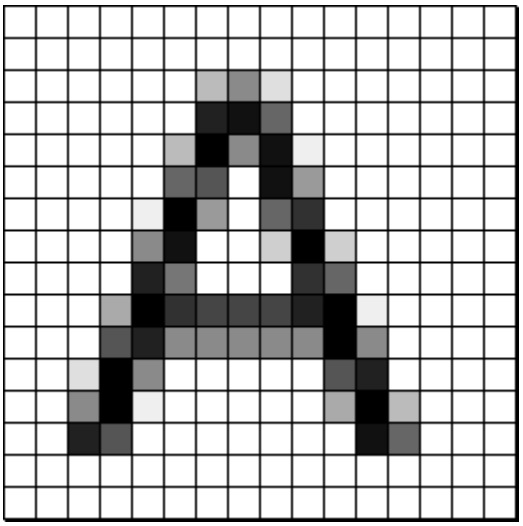
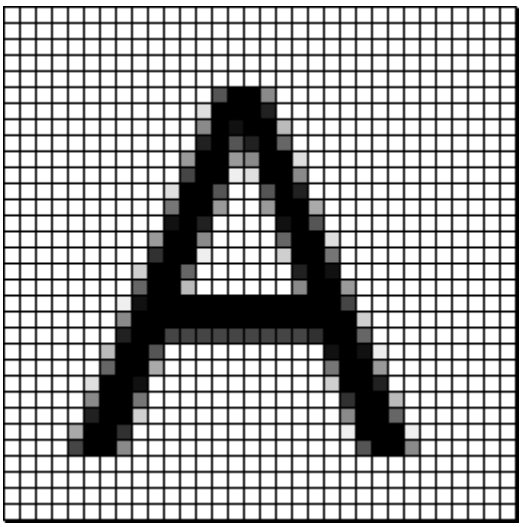
Pixels and images

Pixel stands for *picture element*. Any image (picture) on a computer is made up of a grid of pixels arranged in rows and columns:



Enlarged Normal

Each square pixel holds a single color or shade of grey. The number of pixels determines an image's *dimensions*: the above image is 16x16, because it has 16 columns and 16 rows. How large the pixels are determines the image's *resolution*: how finely detailed the image is.

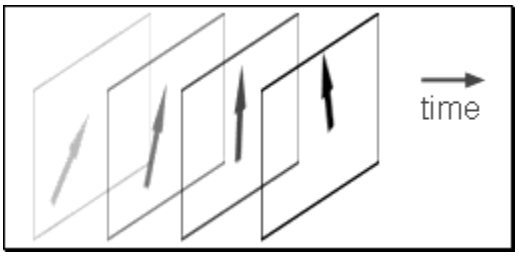


Enlarged Normal size

Here we have doubled the image's resolution, and the image looks better as a result. Because the image is the same size, the image's dimensions have also been doubled to 32x32.

Frames and framerates

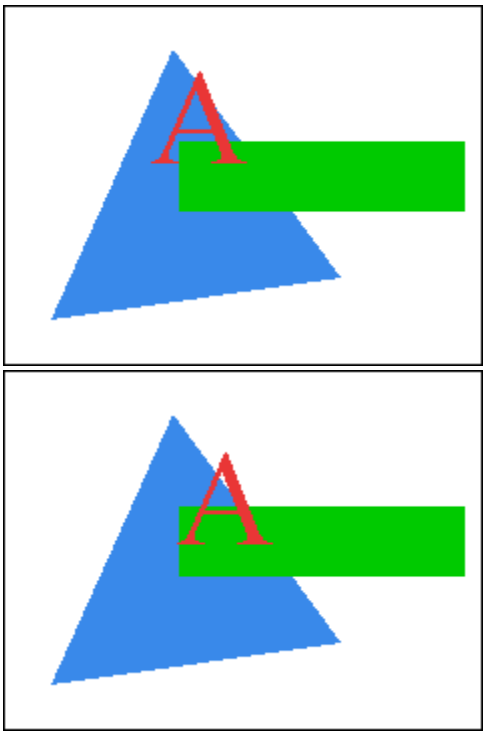
Digitally stored video consists of a series of images, or *frames*. By playing a series of frames in sequence like a flipbook, we can simulate motion:



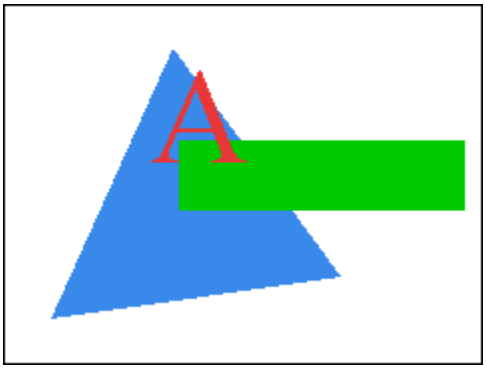
The speed at which successive frames are played is measured in *frames per second (fps)*. The more frames per second, the smoother the video is; professional movies are filmed at 24 or 48 fps, and television runs at 25 or 30 fps. Because high framerates take a *lot* of space for all the individual frames, a lot of computer video only runs at 10 or 15 frames per second, which may look a bit jerky to some people.

Keyframes vs. delta frames

Playing a bunch of individual frames is fine, but it's a lot of trouble to keep track of all the separate frames. For instance, take these two frames:

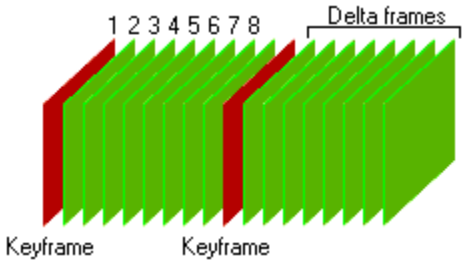


The only difference between these two frames is that the red A moved down and to the right, and yet we have to store the whole picture for both frames -- including the blue triangle and green square which don't change. But what if we stored the frames like this:



The first frame hasn't changed, but instead of storing the second frame, we make a *delta frame* that only stores the differences between the two frames, and store the delta frame instead. Now the second frame only takes up a tiny bit of space, because only a small part of the frame changed. The first frame is now a *keyframe*, because it is the key to decoding the second. When we add a third frame, we can do the same trick and store the differences between the second and third frames.

So what's the drawback to delta frames?



The drawback is that to decode a delta frame, we need to have the frame before it. But if that frame is also a delta frame, we need to know the frame before that, and so on. In fact, we'd need to go all the way to the last keyframe, and start decoding delta frames one-by-one until we got to the frame we wanted. For instance, let's say we wanted frame 8. Frame 8 is a delta frame, so we need to decode frame 7. Frame 7 is also a delta frame, so we need 6. We'd need to decode frames 1-6 before we could get to frame 7. There's another keyframe at frame 10, though, so if we needed frame 12, only 10 and 11 would need to be fetched. We can reduce the problem by putting in keyframes more often, but keyframes are usually bigger, and so our video file would get bigger as well. Most video files have keyframes every 5-30 frames.

There's another reason to watch the keyframe interval. Delta frames aren't normally stored exactly; they approximately reconstruct the original frame. This saves a lot of space, because tiny details that take up a lot of space can be tossed. Unfortunately, this sometimes means that the image degrades as more and more delta frames are generated, and so the longer the keyframe interval, the worse the image gets between keyframes.

It all seems so big…

…and it is. Raw, uncompressed video takes anywhere from 1-3 bytes per pixel, and so a single 320x240 frame takes from 77K to 230K of space. 16-bit uncompressed video is common and takes 2 bytes per pixel, so that's a whopping 2.3 megabytes of space per *second*. That's a ton of space!

There are three ways to save space. The first is to shrink the picture. Shrinking a video file from 320x240 to 160x120 makes it look half-size (or half-resolution) and so it doesn't look as good. But you have half the columns *and* half the rows, so it actually takes *one-fourth* the space. Unfortunately, it also works the other way around: 'doubling' the size to 640x480 takes four times the space. Nowadays, any reducing video below 240x180 is usually too small to use, and 320x240 or 352x240 is the norm. Above 320x240, video very quickly takes too much space and computing power to process. There is no way you're going to store your full-screen 800x600 Quake II game on disk as a video file.

The second is to reduce the framerate. There are no tricks here; half the framerate leads to half the data. However, the more you reduce the framerate, the more quickly people notice. People won't notice if you drop half the frames in a 30fps video and make a 15fps one instead. But they'll cry murder if you drop half the frames again and make a 7.5fps video. You can't cleanly drop the framerate by a fraction; for instance, reducing a 15fps video to 10fps doesn't work so well because you have to drop one frame out of every three, making a jerky video. The best solution is simply to figure what framerate you want, and work at that framerate to begin with.

The third is compression. We'll cover that next.

Video compression

You can't take raw video and throw it through ZIP or RAR. Regular compression algorithms are *lossless* and have to give back exactly what they packed; it would be very bad if a program file came out different than when it went in. But lossless compression doesn't work for video; even though your eye can't tell the difference, there's often a little noise or tiny details that the computer knows about. Lossless compression algorithms usually can't pack video down to less than one-half original size.

The solution is *lossy compression*; figure out what the eye doesn't care about, and throw it away. Then figure out the best way to store what's left. Instead of one-half, we can now compress down to *one-tenth to one-twentieth* original size. The decompressed picture doesn't exactly match the original, but it's close enough and so much smaller than basically every video compression algorithm is lossy, including Intel Indeo, MPEG, Cinepak, Sorenson, and MJPEG.

There's another big advantage to lossy compression: you can control how big you want the video file, by controlling how much data is thrown out. Of course, the more you compress, the lousier the data looks. At a certain point, more compression isn't worth it, because reducing the video size by even 10% makes it look 10 times worse. The *compression artifacts* that result from high levels of compression vary from algorithm to algorithm. Some methods cause the picture to get blurry, and others add edges and specks to the picture.

One quick note: sometimes you'll see MPEG advertised as getting 100:1 or similar ridiculous compression ratios. This is complete B.S.; realistic ratios for MPEG are around 26:1, according to people who *created* the MPEG standard. Real 100:1 compression results in video that looks more like colored bathroom tiles.

Compression artifacts

You've taken a 100Mb raw video masterpiece, and packed it down into a 5Mb file using Intel Indeo or some other compression codec. When you play it back, though, some parts are too blurry to see, and other areas have blocky edges, and large, flat gradients have stairsteps on them. What happened?

The compression algorithm threw out a bit too much data. The codec couldn't store all the information about the gradients, edges, and shapes in the picture, so it started discarding parts of the picture until it could. All of the artifacts in the compressed video are a result of this discarding.

Blurriness

Blurring an image simplifies it, because sharp details become smooth shapes and curves. As a result, when a compression algorithm is short on space, it usually tries to figure out which details in the picture are the most important and should be saved, and blurs less important details. If a particular spot in the picture frame has sharp details but doesn't change for several seconds, some compression algorithms will figure that out and only encode the details once. But if that spot moves, the algorithm might not be able to efficiently detect or store the movement, and just blurs it instead.

Stairstepping and blockiness

What happened here? The artifacts in the compressed image are the result of *tiling* or *blocking*; most compression algorithms split the picture into 4x4 or 8x8 pixel tiles to speed up compression. Tile boundaries don't become visible until you lower the compression quality greatly; then the compressor starts to drop so much data that adjacent tiles change very differently, resulting in a visible boundary between them.

Stairstepping is an extreme case, where the codec discards all the data about a tile except for the average color of all the pixels in the tile, reducing any detail in the tile down to a flat block of color. Large, flat gradients of color, like sky, tend to be reduced to blocky stairsteps at high compression ratios.

Warning: About to save an uncompressed file

By default, VirtualDub does not automatically recompress video data that it processes, resulting in raw video data being written to the AVI file. This results in the absolute highest quality, because the processed output is written straight to disk, but also results in huge files. In many cases, writing uncompressed data slows down processing because of the sheer amount of data being written to disk.

The simple fix is to enable video compression:

- Under the *Video* menu, make sure *Full Processing Mode* is set.
- Select *Video, Compression…* and choose a suitable video codec.

Warning: Video compression not active

Direct Stream Copy mode tells VirtualDub to send data straight from input to output. This is very fast and makes sure the output video looks exactly like the video, which is very useful for audio processing. However, few of VirtualDub's processing functions are available in this mode.

To enable video compression, the video mode must be set to either *Normal Recompress* or *Full Processing Mode*.

Warning: Outdated compression codec

You have selected a video compression algorithm which is inferior to other available codecs. By coincidence, the outdated codecs listed here have been included with Video for Windows ever since Windows 3.1, and are the only ones shipped with the original retail version of Windows 95, so you may want to ignore this warning and compress anyway.

Microsoft Video 1 This codec produces large files that appear very blocky, and more importantly, *cannot produce a perfect frame no matter how much space you throw at it*. It had appeal during the days of the 80286 and 80386 CPUs because of its high speed, but finds little use today.

Intel Indeo R3.2 Intel's first widely available codec. It's very blocky with highly saturated color gradients, and produces dithering in its output. It has been shadowed by Intel's own newer codecs; Indeo 4.1/4.5 is available on both Windows and Macintosh, and 5.0/5.1 for Windows. The Indeo 5.1 codec decompresses slower but compresses several times faster than its predecessors, and with much improved quality.

SuperMac/Radius Cinepak The long-standing favorite of Macintosh users before the appearance of Intel Indeo 4 for QuickTime and the recent reigning champion, the Sorenson codec. This codec decompresses several times faster than Indeo 5 but is slower at compression and produces blockier output. It attempts to intelligently mark scene changes with keyframes, but in general the output of this codec just sucks in both quality and size.

Warning: Audio compression inactive

The audio mode is set to *Direct Stream Copy* mode, so the audio track in your input file will not be touched, and the current audio compressor will be ignored. To use the audio compressor you have selected, the audio mode needs to be set to *Full Processing Mode*.

Warning: Filters inactive

VirtualDub's filters only activate in *Full Processing Mode*. Most filters modify video data too extensively for any hope of preserving the original compressed video, and require recompression. If you have not already set one, you will need to select a video compression codec as well with the *Video, Compression…* menu option.

Warning: Video compression quality >90%

Most video codecs are designed to work at a moderate compression level, and work less efficiently as you increase the compression quality level; the step from 90-95% usually takes more space than the step from 80-85%. Usually, reducing quality to 90% or so saves a lot of space without visible change in the output video.

Warning: No keyframe interval set

Video players need to rewind to the last keyframe and ‘fast forward’ when you seek to the middle of a file. If there are no keyframes in a file, the player has to fast forward *from the beginning every time you seek in the video*. This can take more than five minutes in a large file. For this reason, keyframes are almost mandatory in a video file.

To alter the keyframe interval, select *Video, Compression…* from the menu, make sure ‘keyframes’ is checked, and enter in a non-zero value.

Warning: Output color depth is set to 16-bit

VirtualDub has been set to 16-bit output for either speed or space savings. When video compression is enabled, this is usually undesirable – it won't save any space, and usually adds banding to the file. You can fix this by using the *Video, Color Depth…* option.

Warning: Audio interleaving is off

Audio interleaving puts small chunks of audio in between video frames in the file, so the player can walk through the file in one pass and pick up both audio and video as it goes. Without interleaving, the player must jump back and forth between audio and video portions of the file. Enable audio interleaving through the *Audio, Interleaving…* menu option.

Warning: Fast repack mode selected

Fast repack mode can significantly speed recompression time if the video codecs are able to negotiate a common YUV format. However, some codecs do not handle YUV properly, occasionally resulting in upside-down video. If this happens, switch to slow repack or full processing mode.

Warning: Attempt to use a hacked compression driver

The “DivX” codec is a hacked version of the Microsoft MPEG-4 codec. Because this is a modified binary, there is always the possibility that the changes to the codec may have destabilized the driver, causing VirtualDub to act erratically or crash. If you decide to continue using this codec, you’re completely on your own with any problems you encounter.

Warning: Using MPEG-4 V3 in an AVI file

The Microsoft MPEG-4 driver was a part of Microsoft's NetShow package, which has now become Windows Media Tools and Player. Beginning with build 4.00.3845 of Windows Media, this format cannot be used in AVI files; the drivers will refuse to compress and decompress the format. If you generate an MPEG-4 V3 stream in an AVI file, Windows Media Player will not be able to find an appropriate decompressor for it and will refuse to play the file. However, you can convert the resulting AVI file to a playable ASF using Microsoft's `vidtoasf` tool.

If this is troubling to you, consider using MPEG-4 V2. This format is close to or similar to V3 quality-wise, has better speed, particularly on non-Intel CPUs, and can freely be used in AVI files. You may have to upgrade to the latest version of Windows Media Tools, however, to enable MPEG-4 V2 compression.

Warning: Miro/Pinnacle capture drivers detected

Some miroVIDEO capture cards from Pinnacle Systems, such as the DRX/DC10, resort to an emulated overlay when used with a graphics card that cannot provide a true video overlay. This “primary overlay” mode is still very fast but can result in a lot of flickering, garbage on screen, and very slow GUI window operation, even on fast Pentium II systems. Disabling overlay, by switching to preview mode or turning off the capture view altogether solves the problem. This doesn’t occur if the DCxx is using a true video overlay on the graphics card.

VirtualDub has built-in workarounds to alleviate this problem. To enable the workarounds, select the *Preferences…* dialog, select the miroVIDEO driver, and click the *Disable overlay for menus and dialogs* option. VirtualDub will then automatically disable and re-enable overlay as appropriate to speed up the user interface.

Warning: Zoran drivers detected

Some TV tuner cards are modified Zoran H20 reference designs, and at least one device – the Iomega Buz – is based on the H22 design. Drivers for these cards that were written by Zoran or were based off of such drivers cannot time frame grabbing more accurately than the nearest millisecond. As a result, some of the popular framerates – such as 15.000 and 29.970 fps – cannot be hit with this driver, and attempting to do so can result in dropped frames and/or jerky capture files.

The only known workaround at this time is to adjust the capture framerate to the nearest millisecond. There is a button in VirtualDub's *Settings…* dialog to do this. Adjusted framerates, such as 15.151 and 30.303 fps, are also available in the quick framerate menu. The *Average Frame Rate* field in the side panel will tell you if the capture card is capturing off-rate.

Warning: Brooktree Bt848/878 drivers detected

Depending on the modifications your capture card maker has made to the Brooktree reference drivers, you may have to manually disable overlay to capture above 240 scanlines, i.e. 640x480. You may also get field swapping when capturing in the YUY2 (YUV 4:2:2) format.

On the good side, the reference drivers allow arbitrary image sizes (352x240) and some additional YUV formats (UYVY) that may not be exposed by your particular driver's Video Format dialog box. These extra features, if present, can be accessed through VirtualDub's *Set Custom Format… (Shift-F)* function.

Video codec information: delta frames

Set to **Yes** if the compressor produces delta frames. Delta frames record only differences from the previous frame, so all frames from the last key frame need to be decompressed first to view a delta frame. Some compressors set this value to **No** even though they support delta frames, such as the MS MPEG-4 series of video compressors. These codecs internally perform more advanced key frame analysis.

Video codec information: FOURCC code

All video compression drivers respond to one or more unique *fourcc* (four character code) IDs that uniquely identify the video compression algorithm. For instance, any video stream tagged with '`IV50`' is compressed with the Indeo 5 algorithm. This information is useful for identifying driver conflicts, manually installing a codec driver, or forcing a video stream to another compatible driver.

