# MJPEG HOWTO − An introduction to the MJPEG−tools

# Table of Contents

**MJPEG HOWTO − An introduction to the MJPEG−tools** ..... **1**
Praschinger Bernhard ..... 1
1. Introduction ..... 1
2. Unsorted list of useful Hints ..... 2
2.1 Some books we found usefull ..... 4
3. Recording videos ..... 4
3.1 lavrec examples ..... 4
3.2 Other recording hints ..... 6
3.3 Some information about the typical lavrec output while recording ..... 6
3.4 Notes about "interlace field order − what can go wrong and how to fix it" ..... 7
There are three bad things that can happen with fields ..... 7
How can I recognize if I have one of these Problems ? ..... 7
How can you fix it? ..... 8
Hey, what about NTSC movies ? ..... 9
4. Creating videos from other sources ..... 9
4.1 Creating videos from images ..... 9
4.2 Decoding streams with mplayer ..... 11
4.3 Decoding MPEG2 streams with mpeg2dec ..... 11
4.4 Other things to know ..... 11
5. Checking if recording was successful ..... 12
6. Edit the video ..... 13
6.1 Edit with glav ..... 13
6.2 Unify videos ..... 14
6.3 Separate sound ..... 14
6.4 Separate images ..... 14
6.5 Creating movie transitions ..... 14
7. Converting the stream to MPEG or DIVx videos ..... 16
7.1 Creating sound ..... 17
7.2 Converting video ..... 18
Scaling ..... 18
7.3 Putting the streams together ..... 19
7.4 Creating MPEG1 Videos ..... 20
MPEG1 Audio creation Example ..... 20
MPEG1 Video creation Example ..... 20
MPEG1 Multiplexing Example ..... 22
7.5 Creating MPEG2 Videos ..... 22
MPEG2 Audio creation Example ..... 23
MPEG2 Video creation Example ..... 23
Which values should be used for VBR Encoding ..... 23
Encoding destination TV (interlaced) or Monitor (progressive) ..... 24
MPEG2 Multiplexing Example ..... 25
7.6 Creating Video−CD's ..... 25
VCD Audio creation Example ..... 25
VCD Video creation Example ..... 25
VCD Multiplexing Example ..... 26
Creating the CD ..... 26
Notes ..... 26
Storing MPEGs ..... 26

# Table of Contents

**MJPEG HOWTO − An introduction to the MJPEG−tools**
7.7 Creating SVCD....27
SVCD Audio creation Example....27
SVCD Video creation Example....27
SVCD Multiplexing Example....28
SVCD Creating the CD....28
7.8 Creating DVD's....29
DVD Audio creation Example....29
DVD Video creation Example....29
DVD Mplex Example....29
DVD creation Example....30
7.9 Creating DIVX Videos....30
lav2avi.sh....30
8. Optimizing the stream....31
8.1 Scaling and offset correction....34
8.2 Frame rate conversion....35
9. Transcoding of existing MPEG−2....36
9.1 If you want to do every step on your own it will look something like this....38
10. Trading Quality/Speed....40
10.1 Creating streams to be played from disk using Software players....41
11. SMP and distributed Encoding....41
12. Interoperability....42

# MJPEG HOWTO − An introduction to the MJPEG−tools


## Praschinger Bernhard



v1.48



*MJPEG capture/editting/replay and MPEG encoding toolset description*


## 1. Introduction

I wrote this things down, because I had many sheets with notes on them. This should be some kind of summary of collected knowledge of this sheets. Andrew Stevens helped with encoding and VCD knowledge and hints.

The mjpegtools are a set of programs that can do recording, playback, editing and eventual MPEG compression of audio and video under Linux.

Although primarily intended for use with capture / playback boards based on the Zoran ZR36067 MJPEG codec chip, the mjpegtools can easily be used to process and compress MJPEG video streams captured using xawtv using simple frame−buffer devices.

The HOWTO for the tools intended to give an an introduction to the MJPEG−tools and the creation of MPEG 1/2 videos. VCD and SVCD, and the transcoding of existing mpeg streams.

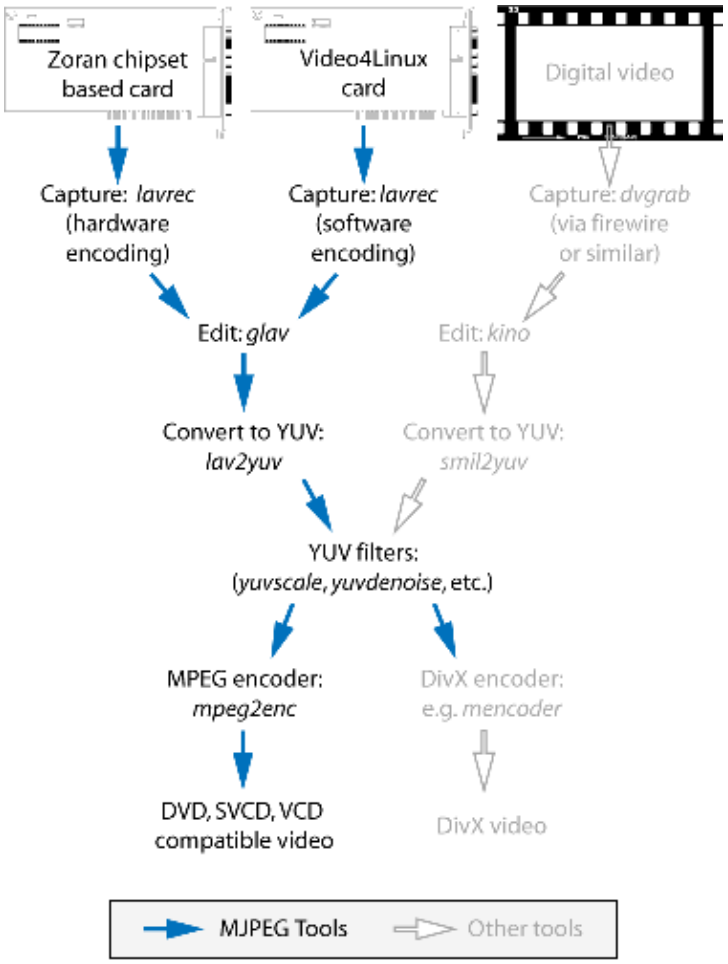
For more information about the programs read the corresponding man−page.

Achtung es gibt auch eine deutsche Version bei: http://sourceforge.net/projects/mjpeg

There is also a manpage of this text, you can read it with "man mjpegtools" if installed.

The text version of this text is available via cvs, you should get it with a tarball or the precompiled package (RPM and deb).

In the following picture you see the typical workflow when you record a video. Cut it afterwards and encode it. In the picture you also see the connections to other programs. These parts are in grey, the parts in blue can be done with the mjpegtools.



Video encoding workflow

# 2. Unsorted list of useful Hints

You have to compile and install the mjpeg_play package, for this read the README & REQUIRED_SOFTWARE & INSTALL. If you do not want to compile it, you can download the mjpeg .RPM or .DEB package at Sourceforge.

There is a script in the scripts/ directory. This script is something that show's you a way how it can be done. It also creates (under certain circumstances) videos that look quite good. Better videos you only get by tuning the parameters yourself.

You will usually have to load the drivers for the Buz or DC10 or LML33 cards.So you have to run the update script providing as option the name of your card you have. The script is usually in /usr/src/driver−zoran/. The zoran kernel driver below the kernel 2.4.4 do not work. You have to use the driver available from: http://mjpeg.sourceforge.net/driver−zoran

The driver for the Matrox Marvel card also works, more information about it: http://marvel.sourceforge.net

If you compile the tools on a P6 based computer (PPro, P−II, P−III, P−4, Athlon,Duron) then never try to let them run on a P5 based computer (Pentium, Pentium−MMX, K6, K6−x, Cyrix, Via, Winchip). You'll get a "illegal instruction" and the program won't work.

If lav2yuv dumps core then one possible cause is no dv support was included. To enable it make sure that libdv is installed on the system. This will be necessary if you are using a digital camera (or analog to DV converter such as the Canopus ADVC100) and converting the dv avi format into the MPEG format.

Start xawtv to see if you get an picture. If you want to use HW−playback of the recorded streams you have to start xawtv (any TV application works) once to get the streams played back. You should also check the settings of your mixer in the sound card.

If you compile the tools on a platform other than Linux not all tools will work. Mjpegtools on a OS/X system for example will not have V4L (video4linux) capability.

Never try to stop or start the TV application when lavrec runs. If you start or stop the TV application lavrec will stop recording, or your computer could get "frozen". This is a problem of v4l (video4linux).

This problem is solved with v4l2. If you use v4l2 you can record the video and stop and start the tv application whenever you want. But v4l2 is currently (7. Jan. 2003) only supported for the zoran based cards (BUZ, DC10, DC10+, LML33) if you use the CVS driver from mjpeg.sf.net tagged with ZORAN_VIDEODEV_2. And this driver only works with the 2.4.20 kernel and the 2.5.* development kernel.

One last thing about the data you get before we start:

```
Audio: ( Samplerate * Channels * Bitsize ) / (8 * 1024)
CD Quality:(44100 Samples/sec * 2 Chanels * 16 Bit) / (8 * 1024)=172,2 kB/sec

The 8 * 1024 convert the value from bit/sec to kByte/sec

Video: (width * height * framerate * quality ) / (200 * 1024)
PAL HALF Size : (352 * 288 * 25 * 80) / (200 * 1024) = 990 kB/sec
PAL FULL size : (720 * 576 * 25 * 80) / (200 * 1024) = 4050 kB/sec
NTSC HALF size: (352 * 240 * 30 * 80) / (200 * 1024) = 990 kB/sec
NTSC FULL size: (720 * 480 * 30 * 80) / (200 * 1024) = 4050 kB/sec
```

The 1024 converts the Bytes to kBytes. Not every card can record the size mentioned. The Buz and Marvel G400 for example can only record a size of 720x576 when using −d 1, the DC10 records a size of 384x288 when using −d 2.

When you add audio and video datarate this is what your hard disk has to be able to write constantly streaming, else you will have lost frames.

If you want to play with the **−−mjpeg−buffer−size**. Remember the value should be at least big enough that one frame fits in it. The size of one frame is: (width * height * quality ) / (200 * 1024) = kB If the buffer is too small the rate calculation doesn't match any more and buffer overflows can happen. The maximum value is 512kB.

How video works and the difference between the video types is explained here: http://www.mir.com/DMG/

There you also find how to create MPEG Still Images for VCD/SVCD.

A good description of DV (Digital Video) can be found here: http://www.iki.fi/znark/video/conversion/

## 2.1 Some books we found usefull

written in English:

- Digital Video and HDTV by Charles Poyton (ISBN 1−55860−792−7)
- Digital Video Compression by Peter Symes (ISBN 0−07−142487−3)
- Video Demystified by Keith Jack (ISBN 1−878707−56−6)

written in German:

- Fernsehtechnik von Rudolf Mäusl (ISBN 3−7785−2374−0)
- Professionelle Videotechnik − analoge und digitale Grundlagen von U. Schmidt (ISBN 3−540−43974−9)
- Digitale Film− und Videotechnik von U. Schmidt (ISBN 3−446−21827−0)

If you know some other really good book about that, write us!

# 3. Recording videos

## 3.1 lavrec examples

Recording with lavrec look's like this:

`> lavrec −f a −i P −d 2 record.avi`
: Should start recording now,

***−f a***
: use AVI as output format,

***−i P***
: use as input source the SVHS−In with PAL format,

***−d 2***
: the size of the pictures are half size (352x288)

***record.avi***
: name of the created file.

Recording is finished by pressing Crtl−C (nowadays: Strg−C). Sometimes using **−f A** instead of **−f a** might be necessary

Other example:

`> lavrec −f q −i n −d 1 −q 80 −s −l 80 −R l −U record.avi`
: Should start recording now,

***−f q***
: use Quicktime as output format,

***−i n***
: use Composite−In with NTSC format,

***−d 1***
: record pictures with full size (640x480)

***−q 80***
: set the quality to 80% of the captured image

***−s***
: use stereo mode (default mono)

***−l 80***
: set the recording level to 80% of the max during recording

***−R l***
: set the recording source to Line−In

***−U***
: With this lavrec uses the read instead of mmap for recording this is needed if your sound card does not support the mmap for recording.