Video codec information: Driver name

This is the name of the DLL that contains the video codec. Usually a video codec can be installed by adding a `vidc.`*`fourcc`* `=` *`driver name`* line to system.ini under the `[drivers]` or `[drivers32]` sections. This field also shows you if a driver is handling more than one format.

Video codec information: Format restrictions

When you select a video compression algorithm, VirtualDub will probe the driver and attempt to figure out which formats it can and cannot accept. For instance, codecs which block the image into 8x8 tiles may refuse to compress if the width and height are not multiples of 8. Due to limited error reporting capability in the Video for Windows architecture, the driver cannot tell you what part of the image format is wrong. Use the information from this box to determine an output size that the compressor will accept.

Quality setting

The quality factor for video compression has several meanings. Ultimately, the compressor can interpret this value any way it wants – even ignore it – but there are two general uses for this value:

If you do not have the "data rate" box checked, this slider usually sets a threshold for compression. The video compressor will then use just enough space for each frame to hit this quality level. More complex scenes use more space, and the size of the output file cannot be determined until processing has finished. A higher quality level corresponds to a better quality frame. This mode (thresholding) is sometimes faster than data rate control.

If you have the "data rate" box checked, this slider determines how hard the compressor works to hit that data rate. Lowering the slider speeds up compression, but can result in video streams over bitrate and/or low quality output. With data rate control, this value should almost always be set to 100%.

Target data rate

Some compressors support *data rate control* as an alternative to thresholding. Rather than using a varying amount of space to hit a fixed quality, the compressor varies quality to maintain a constant data rate. This has a number of advantages, including a predictable file size, much better quality during “quiet” scenes, and dependable CD-ROM playback due to the constant data flow.

To use this feature, type in the desired data rate in kilobytes per second (K/s or Kb/s). For instance, a 1X CD-ROM rate would be no higher than 150 Kb/s. The video compressor will then do its best to make sure that the video stream does not exceed 150 Kb/s *on the average*. The video stream may end up slightly larger or smaller than expected depending on the compressor’s ability.

Key frame interval

Enable this value to guarantee key frames at specific intervals throughout the file. When seeking in a file, players must play to the desired point from the previous key frame, so larger key frame intervals increase seek time. Key frames also tend to be larger and higher quality than compressed ones. In general, use a value between a quarter of a second to a 3 seconds for a good balance between seek time, image quality, and file size.

If this feature is not enabled, the only guaranteed key frame is the first frame in the stream.

Sampling rate

Select a new sampling rate for the output audio here, or choose **No change** to leave the audio sampling rate alone. Raising the sampling rate doesn't help quality, but increases the audio track size; lowering the sampling rate diminishes quality but also lowers the track size. However, you may need to force a conversion here if an audio compressor cannot accept the current audio format. If you don't want to change the rate, be sure to choose *No change* rather than clicking the appropriate sampling rate because the sampling rate might be slightly off, as in 22047 Hz, causing the stream to be converted to 22050 Hz.

Depending on the audio input, sample conversion can introduce noise into the output. If the audio contains a lot of high-pitched sound, such as a high female voice or percussion, you may want to click the **High Quality** box to enable filtering on the output. For even better quality, extract the audio track and perform the conversion using a good sound editor.

CD quality audio has a sampling rate of 44100 Hz.

Sampling rate: Integral conversion

Check this box to force the new sampling rate to be an integral multiple or division of the old rate. For instance, if you have an audio track with a slightly nonstandard sampling rate, such as 22047 Hz, choosing 11025 Hz with integral conversion enabled converts the track to 11023.5 Hz. Similarly, the 44100 Hz setting would cause a conversion to 44094 Hz instead of 44100.

Sampling rate: High quality

This option enables a triangular Tent filter on the resampled output. When upsampling to a higher rate, the filter reduces quantization noise and hiss. When downsampling to a lower rate, the filter attenuates high frequencies that are normally aliased to lower frequencies. Without the filter, noisy high-frequency sounds like the hiss of a trailing *s* can drop in pitch and drown out the audio.

Precision

Allows you to change the precision of audio samples. If the volume is sufficiently well adjusted, lowering the audio sample precision from 16-bit to 8-bit halves the audio track size with little difference in quality. You may have to expand an 8-bit track to 16-bit for audio compression; however, it is better to start with 16-bit audio to begin with. Never reduce a 16-bit track to 8-bit before audio compression, if possible.

CD audio is 16-bit.

Channels

Allows you to mix a stereo track into a single mono track, halving the track size, or duplicate a mono track to form a stereo track, doubling the size. Conversion to mono is performed by averaging the two samples together. Stereo tracks take twice the space, so if the audio is highly central, such as voice, then consider mixing to mono for the final output.

CD audio is stereo.

Bandwidth required

Tells you how much space is required for each second of uncompressed (PCM) audio at the selected values. For instance, dropping a CD-quality track to 22KHz, 8-bit mono reduces the bandwidth requirements from 176K/s to 22K/s, saving a lot of space. Reducing the sampling rate to 11KHz again would only save another 11K/s of bandwidth, and make a much smaller difference in file size.

Video capture troubleshooter

{button ,AL(`trouble_cap_droppedframes')} I’m getting dropped frames during video capture.

{button ,AL(`trouble_cap_deviceabsent')} My capture device isn’t on the list.

{button ,AL(`trouble_cap_overlay')} I can’t get overlay working.

Video capture troubleshooter: dropped frames

Choose the option that best describes the problem.

{button ,AL(`trouble_cap_dropstart')} dropping frames immediately at start

{button ,AL(`trouble_cap_dropcont')} occasional dropped frames throughout capture

{button ,AL(`trouble_cap_droppoint')} frames being dropped at specific points in input video

Video capture troubleshooter: dropped frames at start

It's not unusual to get one or two frames dropped at the start of the capture operation, to align the video to the audio track. Usually this is the very first frame in the video, so no video data has been lost. However, a large drop of 10 frames or more at the start could mean VirtualDub is having trouble starting the AVI file. In this case, consider pre-allocating space through the Allocate option in the File menu, or enable the "wait for OK before starting capture" option in the Settings dialog.

Video capture troubleshooter: occasional dropped frames throughout capture

Try the capture operation using F7 (test capture). This simulates the capture operation without writing data to disk.

{button ,AL(`')} Frames are still being dropped.

{button ,AL(`')} Few or no frames are being dropped.

Video capture troubleshooter: Source or device-based frame drops

If you

Video capture troubleshooter: CPU or timing-based frame drops

Because test mode (F7) bypasses the disk, not dropping frames in Testing mode usually means your disk I/O system is not keeping up with the capture operation. If you are using AVICap mode, try switching to Internal mode. If you are using Internal mode, check the Disk I/O settings. Consider increasing the number of cache chunks to 4 and changing the cache chunk size.

This can also be a sign of excessive CPU usage. If you are using video compression, reduce the load by switching to a faster codec or tweaking the quality and keyframe intervals. Also, if you know how, check to see if bus mastering (DMA) is enabled on your capture drive.

Timing drops are rare but consistent. Constantly drifting jitter and displacement values, shown in the status bar at the bottom of the window, indicate drifting video frame timing. Subtly adjusting the frame rate until these values stay nearly constant should help. If not, dropping one or two frames every few minutes is not a big concern.

Hello.

This is the main documentation and help file for VirtualDub, the AVI capture and processing utility, version 1.4d. Documentation is still very spotty at this point, particularly for the main part of the program (capture mode is better documented). Most of the information is context help, which can be accessed by clicking the question mark buttons and selecting controls.

The MPEG video decoder code is based off the Java MPEG player by Joerg Anders. The source code is available at `http://rnvs.informatik.tu-chemnitz.de/~ja/MPEG/MPEG_Play.html`.

1.0 now uses a new MPEG audio decoder core, NekoAmp. NekoAmp is ©1999 Avery Lee as well (grin), and is also GPL. It contains some heavily modified code from FreeAmp and the MPEG reference decoder.

Complete source code to VirtualDub is available, free of charge, under the same license. At the time of this writing, the URL for this is:

`http://www.virtualdub.org/`

Please check this website for news and updates.

GNU General Public License (GPL)

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying

the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> <one line to give the program's name and a brief idea of what it does.>
> Copyright (C) 19yy <name of author>
>
> This program is free software; you can redistribute it and/or modify
> it under the terms of the GNU General Public License as published by
> the Free Software Foundation; either version 2 of the License, or
> (at your option) any later version.
>
> This program is distributed in the hope that it will be useful,
> but WITHOUT ANY WARRANTY; without even the implied warranty of
> MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
> GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License
> along with this program; if not, write to the Free Software
> Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this

when it starts in an interactive mode:

```
    Gnomovision version 69, Copyright (C) 19yy name of author
    Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.
```