Setting the mixer does not work with every sound card. If you record with 2 different settings and both recordings are equally loud you should setup the mixer with a mixer program. After that you should use the **−l −1** option when you record using lavrec

The size of the image depends on the card you use. At full size (−d 1) you get these image sizes: BUZ and LML33: 720x576, the DC10: 768x576

Other example:

```
> lavrec −w −f a −i S −d 2 −l −1 record%02d.avi
```

Should start recording,

***−w***
: Waits for user confirmation to start (press enter)

***−f a***
: use AVI as output format,

***−i S***
: use SECAM SVHS−Input (SECAM Composite recording is also possible: −i s)

***−d 2***
: the size of the pictures are half size

***−l −1***
: do not touch the mixer settings

***record%02d.avi***
: Here lavrec creates the first file named record00.avi after the file has reached a size of 1.6GB (after about 20 Minutes recording) it starts a new sequence named record01.avi and so on till the recording is stopped or the disk is full

Other example:

```
> lavrec −f a −i t −q 80 −d 2 −C europe−west:SE20 test.avi
```

Should start recording now,

***−f a***
: use AVI as output format,

***−i t***
: use tuner input,

***−q 80***
: set the quality to 80% of the captured image

***−d 2***
: the size of the pictures are half size (352x288)

***−C***
: choose TV channels, and the corresponding −it and −iT (video source: TV tuner) can currently be used on the Marvel G200/G400 and the Matrox Millenium G200/G400 with Rainbow Runner

extension (BTTV−Support is under construction). For more information on how to make the TV tuner parts of these cards work, see the Marvel/Linux project on: http://marvel.sourceforge.net

Last example:

```
> lavrec −f a −i p −g 352x288 −q 80 −s −l 70 −R l −−software−encoding
test03.avi
```

The two new options are **−g 352x288**, which sets the size of the video to be recorded when using **−−software−encoding**, this enables the software encoding of the recorded images. With this option you can also record from a bttv based card. The processor load is high. This option only works for generic video4linux cards (such as the brooktree−848/878 based cards), it doesn't work for zoran−based cards.

## 3.2 Other recording hints

All lavtools accept a file description like file*.avi, so you do not have to name each file, but that would also be a posibillity to do.

Note: More options are described in the man−page, but with this you should be able to get started.

Here are some hints for sensible settings. Turn the quality to 80% or more for −d 2 capture. At full resolution as low as 40% seems to be visually "perfect". −d 2 is already better than VHS video (a *lot*!). For a Marvel you should not set the quality higher than 50 when you record at full size (−d 1). If you use higher settings (−q 60) it is more likely that you will encounter problems. Higher settings will result in framedrops. If you're aiming to create VCD's then there is little to be gained recording at full resolution as you need to reduce to −d 2 resolution later anyway.

you can record at other sizes than the obvious −d 1/2/4. You can use combinations where you use halve horizontal size and full vertical size: −d 21. This would record for NTSC at a size of 352x480. This helps if you want to create SVCDs, scaling the 352 Pixles put to 480 is not that visible for the eye as if you would use the other combination −d 12. Where you have the full horzontal resolution and half vertical this Version will have a size of 720x288 for NTSC

## 3.3 Some information about the typical lavrec output while recording

0.06.14:22 int: 00040 lst:0 ins:0 del:0 ae:0 td1=0.014 td2=0.029

The first part shows the time lavrec is recording. **int:** the interval between two frames. **lst:** the number of lost frames. **ins and del:** are the number of frames inserted and deleted for sync correction. **ae:** number of audio errors. **td1 and td2** are the audio/video time−difference.

- **(int) frame interval** should be around 33 (NTSC) or 40 (PAL/SECAM). If it is very different, you'll likely get a bad recording and/or many lost frames
- **(lst) lost frames** are bad and mean that something is not working very well during recording (too slow HD, too high CPU usage, ...) Try recording with a greater decimation and possibly a lower quality.
- **(ins, del) inserted OR deleted frames** of them are normal −> sync. If you have many lost AND inserted frames, you're asking too much of your machine. Use less demanding options or try a

different sound card.
- **(ae) audio errors** are never good. Should be 0
- **(td1, td2) time difference**is always floating around 0, unless sync correction is disabled (−−synchronization!=2, 2 is default).

# 3.4 Notes about "interlace field order − what can go wrong and how to fix it"

Firstly, what does it mean for interlace field order to be wrong?

The whole mjpegtools image processing chain is frame−oriented. Since it is video material that is captured each frame comprised a top field (the 0th, 2nd, 4th and so lines) and a bottom field (the 1st, 3rd, 5th and so on lines).

## There are three bad things that can happen with fields

1. This is really only an issue for movies in PAL video where each film frame is sent as a pair of fields. These can be sent top or bottom field first and sadly it's not always the same, though bottom−first appears to be usual. If you capture with the wrong field order (you start capturing each frame with a bottom rather than a top or vice versa) the frames of the movie get split *between* frames in the stream. Played back on a TV where each field is displayed on its own this is harmless. The sequence of fields played back is exactly the same as the sequence of fields broadcast. Unfortunately, playing back on a Computer monitor where both fields of a frame appear at once it looks *terrible* because each frame is effectively mixing two moments in time 1/25sec apparent.
2. The two fields can simply be swapped somehow so that top gets treat as bottom and bottom treat as top. Juddering and "slicing" is the result. This occasionally seems to happen due to hardware glitches in the capture card.
3. Somewhere in capturing/processing the *order* in time of the two fields in each frame can get mislabeled somehow. This is not good as it means that when playback eventually takes place a field containing an image sampled earlier in time comes after an image sampled later. Weird "juddering" effects are the results.