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

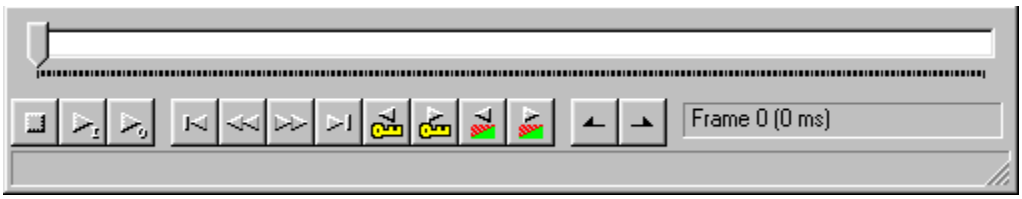
```
  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  `Gnomovision' (which makes passes at compilers) written by James Hacker.

  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Position Control

The position control appears in VirtualDub's main window and in some video dialogs to help you browse through frames and choose the ones you want to work with. All controls are available in the main window, but in dialogs the playback, scene shuttle, and marking controls are often absent.



Click on a control or button on the position control for help on that item.

Stop

Halts a currently running preview or dub. This button only appears in the main VirtualDub screen.

Play input

Starts playing the input video at the current position. Most of VirtualDub's processing features are disabled, including range selection. This button only appears in the main VirtualDub screen.

Play output

Starts playing processed video from the current position. All of VirtualDub's processing features are functional. Preview begins at the current position, will end at the end marker if you have placed it. This button only appears in the main VirtualDub screen.

Start

Moves the current position to the start.

Back

Backs up by a single frame.

Forward

Advances by a single frame.

End

Moves the current position to the end.

Key reverse

Backs off to the previous keyframe. With MPEG files, this jumps to the previous I-frame

Key forward

Advances to the next keyframe. With MPEG files, this jumps to the next I-frame.

Scene reverse

Shuttles backward to the previous scene transition. VirtualDub will then guess where the next transition is and stop, according to the thresholds set in *Preferences*…. You can click this button again to stop the shuttling or click *scene forward* to switch direction. This button only appears on the main VirtualDub screen.

Scene forward

Shuttles forward to the next scene transition, determined by thresholds set in *Preferences*…. Click the button again to halt, or click *scene reverse* to switch shuttle directions. This button only appears on the main VirtualDub screen.

Mark in

Sets the beginning of a range of frames to delete. Appears only in the main VirtualDub screen.

Mark out

Sets the end of a range of frames to delete. Appears only in the main VirtualDub screen.

Position slider

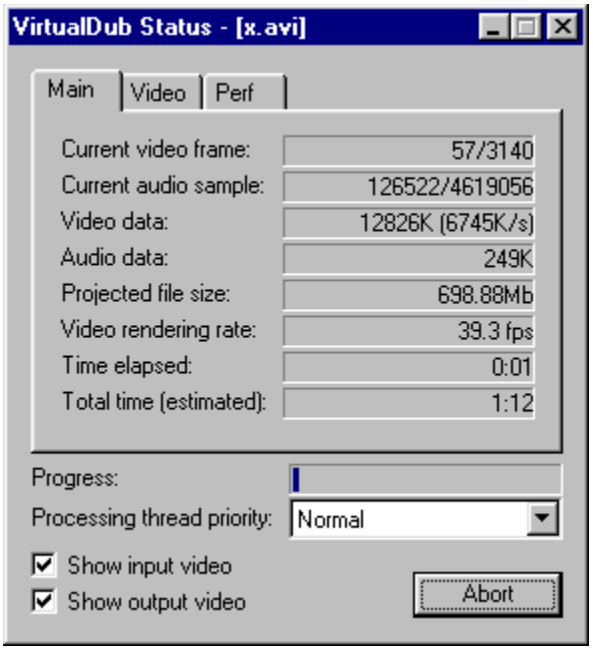
Controls which frame you're on; drag to move around in the source file. Tick marks will appear every frame, or a black stripe if you have a lot of frames.

Information panel

Tells the current frame number and time index from the start, in milliseconds.

Monitoring dub status

When a processing operation starts, VirtualDub will pop up a status window showing the progress of the dub.



Click on a region for an explanation (Ctrl-Tab to show clickable areas). Click on a tab to switch to another status page in the window. When VirtualDub is minimized, the percent complete will show up in the taskbar.

Current video frame and audio sample

Displays VirtualDub's progress on the input streams. This allows you to check, for example, if range clipping and edits are active. A one second clip shouldn't need 1000 input frames and 4 million audio samples. However, frame rate decimation doesn't affect the total count; if FRD is set to 3, then the video frame total will stay the same, but the current frame will count up by threes.

Video and audio sizes

How large the video and audio streams in the output file are, at this point. This allows you to check which stream currently occupies more space. Normally, the audio stream will be much smaller than the video. VirtualDub also displays the average bandwidth of the video stream in kilobytes per second (K/s), so you can check if the video compressor is meeting its data rate quota or not. Generally, most will come fairly close by the end of the file.

Projected file size

The estimated final size of the output file, based on how big the video and audio streams are at this point. The audio portion of this statistic is usually very stable, but the video frames can vary in size, and this value may swing up or down depending on how well the remainder of the input compresses. Thus, this value will sometimes be accurate to the kilobyte, and other times it can be off by a few megabytes at first.

If this value shows a few gigabytes after several minutes when you expected a 100Mb file, though, then you should probably check your settings.

Video rendering rate

Shows how many source frames VirtualDub is processing per second. It's normal for this value to vary slightly, more so in Windows 95/98 than in NT. Obviously, when dubbing, you want this value to be as high as possible.

Time elapsed / Estimated Time Remaining

Displays how long the dub has gone and the estimated total time in *days* : *hours* : *mins* : *secs*. The estimated time is continually updated; it will vary a lot at first but after a few minutes it settles to a stable time. If the computer speeds up or slows down though, usually due to other programs, the estimated time will change accordingly.

Progress bar

Shows how far along VirtualDub is in the current operation. You know, the bar grows right as more is done, and when it's full, everything's done….

Processing thread priority

Controls how much CPU time VirtualDub receives relative to other tasks in the system. Normally, VirtualDub gets CPU time roughly equally with other tasks. If its priority is raised, it will tend to get more CPU time, and if its priority is dropped, it will tend to get less. This is particularly useful if you want to compress in the background but don't want the system bogged down too much. Note that if needed, VirtualDub will suck up any leftover CPU time that other tasks don't use, regardless of this setting, so if your system isn't running any other tasks, it's generally not useful to boost this value.

Show input/output video

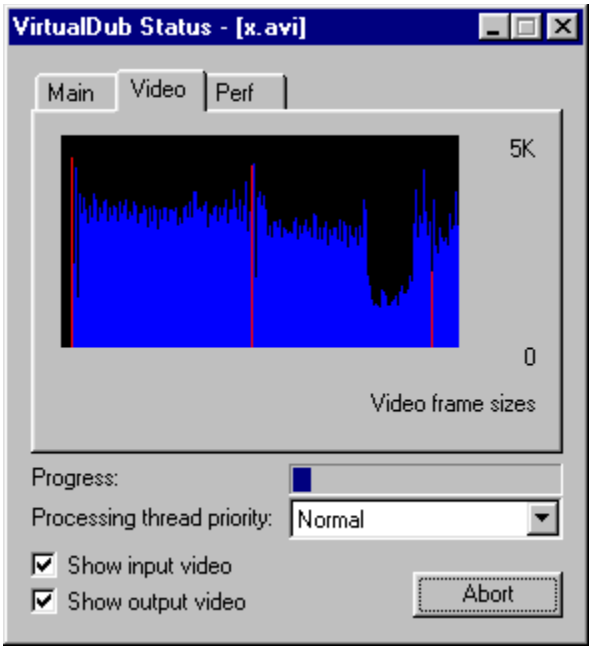
Enables or disables the display of input and output video panes. During preview, turning off the input pane can speed up the preview if the CPU isn't powerful enough to draw both panes. In processing mode, VirtualDub only draws both panes every half-second, so turning them off gains very little speed.

Abort operation

Stops the preview or dub operation. In the case of a dub, the output file will be finalized and properly closed, if possible, to create a fully usable file. All processed data up to the abort point will be valid. Use this option with care, since an aborted operation cannot be resumed.

Monitoring dub status

When a processing operation starts, VirtualDub will pop up a status window showing the progress of the dub.



Click on a region for an explanation (Ctrl-Tab to show clickable areas). Click on a tab to switch to another status page in the window. When VirtualDub is minimized, the percent complete will show up in the taskbar.

Video frame size monitor

This window is a moving bar graph showing the output frame sizes as video frames are compressed. Red bars are key frames, and blue bars are compressed delta frames. The taller the bar, the larger the frame. The graph automatically switches scales to accommodate the frame sizes.

In preview mode, or when uncompressed video is being output, this graph will be a solid block of red.

Monitoring dub status

When a processing operation starts, VirtualDub will pop up a status window showing the progress of the dub.

Click on a region for an explanation (Ctrl-Tab to show clickable areas). Click on a tab to switch to another status page in the window. When VirtualDub is minimized, the percent complete will show up in the taskbar.

Default Capture Driver

You can tell VirtualDub to either use the First Available driver for capture, or to request a specific driver on startup. If you choose the latter, the driver's name is stored in the Registry, and VirtualDub will select that driver even if you change capture driver numbers by adding or removing capture cards.

Per-driver settings

Settings which can be adjusted for each capture driver. These settings go into effect when the capture driver is selected. They are saved by capture driver name, and will be matched properly even if driver IDs change. Note that if two capture cards have the exact same name, the higher ID card will override the settings of the lower ID card, and the two cards will use the same settings.

Initial Display Mode

Selects which display mode (preview/overlay), if any, that you want the capture driver to appear with when selected. Some capture drivers spray garbage on the screen when overlay mode is first started, or impose irritating delays.

Slow Preview/Overlay mode

Some capture card drivers, such as the miroVideo DRX, have trouble rendering real-time overlays to certain video cards. As a result, overlay video may be incorrectly rendered, or Windows may slow down, when windows, dialogs and menus appear on screen. The *Slow Overlay* mode disables overlay when menus and dialogs are selected from VirtualDub's menu to avoid this problem. Most TV tuner cards will not have this problem, though.

Preview mode requires that the CPU manually request frames and then submit them for decompression. This can cause the interface to pause if hardware support is continually flipping between capture and decompression. Choose *Slow Preview* to alleviate this situation.

Frame rate

Selects the number of frames to be captured per second. Note that this value is not adjusted for the characteristics of the input device; if you select a frame rate of 30.00 fps against a live video signal with a 59.94Hz refresh, you'll probably get a 30.00 fps video stream with one frame duplicated every 33 seconds.

Limit capture length

When enabled, this option causes AVICap to stop the capture after a specified duration. If the approximate bandwidth and disk limits are known, this option is useful for making sure you do not run out of disk space; if this happens during a capture, the index block cannot be written out, and seeking in the resultant capture file will be very painful.

Drop limit

Specifies the maximum rate at which AVICap is allowed to drop frames before aborting streaming capture. This value is ignored for normal capture mode. Note that AVICap seems not to care about this parameter, and in any case, it ignores the bursty nature of dropped frames. It is included for completeness.

Maximum index entries

A capture AVI file requires one index entry for every video and audio frame in the file. If this value is too low, the capture will prematurely stop. A good rule of thumb is to add 1 index entry for every video frame, plus 4 for every second for audio, plus another 10% for your faith in your estimation skills. If you are very tight on memory, you should consider reducing this value. If not, the default 300K value is fine for captures smaller than 3 hours.

Normal capture mode does not use this value, since its AVI index buffer is dynamic.

Video buffer limit

Specifies the *maximum* number of frame buffers to allocate; the capture driver may not be able to allocate all requested buffers. Video buffers hold data when the output device is unable to immediately store images. If a video buffer is not available when a frame is captured, the frame is dropped. Increase this value if you notice frames are consistently dropped when hard disk access occurs.

Normal capture mode requires a high number of video buffers, about enough to cover the time spent in writing a megabyte of data to the hard disk.

Audio buffer limit and size

Audio buffers hold sound while the output device is busy. The AVICap default, used when 0 is specified in the dialogs, is 4 buffers of 10K each, which will hold anywhere from ¼ to 4 seconds of PCM audio data. Since avoiding dropped audio is very important, this value should be increased when high-bandwidth audio formats are used. AVICap imposes a limit of 10 buffers.

For normal capture mode, you should have enough audio buffers to hold at least a couple seconds of sound.

Lock video stream to audio

If this box is checked (VfW 1.0 mode), the video stream's framerate is tweaked so that the audio and video streams are equal length. When unchecked (VfW 1.1+ mode, the default), the video stream will not be modified, and the two streams are of different lengths. This box should normally be left unchecked, since checking it may desynchronize the two streams, but some applications may require VfW 1.0 mode.

This option does nothing if you are not capturing in compatibility (AVICap) mode.

Histogram

This graph allows you to check if the incoming capture is too bright, dark, washed out, or has too much contrast. Click on the graph to switch between, luminance, red, green, and blue modes. The graph is logarithmic; every 8 pixels is another power of two. Histogram mode uses whatever capture settings are currently set.

Note that the histogram may not function correctly if a capture board supporting hardware compression such as MPEG or MJPEG. If this is true, you may get decompression errors instead of a histogram. This is unfortunately normal.

Volume meter

Note: You should set the audio format before using this.

The Volume Meter shows how loud incoming audio is. If you have a mono PCM audio format, only the center channel will be displayed; otherwise, both L/R channels. The blue bars end at the average volume, and the red bars end at the peak volume. You'll want to adjust the mixer controls in Volume Control so that the average volume is high, but little or no clipping of peaks occur.

Volume Meter doesn't work when a non-PCM format such as ADPCM or MPEG Audio Layer 3 is selected, and also uses Wave Mapper, and so may use the wrong audio device when multiple sound cards or a capture card with built-in audio support is installed.

Frame rate

Controls how fast video frames are displayed, in frames per second (fps). Common values are 10, 15, 24, and 30. Decimal values are also permitted, although some rounding may occur.

Note that if an audio track exists, the two will lose (or regain) synchronization if the video frame rate is changed. This can sometimes restore sync to a file that was improperly stored or modified. In some cases, the proper frame rate can be found by matching the lengths of the two streams (the 3rd option); this most often works with capture files.

The frame rate can be adjusted without requiring recompression.

Frame rate decimation

Directs VirtualDub to cut the video stream down to some integral factor of its original frame rate (VirtualDub does not do fractional decimation). This ensures that you will always get exactly 1/2, 1/3, etc. of the original frames in the file. Fractional decimation, which is what some utilities like VidEdit use, can result in uneven results if the original was at an odd frame rate.

Frame rate decimation only works properly without recompression if only keyframes are left after decimation. The framerate is always dropped accordingly, so audio and video will synchronize the same as before the decimation.

Decompression video depth

Allows you to select 16-bit HiColor (5:5:5) or 24-bit TrueColor (8:8:8) as the destination depth for input video decompression. It has no effect on uncompressed files. In most cases, you should choose 24-bit mode, since 16-bit mode may lose some color depth. However, if the decompressor functions better with a 16-bit output, then 24-bit may not be a viable option. Some video formats only store 16-bits worth of color anyway, so 16-bit might be faster with no penalty.

The Microsoft Video 1 compressor, for example, only supports 8-bit and 16-bit compression, and does not store 24-bit color.

Output video depth

Allows you to use 16-bit HiColor (5:5:5), 24-bit TrueColor (8:8:8) or 32-bit TrueColor (8:8:8:8) as source depth for output compression. In most cases, 24-bit is best because it preserves the most color while only requiring 75% as much memory as 32-bit mode (and thus is faster). Use 16-bit or 32-bit mode if you are outputting uncompressed video, or if the compressor functions better with those color depths. Some compressors may not function with 32-bit input.

With uncompressed video, this makes the biggest difference in file size. A 16-bit raw AVI is a full third smaller than a 24-bit AVI.

Start/end offsets, length

Controls what subsection of the video stream is processed. You can enter in offsets in either source frames or milliseconds (ms), but VirtualDub only remembers the offsets in milliseconds. Length can be entered in instead of the end offset, but again VirtualDub only remembers the end offset, and not the length you specify.

Offset audio to maintain a/v sync

When checked, adjusts the audio start to correspondingly match the video start you specify. If you start 500ms past the beginning, the audio will also begin 500ms past its original start. When unchecked, the audio stream is processed from the beginning regardless of how far ahead the video processing starts.

In most cases, this should be checked.

Cut off audio when video stream ends

Slices off enough audio so the video and audio streams are of equal lengths; has no effect when the audio stream is same or shorter length than the video. Most video players, though, when confronted with an audio stream that is longer, will simply display the last video frame again and again while the remainder of the audio stream plays. If the tail of the audio stream is silence, this can be very annoying. Use this option to cut off this '"tail."

Benchmark

Allows you to specify capture settings and run a disk benchmark test to see if your hard disk can handle the necessary bandwidth. This tests gives more information than VirtualDub does during capture, and can help you figure out if dropped frames are due to a disk I/O problem, not enough CPU power, or a combination of the two.

This test is more appropriate for video captures than a general-purpose disk performance test such as the one in Norton Utilities.

Install AVIFile handler

This AVIFile handler allows VirtualDub to hook into the generalized AVI file parsing routines and fake an AVI file for applications that expect one. You can then set up VirtualDub for clipping and filtering options and stream the processed AVI directly to the application without having to save a temporary AVI to disk first. This is perfect for when the temporary AVI file would be too large to store on disk.

Some applications bypass AVIFile and thus cannot take advantage of this support. Xing Technology's XingMPEG Encoder is an example.

Three registry keys are added and two DLL files copied to your WINDOWS\SYSTEM directory.

Deinstall AVIFile handler

Removes the AVIFile handler from your system. This deletes the registry keys and DLL files added before. You can always reinstall the AVIFile handler if you need it later.

Remove VirtualDub personalized settings

Deletes VirtualDub preferences from your Registry. This includes all capture settings, recently used file lists, etc. Note that this deletes settings for *all* users!

This does not delete any of VirtualDub's program files, nor does it deinstall the AVIFile handler.

Frame size

Specifies the size of a video frame. You can specify a size in bytes, as in **153600**, or in 1024-byte kilobytes, as in **150K**, or a frame dimension in width-height-depth, as in **320x240x16**. This defines the basic unit around which other parameters depend.

If you use video compression, you will want to approximate the frame size here. A good estimate is the average frame size; if the video compression is not very consistent with frame size, then overestimate a bit to be sure.

Frame count

How long the test should last, in frames. The longer the test is, the more reliable the results; if you want to be absolutely sure, make the test long enough that your hard drive's thermal calibration will have kicked in at least once during the test. If this value is too small, parts or all of the captured data may fit in the disk buffer, which will exaggerate your performance; but if you plan to do very short captures, this may be what you want.

Frame buffers

The number of frames that can be held in memory while disk I/O is in progress. This is a very important value; set it to the actual number of video buffers you will have during a real capture. More buffers means more memory, but decreases the chance of intermittent dropped frames. However, extra frame buffers cannot compensate for an insufficient hard disk write speed. You will seldom need more than 50 buffers.

Frame rate

Sets the number of frames per second to simulate. Decimal values are allowed. The actual test may be slightly slower or faster than the specified value, because video capture speeds are measured in microseconds, while the system timer used for this test is only accurate to the nearest millisecond at most. For values below 60fps, though, it is accurate enough for the purposes of this test.

Disk buffer

Sets the size of the disk buffer in kilobytes, and thus the I/O size, for writes to disk. The larger this is, the better disk performance you'll get, but the more buffers you'll need. Buffers smaller than 256K tend to cause too much disk activity, while buffers larger than 2-4Mb require too many frame buffers without giving much more performance. 512K and 1024K are usually good values.

Data rate

How much disk bandwidth will be necessary just for writing out capture data alone. The ‘actual’ bandwidth needed will be about 10-25% higher with disk caching off or possibly as much as 100% more with disk caching on, for head seeks and overhead. If your hard drive sustains a 4Mb/s transfer rate, you should not expect to be able to get more than 3-3.5Mb/s during actual capture.

Buffering modes

Tells Windows 95/98/NT how it should handle disk I/O for the capture file. The default mode allows Windows to both delay and buffer writes, and is the mode that AVICap uses. The second mode uses the FILE_FLAG_WRITE_THROUGH mode, and tells Windows not to delay writes, although it may buffer them. The third mode additionally uses the FILE_FLAG_NO_BUFFERING mode, which also prohibits buffering; this is the mode used by VirtualDub with normal capture mode.

Of the three modes, the second seems to be the worst, but may depend on your hardware – YMMV. Full buffering tends to be inappropriate for video capture because of the high data rate, especially when Windows delays writing the data to disk. No buffering tends to be the best solution when your data rate starts to push the limits of your hard disk.

Output color depth

Selects the default color depth for quick previews activated through the position control. This will normally be set to **16-bit (Fastest)** because on most accelerated video cards, 16-bit drawing is faster even in 24-bit mode. **Match display depth** is the best choice if you want the highest quality possible (i.e. 24-bit depth in 24-bit video modes) while still using lower depths in shallower displays for speed. **Use output setting** forces VirtualDub to use existing dub settings.

If your video drivers prefer one format over another, you may find it beneficial to force the format by specifying it directly.

Process priority

Specifies how much priority VirtualDub should have over other tasks in the system. By default, this is **Normal** for regular dubs and **Higher** for previews under Windows 95 (it stays **Normal** under NT).

Automatically add extension to filename when saving

Forces an `.avi` extension on output AVI files, meaning that entering `junk` in the Save AVI dialog will save the output as `junk.avi`. This also causes extensions to double if you enter in another extension; `junk.vid` would become `junk.vid.avi`, effectively making it impossible to save with an alternate extension. This annoying behavior is disabled by default, but may be useful to those who dislike typing full filenames.

Enable 16-bit dithering

16-bit RGB is very good for video display, but can still show banding artifacts in large, shallow gradients. In very specific situations, banding can be quite noticeable. This option causes VirtualDub to dither 24-bit video down to 16-bit by combining the closest colors in patterns. It is a purely display-oriented option and only affects Preview mode.

Interframe threshold

Controls how much of a difference between successive frames constitutes a scene change. If this is set too low, VirtualDub will stop more often than it needs to; if the threshold is too high, transitions may be missed. Generally, this setting affects VirtualDub's ability to detect cuts and other quick scene changes. The threshold applies only to luminance changes, since chrominance is completely ignored during scene detection.

Intraframe threshold

Sets the lower threshold at which VirtualDub considers a frame "empty." Empty frames usually signify gradual fades, and the intraframe threshold allows VirtualDub to spot such transitions – they are too gradual to be caught by the interframe system. Increasing this threshold makes VirtualDub less picky about detail in the picture, and thus raising the threshold can be beneficial when working with noisy video.

CPU optimizations

This allows you to override CPU-specific optimizations that VirtualDub normally enables and disables automatically. **MMX optimization** is very important; if you have an MMX-capable CPU, many of VirtualDub's video operations double or triple in speed. Turning it on with a non-MMX CPU is harmless. **FPU optimization** currently affects only one filter – resize, in bilinear mode – but really speeds up bilinear resizes on 486s and Pentiums without MMX.

For all Intel CPUs and AMD K6's, default optimizations should suffice.

Restrict legacy AVI support to 1Gb

AVI 2.0 files have a “legacy” portion that can be read by older applications; the rest is inaccessible except by applications that can read AVI 2.0 files, such as Windows Media Player. By default, VirtualDub will try to squeeze up to 2Gb of AVI data into the legacy portion before switching to the extended partition. However, older programs like the 16-bit Media Player will refuse to access the AVI file at all if the legacy partition exceeds 1 gigabyte. Turning this option allows such programs to access the legacy partition. However, it shortens the amount that can be read by applications that accept up to 2Gb of AVI data.

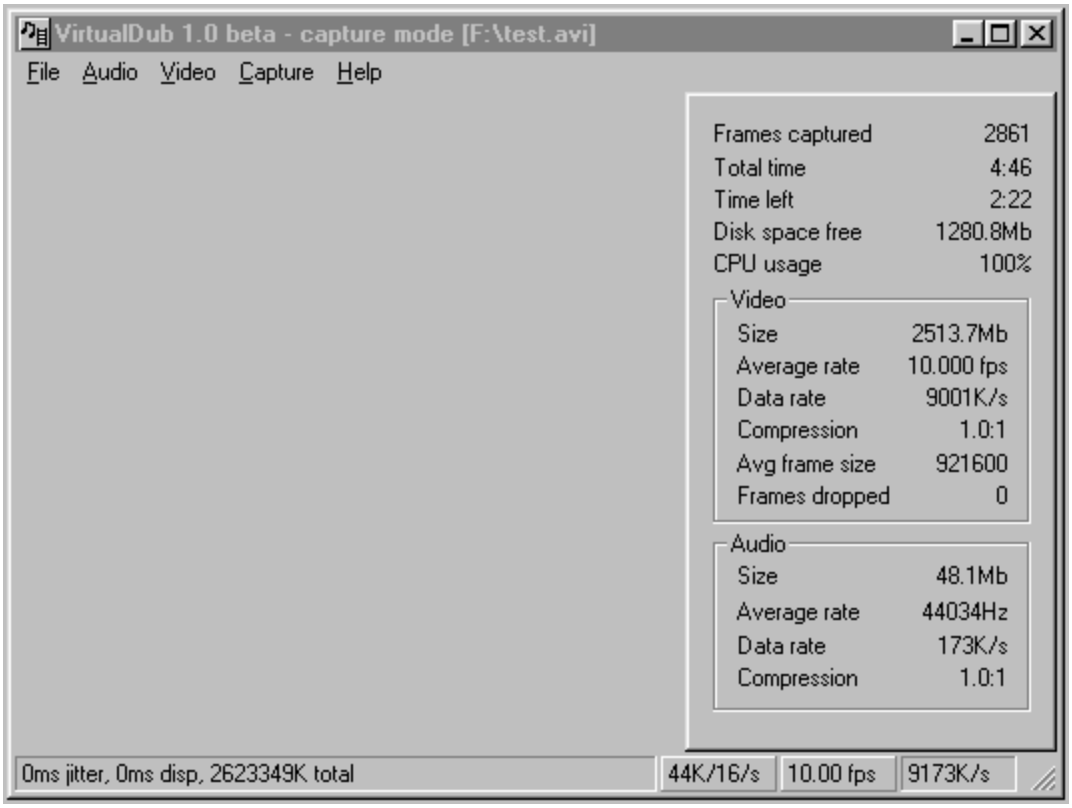
Do not correct MPEG Layer III auto stream byterate

All AVI audio streams have a value that specifies how much space a second of audio takes up. Players then use this value to convert between time and byte position in the audio stream. Unfortunately, the Fraunhofer-IIS MPEG audio layer 3 codec does not always compute this value properly, resulting in files that do not have proper synchronization when played back. By default, VirtualDub will detect the flawed codec and will automatically correct the bad byterate value for you.

If you have applications that expect the flawed value, or want to manually correct for the error yourself by adjusting the framerate, enable this option to disable VirtualDub's automatic byterate correction.

The video capture window

This is what VirtualDub's main window looks like in video capture mode.



Click on a feature for more information. Hold down Ctrl and Tab at the same time to highlight all the areas you can click on.

Bandwidth indicator

This shows VirtualDub's best guess as to how fast data will be written to disk, including video, audio, and AVI file overhead. Thus, 5000K/s means your hard drive will have to write about 5 megabytes per second to not drop frames. The value is most accurate with uncompressed RGB video and PCM audio; it can get confused with compression, even MJPEG.

Quick framerate chooser

This small button shows the currently selected frame rate. Click on it to choose one of many predefined frame rates. A couple of “millisecond friendly” rates are included for those stuck with Zoran drivers.

Quick audio format chooser

This small button displays the currently selected audio format. For PCM, it displays a string such as “44K/16/s”, which in this case means 44KHz, 16-bit stereo audio, or CD-quality. Compressed audio formats show up only as a sampling rate (“44.100Hz”). Click on the button to choose one of the standard PCM audio formats.

Status bar

Generally helpful information appears here, such as Video for Windows (VfW) status messages. During capture, this bar displays frame timing diagnostics.

Frames captured

Exactly what it says: the number of video frames captured so far. Not very interesting, but very important. If this value stops, you're in big doo-doo.

Total time

How long the capture has been going, in *days* : *hours* : *minutes* : *seconds*.

Time left

VirtualDub's projection for how long the capture can run before you run out of disk space or hit the operating system file size limit. This value is updated in real time, so if the capture card starts putting out bigger video frames, or another program starts stealing space on the disk, the value will change accordingly. Note that this doesn't take into account pre-allocated space in the file.

Disk space free

Self-explanatory, except that the value tends to drop in half-megabyte increments (0.5Mb). This is because Windows allocates half-megabyte chunks to try to reduce disk fragmentation. This allocation strategy doesn't affect the size of your files or how much free space you'll have after the capture is finished.

CPU Usage

Shows you how much of your CPU power is being used. Running above 90% is usually an indication that you're overcommitting your CPU or your hard drive and need to back down some of the capture parameters, like frame rate or size. If you hit 100% for more than a short time, you will usually drop frames.

This is the same value reported by System Monitor, by the way.

Video: Size

How much video data has been written to disk.

Video: Average Rate

The average frame rate for frames written to disk. If you have a good capture card and haven't dropped any frames, then this value will be exact or very, very close to the frame rate you've picked. If you do drop frames, then this value will drop accordingly. If you haven't dropped any frames, and this value doesn't match your selected frame rate, then it'll tell you what rate your capture card is really capturing at.

Video: Data Rate

How much bandwidth the video stream is taking on disk. This value is always accurate. It's not uncommon for this value to be different than what you tell your capture card to capture at, particularly if the scene is very simple; it doesn't take that many bytes to represent a black screen.

Video: Compression

An approximate measure of what kind of video compression ratio you're getting. 1.0:1 means you're capturing uncompressed. This value is dead-on if VirtualDub is doing the video compression; its accuracy with hardware compression is dubious at best since the "uncompressed" video format isn't clear. Obviously, higher is better. This value will unfortunately always be 1.0:1 in AVICap mode due to Video for Windows restrictions.

Video: Avg Frame Size

The average size of a video frame. For instance, this panel was taken from a screenshot of a 640x480x24-bit uncompressed capture, so 640 x 480 x 3 = 921600 bytes per frame. Compressed frames vary in size, so this value will usually wander a bit with compressed video, particularly when the scenes get very simple, such as a fade-out, or very complex. In AVICap mode, this value only shows the pre-compression (uncompressed size).

Video: Frames Dropped

How many frames are missing. VirtualDub only receives timing information, and this value simply indicates how many dummy frames it inserted to maintain proper timing. You can get dropped frames because your computer can't keep up, and thus the capture hardware didn't have a buffer to place a frame in and simply dropped it, or the frame wasn't available to begin with, as might happen with an awful VHS tape. Note that this is a bit of a misnomer – a "dropped frame" means you have an *extra* frame in your output to replace the holes in the capture stream.

If you aren't using compatibility mode capture and have timing correction enabled (default), then this value will also reflect correction actions by the timing corrector to keep the audio track synchronized to the video. This includes inserting and dropping frames; if the timing corrector drops a frame, you won't be able to find it in the resulting file.

Audio: Size

How much audio data has been written to disk. Didn't know that CD-quality audio consumes 176K per second? Now you know.

Audio: Average rate