## How can I recognize if I have one of these Problems ?

1. This can be hard to spot. If you have mysteriously flickery pictures during playback try encoding a snippet with the reverse field−order forced (see below). If things improve drastically you know what the problem was and what the solution is!
2. The two fields can simply be swapped somehow so that top gets treat as bottom and bottom treat as top. Juddering and "slicing" is the result. This occasionally seems to happen due to hardware glitches in the capture card. That problem lookes like that:

Interlacing problem

3. Somewhere in capturing/processing the *order* in time of the two fields in each frame can get mislabeled somehow. This is not good as it means that when playback eventually takes place a field

containing an image sampled earlier in time comes after an image sampled later. Weird "juddering" effects are the result.

If you use glav or lavplay be sure that you also use the **−F/−−flicker** option. This disables some things that make the picture look better.

If you want to look at the video you can also use yuvplay:

```
> lav2yuv | ... | yuvplay
```

If there is a field order problem you should see it with yuvplay.

## How can you fix it?

1. To fix this one the fields need to be "shifted" through the frames. Use yuvcorrect's **−T BOTT_FORWARD/TOP_FORWARD** to shift the way fields are allocated to frames. You can find out the current field order for an MJPEG file by looking at the first few lines of debug output from: **> lav2yuv −v 2 the_mjpeg_file > /dev/null** Or re−record exchanging **−f a** for **−F A** or vice−versa.
2. This isn't too bad either. Use a tool that simply swaps the top and bottom fields a second time. yuvcorrect can do this use the **−T LINE_SWITCH**.
3. Is easy to fix. Either tell a tool someplace to relabel the fields or simply tell the player to play back in swapped order (the latter can be done "indirectly" by telling mpeg2enc when encoding to **reverse the flag (−z b|t)** that tells the decoder which field order to use.

**In order to determine exactly what type of interlacing problem you have, you need to extract some frames from the recorded stream and take a look at them:**

```
> mkdir pnm
> lav2yuv −f 40 video.avi | y4mtoppm | pnmsplit − pnm/image%d.pnm
> rm pnm/image?.pnm
> cd pnm
> xv
```

First we create a directory where we store the images. The lav2yuv −f 40 writes only the first 40 frames to stdout. The mjpegtools y4mtoppm converts the frames to pnm images and the pnmsplit splits the picture into two frames in the picture to two single pictures. Then we remove the first 10 images because pnmsplit does not support the %0xd numbering. Without a leading zero in the number, the files will be sorted in the wrong order, leading to confusing playback.

Use your favorite graphic program (xv for example) to view the pictures. As each picture only contain one field out of two they will appear scaled vertically. If you look at the pictures you should see the movie slowly advancing.

If you have a film you should always see 2 pictures that are nearly the same (because the film frame is split into two field for broadcasting) after each other. You can observe this easily if you have comb effects when you pause the film because both fields will be displayed at the same time. The two pictures that belong together should have an even number and the following odd number. So if you take a look on pictures: 4 and 5 are nearly identical, 5 and 6 differ (have movement), 6 and 7 identical, 7 and 8 differ , ....

To fix this problem you have to use yuvcorrect's **−T BOTT_FORWARD or TOP_FORWARD**. You can also have the problem that the field order (top/bottom) is still wrong. You may have to use yuvcorrect a

second time with **−M LINE_SWITCH**, or use the mpeg2enc **−z (b|t)** option.

To see if you guessed correctly, extract the frames again, reordering them using yuvcorrect:

```
> lav2yuv −f 40 video.avi | yuvcorrect −T OPTION | y4mtoppm | pnmsplit −
pnm/image%d.pnm
```

Where "OPTION" is what you think it will corrects the problem. This is for material converted from film. Material produced directly for TV is addressed below.

### Hey, what about NTSC movies ?

Movies are broadcast in NTSC using "3:2" pulldown which means that half the capture frames contain fields from 1 movie frame and half fields from 2 frames. To undo this effect for efficient MPEG encoding you need to use yuvkineco.

If you have an interlaced source like a TV camera you have a frame consists of two fields that are recorded at different points in time and shown after each other. Spotting the problem here is harder. You need to find something moving horizontally from the left to the right. When you extract the fields, the thing should move in small steps from the left to the right, not one large step forward, small step back, large forward, small back..... You have to use the same options mentioned aboth to correct the problem.

Do not expect that the field order is always the same (top− or bottom−first) It may change between the channels, between the films, and it may even change within a film. If it changes constant you may have to encode with the mpeg2enc −I 1 or even −I 2.

You can only have this problems if you record at full size !!!

## 4. Creating videos from other sources

Here are some hints and descriptions of how to create the videos from other sources like images and other video types.

You might also be interested in taking a look at the **Transcoding of existing MPEG−2** section.

## 4.1 Creating videos from images

You can use jpeg2yuv to create a yuv stream from separate JPEG images. This stream is sent to stdout, so that it can either be saved into a file, encoded directly to a mpeg video using mpeg2enc or used for anything else.

Saving an yuv stream can be done like this:

```
> jpeg2yuv −f 25 −I p −j image%05d.jpg > result.yuv
```

Creates the file result.yuv containing the yuv video data with 25 FPS. The −f option is used to set the frame rate. Note that image%05d.jpg means that the jpeg files are named image00000.jpg, image00001.jpg and so on. (05 means five digits, 04 means four digits, etc.) The −I p is needed for specifing the interlacing. You have to check which type you have. If you don't have interlacing just choose p for progressive

If you want to encode a mpeg video directly from jpeg images without saving a separate video file type:

```
> jpeg2yuv −f 25 −I p −j image%05d.jpg | mpeg2enc −o mpegfile.m1v
```

Does the same as above but saves a mpeg video rather than a yuv video. See mpeg2enc section for details on how to use mpeg2enc.

You can also use yuvscaler between jpeg2yuv and mpeg2enc. If you want to create a SVCD from your source images:

```
> jpeg2yuv −f 25 −I p −j image%05d.jpg | yuvscaler −O SVCD | mpeg2enc −f
4 −o video.m2v
```

You can use the −b option to set the number of the image to start with. The number of images to be processed can be specified with the −n number. For example, if your first image is image01.jpg rather than image00.jpg, and you only want 60 images to be processed type:

```
>jpeg2yuv −b 1 −f 25 −I p −n 60 −j image*.jpg | yuv2lav −o
stream_without_sound.avi
```

Adding the sound to the stream then:

```
> lavaddwav stream_without_sound.avi sound.wav stream_with_sound.avi
```

For ppm input there is the ppmtoy4m util. There is a manpage for ppmtoy4m that should be consulted for additional information.

So to create a mpeg video try this:

```
>cat *.ppm | ppmtoy4m −o 75 −n 60 −F 25:1 | mpeg2enc −o output.m1v
```

Cat's each *.ppm file to ppmtoy4m. There the first 75 frames (pictures) are ignored and next 60 are encoded by mpeg2enc to output.m1v. You can run it without the −o and −n option. The −F options sets the frame rate, default is NTSC (30000:1001), for PAL you have to use −F 25:1.

Other picture formats can also be used if there is a converter to ppm.

```
>ls *.tga | xargs −n1 tgatoppm | ppmtoy4m | yuvplay
```

A list of filenames (ls *.tga) is given to xargs that executes the tgatoppm with one (−n 1) argument per call, and feeds the output into ppmtoy4m. This time the video is only shown on the screen. The xargs is only needed if the converter (tgatoppm), can only operate on a single image at a time.

If you want to use the ImageMagick 'convert' tool (a Swiss Army Knife) try:

```
>convert *.gif ppm:− | ppmtoy4m | yuvplay
```

That means take all '.jpg' images in directory, convert to PPM format, and pipe to stdout, then ppmtoy4m processes them ....

## 4.2 Decoding streams with mplayer

Decoding the streams with mplayer is a nice way of bringing every video that mplayer can play back to something you can edit or encode directly to a mpeg video with the mjpegtools. This method works with mplayer 1.0pre2 or newer

```
>mkfifo stream.yuv

>mplayer −nosound −noframedrop −vo yuv4mpeg anyfile.mpg &

>cat stream.yuv | yuv2lav −o mjpeg_wo.avi

>mplayer −vo null −ao pcm −aofile anyfile.wav anyfile.mpg
```

Now you have for example a mjpeg encoded AVI without sound. The sound will be in anyfile.wav. Now you can choose if you want to add the sound to the AVI with `lavaddwav` and edit the file and encode it.

You can also use instead of yuv2lav, mpeg2enc or any other tool from the mjpeg tools so your command might also look like that:

```
> cat stream.yuv | yuvdenoise | yuvscaler −O SVCD | mpeg2enc −f 4 −o
video_svcd.m2v
```

And cat the wav file into `mp2enc` to encode it to MP2 audio. The **−vo yuv4mpeg** option works well with other input types mentioned in the mplayer documentation.

## 4.3 Decoding MPEG2 streams with mpeg2dec

You can decode mpeg2 streams with the patched mpeg2dec version which creates yuv streams. You can pipe that into any other mjpegtools program. Or you use a mpeg2dec version directly from the libmpeg2 project and use the output mode pgmpipe. With the `pgmtoy4m` program from the mjpegtools you can convert that pgm output back to yuv.

If you ask yourself why there is a patched version and pgmtoy4m. The answer is that the patch for yuv output was sent several times to the libmpeg2 developers but was never included. Now we have two ways around that problem. Decoding looks like this:

```
> mpeg2dec −s −o pgmpipe ANYTS.VOB | pgmtoy4m −i t −a 10:11 −r 30000:1001
| mpeg2enc −f 8 newvideo.m2v
```

You can decode the audio as described in the **Transcoding of existing MPEG−2** Section.

## 4.4 Other things to know

If you have Transport Streams from your DVB card, or os Satelite Receiver you might want to demultiplex or cut them. A nice tool for that is **Project X** available from: http://forum.lucike.info/

You can process the streams afterwards as you would do with any mpeg movie or demultiplexed audio video. So the **Transcoding of existing MPEG−2** section of this document will be of interest.

# 5. Checking if recording was successful

You can use lavplay or glav. **IMPORTANT: NEVER** try to run xawtv and lavplay or glav with hardware playback, it will not work. For software playback it works fine.

```
>lavplay −p S record.avi
```

You should see the recorded video and hear the sound. But the decoding of the video is done by the CPU which will place a heavy load on the system. The advantage of this method is you don't need xawtv.

The better way:

```
>lavplay −p H record.avi
```

The video is decoded and played by the hardware. The system load is now very low. This will play it back on−screen using the hardware rather than software decoding.

You might also try:

```
> lavply −p C record.avi
```

Which will play it back using the hardware but to the video output of the card.

```
> glav record.avi
```

Does the same as lavplay but you have an nice GUI. The options for glav and lavplay are nearly the same. Using no option SW playback is used.

Using hardware playback a signal for the Composite and SVHS OUT is generated so you can view the movie on your TV.

```
> lav2yuv test.eli | yuvplay
```

Is a other way to get the video without sound. You can use yuvplay once in the encoding command. When you use yuvplay in the encoding command you see the changes made by filters and scaling. You can also use it for slow−motion debugging.

**NOTE:** After loading the driver's you have to start xawtv to set up some things lavplay and glav do not, but they are needed for HW−Playback. Don't forget to close xawtv !!