The rate at which the sound card is supplying audio data. This should be close to the sampling rate; because of varied timing in the multimedia system, it's normal for this value to be a little bit off. If you're getting 40000Hz when you requested 44000Hz, though, then you're in trouble. The value gets more accurate with longer captures.

Audio: Data rate

How much bandwidth the audio stream is taking up, in kilobytes per second. This value should be relatively constant. Watch this value; audio shouldn't be taking up a huge amount of bandwidth compared to video. If you're capturing at low frame rates, you might be able to significantly increase your maximum capture time by dropping the audio format a bit.

Audio: Compression

The compression ratio you're getting, versus uncompressed pulse code modulation (PCM) data. Most people capture PCM audio, so this value will normally be 1.0:1. ADPCM normally gets 3:1 or 4:1, and formats such as MP3 can get 10:1. For compressed audio, VirtualDub always assumes a 16-bit input, even if your sound card is 8-bit, because audio compression is usually designed to accept 16-bit input for quality.

AVICap (compatibility) mode capture

VirtualDub supports two main methods of capturing video to disk.

AVICap is the normal capture method used by Video for Windows; an application says ‘capture’, and Video for Windows creates the AVI file automatically. In regular capture mode causes VirtualDub to take over the job of creating the AVI file, which allows it to do neat things to the file. However, you may find the need to drop back to AVICap, depending on your system configuration and capture settings.

These features function in both modes:

- Stop conditions
- RGB filtering
- File size, disk space, CPU usage, and frame timing monitoring

These features only work in normal mode:

- Adaptive data rate regulation during compression
- OpenDML 1.02 AVI file creation (>2Gb)
- Striped capture
- Multisegment capture
- High-performance disk I/O with enhanced buffering
- Audio sync correction

Depending on your capture drivers, disk performance, and CPU speed, you may get fewer dropped frames in AVICap mode, because VirtualDub moves more data around in normal mode for buffering. Conversely, Video for Windows has a tendency to redline the CPU when disk buffers fill up, causing dropped frames; normal capture mode is often able to avoid this.

Picking a video format

There are actually two values to choose: how big of a video frame (resolution) and how fast you want those frames (framerate). Unless your system is buff enough to support the maximums for both, you'll have to trade off between frame size and frame rate.

- Most "hobby" video is captured at 320x240, at 10, 15, or 30 fps.

- For non-linear editing (NLE), or any situation where the video will be output to tape again, capture at as high a quality as possible to preserve the video. If you can't do that, emphasize frame rate over resolution.

- Try and pick the frame size you're going to use. If you're going to make a 320x240 video, it doesn't make much sense to capture at 640x480, especially since it takes four times the space to do so. Some vidcap hardware can't capture below 320x240, so you might be stuck capturing larger than you want. Also beware of reducing the vertical resolution; sometimes this results in scanlines simply being dropped (point sampling), making smooth edges jagged. In this case, it's better to capture at x240 or x480 and scale the results down later.

  Warning: If you are capturing with an SGI Visual Workstation, capturing all scanlines (a height of 480 in NTSC) is highly recommended. When capturing at smaller sizes, the Cobalt drivers scale down the two fields independently and then interlace the results. A 320x240 interlaced video looks *nasty*.

- It's best to capture at even divisions of the source frame rate. For NTSC, the highest frame rate you can get is exactly 29.97 fps – if you want to capture all frames, use **exactly** this value. It's on the quick menu, incidentally.

  Those with Zoran drivers (including the Iomega Buz drivers) won't be able to capture at 29.97 fps. In that case, use 29.412 or 30.303 fps instead. These values are also on the quick menu.

- Ordinarily, do not set the frame rate higher than 30 fps. NTSC video runs at 59.94 fields per second, but these are half-frames and two of them interleave together to form a full frame. Capturing video with a vertical resolution of 480 – 640x480, for example – at a frame rate of 29.97 fps will get all the fields.

  This presents a problem for video games, which often do output 59.94 or 60 frames per second, because capture cards still won't capture at 60fps in these cases. There is a workaround, though, if you have an MJPEG card. See workaround

Adjusting video and audio levels

For a successful capture, you must adjust your audio and video levels so that the sound is at the right volume and the picture is neither too bright nor too dark. If you adjust levels correctly the first time, you can avoid a corrective production step later on, which is especially important when working with very large files.

Adjusting audio levels

Before you capture, bring up VirtualDub's volume meter (V), and check the volume levels. You want the volume level high enough for the red bar to almost reach full volume at the loudest point in the video. Always choose quieter over louder; as long as the volume level isn't *too* low, you can always amplify the audio afterward, but if the volume level is too high the sound will clip and you'll lose sound information. 16-bit audio helps greatly here, because it has much more precision than 8-bit and so the volume level need not be as perfect for great sounding audio.

Adjusting video levels

The same basic principle for audio applies here as well: too soft is correctable, too loud is not. Bring up VirtualDub's video histogram (H), or preferably, the preview histogram (Shift-P). The preview histogram won't work on all cards but is much easier to use because you can view the histogram as you adjust. Use the capture card's brightness and contrast settings, usually under the *Video Source* dialog, to tweak the incoming video. The histogram indicates the breakdown of brightness levels in the image; far left is darkest, and far right is brightest.

Pick a scene with plenty of dark and bright areas, and examine the histogram. Use the brightness control to center the histogram, and the contrast adjustment to control the width. Ideally, a scene with black and white should have the tips of the histogram touching the left and right sides of the graph. If your settings are too dark, counts will pile up into a high line on the left, and similarly, too bright settings will cause piling up on the right.

Finally, do not rely on your monitor to check the black level. If black areas are showing up as gray, the brightness definitely needs to be lowered, but monitors are often adjusted so that a lot of dark grays appear as black. Failing to properly adjust so that black is, well, black, can cause washed-out video when you output back to tape.

Optimizing your hard disk

Rule #1: Upgrade to Windows 98 (if you're running 95).

Windows 98 admittedly doesn't do much for normal disk access, but it has changes in its disk subsystem that can optimize large, unbuffered writes to the hard disk. I noticed a significant drop in disk activity during 1.5Mb/s captures after upgrading to 98.

Make sure that your capture hardware supports Windows 98, however.

Rule #2: Fastest PIO/DMA mode your hard disk supports.

Warning: This is not for the faint-hearted. There is a remote possibility that these instructions may cause data corruption under certain circumstances. If you don't want to risk it, feel free to skip this step since it is by no means necessary.

Reboot your computer, and mark down the PIO modes your BIOS supports when it boots. Then enter BIOS setup and see if you can change the I/O modes of your IDE hard disks. For every drive that is not at PIO Mode 4, bump the PIO mode up by 1 and test thoroughly each time. You probably won't speed up your hard disk this way, since Mode 0 is already 5Mb/sec, and Mode 2 is 10Mb/s, but it'll take less CPU time for disk transfers. Don't rely on BIOS auto-detection, since some drives are detected as Mode 2 but can do Mode 4. Note that not all BIOSes support this tweaking.

Once you've done that, if you have the OSR2 version of Windows 95 (4.00.950b) or Windows 98, then open up System Properties in Control Panel, and choose the Device Manager. Select each IDE hard drive under "Disk Drives" and under its Properties, check the "DMA" box. Not all drives and controllers support DMA, so the next time you reboot, some of these boxes may be unchecked.

Note that some drives don't transfer fast enough to push beyond Mode 0 or 1 limits. That doesn't mean you still shouldn't raise the mode, or enable DMA; both will decrease the amount of CPU time required to read/write to the disk, as well as free up the IDE bus for the other drive (if you have another drive on the same port) to transfer data.

If you have a SCSI drive, try running the configuration utility for your SCSI card. On Adaptec controllers, you may be able to run the SCSISelect utility by pressing Ctrl-A right before the boot process starts. There, you can increase the SCSI transfer rate.

If at *any* time you notice funnyness happening with your computer, immediately degrade the PIO/DMA rates back to their original levels until you've ascertained that higher values are safe.

Rule #3: Faster partitions to the front.

In almost all hard drives, partitions nearer to the outside (start) of the disk are faster. On my 3.5Gb Maxtor, a partition on the outside of the disk is about 20-30% faster than a partition on the inside. If you only have one hard drive, you should partition it so that the capture partition is either C: or D:. PowerQuest's Partition Magic is a good utility for doing this. Generally, any good advice that applies to the placing of swap files also applies to video captures.

Note that the position of the actual data matters, not the size or position of the partition it's in. If you split a hard drive in half, data at the end of the first partition will be read at the same speed as data at the start of the second. This also means that if you have a very big partition, the start of the partition may be significantly faster than the end.

Rule #4: Either defragment, or delete everything

Windows 95/98 exhibits better disk performance when writing to an empty disk partition than one with files on it, especially when AVICap's I/O system is being used. If this isn't possible, at least defragment the partition with a utility such as Microsoft Defrag or Norton SpeedDisk. If you haven't created the capture file yet, you can choose 'Defrag free space'; if you have already created the right size capture file and have plenty of time, defragment the capture file as well by choosing Full Optimization.

For Norton SpeedDisk users, use the Advanced options to push files on the capture drive to the end of the partition, and to force any prepared capture files to the front.

Dropped frames and how to avoid them

A *dropped frame* occurs any time a frame is supposed to be written to disk, but isn't because the system isn't fast enough. The result is a "null frame" in the file that takes up virtually no space and simply mimics the last frame. More dropped frames means a jerkier video file.

Ultimately, the goal is to have no dropped frames. This is important for the following reasons:

- Dropped frames usually come in bursts. Your AVI file will usually have large jerks in it instead of slight, regular jitters.
- If you are compressing the video in real-time, dropped frames occur when the data rate is the highest. Usually, this is when the most action occurs in the video – the *worst* time.
- Dropped frames are noticeable at 15fps and very irritating at 10fps and lower.

If you are getting dropped frames at a certain framerate, you should *never* try to increase the framerate to "grab as many frames" as you can, because you'll end up with a very jerky video and less frames than if you used a lower data rate. Instead, try the following:

- Optimize your hard disk.
- Increase the number of video buffers.
- Make sure no other applications are running when you capture.
- Try VirtualDub's normal capture mode.
- Before you start capturing, turn off real-time overlay or even preview. If you need to see the video to start the capture, use the *Hide on Capture* option.

In addition, if you are using video compression:

- Try decreasing the quality or modifying the keyframe interval.
- If your hard drive can't keep up, but your CPU can, try *increasing* the compression quality.
- Use a better video source, or capture live if you can. The better the source quality, the higher and faster the compression.

If your video capture device isn't the same as your sound capture device (i.e. audio and video capture aren't integrated onto the same board), you may not be able to avoid dropped frames, because VirtualDub will have to insert or drop frames to keep your audio in sync. This should only occur rarely, though – no more than once every several hundred frames – and only in sporadic drops, never large bursts.

Finally, one last cause of dropped frames, especially in normal capture mode, is hardware that isn't capturing at the average rate you tell it; this is reflected in normal capture mode by a steadily increasing or decreasing jitter value. This is true for capture cards based on Zoran drivers, which will round all frame intervals to the nearest millisecond; a framerate of 15.000 fps results in 15.151 actual fps, for instance. (VirtualDub will detect some of these drivers automatically and warn you about this problem.)

- Watch the **Average Frame Rate** value. After a few minutes, it should stabilize to the frame rate you have selected. If it settles to a value different than what you chose, and you are getting dropped frames or bad synchronization, try using the Average Frame Rate value as your capture frame rate and see if that clears up the problem.

- **Redlining the CPU at 100%** for more than a second or so often results in at least one dropped frame. Watch the CPU usage meter to determine if your CPU is too weak to support the current capture configuration. Slower hard disk performance often results in greater CPU usage.

Color formats

RGB (Red/Green/Blue) formats

The RGB format encodes color by breaking it down into red, green, and blue components. No red, green, or blue encodes black; full red, green, and blue encodes white. Two common formats are 16-bit (HiColor) and 24-bit (TrueColor). 16-bit (HiColor) can actually refer to either 15-bit RGB or the “true” 16-bit RGB. Both encode 32 levels of red and blue, but 15-bit RGB, known as 5-5-5 RGB, encodes 32 levels of green while 16-bit RGB, or 5-6-5, encodes 64 levels. However, 15-bit RGB takes the same amount of space as 16-bit RGB. 24-bit TrueColor, or 8-8-8 RGB, encodes 256 levels of all three channels. Because 16-bit RGB takes two-thirds the space of 24-bit RGB without much loss in video quality, it is usually a better choice if you have to capture uncompressed video.

You may also find 32-bit RGB. Most often this is simply 24-bit RGB with an extra pad byte added per pixel. It thus takes one-third more space than 24-bit without any improvement in the picture. Probably should be avoided.

If you happen to find an 8-bit mode, it is probably a *palette* mode, where a set of 256 colors is constructed from a larger color space, typically 18-bit or 24-bit, and colors from captured frames are matched to one of the 256 colors. Video that is one-channel only (grayscale) can take advantage of 8-bit mode, where 256 levels of gray can be encoded with half the space requirements of 16-bit, which can only encode 32 levels; but in most cases, 8-bit modes can be avoided.

Computer displays are RGB, so other video formats must be converted to RGB before they can be displayed.

YUV and YCrCb (Luminance/Chrominance) formats

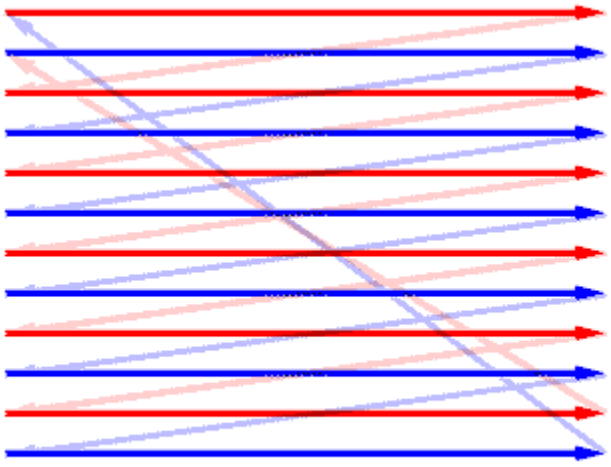
These two formats are very similar, and for convenience’s sake both will be referred to as **YUV** here.

YUV encodes color as a single Y value, plus two U and V values. The Y value holds the luminance, or brightness, of the pixel, while the U and V values determine the color. This is based on the fact that the eye is more sensitive to luminance than color. For this reason, some YUV formats also use *chrominance subsampling*, where U/V color values are encoded for groups of pixels instead of single pixels, as with the luminance.

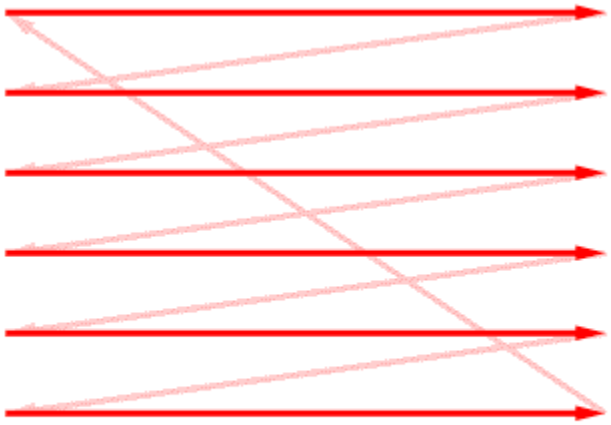
The most common YUV format is YUV12, the native output of MPEG video decoders. In YUV12, 8-bits of Y, U, and V are encoded, but only one U and one V value is included for every 2x2 group of pixels. Each pixel has its own Y value. This is equivalent space-wise to encoding 12 bits for every pixel. YUV9 is similar, except the U and V values occur for every 4x4 group of pixels, changing the total to 9 bits per pixel.

RGB-to-YUV conversion is a common early step in many video encoders. If you are encoding computer-generated graphics, you should beware of the YUV conversion, since chrominance subsampling can cause colors in the image to bleed slightly.

Interlaced vs. Noninterlaced video



Interlaced Display

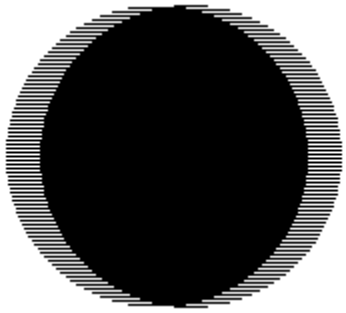


Non-interlaced Display

NTSC and PAL video both use a system called *interlacing* to increase visible resolution. A whole frame – 525 horizontal lines in NTSC, 625 in PAL – is split into two *fields*, which are sent back-to-back. Scanlines from the two fields interleave with each other to form a complete picture with twice the vertical resolution, as shown on the left. NTSC sends alternating fields of 262 and 263 scanlines at 60 fields per second; PAL sends alternating fields of 312 and 313 lines at 50 fields per second.

Frame sizes with **240** lines or less of vertical resolution usually cause *non-interlaced* captures, as shown on the right. This is the same way your computer usually paints its screen: right-to-left, and top-to-bottom, just like how you read a book. Your capture card continually grabs single frames and saves them to disk. Non-interlaced captures off of video are limited to 60fps for NTSC, 50fps for PAL. However, most capture cards in this mode stick to a single kind of field for the operation – either even or odd -- and so the limit is usually either 30fps or 25fps.

If your vertical resolution is higher, such as **480**, you are probably capturing in *interlaced* mode, shown on the left. Your capture card grabs two consecutive fields and combines them to form a single frame, which is saved to disk. The main problem with interlaced capture is that the two fields don't have to correspond to the same frame. Instead of capturing 30 (25) frames per second and encoding them into pairs of fields, video cameras actually send alternating fields of 60 (50) different pictures. You'll often end up with frames like this:



The ball was moving right in the original video, and captured with 480 lines of resolution. Notice that the two fields captured don't match. It wouldn't have mattered if the capture had started one field earlier; the result would have been the same.

Any standard video compressor used with this video, except for MPEG-2, will probably mistake the interlacing for fine detail. As a result, detail in other parts of the picture will suffer, or the file will be larger than expected. Motion-JPEG compressors also do not have a problem with interlaced video, because they compress interlaced frames as two separate fields.

The caveats of composite video

Composite video is so named because it mixes luminance and chrominance information into a single signal. Specifically, a technique known as *quadrature encoding* piles the color information in on a 3.579545MHz color subcarrier. Go through the math for NTSC:

Color subcarrier	3.579545 MHz
Horizontal refresh rate	15.735 KHz
(Color subcarrier) / (Horiz rate)	approx. 227.5 color clocks per line

What this means is that you must have at least enough signal quality to support 227.5 luminance pixels across an entire scanline before you even begin to pick up color. Not only that, but you'll need more bandwidth to be able to accurately represent color. Basically, the color in any composite signal will degrade long before brightness information will; this is extremely apparent in VHS video, where color variation sets in after one generation or so. For this reason, the TV filter should be used with composite video to clean up color artifacts unless the incoming signal is very clean.

After two or three VHS generations, sync degredation tends to be more of a concern than gross color, but the *5x5 I/Q average* option of the TV filter may help.

A special case: Video game consoles

Video game consoles present a special problem because they often output non-interlaced video. This means a full 59.94 **frames** and not **fields** per second. If you try and capture 59.94 frames per second, the capture card will usually only supply half and drop the other half. You can get all the picture information by capturing at twice the vertical resolution (usually x480), but then you have two frames meshed together.

VirtualDub can work around this problem if you can compress the video into Motion-JPEG (MJPEG) format. Motion-JPEG compresses the two fields of an interlaced picture as two separate pictures, and VirtualDub can split these apart. To do this, click the “extended options” box when you open the AVI file, and select one of the MJPEG frame split options. VirtualDub will then split all the frames in half and you can write the result as a true 60fps file. Normally, the unswapped mode will work, but you might have to choose the swapped option if frames are coming in out of order. This works with both hardware and software compression.

Note that you must capture *all* the frames (29.97 fps) and must capture the whole frame for this to work. The trick relies on two frames being interlaced together, and so even if you capture at a lower frame rate you’re always pulling pairs of back-to-back frames. If you capture at odd vertical resolutions like 360, the two frames won’t be evenly interlaced anymore and it’ll look like crap. You can reduce the horizontal resolution, though, so capturing at 320x480 will work.

Video came consoles also vary in resolution. The Nintendo 64 cannot output high resolution video, and so its maximum resolution is 320x480. The Sony Playstation can output both lores (320) and hires (640) video in both interlaced and non-interlaced modes, although most games such as Final Fantasy VII run lores, non-interlaced. The Super Nintendo normally only outputs 256 pixels across and stretches the video quite a bit, although it can also do 512. Pay careful attention to what mode the game is running in and adjust your capture settings accordingly, *especially if the game switches between interlaced and non-interlaced modes*. If the console switches to interlaced video, and you don’t notice and split the frames anyway, the resulting video will look very jittery and strange.

Incidentally, some of the really old consoles, like the Atari 2600, are limited to as low as 160 pixels across.

Introduction to segmented AVI files

A number of video programs for Windows are now capable of skirting by the infamous two-gigabyte limit by splitting their output into a series of AVI segments. Each segment is a legal AVI file by itself, and video editors which cannot accept the OpenDML extensions for large AVI files can process the segmented video by concatenating the segments in the timeline.

As of version 1.2, VirtualDub can now generate and process such segmented files. This method is not as clean as OpenDML-extended files, but is the most compatible method of escaping the two-gigabyte limit.

Reading and writing segmented files within VirtualDub

VirtualDub automatically tags segments so that when you open the first segment, it will automatically load all of the rest. To write a segmented file, simply use the *Save Segmented AVI…* command on the File menu instead of *Save AVI* or *Save extended AVI*. VirtualDub will take your base filename and insert a two-digit number right before the file extension. Thus, if you choose to save as `nuku-nuku.avi`, the first section will be saved as `nuku-nuku.00.avi`, the second as `nuku-nuku.01.avi`, and so on. You can specify the maximum file size in megabytes, and also optionally a video frame limit. VirtualDub will switch to the next segment before either limit is reached.

To open the segments in VirtualDub, simply use *Open AVI* and leave the *Automatically detect segments* box checked. VirtualDub will detect its segment marker in the first file and search for subsequent files. If you have to move some of the segments for disk space reasons, VirtualDub will ask you to point it to the next section whenever it cannot find a section by itself. Alternatively, you can uncheck the *Automatically detect segments* box on the initial open, and manually append segments with *File > Append video segment*.

Capturing to AVI segments

VirtualDub can also capture on-the-fly to sequential AVI segments. This removes file size barriers and limits file sizes essentially to only disk space. It also places additional strain on the system's performance capabilities, so a system that is barely capable of capturing 30fps video will likely drop frames when trying to capture segmented.

The general process for capturing in segments:

- Set up a spill system of hard drives.
- Enable segmented capture.
- Don't use compatibility mode capture.

Video for Windows does not natively understand segments, so you cannot capture segmented using the compatibility (AVICap) mode.

Spill systems

VirtualDub can keep track of multiple drives during capture, and scatter captured segments between them. This means you can dedicate two hard drives for video capture and have VirtualDub use the fastest parts of both drives during a single capture. This is done by defining a *spill system* of drives for VirtualDub to use when an individual segment or drive is full and the program needs to find a place to put the next file. For this, choose the *Capture* > *Spill System…* menu option.

Each spill drive entry has three values: priority, space threshold, and path. Spill drives are ranked first by priority; VirtualDub will not use lower priority drives until all higher priority drives are full. This allows you to relegate less useful drives to "emergency status," for use only when space is very short. For drives with the same priority, VirtualDub chooses the drive with the most space for the next segment. Thus, if two 10Gb drives are on the list, AVI segments will be alternately placed between them, and both drives will be used roughly equally as the capture progresses. The path allows you to choose where the files go, and the space threshold allows you to specify how low the amount of free disk space can drop on the drive before VirtualDub deems it unusable for capture. A value of 50 would tell VirtualDub not to use the drive if it cannot create a decent-sized segment without dropping the free space *roughly* below 50 megabytes,

Note: Windows is only accurate to roughly one megabyte when computing free disk space that has files being expanded. Disk space estimates can be off by several megabytes during the capture operation.

Note: Do not add a hard disk partition to the list more than once, even with different paths. This causes incorrect free disk space estimates, and can cause VirtualDub to act unpredictably.

Exporting segmented files to traditional video editors

Traditional video editors are designed to connect independent video clips, and not to combine video segments chopped from a continuous stream. Thus, they will insert dead space or trim the ends of the video as necessary to ensure that the video clips maintain their audio/video synchronization. VirtualDub attempts to cut the segments as cleanly as possible, but depending on the processing settings, it may have varying levels of success at doing so. The segments will always be readable in VirtualDub, but here are some tips on maintaining compatibility with other programs:

- Do not compress the audio, and leave it as PCM. This gives the audio stream a very fine granularity, less than one-thousandth of a second, and allows VirtualDub to make a very precise cut. In contrast, VirtualDub will have trouble choosing a clean cut point with a 29.97 fps video stream if the audio stream only has 20 blocks per second.

- Use uncompressed video, or set video compression such that only key frames are generated. This can be done with compression that inherently does no delta compression, such as Motion JPEG, or by setting the key frame interval to 1. Otherwise, VirtualDub can cut the video on a non-key frame, possibly resulting in decompression errors in the receiving application.

Importing segmented files from other applications

Because there is no standard way to mark AVI segments, VirtualDub requires manual assistance when importing segments from other programs, such as Markus Zingg's AVI_IO. This must only be done when first opening the file; any subsequent jobs created from the video will contain the complete list of segment filenames and you won't have to select them again.

To manually combine segments:

- Open the first segment as you would open any other video file.
- Choose *Append Video Segment…* from the File menu for each additional segment, in order.

The first segment is placed in the most recently used (MRU) list, as usual, but subsequent segments are not. If you want to open the file again, you will have to repeat the process.

How the automatic segment joining works

VirtualDub automatically writes out a special tag to each segment it creates. This tag indicates whether the segment is the final segment in the chain, and also supplies a path hint to guide the program to the next segment. *VirtualDub cannot automatically join segments that do not have this special tag.*

The tag is a small chunk with a fourCC of `segm` that resides somewhere in the main AVI header LIST chunk. It must not be embedded within another LIST, such as one of the stream header blocks. The segment chunk consists of a single byte, followed by a null-terminated C-style string. The first byte is a hex `01` byte if there is a subsequent segment, and hex `00` if there is not. The optional C-style string that follows specifies *only the path* to the segment. Applications writing this chunk should not write in UNC-style paths, because they can cause unacceptable delays if the network is down or the server is otherwise unavailable. In this case, omit the string or write only the null terminator. The path should always be fully specified, and never relative to the current directory or to a drive. In the case that a segment is moved, VirtualDub will query the user if it detects a next segment but cannot find it.

Example:

`'segm' 00 00 00 01 00 00`
This file is the last, or only, segment in a segmented video.

`'segm' 00 00 00 06 01 'F:\x' 00`
There is at least one segment beyond this file. Programs should look for a file called `F:\x\`*`basename.nn`*`.avi`.

Segments *must* have the filename format `basename.00.avi`, where the first segment is *basename*.00.avi, the second is *basename*.01.avi, etc. If there are more than 100 segments, the next segment is *basename*.100.avi. These filenames can be generated with the printf() formatting specifier `%s.%02d.avi`.

RAID striping for AVI files?

Deprecated. Expected to be removed in later releases.

Well, not quite. VirtualDub's striped AVI support allows you to direct the flow of data that would normally go into a single AVI file toward a number of AVI files. Among the advantages of striping:

- You can take advantage of the disk bandwidth of multiple drives.
- You can dictate buffer and chunk sizes for more reliable video captures.
- Because multiple AVI files are used, you can have 2 gigabytes (2,147,483,647 bytes) per stripe instead of 2Gb for the entire file.
- You can separate audio and video into separate files.
- Since the stripes are still valid AVI files, you can fork off all the audio or video into one of the stripes, which can still be used in a regular AVI processing program.

The disadvantages:

- The output buffering can take a lot of memory, as much as 5-10 megabytes if a lot of stripes are being used.
- If multiple stripes lie on the same physical disk, VirtualDub will try to write to both stripes at the same time.
- Only VirtualDub can use the whole striping system.

Creating a stripe definition file

Stripe definition files have a `.stripe` extension and tell VirtualDub what stripes are active and what data can go into each stripe. This is the stripe definition file I use for capture:

```
#stripe
2
0       m       1048576 524288   i:\capture_master.avi
0       v       2097152 1048576 "e:\capture_stripe_1.avi"
```

Let's break this down part-by-part.

```
#stripe
2
```

The first line of a stripe definition file must be "#stripe" to identify the file as a stripe definition file. The second is a single number indicating the number of stripes. Each successive line is for the next stripe, so the third line is for stripe #1, the fourth line stripe #2, and so on. Extra lines are ignored.

```
0       m       1048576 524288   i:\capture_master.avi
```

The first item on a stripe line is the priority of the stripe. Stripe priority isn't implemented yet, so you can leave this at zero.

```
0       m       1048576 524288   i:\capture_master.avi
```

The second item is a single character indicating the stripe type. It can be one of these characters:

- **a** udio for an audio-only stripe
- **v** ideo for a video-only stripe
- **b** oth for a stripe that holds both audio and video
- **i** ndex for a index-only stripe
- **m** aster for an audio and index stripe

You can create a stripe system without video or audio stripes. In that case, data is discarded. You must have one and only one index stripe.

```
0       m       1048576 524288   i:\capture_master.avi
```

The third item is the total buffer size of the stripe. This is the total amount of data that the stripe output can accommodate before data must be diverted to other stripes to avoid data loss. This should normally be at least 65,536 bytes (64K) for audio/index stripes and 262,144 bytes (256K) for video stripes for decent performance. This value will automatically be adjusted up for sector and chunk size.

```
0       m       1048576 524288   i:\capture_master.avi
```

The fourth item in the stripe line is the total chunk size of the stripe. It tells VirtualDub how much data must accumulate in the stripe's output buffer before a flush can occur. This prevents VirtualDub from making small data writes that hurt disk performance. If this value is too small, tiny writes occur often, sucking down CPU and disk bandwidth. If this value is too large, the disk write can take too long and cause slowdowns or, during a capture, dropped frames. This value should be at least 16,384 bytes (16K) for audio and index streams and preferably at least 262,144 bytes (256K) for high-bandwidth streams such as video streams.

Note that if you set the chunk and buffer sizes the same, VirtualDub will be forced to wait until the entire buffer is full before flushing, but won't be able to place any more data in the buffer until the disk write is complete. This isn't good unless you have other stripes of the same type (audio/video/index).

```
0       m       1048576 524288   i:\capture_master.avi
```

The last item on the line is the filename for the stripe. You should usually use the full name of the stripe so that the same file is used no matter what the current directory is when VirtualDub starts. If the stripe name has spaces in it,

enclose the name in quotes: `"i:\capture master.avi"`

Capturing to a striped system

Striped captures don't work with compatibility mode capture, but aren't much different than normal captures. Use *Set Striping System* instead of *Set Capture File* in the menu, and that's it. The major gotcha with capturing to a striped system is that performance is critical, so follow these optimization tips:

- Do not use multiple stripes on the same physical drive if any of them are high performance stripes. Audio and index data is okay, but any video stripes usually need to be separate. Remember that `C:\` and `D:\` are different **logical** drives, but may be different partitions on the same **physical** drive.

- Keep the buffer and chunk sizes high, especially for video. Normal capture mode uses a 1Mb buffer with a 512K chunk size (double buffering).

One of the simplest striping systems you can use is to put index/audio data in one stripe, and video data in another. This has the advantage of producing two AVI files that are usable in most movie encoders, while allowing you to capture a bit more video before hitting the 2Gb limit, especially if you are using CD-quality audio. (This is how I captured an entire Fushigi Yuugi episode in MJPEG format at 30fps once.) The stripe file I use happens to be the same one I used in the previous topic:

```
2
0       m       1048576 524288   i:\capture_master.avi
0       v       2097152 1048576 "e:\capture_stripe_1.avi"
```

My `I:\` drive is slower than my `E:\` drive, so the video data goes to the faster `E:\` drive, and the low-bandwidth audio and index data goes to the slow drive. The slow stripe is about 180K/s, and the video stripe is 1300K/s. This stripe system allows me to pack about 300 megabytes more video into the capture.

Stripe systems for VCM frameclients

(For an explanation of VCM frameclients, take a look at An introduction to VirtualDub frameservers and Using the VCM driver.)

To make an AVI file for VCM frameclients, you must use a stripe system with a master (**m**) stripe in it. This means all the audio must go into a single stripe along with the index data. The video can be in any configuration, since VirtualDub will be handling the video.

If you're trying to process an existing video with an application that requires the VCM client, you can make a striping system with *only* the master stripe. Then save out the processed AVI. VirtualDub will still process the video, but save only the audio and index marks. To circumvent the filters, set the video mode to *Direct Stream Copy*. (This will be fixed in a later version of VirtualDub.) Then start the frameserver and load the new AVI into the other application.

An introduction to VirtualDub Frameservers

The *frameserver* is a special mode of VirtualDub that allows the output of the dub process to be rerouted to another active process on the same computer. Most video options such as clipping and filtering can be used, allowing you to preprocess video and send it directly to an encoding application without having to create a temporary file. The data is always transferred in full 24-bit color, so no data is lost in the process. Audio data is transferred verbatim, although clipping and offset options are functional.

Frameserver mode is currently only usable with applications that use the AVIFile or VCM interfaces for reading video data. This includes RealVideo Encoder and AVIEdit. Applications such as XingMPEG Encoder and 16-bit Media Player which bypass the AVIFile system must use the VCM driver and cannot currently take advantage of full frameserver support. Unfortunately, ActiveMovie does not work with either driver and cannot use frameservers at this time.

Windows NT support is untested at this time. Frameserver support has been tested under Windows 95 and 98 without problems.

A video compression driver for the Installable Compression Manager (ICM) is available to support applications that do not use the AVIFile interface. Remote serving through TCP/IP is also being considered.

Installing Frameserver support

Frameserver support requires that two DLL files, `vdserver.dll` and `vdremote.dll`, be installed into your `Windows\System` directory, and that entries be made in the Registry to alert AVIFile to the presence of these files. The simplest way to do this is to run VirtualDub Setup and click “Install.” Frameserver support is then immediately available.

If you must do this manually, this is the registry file:

```
REGEDIT4
[HKEY_CLASSES_ROOT\Clsid\{894288e0-0948-11d2-8109-004845000eb5}]
@="VirtualDub link handler"

[HKEY_CLASSES_ROOT\Clsid\{894288e0-0948-11d2-8109-004845000eb5}\InprocServer32]
@="vdremote.dll"
"ThreadingModel"="Apartment"

[HKEY_CLASSES_ROOT\Clsid\{894288e0-0948-11d2-8109-004845000eb5}\AVIFile]
@="1"

[HKEY_CLASSES_ROOT\AVIFile\Extensions\VDR]
@="{894288e0-0948-11d2-8109-004845000eb5}"

[HKEY_CLASSES_ROOT\AVIFile\RIFFHandlers\VDRM]
@="{894288e0-0948-11d2-8109-004845000eb5}"
```

Starting the frameserver

Launch VirtualDub, and open the AVI file you wish to serve. Select video and audio settings as you would for a dub. Note that the following settings do not function in Frameserver mode:

- Video processing mode. Full Processing is always enabled.
- Video output depth. Frames are always sent in 24-bit color. Note that the *decompression color depth* **is** used.
- Video compression. Frames are sent uncompressed.
- Any video filter which requires previous frames (Temporal Soften).
- Audio interleaving. A/V interleaving is completely up to the client.
- Audio conversion, compression and processing modes. Audio is always sent in Direct Stream Copy mode, so none of these apply.

If you are clipping the audio stream in any way through the video clipping options, it is strongly recommended that you uncompress the audio stream first, and use the *Wav Audio…* option to serve audio from the external, uncompressed file instead of the compressed stream. This gives you the cleanest and most accurate clipping.

Once this is done, select *Start Frame Server…* and enter in a name for your frameserver on this computer. This can be anything short (under 50 characters or so); the name of the source is a good choice, but "x" will do as well (grin). Now hit OK, and you will be requested to save a `.vdr` signpost. The signpost file tells the client which frameserver to use. If you already have a signpost with the right frameserver name, you do not need to save another.

The frameserver should now be running. Now you can set up the client.

Adding a client to the frameserver

Open up the application, and select its *Open AVI*… option, or use whatever method the application uses for opening an AVI file. Attempt to open the `.vdr` signpost. If all goes well, the application should be treating VirtualDub's frameserver like any other AVI file.

If it doesn't work:

- Try forcing the file type to AVI.

  If you use the *All Files* or *Autodetect Type* option when opening the signpost file, the application may not recognize the signpost file as being compatible with AVIFile, since the signpost is not a regular AVI file. Forcing the file type to AVI may work.

- You may need to rename the `.vdr` file.

  The application may be checking the extension of the file and rejecting it as an AVI file since it does not have an `.avi` extension. Try renaming the file so it has an `.avi` extension instead of `.vdr`.

- The application does not support AVIFile.

  Some applications simply do not use AVIFile and thus are incompatible with frameserver support. One such application is Microsoft's ActiveMovie Control, which uses its own routines for reading AVI files. If this is true, you may be able to use the VCM driver instead (see Installing the VCM driver and Using the VCM driver).

  A telltale sign that the application does use AVIFile is the acceptance of audio files (`.wav`) as AVI streams, since AVIFile has a built-in handler for such files. Not all AVIFile applications may support this, though.

Installing the VCM driver

[Note: The VCM driver is still in alpha stage. Use at your own risk.]

The VCM (Video Compression Manager) driver should be used when the traditional AVIFile driver does not work. It will work with some applications that do not work with AVIFile, such as XingMPEG Encoder and the 16-bit Media Player. It does **not** work with ActiveMovie yet (this includes Media Player 5). Also, it probably won't work under NT.

Because the VCM driver is still in alpha, installation is not automated. You will have to install it by manually editing the `system.ini` file. Here's how:

Warning! This section tells you how to edit critical system files of Windows. Even the slightest screwup can render your Windows installation inoperable. If you're the least bit squeamish, find something else to do. If you decide to try this, and you screw up your system, you agree that the author wasn't responsible and that it was **all your fault**.

- Copy the `vdsrvlnk.dll` file to your `Windows\System` directory. This may already be done for you if you installed the AVIFile driver.
- Open the `system.ini` file in your Windows directory with Notepad or a similar text editor.
- Look for the section where 32-bit drivers are added. It should look similar to this:

  ```
  [drivers32]
  vidc.IV50=ir50_32.dll
  vidc.MSVC=msvidc32.dll
  ```

  If you do not have a `[drivers32]` section, you will need to add one. At the end of the file or after the [drivers] section is fine.

- Add a line like this after the `[drivers32]` marker:

  ```
  vidc.VDST=vdicmdrv.dll
  ```

  Case is unimportant except after the period: it **must** be .VDST or else Video for Windows will not find the codec. `vidc.vdst=` will not work! If your vdicmdrv.dll is not in your `Windows` or `Windows\System` directory, specify the full path (i.e. `g:\projwin\virtualdub\vdicmdrv\release\vdicmdrv.dll`). It is suggested you place this at the end of any other `vidc` drivers, because the order of the driver lines in this section is the same order the drivers will appear in the Compression… dialog box.

- Reboot.

If all goes well, the driver should show up in Control Panel, Multimedia.

Using the VCM driver

[Note: The VCM driver is still in alpha stage. Use at your own risk.]

The VCM driver operates at the installable compressor (IC) level instead of the AVIFile level. This means it can only affect video, and not audio. For this reason, it is not possible to stream processed audio from VirtualDub in real-time; the audio track must be prepared ahead of time. Furthermore, a special file must be prepared to use the VCM driver.

The first step is to make sure the VCM driver is not installed. If it has not, read the instructions and do it now. Afterward, you must create a special *striped AVI file* for the VCM driver to work. You may want to read the introduction to striped AVIs section first, although this is not necessary. Assuming your source file is called `source.avi`, and your new VCM-ready AVI file will be called `e:\vcmready.avi`, you will need to prepare a `.stripe` file like this:

```
1
0   m   131072   65536   e:\vcmready.avi
```

This stripe definition file tells VirtualDub that you are creating a stripe system consisting of 1 stripe, and that this stripe will contain only index marks and audio. Note that there's no video stripe included. Fine with us.

Now, in VirtualDub, open your source.avi file. Select *Direct Stream Copy* for the video mode, and set audio settings as you please. Select *Save Striped AVI…* and choose your newly created stripe definition file. When VirtualDub finishes, you'll have a `vcmready.avi` file that contains only an index track and an audio track.

To use the VCM-ready file, setup video options in VirtualDub and start the frameserver. Name the frameserver `VCM` and start it. Make sure VirtualDub says the VCM driver is installed, and then select the `vcmready.avi` file as the source file in the other application. If all goes well, the application will read audio from vcmready.avi and processed video through VirtualDub from `source.avi`.

Important note: Only one application can use frameservers through VCM at this time. This is because the frameserver *must* be named VCM.

Using the proxy mode of the AVIFile driver

VirtualDub's main frameclient driver has a special mode to work with programs that require the .AVI extension on input video files. Directions and registry patches to control this special mode are in the `AVIPROXY` directory of the VirtualDub binary distribution.

Frameserver/frameclient performance

Frameserver mode is, in a word, slow. One reason is the overhead; there is a process switch for every video and audio frame transferred at least, and sometimes two depending on the client behavior. The second is that frameserver mode cannot read ahead and pipeline frames as the dubbing routines do for performance.

Another gotcha is that frameservers always run at High priority. Even if the client application is running at below normal priority, its requests execute on the frameserver's process. As a result, disk I/O and filtering operations on the server side will have priority over most other user tasks in the system. This degrades performance on 95 and can become extremely irritating under Windows NT. Therefore, running WinAmp to record MP3s to tape while a frameserver is in use is not a good idea.

Finally, chaining two copies of VirtualDub may be fun, but isn't particularly fast compared to a direct file-to-file dub.

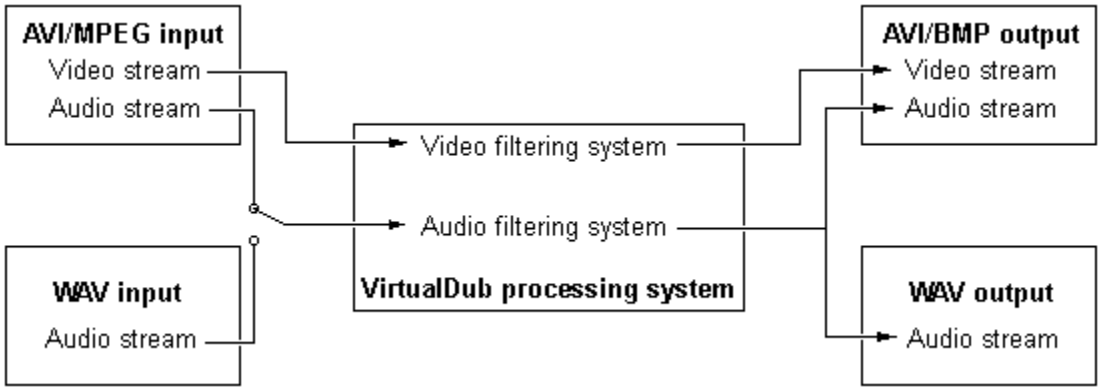
Signpost files

A `.vdr` signpost file only holds the full name of the frameserver. This includes your NetBIOS computer name if you have one. However, there is no support yet for cross-computer frameserving. The signpost serves only to tell the frameclient software which frameserver to connect to. The frameclient then communicates with the server directly for all other information. Signpost files are not needed for the VCM driver, only for the AVIFile driver.

In case you get any smart ideas about sharing `.vdr` signposts across the network, it won't work. The client on the other end will notice the computer name is incorrect and will post an error.

What is VirtualDub?

A *dub* usually refers to a copy. You can dub a videotape, an audio tape, or even do an *audio dub* on videotape, where you copy only the audio track without disturbing the video. *VirtualDub* allows you to perform similar operations on your own computer with audio and video files – with a little bit more power.



Generalized VirtualDub flow diagram.

VirtualDub is primarily a linear editing solution; it does not offer general purpose editing capabilities such as splicing or matte effects. It makes an excellent conversion utility, though; you can recompress, clip, and filter the video to your liking.

Video filtering?

VirtualDub has many of the filters you'd expect to find in an image processing program, such as smooth, sharpen, filtered resize, and brightness/contrast adjustment. You can set up multiple filters in a chain, and apply them all in a single pass. Once you've got the filter parameters ready, reworking a file is as easy as Open and Save.

Video processing modes

VirtualDub has four video processing modes. They are, as follows:

1. Direct stream copy

In this mode, video data is copied directly from input to output without any processing.

Advantages:
- Fastest mode; as fast as 400fps with many input files.
- No quality loss.

Disadvantages:
- Processing must begin on a keyframe.
- All frames in the range must be copied (no decimation).
- The video cannot be seen.

2. Fast repack

Fast repack mode allows a video stream to be recompressed. In this mode, VirtualDub hooks decompressor directly to compressor and tries to find the fastest way from one video format to another.

Advantages:
- Depending on the video codecs involved, recompression might take place in a faster format, such as YUV. This can significantly speed recompression.

Disadvantages:
- May not work if the codecs cannot agree on a common image format.
- Video cannot be viewed during process.
- No image processing possible.
- Some video codecs handle YUV data incorrectly, which can result in upside-down video in this mode.

3. Slow repack

VirtualDub decompresses the input video to RGB and compresses it using the selected video compressor.

Advantages:
- Allows a video stream to be recompressed from any video format to any other, provided compressors and decompressors are available.
- Compatible with more video codecs, because RGB is a very 'safe' format.

Disadvantages:
- Often slower than fast repack mode.
- Compressors and decompressors must be compatible; if the decompressor only decompresses to 16-bit RGB, and the compressor only accepts 24-bit RGB, this mode won't work.

4. Full processing (default)

The full video processing pipeline is enabled. This involves a full conversion to 32-bit RGB and back, and the video runs through the current set of video filters.

Advantages:
- All filtering options are functional.
- Any decompressor-compressor pair can be used as long as the decompressor outputs some form of RGB and the compressor accepts 16- or 24-bit RGB.

Disadvantages:
- The slowest mode.

- Requires the most memory, since all images must be decompressed to 32-bit RGB, and then converted back to a form suitable for the compressor.

Frame rate adjustment and decimation

VirtualDub offers two types of frame rate adjustments. Both are found in the *Video* > *Frame Rate…* menu option.

1. Frame rate adjustment

This option allows you to change the playback speed of the source video stream. It has no effect on the individual frames or on the audio stream, so adjusting this value on a perfect source is *guaranteed* to cause the audio stream to desynchronize. However, it is very useful for adjusting a poorly timed source back into audio/video sync. For sufficiently long videos, VirtualDub can do this automatically if you select *Adjust so audio and video stream durations match*. This option adjusts the video frame rate so that both the audio and video streams have the same time length.

2. Frame rate decimation

Frame rate decimation allows you to reduce the number of frames in a video. For instance, you can reduce a 29.97 fps video to a 9.99 fps one. Because the frame rate is reduced appropriately, audio/video synchronization is not affected by this option.

Note: Frame rate decimation is not compatible with inverse telecine (3:2 pulldown removal).

Inverse telecine (3:2 pulldown removal)

When motion picture films are formatted for NTSC video, they must go through a process called *telecine*. NTSC runs at 29.97 frames per second, while film runs at 24 fps, so the video must undergo a frame rate conversion. This is accomplished by first slowing down the film by 0.1%, reducing it to 23.976 fps, and then inserting one extra frame every fourth source frame.

Capturing telecined video poses a special problem because the frame insertion is done on a *field* basis, by alternating between two and three fields output per source frame. Given a series of four source frames:

Even field	A	B	C	D
Odd field	A	B	C	D

The output five frames will look like this:

Even field	A	A	B	C	D
Odd field	A	B	C	C	D

As a result of the telecine process, the second and third frames out of every five are interlaced. Deinterlace filters can reduce or eliminate the interlacing, but may lower the image quality and still doesn't remove the inserted fields.

VirtualDub can properly remove the telecine effect. To do this, the "enable inverse telecine" button must be checked in *Video* > *Frame Rate*…, and the video mode must be set to Full Processing Mode. When enabled, the telecine remover reduces the frame rate by 20%. The telecine remover is adaptive and will automatically relock whenever the pattern offset changes.

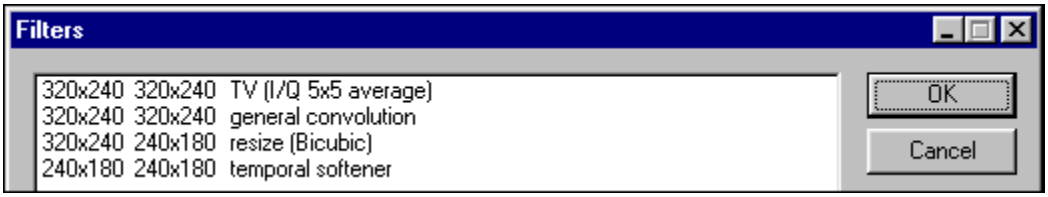
- Do not enable inverse telecine on a video that has not undergone 3:2 pulldown. Doing so interlaces fields together improperly. VirtualDub cannot detect the lack of telecining.
- The remover may lose sync with poor video or after a cut. When this occurs, a few frames may be incorrectly interlaced. VirtualDub reads ten frames ahead and uses two different detection techniques to try to prevent this.
- Telecine removal is not compatible with frame rate decimation.

Processing video the powerful way

Video filters are one of VirtualDub's most powerful features. You can chain multiple video operations together in a sequence and have VirtualDub apply them to every frame in the movie sequence automatically. Many of VirtualDub's filters are designed for speed; most filters will run at several frames per second, and some filters will even run in real time on a reasonably fast Pentium system. Other filters are designed for quality; for instance, the resize filter rivals some image processing programs for quality.

As powerful as this is, VirtualDub only supports linear filter chains; it cannot support *filter graphs* where the processing path can branch off or even loop back.

The *Filter* dialog box



The *Filter…* dialog box displays your current filter chain. Each line indicates the input frame size, the output frame size, and the filter used at that step.

- Use **Add** to add another filter to the list.
- **Delete** nukes a filter entry.
- **Move Up** moves a filter earlier in the order. This can change how other filters operate by changing the input frame sizes of subsequent filters.
- **Move Down** moves a filter later in the order.

You can add multiple instances of a filter at different points in the sequence. Each filter will act independently. Sometimes it is useful to use multiple runs of a filter, such as 2:1 reduce.

- **Configure** changes options for a filter instance, if the filter supports extra options.

Some filters have additional options you can change, such as quality/speed tradeoffs or output sizes; when you **Add** an instance of a configurable filter, the configuration box for that filter will automatically pop up. You can manually select the filter instance and click **Configure** to change the options.

- **Clipping** allows you to keep a rectangle in an image and discard the area around it.

If there's junk around a message, you can attach a clipping rectangle to a filter, so it only works on that rectangle; everything else goes away. The clipping rectangle occurs immediately before the filter and occurs for free, since no extra work goes into discarding the extra data. However, it is advantageous to clip as soon as you can in the order since the earlier you clip, the less data subsequent filters will have to process.

Because a clipping rectangle can only be attached *before* a filter, you have to add a null transform filter to clip after the last filter or clip when no filters are being used.

2:1 Reduce (normal and high quality)

- Shrinks the image to half its size in both the x and y directions.

Both filters do the same operation, but with slightly different filtering. The high quality filter is also known as a *loop filter*; it combines pixels in overlapping 3x3 matrices, similar to blur filters. The lower-quality filter combines non-overlapping 2x2 blocks, and is more suited to certain types of computer-generated images.

The general *reduce* filter can also be used for these tasks, but the 2:1 filters are much faster. It is recommended that the *resize* filter be used when quality is important, however, because the 2:1 filters use the equivalent of bilinear filtering, while the resize filter can use bicubic.

3x3 Average

Deprecated. Likely to be removed in a later release.

- Replaces each pixel by the average of itself and its neighbors.

This filter is a fast blur operation. The weighting is actually 32/256 for the center pixel and 28/256 for each of the surrounding pixels, so it's not exactly an even average. 3x3 Average is good for large (320x240 or bigger) images that have noise or other small, undesirable artifacts.

Blur

- Applies a radius-1 gaussian-blur to the image.

This is a nice, soft, rounded blur. It's better than the 3x3 average.

Blur more

- Applies a radius-2 gaussian blur to the image.

A nice, soft, rounded blur. This isn't very good for noise reduction, though, since it blurs the image too much.

Box blur

- Applies one, two, or three passes of a box blur to the image.

Box blurs, also known as moving averages, are extremely fast, but poor quality blurs, because of the boxy artifacts they produce. However, applying box blurs multiple times gives good approximations to the Gaussian with a fraction of the computing load. Set the filter to power 1 to use a box blur, power 2 for a triangle filter, and power 3 for a cubic. Box blur on power 3 is a very good approximation, and in addition, is nearly independent of filter width – a radius-50 blur with this filter is extremely fast.

Brightness/contrast

- Brightens, darkens, and adjusts contrast for an image.

The brightness slider linearly brightens or darkens a image; parts of the image that are already bright or dark will saturate at one end. The contrast slider enhances the difference between light and dark images; it consistently makes everything brighter, so you'll probably want to use the brightness slider at the same time.

Deinterlace

- Removes interlacing artifacts from interlaced video.

Interlacing artifacts occur when two fields of a frame don't match very well. As a result, alternating horizontal lines appear where the match is poor. This can screw up video processing, particularly those such as compression and resizing.

- **blend** (or **interpolate**) is the best mode to choose. This is effectively a vertical blur on the image, arranged just right to remove the interlacing artifacts. You might notice slight edge blurring with this mode, but in general you won't.

- **duplicate** discards one of the fields and replaces it with the other. This is a good mode to choose if you need to preserve the frame size, but don't want one of the fields. In particular, this prevents "half-and-half" images from showing up from a 3:2 pulldown (telecine) video. However, this results in jagged edges as well.

- **discard** is similar to duplicate, but halves the vertical size of the frame as well. This is good if the interlaced video has an odd aspect ratio, such as 320x480.

Emboss

- Replaces an image with a 3D lifted/depressed-face relief based on pixel brightness.

Depending on how bright a pixel is, it appears to be higher or lower than surrounding pixels after an emboss operation. You can alter the direction of the light source; choosing the opposite direction effectively inverts the relief. The *emboss* operation treats all three RGB channels separately, so color fringes will appear where the chrominance changes. If this is a problem, use the *greyscale* filter first to emboss purely on luminance.

Fill

- Fills a rectangle on the image with a solid color.

You set the rectangle for the *Fill* filter similarly to how you set up clipping, but instead of affecting areas outside the rectangle, *Fill* blots out the area inside. Click on *Pick Color* to select a color other than black to fill the area with.

Flip horizontally

- Flips an image so that what was on the right is on the left, and vice versa.

If you find anything neat in a mirror in a video, you can use this filter to flip it back to a reasonable orientation. Further explanation would be futile, I guess…

Flip vertically

- Flips an image, so that the bottom is on top and vice-versa.

This filter is mainly for programmers, and less for users, although there may be some perverse use for it in the video arena. Windows Device Independent Bitmaps (DIBs) are stored upside-down, so quick programs that dump picture data to AVI format may accidentally place it inverted. This filter allows you to rectify that situation without having to rewrite your AVI file.

Actual origin of this filter: Hacked version of Snes9x 1.05 for DOS, writing out frames from the Final Fantasy 5 intro to a 320x240x16-bit AVI. Inverted, of course.

General convolution

- Replaces each pixel by a linear function of itself and its neighbors.

This is a versatile filter. Many of the other filters, such as 3x3 average, emboss, sharpen, and brightness/contrast can be replaced by a general convolution filter, but the specific filters are much better optimized for the task speed-wise. The convolution filter simply allows you to multiply the center pixel and its 8 surrounding neighbors by fractional values, add them all up, add a bias, and then use the result as the new pixel. The fractional values are multiplied by 256. Example matrices:

Identity matrix (straight source-to-destination copy):

0	0	0
0	256	0
0	0	0

3x3 Average matrix (blur):

28	28	28
28	32	28
28	28	28

Sharpen:

-16	-16	-16
-16	384	-16
-16	-16	-16

Doubling contrast:

0	0	0
0	512	0
0	0	0

Emboss from top-left, not rounded:

-32	0	0
0	0	0
0	0	32

Grayscale

- Replaces each pixel by its luminance value, stripping the image of chrominance information.

Grayscale rips out color from your image, reducing it to a series of gray pixels. The equation used for this is:

```
Y = 0.213R + 0.715G + 0.072B
```

This is the recommended equation for contemporary monitors. Note that this is different from the traditional NTSC equation:

```
Y = 0.30R + 0.59G + 0.11B
```

Invert

- Inverts the entire image, just like a negative.

A very simple filter. Watch the video histogram carefully when using this filter, since you might have your monitor considerably darker than it should be, and an inverted “black” might turn out grayish.

Levels

- Adjusts brightness levels in the image.

This filter maps from one brightness range to another. The top bar with the three arrows is the input range; the bottom bar represents the output range. When an image has low contrast, drag the top left and top right arrows to narrow the input range. Anything to the left of the black arrow becomes black; anything to the right of the white arrow becomes white. The gray arrow in the middle controls gamma correction, and whatever brightness level it points to is mapped to 50% in the output range.

Normally, the default output range is okay, but if you want to decrease the contrast, drag the output arrows in. Anything beyond ‘black’ in the input range maps to the black output point, and anything beyond ‘input white’ maps to the white output point.

Generally, to use this filter, you should sample the video to obtain a histogram, and then drag the input bounds inward so that the darkest parts of the image are black, and the brightest parts are white. As usual, it is possible to correct an image that has too little contrast, but not one that has too much.

Motion blur

- Blurs successive images together.

The 3x3 Average filter is a *spatial* filter because it blurs across an image; this filter is a *temporal* filter because it blurs across the time axis. Specifically, the previous frame is averaged with the current. This leads to a very slight "ghosting" effect. A low-motion video with noise may benefit from this filter, because noise in one frame tends to partially cancel noise in another.

Null transform

- Does nothing.

Exactly as it says above. This filter does nothing. Except for one thing. It provides a placeholder for the clipping when no other filters are being used. Because the filter does nothing, the clipping doesn't add any extra work to the processing stage.

Resize

- Resamples an image so that it is bigger or smaller than it used to be.

No, this is not a lame `StretchDIBits()` filter.

VirtualDub's resize filter is both fast and high-quality; it can outstrip the Windows 95/98 graphics routines both in speed and quality. It has five antialiasing filter modes, each of which has advantages:

- **Nearest neighbor**, or *point sampling*, picks the closest pixel in the source image. This is by far the fastest mode but also looks lousy, especially if the new dimensions are not integral multiples or divisions of the old ones. In this mode, VirtualDub basically drops and duplicates pixels.

- **Bilinear** uses a linear approximation in both directions using a 2x2 kernel. This is the second fastest mode, and produces very good results. It can reduce down to about 66% and enlarge up to about 8x without visible stairstepping. Most video cards with smooth video overlays have bilinear filtering in hardware. Bilinear filtering tends to produce a slightly blurry image and at very high magnification results in trapezoidal patterns. Because this mode only uses a 2x2 kernel, at high reduction it results in pixellation much like the nearest neighbor mode.

- **Bicubic** is the best choice for enlarging a picture, producing slightly sharper images at moderate levels and curves instead of trapezoids at high zooms. It is significantly slower than bilinear but produces much better at high magnfications such as 20x. Because bicubic mode treats detail better, very noisy video can appear worse under bicubic filtering than bilinear. Because bicubic mode only uses a 4x4 kernel, it too results in pixellation at high reduction.

- **Precise bilinear** uses a full bilinear filtering kernel rather than a 2x2 and works for both reducing and enlarging pictures. When enlarging, this mode works similarly to bilinear, but when shrinking this filter averages many pixels together. This results in a much more faithful image and effectively squashes noise, even better than the 2:1 filter. Unfortunately, this filter mode is also slightly slower than bilinear mode when enlarging and signfiicantly slower when shrinking.

- **Precise bicubic** uses a full bicubic filtering kernel rather than a 4x4. It works very similarly to precise bilinear, but the bicubic kernel produces sharper images, when either shrinking or enlarging. Shrinking a noisy 320x240 video with this filter can result in a sharp, clean 160x120 video. This filter is also the slowest of them all.

Rotate

- Rotates an image 90, 180, or 270 degrees.

Use this filter if you need to rotate the video by right angles. The frame size is altered to reflect the new size. You may also need to use a flipping filter if your video is in a particularly funny orientation.

Rotate2

- Rotates an image by any arbitrary angle.

This filter can rotate video by any angle, even in sub-degree increments. However, it is slower than the regular Rotate filter for right-angle rotates. The areas added due to the rotation are filled with a user-selected color.

You can choose from several filtering methods:

- **Point samping** gives you a quick and dirty rotate. It is very fast, but gives rough edges and highly cumulative error on multiple rotates. For quick and dirty rotates such as previews, though, it's fine.

- **Bilinear** takes longer but gives smoother edges and better image quality. Cumulative rotates will blur the image significantly, so this filter isn't very good for more than one rotate.

- **Bicubic** gives the better quality than the above two and is much better at preserving detail. An image can survive five or so rotates without significant image degradation. Note that this filter still does bilinear filtering on the edges, but this is a very minor issue. This filter is the slowest.

The **expand bounds** option stretches the picture frame to fit the rotated picture. Ordinarily, VirtualDub keeps the original image size and rotates the image, cropping the parts that don't fit. The image is always rotated around the center in either case.

If you really want to test a rotation algorithm, one good test is to do a 180 degree rotation in 5-degree steps and then do a single backwards 180 flip. The image degradation will show you how good the rotation algorithm is.

Sharpen

- Increases contrast between pixels and their neighbors, sharpening it.

Most people know what “blur” and “sharpen” means, either through glasses or waking up in the morning, so I won’t bore you with an explanation of what this filter does. Two things to watch out for, though, when using this filter. First, what looks like an “edge” in the image is usually two: a transition from light to dark, and again from dark to light. A dark line that is sharpened will get darker, but will also have new bright lines on its sides. This is strange, but a useful technique when greatly reducing an image is to over-sharpen it first to enhance features. Second, never sharpen an image with noise in it. The “ripple” effect of the sharpen transform will show the noise very clearly.

Smoother

- Removes high frequency noise from an image.

This filter is particularly valuable for dealing with VHS video because it can remove noise without affecting sharp edges much. Smoother works very well on cartoon animation, but can soften videotape artifacts from higher quality video, as well. Raising the threshold causes the filter to consider fewer pixels as ‘edge pixels’, and blurs more of the image. Very sharp images will almost never be blurred no matter what the threshold is, though.

Smoother is very effective at reducing the size of final Indeo AVI files (or increasing quality, if you have data rate stricting on). It is ineffective with sharp tile artifacts, such as those from MPEG or Cinepak.

Temporal smoother

- Applies an adaptive, time-based blur to the image.

This filter is an adaptive noise reducer, working along the time axis; it is most effective when the image is not moving much. Increase the filter strength to increase noise reduction, and decrease it to reduce speckling and ghosting artifacts. It is recommended that you combine this filter with a spatial (area-based) noise reducer for greatest effect.

Temporal smoother is VirtualDub's first linear-phase temporal filter, meaning there is a lag between the input and output frames. This results in some lag in the preview during scrubbing, but the lag is compensated for during preview and exporting. It does mean, however, that the filter will not run correctly in frameserver or capture mode.

Threshold

- Splits an image in half, with bright pixels becoming white and dark ones becoming black.

Very simply, this makes your image monochrome, black-and-white. Pixels darker than the threshold become black. Pixels brighter than the threshold become white. No shades of gray, no dithering. Just straight black-and-white. The best use of this filter is a diagnostic filter, to see which areas of the picture are problem areas in terms of brightness; however, there are creative uses for this filter as well.

TV

- Attempts to correct or mask noise from TV video sources, particularly VHS.

Don't expect miracles with this filter. It works by smoothing the chrominance of an image without touching its luminance. No sync correction is performed, so a low-quality input with horizontal or vertical jitter will still appear jittery after this filter.

The first three configuration options, Y/I/Q-channel, allow you to view the three input channels of composite video: brightness, chrominance 1, and chrominance 2. The brightness (Y) channel is computed with the 30/59/11 distribution that NTSC uses rather than the 21/72/7 breakdown that the grayscale filter uses. The chrominance 1 (I) and chrominance 2 (Q) channels together determine the color of the picture.

Because of the way chrominance is encoded in composite video, and especially in VHS recording, the chrominance channels often have a lot of noise and error in it compared to the luminance channel. The TV filter's averaging options try to mask much of the noise. This has only a slight visual impact but can make a big difference when encoding MPEG videos at low bitrates or any other video format that uses motion prediction. The *5x5 average + temporal* option risks greater degredation of the video but can reduce chrominance noise even further than the *5x5 average* can.

This filter without the *temporal* smoothing mode is only partially effective with YUV12 compressors like MPEG and almost completely ineffective with YUV9 compressors like Indeo because the spatial chrominance averaging is already partially done as the early chrominance subsampling steps of these compression algorithms.

Changing the frame rate of a file

- Use the menu option *Video, Frame Rate*.
- If you want to *remove* frames, then use *frame rate decimation*. If you want to change the *speed* at which the frames are played, change the frame rate value.

Remember what the dialog box says: changing the frame rate will affect synchronization, but frame rate decimation won't.

Compressing a file

To compress video:

- Choose *Video, Compression…* and select your compression algorithm.
- Set the video processing mode to *Full Processing Mode*.

To compress audio:

- Choose *Audio, Compression…* and choose an audio compressor. “PCM” is uncompressed audio.
- Set the audio processing mode to *Full Processing Mode*.

The next time you save, the processed video and/or audio will be saved compressed.

Deleting JUNK chunks from a file

VirtualDub never puts JUNK chunks in any AVI file it creates, so the solution is to turn off all processing:

- Set the video processing mode to *Direct Stream Copy*.
- Set the audio processing mode to *Direct Stream Copy*.
- Open the source file, and save to the output file.

Audio interleaving options work in this mode, and video range selection works to the nearest key frame, so you can re-interleave or trim off ends at the same time.

Extracting audio from a file

- Open the source file in VirtualDub.
- Select *Save WAV…* from the *File* menu.

The resultant file can then be loaded into any application which plays WAV files. When doing this, it's best that the audio mode be set to *Direct Stream Copy* so that the audio isn't processed or decompressed. Note that the range selection options don't work when extracting audio, so the whole audio track will be extracted.

Inserting or replacing audio in an existing file

- Make sure the input audio file is in WAV format.
- Open the source video file.
- If you do not want to process the video, select *Direct Stream Copy* as the video processing mode.
- From the *Audio* menu, choose *WAV audio* as the input source, and select the WAV file you are using as audio.
- Save the new AVI to disk.

Repairing a bad video file

If key frame markers are incorrect, causing some frames to display incorrectly when displayed without playing through from earlier parts of the file (this is common after conversion from QuickTime):

- In the Open dialog box, check *Popup extended open options*.
- Select desired file.
- Check the box labeled *Re-derive keyframe flags.*
- Process file as necessary. If you don't want to modify the file in other ways, use the "null pass-through" method described in Deleting JUNK chunks from a file.
- Note: This may not work with all compressors. It is known to work with CinePak and Indeo 5.

If the file has no index block, or has been truncated in some way, and forces many programs to read the entire file first and others like ActiveMovie to crash, then you may be able to fix the file if VirtualDub can open the file. If it can:

- Follow the directions in Deleting JUNK chunks from a file. If this works, the index chunk should be added to the file.
- If the keyframe markers are incorrect in the resulting file, then follow the directions above to fix them.

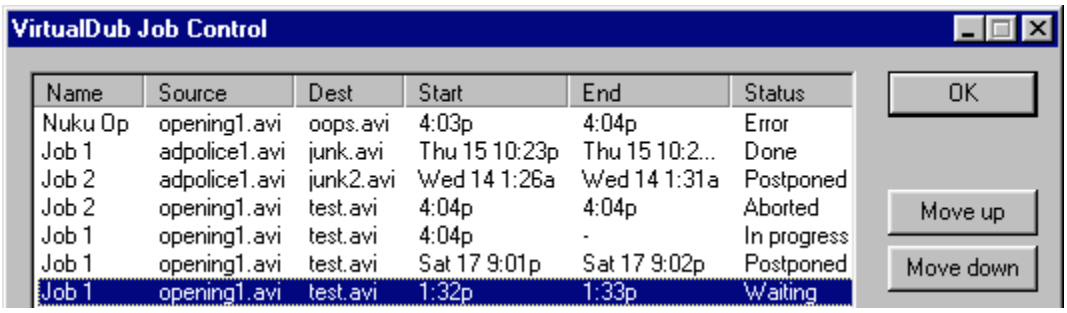
Resizing a video file

- Use the *Filters* option under the *Video* menu.
- Click ‘Add’ to add a filter to the list.
- Select *2:1 reduce* or *2:1 reduce (high quality)* if you want to resize a file to quarter-size (half the width and half the height). Otherwise, choose the regular *resize* filter. Click OK.
- If you are using the regular *resize* filter, then type in the desired size in pixels. *Bilinear filtering* results in higher quality, but takes more time, especially on a 486. *Bicubic filtering* is even slower, but produces even better results at high magnifications.
- You may opt to use the *resize* filter with bilinear/bicubic filtering instead of the *2:1* filters if you find that the results are too blurry. In general, though, the *2:1* filters look better.

Because video frame resizing is a filter operation, it can be combined with any number of other operations, either pre- or post- resize.

Job Control (batch processing)

VirtualDub can queue up processing tasks and run them later, in sequence and unattended. As each job is run, VirtualDub loads the appropriate files, restores all dub settings, and starts the operation in the background. If an error occurs, VirtualDub aborts the operation, records the error, and starts the next job. The job window is independent of the main window, and jobs can be postponed, reordered, and deleted while files are being processed.



Click on a field for more information.

If you are running a version of Windows 95 without the new version of the Common Controls library (pre-OSR2 without Internet Explorer 4), only the job name may highlight instead of the entire job line. This is normal.

Job name

Each job is automatically generated a name by VirtualDub, starting at "Job 1" and ascending in number. By clicking on the name, you can edit it to something more readable, such as "Slow-mo clip." Job names can be anything you want, and are not used for anything other than human identification.

Source and destination filenames

Shows you the source and destination files for the job. The pathname is trimmed off so the name fits in the dialog box.

Start and end times

Indicates the last time a job started and ended; if you request that a job be redone, the old times are erased. Aborted or errant jobs will show stop times here. Jobs that are a day or more earlier will show dates as well.

Status

Describes the current state of the job. It can be one of the following:

Waiting	The job is ready to be processed. You can postpone this job by double-clicking it.
In progress	VirtualDub is currently processing this job.
Done	Job was completed successfully. Double-click to change state to "Waiting," marking this job to be redone.
Aborted	Either the user aborted the job, or VirtualDub was forced closed during the operation (probably due to a crash or hard reboot). Double-click this job to switch it to "Waiting."
Error	VirtualDub detected an error during the dub and started the next job in the queue. Double-clicking the job will display the error and reset this job to "Waiting."
Postponed	This job is in the queue, but will be skipped over. You can click "postpone" or double-click on the job to toggle this state.

You dumba… heh heh…

This section is meant to help you avoid many of the beginners' mistakes that often occur, especially with files distributed over the Internet.

Clip dead space

If there's a static border around your image – for instance, black bars on the top and bottom because the video was letterboxed – clip it out. Someone will point out that the black bars will compress extremely well, and that's true – with decent compression algorithms, the black bars will take practically no space at all. It's the sharp edge between the picture and the black area that does!

Video compression algorithms are designed to optimize the smooth features and edges in video, not artificially hard cliff-like breaks that are perfectly horizontal or vertical. Normally, video compressors block the image into 8x8 or 16x16 tiles, and you're fine if your boundaries like *exactly* on those boundaries. However, if they don't, the compressor either has to devote a lot of space to forming those perfect edges, or smears them.

Instead, move those edges to where the compressor naturally has to handle cliff-drops: the edge of the picture frame. If the video's letterboxed, make a letterboxed movie file. That way, the bandwidth that would have been used for the border and edges can be used on the picture instead.

Don't release in MJPEG, VCR1, VCR2, ASV1, etc.

MJPEG, VCR1, VCR2, ASV1, and WNV1 are all capture formats, and in addition, all but MJPEG are proprietary. This means you either need to hunt down the decompression codec from the manufacturer's site, or even worse, possibly need hardware decompression support. The latter is particularly prone to happen if you're using MJPEG; there are software codecs available now that don't need hardware support, but they run slowly on Pentiums or lower, and aren't free.

Also, capture files tend to be quite large. You would be much better off using a low-bitrate format such as H.263 or Indeo.

Avoid outdated formats.

Microsoft Video 1, Radius Cinepak, and Indeo 3.2 are all outdated formats. Video 1 sucks because it cannot produce a perfect frame in some cases, no matter how much bandwidth you throw at it – it can only assign 8 colors to every 16 pixels, and only compresses up to 16-bit video. It also looks *very* blocky, especially at low data rates. Radius Cinepak produces gross stair-step artifacts due to its motion prediction and takes a lot of space. Indeo 3.2 can produce sick color bleeding and dithering in many cases. In other words, these codecs all stink compared to more modern ones and should be avoided.

Intel Indeo 4.0/4.1 is the exception. It has been outclassed by Indeo 5, and ordinarily I'd suggest you avoid it, except that it is the best codec for the AVI format that Apple Macintosh people can decode; there is no Indeo 5 codec for the Macintosh. If you really care about playback on a Macintosh (although I don't understand why you should), use Indeo 4 instead. Otherwise, go ahead and use Indeo 5.

Compress the audio!

I have seen more than one AVI file distributed with CD-quality audio (176K/s). This is a complete waste of space and often takes as much space as the video track. 16-bit audio at 22KHz or higher should be compressed with, at the very least, ADPCM. This will reduce the audio track size to between one-half and one-quarter its original size with very little loss in quality. In addition, everyone has the ADPCM codec. Compressing the audio in this manner can shave 2-5Mb off a 20Mb file.

MPEG Audio Layer 3 is still superior, but not everyone has the decompression codec and even fewer people have the compression-capable version. It can also introduce sync problems into your video.

Don't compress AVIs to ZIP, ARJ, RAR, etc.

If you can gain any significant compression (>5%) with general purpose utilities such as WinZIP and RAR, then you're not making your AVI correctly. If you can reduce the file a lot, it means either you have wasted space in your file (JUNK chunks) or your compression codecs are inefficient. Remove JUNK chunks with VirtualDub and compress the audio. There are few advantages to packing AVIs and it is a royal pain for those receiving the file.

In addition, packing an AVI file makes it impossible to view a partial download, because the file cannot be decompressed until the entire archive has been received. If the file is not packed, downloaders can play and repair the incomplete file.

Match the frame rate to the source.

I have an Ah My Goddess! ending movie that is recorded at 30 fps. The video is of a cartoon that changes about three times a second. What's wrong with this?

There are two problems:

- Frames aren't free. A frame that changes very little or not at all still takes space in the file, and the computer still has to draw each and every frame.
- The movie was digitized from a video source, and so frames that *look* the same or were originally the same aren't anymore due to noise. The video codec can't tell the difference and will encode the noise.

In other words, if your input source is a 10fps animation, don't digitize it at higher than 10fps unless you intend to perform noise reduction or field-based (IVTC or deinterlacing) operations, and even then you should reduce it down afterwards.

Stop attaching openings, trailers, etc.

Use computer generated graphics sparingly when ‘branding’ video. Video compressors are adapted to natural images and perform sub-optimally on the saturated colors and sharp edges of computer graphics. Depending on the video codec, title screens and ‘bugs’ in the video can adversely affect video quality, especially at low bitrates.

If you must include such items:

- Make openings and trailers self-contained. That is, don't use transitions to blend them with the video, and try to have them start and end on keyframe boundaries. That way, your openings and trailers don't interfere with the compression of the actual video. At lower bitrates, this can avoid artifact-plagued transitions.

- Align the position and size of ‘bugs’ to eight- or sixteen-pixel boundaries, and make them opaque.

  Most video compression algorithms handle video in "tiles" of 4x4, 8x8, or 16x16 pixels, and perform motion compensation and prediction on those tiles. If your bug is not aligned on tile boundaries, the video codec will devote a lot of data to the border tiles, trying to keep your bug crystal clear while still showing the parts of the original video that show through on those tiles. If you align the logo, the codec will notice that the tiles involved never change, and can ignore them with very little overhead.

  For similar reasons, never put holes or transparent regions in the bug. The codec won't be able to tell that you have a odd-shaped static logo floating above a background, and will simply see a complex tile for which it will devote a lot of space.

  As a side effect, these steps will often make your bug cleaner, and reduce the "ringing" effects around it.

It's broken…

These aren't really problems with VirtualDub, but a few miscellaneous fixes related to audio/video on computers in general that we've found to be useful.

Stuttering in ActiveMovie and Windows Media Player

Note that *ActiveMovie* is the name of the movie streaming technology, and *Windows Media Player* is an application that uses *ActiveMovie*.

This problem often shows up as uneven frame rates, where the movie will suddenly play too quickly, and sound may even skip. We've traced this down to two causes:

- The Indeo 5 codec.

  For some reason, movies created with the Intel Indeo 5 codec will occasionally play too quickly under ActiveMovie. The solution is to hit play, then pause the movie immediately, and unpause again. This seems strange, but then again, so is the problem.

- Half-duplex mode in Creative Sound Blaster AWE32/64 drivers.

  Consistent stuttering seems to be a symptom in this case, particularly if Reverb is enabled. It turns out that one way to fix this problem is to download the latest Sound Blaster drivers from the Creative site (ftp.creaf.com) and turn on full-duplex mode. This disables Reverb on the wave out, but you can toggle full-duplex operation without rebooting, so this isn't a problem.

  Note that this problem won't go away if you simply mute Reverb.

Fraunhofer-IIS's MPEG Audio Layer 3 codec

There are three versions of Fraunhofer-IIS's MP3 codec:

- Standard version.

  This version will decode all MP3 streams, but cannot compress. Newer versions of Windows Media Player come with this codec.

- Advanced version.

  The Advanced codec will decompress all MP3 streams and will also encode MP3 streams up to 64 kilobits/sec (22KHz). It usually comes with Microsoft Netshow.

- Professional version.

  Decodes all MP3 streams and encodes up to 128 kilobits/sec (CD quality). This codec is very hard to find, and its distribution outside of a product is… well, "questionable."

Most people do not have the Professional version, and this is the reason why you can often play an AVI file with a 128Kbps audio stream, but can't make one yourself. (It is technically possible to encode an MP3 with another program and directly incorporate it into an AVI with the correct tag, but I haven't been able to do it yet.) Note that since the audio stream is a valid MPEG Layer 3 stream, you can extract the audio track in its entirety with VirtualDub and play it with an MP3 player such as WinAmp.

Another problem with this codec is that it is an enormous hack. In particular, it sets the `nBlockAlign` member of the wave format to `1`. This means that applications think a single byte can be decompressed into audio, when in fact the Fraunhofer-IIS codec will buffer up data until it has enough to decompress a Layer 3 frame. Extracted sections of such audio streams will often have muted tails because the audio codec discards fractions of frames at the start. The root of these problems is the MP3 spec, which allows audio frames to "borrow" unused bandwidth from earlier frames, making it impossible for the MP3 audio codec to specify a fixed size for a self-contained audio block.

Finally, the Fraunhofer-IIS codec is very lazy and incorrectly sets the bitrate of the stream. For instance, a 48Kbit stream encoded by the Fraunhofer-IIS codec has a specified rate of 6000 bytes/sec, when in reality the stream is about 5971 bytes/sec. This 0.0048% difference may not seem like much, except that it causes the audio to race past the video approximately one second for every 200 seconds of video. One way to ‘fix’ this problem is to correspondingly adjust the video frame rate to compensate. VirtualDub will automatically correct for this problem when compressing audio to MPEG format.

Video files play in ActiveMovie applications only

ActiveMovie can use two types of video codec: standard Video Compression Manager (VCM) drivers and ActiveMovie filters. Non-ActiveMovie applications such as VirtualDub can only use the former. Usually, either only the VCM driver or both the driver and the filter for a particular format will be installed, but sometimes only the ActiveMovie filter is installed. One common example is Final Fantasy VII PC, which installs an ActiveMovie filter for the Duck TrueMotion format, but no VCM driver. (Incidentally, all the FF7 PC movies suck compared to the PSX ones, so this isn't a great loss.)

Usually, installing Microsoft Netshow or the Windows Media Toolkit will give you the needed drivers.

Can't compress to a particular audio/video format

Microsoft Netshow and Windows Media Player will sometimes install decompression-only versions of drivers that say they can compress, but refuse to do so when asked, either because the driver author was a moron or the driver requires a "authorization key" of some sort. Nothing can be done about this except to find a compression-capable version of the driver. VDOWave and Duck TrueMotion are two common decompression-only codecs.

More detail is available on the following drivers:

- Fraunhofer-IIS MP3 (audio)
- Windows Media Audio v1 and v2 (audio)
- Microsoft High-speed MPEG-4 V1 through V3 (video)

AVI file very slow at seeking

There are complete idiots out there who encode AVI files with very long keyframe intervals, or sometimes, no keyframes at all except for the beginning. These people need to be exterminated because a player app must read all frames from the last decoded frame or keyframe up to the desired frame, which can potentially be more than half the file. The author has personally seen AVI files more than twelve minutes long without keyframes that took over five minutes to seek to the middle of the movie, because the player had to decode each and every frame from the beginning onward.

Some compressors are designed to insert keyframes as needed. The Microsoft MPEG-4 series of compressors does this. Even though MS MPEG-4 V1 defaults to a keyframe every hour, and the V3 compressor every 8 seconds, you'll often see keyframe intervals shorter than that.

For most video formats, there is no way to insert keyframes into an existing file without recompressing the entire movie.

Cannot encode audio with Windows Media Audio v1 or v2

Microsoft does not allow the use of Windows Media Audio encoding in AVI files. As a result, they have locked the codec driver accordingly so that it only works with the ASF encoding tools. It would sure be nice if they kept their ASF only drivers from appearing in the lists, instead of creating support hassles for video tool makers.

Cannot encode video as Microsoft High-Speed MPEG-4 V1-V3

Microsoft does not allow the use of their MPEG-4 codec in AVI files. As a result, they have locked the MPEG-4 codec driver accordingly so that it only functions with the ASF encoding tools. All other applications receive cryptic errors (`ICERR_ERROR`) on both compression and decompression requests. The first version of `MPG4C32.DLL` to have this lock was build 3845, and build 3876 also has the lock. The earlier, beta builds of the codec, build 2700 and 3688, do not have any such locks.

Some later versions of the codec (3917+) have V1 and V2 unlocked.

Congratulations!

You've managed to crash VirtualDub. So what happens next?

- You'll have to restart any dub or capture operation that was in progress.
- First, make sure you are using an official build of VirtualDub, not somebody's hack. Crash dumps generated by a hacked version are useless to me because I didn't build it and thus don't have the link map to diagnose the problem.

  For technical support, click *Save…* to record the crash information to a file called *crashinfo.txt* in the VirtualDub directory. Then send an email to <phaeron@virtualdub.org> with this report and an explanation of what you were doing at the time. I'll try to track down the problem.

  Please do not do this if you are using any of the following codecs. I will discard the dump because the codecs themselves are known to be problematic and/or buggy:

Driver name	Driver filename
Microsoft MPEG-4 V1/V2/V3	MPG4C32.DLL
DivX low-motion/fast-motion	DIVXC32.DLL, DIVXC32F.DLL
AngelPotion Definitive	APMPG4V1.DLL, ~AP****.TMP
Fraunhofer-IIS MPEG layer III	L3CODECA.ACM, L3CODECP.ACM

- If you are fluent in assembly language, check the instruction referenced by CS:EIP to see if it is an MMX instruction. If it is, and you do not have an MMX-capable CPU, force MMX optimizations off in VirtualDub's *Preferences…* dialog the next time. This indicates a bug in VirtualDub.
- After you're done looking at code and registers, click OK to bring up the normal Windows application failure dialog.

Warning: Problematic MPEG-4 codec detected.

Newer versions of the Microsoft MPEG-4 High Speed V2/V3 codec do not work with AVI-based programs; they will report the error `ICERR_ERROR` on either compression or decompression. This is a generic error. In VirtualDub, the symptom of this problem is `VideoSourceAVI error: unspecified error (-100)` errors when trying to seek in MPEG-4 V2 or V3 files. Other AVI programs will not be able to use the codec as well. The only applications that can are from the Windows Media Tools collection, and applications playing .ASF files back through DirectShow (Windows Media Player cannot even play MPEG-4 V3 AVIs using this codec).

The culprit is the `MPG4C32.DLL` driver in the `WINDOWS\SYSTEM` directory. As of this writing, all known MPEG-4 codecs work with VirtualDub. If you have a version that doesn't, try downloading and installing Windows Media Tools V4 from Microsoft. This will updates your MPEG-4 codec driver and may solve your problem. These codec versions are known to work:

Version	File size	Status
`4.00.0.3688`	`416,304`	works with all programs
`4.00.0.3845`	`413,248`	not usable
`4.1.00.3918`	`413,760`	only usable for MPEG-4 V1 and V2

Command line options

VirtualDub supports limited control via command line options. Allowable switches:

- `/b`*source_dir*`,`*dest_dir* adds a batch of jobs to translate from one directory to another, using the current options. (Note: there are no spaces in the switch.)
- `/c` clears all jobs in the job list.
- `/r` runs the job list.
- `/s`*script_name* runs a script.
- `/x` forces an exit after the job list has run.

The order of the switches matters, since switches are run from left-to-right. Because VirtualDub configuration files are simply Sylia scripts, just like the job list, they can be invoked using the `/s` option to set the dub processing parameters. So to do automated processing of video files, you can set the parameters you want and save them in a configuration file, and then have a scheduling program invoke VirtualDub using a command line like this:

```
virtualdub /sparms.vcf /b"f:\ready","f:\output" /x /r
```

VirtualDub will then run the script to load parameters, scan the `f:\ready` directory and add job entries to process files there to `f:\output`, run the job list, and then exit.

Because VirtualDub is a GUI application, any console it is run from returns immediately, even before VirtualDub has finished processing. This is fine if you want to process in the background, but poses problems in a batch file. The solution is to use the `start` command to force a wait:

```
start /wait virtualdub /s"pack indeo5.vcf" /bin,out /x /r
```

This works under both Windows 95/98 and NT4.

Finally, if you need more specific control, you can always write a program to generate job scripts and append them onto the `virtualdub.jobs` file, which is simply text. You must be careful about the format, or VirtualDub can get confused, but this allows you much greater freedom in file and mode selection. Then, in a batch file, simply invoke VirtualDub with the `/r` and `/x` flags alone to run the job list.

Glossary

aspect ratio

The ratio of a picture's horizontal size to its vertical. Note that this is the ratio of *actual size* and not *resolution*; for instance, the 320x200 and 640x480 modes of a PC video card normally both have 4:3 aspect ratios on screen, but 320 divided by 200 is not the same as 640 divided by 480.

This can also refer to pixel aspect ratio. Nearly all video modes have 1:1 pixel aspect ratios, meaning the pixels are square (or round). The VGA's 320x200 mode, however, has a slightly elongated pixel, approximately 1:1.1.

bidirectional prediction

A form of motion prediction. Unlike normal motion prediction, bidirectional prediction allows a frame to be predicted from frames that both precede and follow it in the display order. This requires that frames be decoded out-of-order, for the later frame to be available for the backward prediction. MPEG B-frames are predicted this way.

bilinear filtering

A better way to process images when they must be sub- or supersampled. The fastest way to sample an image is to pick the closest pixel to the point being examined; bilinear filtering improves on this by grabbing the four closest pixels and adding them together in a weighted average, based on how close the point is to each of the four pixels. Bilinear filtering is most commonly associated with image resizing and texture mapping.

bicubic filtering

Resampling method which usually gives better results than bilinear filtering. Bicubic uses a 4x4 area instead of the 2x2 that bilinear uses, and maps cubic functions to the points instead of lines. Bicubic filtering is especially better at blowing up images, as opposed to shrinking them.

chrominance

The color of an area, ignoring its brightness (luminance).

chrominance subsampling

Encoding chrominance information at a lower resolution than luminance information, due to the fact that the human eye senses brightness detail better than color detail. Chrominance subsampling is often the first step in video compression, since it cuts out a good part of the data without sacrificing much quality. MPEG uses 2:1 subsampling in both horizontal and vertical directions, cutting out 25% of the data. Each 2x2 square of pixels thus has its own brightness, but share the same color. Intel Indeo goes farther, using 4:1 subsampling in each direction, cutting out 43% of the data. Chrominance subsampling is the major reason high-quality video codecs are often cause "color bleeding" around sharp edges of computer generated video.

composite video

A way of transferring both video luminance and chrominance information over only one wire, instead of two as in S-Video. This causes a loss in video quality, particularly in the chrominance, but it's ubiquitous; nearly every VCR accepts and outputs composite video, and some TVs will accept it as well. Older computer monitors will also accept composite video. Lousy as it is, composite video is better than modulated video, video attached to a TV channel.

compression

Removing patterns in data to reduce its size. "Random" is defined as the absence of patterns, so removing patterns makes data more random, or rather, increases its *entropy*; this is why a fancy phrase for compression is *entropy encoding*.

discrete cosine transform (DCT)

A method of translating waveforms, such as video data, into the sum of cosine waves. For a given quality, an image can usually be encoded as sums of cosine waves in less space than with raw pixels. However, the DCT gets a *lot* more complex as image size increases, so images are chopped into tiles, usually 8x8, to speed things up. The DCT is the major reason why "ringing" occurs in compressed video when the quality value is dropped a lot. MPEG uses

an 8x8 DCT for both luminance and chrominance information. See also wavelet transform.

entropy encoding

A longwinded term for compression.

frame rate

How fast successive images are displayed. Motion picture video displays 24 frames per second (fps); TV displays have frame rates anywhere between 25 fps and 60 fps, depending on how you look at it. Computer displays are usually at least 60 fps and may display as many as 120 frames per second. The faster the frame rate, the smoother the video will get, but the more data that needs to be encoded.

Huffman encoding

A form of compression that compresses data on the basis that values which appear more frequently are given encoded forms that are shorter than less frequent values. For instance, in Huffman-coded text, the letter 'e' would usually be given a shorter code than 'z', because 'e' occurs more often.

interlacing

Mixing parts of successive images to improve resolution without increasing necessary bandwidth. For instance, both NTSC and PAL transmit only half the total scanlines in each field. The scanlines from one field alternate with the scanlines from the next, so the picture has twice the vertical resolution. Unfortunately, this means at any one time the frame can have part of one image interleaved with part of another, which leads to big headaches when processing the video with a computer.

keyframe (also key frame)

A frame which is not dependent upon any others to be decoded. Anytime a video is started in the middle, the decoder must begin decoding frames from the last keyframe, since all subsequent frames up to the next keyframe are dependent on it. MPEG I-frames are keyframes.

luminance

The brightness of an area, without taking its color (chrominance) into account.

motion compensation (MC)

A way of correcting for errors in motion prediction. For instance, MPEG includes additional picture data that is added to the predicted video to make it better resemble the actual frame.

motion prediction

Using parts of previously decoded frames, in conjunction with motion data, to "predict" what data in subsequent frames will look like, based on that motion. The most common form, forward prediction, constructs later frames based on earlier ones. See also motion compensation and bidirectional prediction.

nearest neighbor

The quickest and dirtiest way to sample an image, particularly for resizing or texture mapping. When you need a pixel that lies between ones you actually have, pick the closest one. This is fairly acceptable for integral enlargements, but looks bad when shrinking images or when the scale values aren't integers. On the other hand, nearest neighbor is *extremely* fast compared to bilinear or bicubic methods.

NTSC

Abbreviation for *National Television Standards Committee*, as well as the video encoding standard predominant in the USA (and other areas that the author doesn't know enough about). It encodes video at 59.94 interlaced fields per second (29.97 frames per second), with alternating fields of 262 and 263 lines, making 525 lines of total vertical resolution. NTSC has also been called *Never The Same Color* due to its color encoding scheme, needed for compatibility with black-and-white TV sets. See also PAL.

PAL

Abbreviation for *Phase Alternating Line*, a video encoding standard common in Europe. It is similar to NTSC, but

encodes 625 total lines instead of 525, and has a 50Hz field rate instead of NTSC's 60Hz.

pixel (pel)

An abbrevation for *picture element*, but it has basically become a regular word due to common use (and few people remembering its heritage). It represents the smallest independent area of an image that can be manipulated and displayed.

quantization

Losing less-important parts of data to save space. For instance, if we have the sequence of numbers 17, 22, 24, 26, and 37, and we round each number to the nearest 10, the sequence becomes 20, 20, 20, 30, 40, and can be compressed much more easily. Obviously, once image data undergoes quantization, part of the image is lost and can't be recovered. Quantization of wavelet/DCT coefficients is the major cause of "ringing" in highly compressed video.

S-Video

A way of connecting two video devices using two separate connections, one for luminance information, and another for chrominance. This saves the trouble of mixing the two together into a single signal and splitting it out again, as with composite video, and results in better quality.

synchronization

Making sure that the video runs at the same speed as the audio, and at the right place. When the two streams run at different speeds or are offset from each other, events that are supposed to happen simultaneously in both streams do not. For instance, a person's lips move in the video, and then his words come out several seconds later.

vector quantization (VQ)

Representing an arbitrary vector by one of a fixed set of vectors. This is most commonly used in motion prediction, where an image block's movement from one frame to the next is represented by a vector. Instead of representing the vector with arbitrary (x,y) coordinates, VQ picks the closest vector in a table and then just records the place in the table instead of the whole vector.

wavelet transform

An alternative to the discrete cosine transform (DCT), the wavelet transform changes data, such as video data, into the sum of varying frequency wavelets. (Don't ask the author to explain what a wavelet is; he doesn't quite know himself.) Wavelets are sometimes used instead of the DCT because they are more versatile and don't slow down as much with larger images as the DCT does. Intel's Indeo technology makes use of wavelets.