**NOTE2:** Do not try to send glav an lavplay into background, wont work correct !!!

**NOTE3:** SECAM playback is now (12.3.2001) only in monochrome, but the recording and encoding is done right.

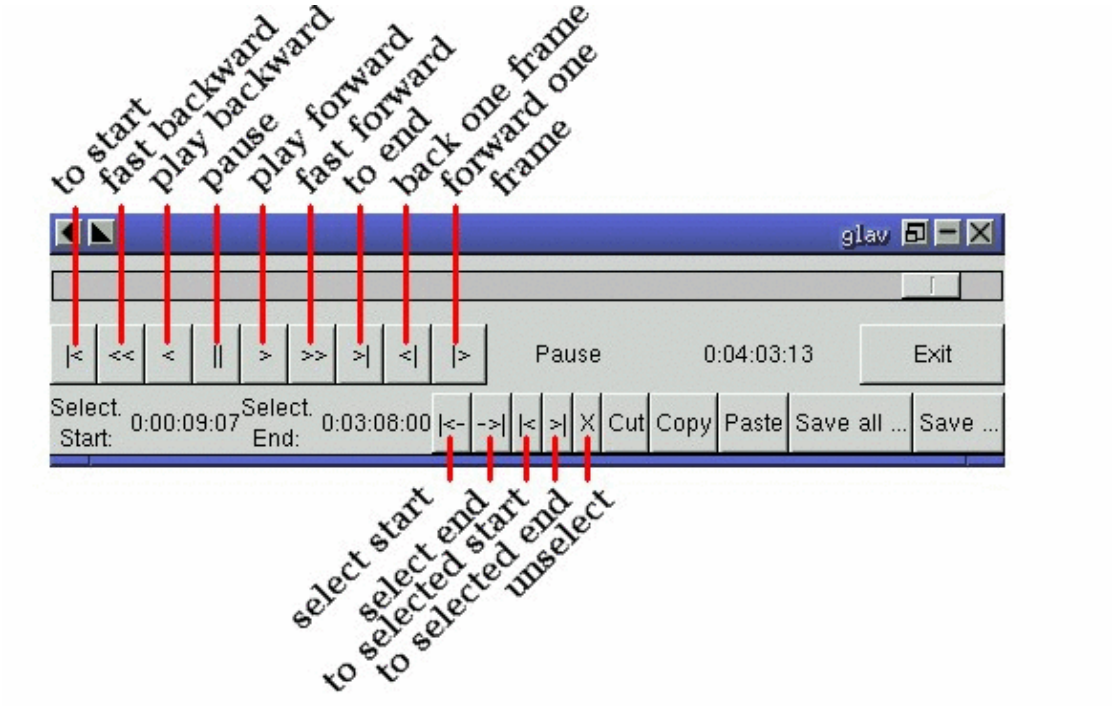
**NOTE4:**Bad cables may reduce the quality of the image. Normally you can't see this but when there is text you might notice a small shadow. When you see this you should change the cable.

**Coming soon:** There is a tool which makes recoding videos very simple: Linux Studio. You can download it at: http://ronald.bitfreak.net

# 6. Edit the video

## 6.1 Edit with glav

Most tasks can be easily done by glav. Like deleting parts of the video, cut paste and copy parts of the videos.




glav button

description

The modifications should be saved because glav does not destructively edit the video. This means that the original video is left untouched and the modifications are kept in an extra "Edit List" file readable with a text editor. These files can be used as an input to the other lavtools programs such as lav2wav, lav2yuv, lavtrans.

If you want to cut off the beginning and the end of the stream mark the beginning and the and, and use the "save select" button. The edit list file is than used as input for the lavtools. If you want to split a recorded video to some smaller parts simply select the parts and then save each part to a different listfile.

You can see all changes to the video and sound NOW and you do not need to recalculate anything.

If you want to get a "destructive" version of your edited video use:

```
> lavtrans −o short_version.avi −f a editlist.eli
```

***−o***
: specifies the output name

***−f a***
: specifies the output format (AVI for example)

***editlist.eli***
: is the list file where the modifications are described. You generate the list file with the "save all" or "save select" buttons in glav.

## 6.2 Unify videos

```
> lavtrans −o stream.qt −f q record_1.avi record_2.avi ... record_n.avi
```

***−o***
: specifies the outputfile name

***−f q***
: specifies the output format, quicktime in this case

This is usually not needed. Keep in your mind that there is the 2GB file−size−limit on 32Bit systems with an older glibc.

## 6.3 Separate sound

```
> lavtrans −o sound.wav −f w stream.avi
```

Creates a wav file with the sound of the stream.avi Maybe needed if you want to remove noise or if you want to convert it to another sound format.

Another way to split the sound is:

```
> lav2wav editlist.eli >sound.wav
```

## 6.4 Separate images

```
>mkdir jpg; lavtrans −o jpg/image%05d.jpg −f i stream.avi
```

First create the directory "jpg". Then lavtrans will create single JPG images in the jpg directory from the stream.avi file. The files will be named: image00000.jpg, image00001.jpg ....

The jpg images created contain the whole picture. But if you have recorded at full size the images are stored interlaced. Usually the picture viewers show only the first field in the jpg file.

If you want to have the image in a single file you can use that version

```
> lav2yuv −f 1 stream.avi | y4mtoppm −L >file.pnm
```

If you want to split the fields into single files use that:

```
> lav2yuv −f 5 ../stream.avi | y4mtoppm | pnmsplit − image%d.pnm
```

Maybe interesting if you need sample images and do not want to play around with grabbing a single image.

## 6.5 Creating movie transitions

Thanks to Philipp Zabel's lavpipe, we can now make simple transitions between movies or combine multiple layers of movies.

Philipp wrote this HOWTO on how to make transitions:

Let's assume simple this scene: We have two input videos intro.avi and epilogue.mov and want to make intro.avi transition into epilogue.mov with a duration of one second (that is 25 frames for PAL or 30 frames for NTSC).

Intro.avi and epiloque.mov have to be of the same format (the same frame rate and resolution). In this example they are both 352x288 PAL files. intro.avi contains 250 frames and epilogue.mov is 1000 frames long.

Therefore our output file will contain:

the first 225 frames of intro.avi
a 25 frame transition containing the last 25 frames of intro.avi and the first 25 frames of epilogue.mov
the last 975 frames of epilogue.mov

We could get the last 25 frames of intro.avi by calling:

```
>lav2yuv −o 225 −f 25 intro.avi
```

**−o 255**, the offset, tells lav2yuv to begin with frame # 225 and **−f 25** makes it output 25 frames from there on.

Another possibility is:

```
> lav2yuv −o −25 intro.avi
```

Since negative offsets are counted from the end.

And the first 25 frames of epilogue.mov:

```
> lav2yuv −f 25 epilogue.mov
```

**−o** defaults to an offset of zero

But we need to combine the two streams with lavpipe. So the call would be:

```
> lavpipe "lav2yuv −o 255 −f 25 intro.avi" "lav2yuv −f 25 epilogue.mov"
```

The output of this is a raw yuv stream that can be fed into transist.flt.

transist.flt needs to be informed about the duration of the transition and the opacity of the second stream at the beginning and at the end of the transition:

***−o num***
: opacity of second input at the beginning [0−255]

***−O num***
: opacity of second input at the end [0−255]

***−d num***
: duration of transition in frames

An opacity of 0 means that the second stream is fully transparent (only stream one visible), at 255 stream two is fully opaque.

In our case the correct call (transition from stream 1 to stream 2) would be:

```
> transist.flt −o 0 −O 255 −d 25
```

The −s and −n parameters equate to the −o and −f parameters of lav2yuv and are only needed if anybody wants to render only a portion of the transition for whatever reason. Please note that this only affects the weighting calculations − none of the input is really skipped. If you use the skip parameter (−s 30, for example) you also need to skip the first 30 frames in lav2yuv (−o 30) in order to get the expected result. If you didn't understand this send an email to the authors or simply ignore −s and −n. The whole procedure will eventually be automated.

Now we want to compress the yuv stream with yuv2lav:

```
> yuv2lav −f a −q 80 −o transition.avi
```

Reads the yuv stream from stdin and outputs an avi file (−f a) with compressed jpeg frames of quality 80.

Now we have the whole command for creating a transition:

```
> ypipe "lav2yuv −o 255 −f 25 intro.avi" "lav2yuv −f 25 epilogue.mov" |
transist.flt −o 0 −O 255 −d 25 | yuv2lav −f a −q 80 −o transition.avi
```

The resulting video can be written as a LAV Edit List, a plain text file containing the following lines:

```
LAV Edit List
PAL
3
intro.avi
transition.avi
epilogue.mov
0 0 224
1 0 24
2 25 999
```

This file can be fed into glav or lavplay, or you can pipe it into mpeg2enc with lav2yuv or combine the whole stuff into one single mjpeg file with lavtrans or lav2yuv|yuv2lav.

# 7. Converting the stream to MPEG or DIVx videos

First there is some general description in the encoding process and afterwards there is a detailed description of some commonly used output formats.

If you want a one command conversation to mpeg videos try lav2mpeg in the scripts directory

The encoding with the lav2mpeg script looks like this for mpeg1 output:

```
>lav2mpeg −a 160 −b 2110 −d 320x240 −m mpeg1 −o output.mpg file.eli
```

- Will create a mpeg1 with videobitrate of 2110kBit/sec and audiobitrate of 160 kBit/sec
- at a resolution of 320x240

Or for the generation of mpeg2 output:

```
lav2mpeg −o mpeg2 −O output.mpg file.eli
```

- Will create a mpeg2 with default bitrate in same resolution as the input resolution

Better results can be accomplished, however, by trying various options and find out which ones work best for you. These are discussed below.

The creation of MPEG1 movies is explained with more examples and in greater detail because most of the things that can be used for MPEG1 also work for the other output formats

For the creation of of VCD/SVCD Stills sequences (−f 6, −f 7 in mpeg2enc) you should see: http://www.mir.com/DMG/ Still sequences are needed for the creation of menus in VCD/SVCD. The creation of menus is described in the doku of vcdimager.

# 7.1 Creating sound

MPEG−1 videos need MPEG1−layer2 sound files. For MPEG−2 videos you can use MPEG1−Layer2 and MPEG1−Layer3 (MP3). Layer 3 (MP3) audio is not an offically valid audio format but many VCD players will recognize it. MP3 audio is not valid for DVDs. You should stick to MPEG1−Layer2 because most of the MPEG2 players (DVD Player for example usually the different Winxx Versions have great problems with this too) are not able to play MPEG2−Video and MPEG1−Layer3 sound.

mp2enc is a MPEG1−layer 2 Audio encoder. The toolame encoder is also able to produce an layer 2 file. You can use that one as well. Toolame is much faster than mp2enc but toolame does not peform resampling (48000 to 44100 samples/second). Many hardware players will play SVCDs using 48000 rate audio. For mp3 creation I'm be sure you have an encoder.

Example:

```
> lav2wav stream.avi | mp2enc −o sound.mp2
```

This creates a mpeg sound file out of the stream.avi with 224kBit/sec bitrate and a sample rate of 48kHz. If you audio file has 44.1kHz mp2enc resamples the audio to create a 48kHz output. If you want a 44.1kHz output sample rate you have to add −r 44100 to the mp2enc command

Example

```
> cat sound.wav | mp2enc −v 2 −V −o sound.mp2
```

This creates a VCD (−V bitrate=224, stereo, sampling rate:44100) compatible output from the wav file.

With −v 2 mp2enc is more verbose, while encoding you see the number of sec of audio already encoded.

You can test the output with:

```
> plaympeg sound.mp2
```

**NOTE:** plaympeg is a MPEG1 Player for Linux, you can use other players as well, for MPEG audio testing you can also use mpg123.

# 7.2 Converting video

You can create MPEG1 and MPEG2 videos.

Normally the first video you create is not the best. For optimal quality/size you need to play with the bitrate, search radius, noise filter .... The options of mpeg2enc are described in the manpage of mpeg2enc.

Example:

```
lav2yuv stream.avi stream1.avi | mpeg2enc −o video.m1v
```

This creates an video file with the default constant bitrate of 1152kBit/sec. This is the bitrate you need if you want to create VCDs. You can specify more files and also use the placeholder %nd. Where **n** describes the number.

Example:

```
> lav2yuv streami%02d.avi | mpeg2enc −b 1500 −r 16 −o video.m1v
```

mpeg2enc creates a video with a bitrate of 1500kBit/s uses an search radius of 16. That means when trying to find similar 16*16 macroblocks of pixels between frames the encoder looks up to 16 pixels away from the current position of each block. It looks twice as far when comparing frames 1 frame apart and so on. Reasonable values are 16 or 24. The default is 16 so adding the option here is silly. Lower values (0, 8), improve the encoding speed but you get lower quality (more visible artifacts) while higher values (24, 32) improve the quality at the cost of the speed. With the file description of stream%02d.avi all files are processed that match this pattern with 00, 01....

## Scaling

Using yuvscaler one can now also scale the video before encoding it. This can be useful for users with a DC10 or DC10+ cards which captures at −d 1 768x576 or −d 2 384x288 (PAL/SECAM) or −d 1 640x480 (NTSC).

You get a full description of all commands by reading the manpage or running:

```
>yuvscaler −h
```

Example:

```
> lav2yuv stream.avi | yuvscaler −O VCD | mpeg2enc −o video.m1v
```

This will scale the stream to VCD size which for PAL/SECAM is 352x288 and for NTSC is 352x240. The scaled yuvstream is encoded to MPEG−1.

It can also do SVCD scaling to 480x480 (NTSC) or 480x576 (PAL/SECAM):

```
> lav2yuv stream.avi | yuvscaler −O SVCD −M BICUBIC | mpeg2enc −o
video.m1v
```

The mode keyword (−M) forces yuvscaler to use the higher quality bicubic algorithms for downscaling and not the default resample algorithms. Upscaling is always done using the bicubic algorithm.

Example

```
> lav2yuv stream.avi | yuvscaler −I USE_450x340+20+30 −O SIZE_320x200 |
mpeg2enc −o video.m1v
```

Here we only use a part of the input and specify a nonstandard output resolution.

**NOTE:** yuvscaler can set a active area, and set everything else to black using: −I ACTIVE_WidthxHeight+WidthOffset+HeightOffset

Testing is done by:

```
> plaympeg video.m1v
```

**NOTE:**These are only examples. There are more options you can use. You can use most of them together to create high quality videos with the lowest possible bitrate.

**NOTE2:**The higher you set the search radius the longer the conversion will take. In general the more options used the longer encoding takes.

**NOTE3:**MPEG1 was not designed to be a VBR (variable bitrate stream) !! So if you encode with −q 15 mpeg2enc sets the maximal bitrate −b to 1152. If you want a VBR MPEG1 you have to set −b very high (2500).

**NOTE4:**Maybe you should give better names than video.mpg. A good idea is to use the options as part of the filename (for example: `video_b1500_r16_41_21.m1v`). Another possibility is to call all the layer 2 files ".mp2" all the MPEG−1 video files ".m1v" and all MPEG−2 video files ".m2v" Easy to see what's happening then. Reserve .mpg for multiplexed MPEG−1/2 streams.

## 7.3 Putting the streams together

Example:

```
> mplex sound.mp2 video.m1v −o my_video.m1v
```

Puts the sound.mp2 and the video.m1v stream together to my_video.mpg

Now you can use your preferred MPEG player and watch it. All players (gtv for example) based on the SMPEG library work well for MPEG−1. Other players (which can play MPEG−2 as well as MPEG−1 movies) are: xmovie, xine, and MPlayer.

**NOTE:** If you have specified the **−S** option for mpeg2enc mplex will automatically split the files if there is in the output filename a %d (looks like: −o test%d.mpg) The files generated this way are separate stand−alone MPEG steams!

**NOTE2:** xine might have a problem with seeking through videos. mplayer has a problem with the "seek backward/forward" with variable bitrate streams because it goes forward in the file the amount of data for a constant bitrate stream. That amount might be significantly more than 10 seconds or one minute (those are the amount mplayer seeks for each press of the arrow keys). So don't wonder if it seeks much more time forward or backward than you expect.

**Variable bit−rate multiplexing:** Remember to tell mplex you're encoding VBR (−V option) as well as mpeg2enc (see the example scripts). It *could* auto−detect but it is not working yet. You should tell mplex a video buffer size at least as large as the one you specified to "mpeg2enc" Sensible numbers for MPEG−1 might be a ceiling bit−rate of 2800Kbps, a quality ceiling (quantization floor) of 6 and a buffer size of 400K.

Example:

```
> mplex −V −r 1740 audio.mp2 video_vbr.m1v −o vbr_stream.mpg
```

Here we multiplex a variable bitrate stream. mplex is now a single pass multiplexer so it can't detect the maximal bitrate and we have to specify it. The data rate for the output stream is: audio bitrate + peak videobitrate + 1−2% for mplex information. If audio (−b 224) has 224kBit, video has 1500kBit (was encoded with −b 1500 −q 9) then we have 1724 * 1.01 or about 1740kBit.

Example:

```
> plaympeg my_video.mpg
```

oder

```
> gtv my_video.mpg
```

# 7.4 Creating MPEG1 Videos

For MPEG1 you can use mpeg layer 2 Audio and mpeg1 video. A subset of MPEG1 movies are VCD's. You can use VBR (Variable BitRate) for the Video (although VCDs are almost always use CBR video) but the Audio has to be CBR (Constant BitRate).

MPEG1 is recommended for picture sizes up to 352x288 for PAL and 352x240 for NTSC for larger sizes MPEG2 is the better choice. There is no exact resolution where MPEG1 is better than MPEG2. Just to make soure, MPEG1 can't handle interlaced sources. If you video is interlaced you need MPEG2 to get it proper encoded. .

## MPEG1 Audio creation Example

```
> lav2wav editlist.eli | mp2enc −r 44100 −o sound.mp2
```

You can save some bits by telling mp2enc to use a lower bitrate (−b option) like 160 or 192 kBit/s. The −r 44100 option forces mp2enc to generate a 44.1kHz audio file.

```
> lav2wav editlist.eli | mp2enc −b 128 −m −o sound.mp2
```

This creates a mono output with an bitrate of 128kBit/sec bitrate. The input this time is the editlistfile (can have any name) created with glav so all changes you made in glav are direct processed and handed over to mp2enc. You do NOT have to create an edited stream with lavtrans to get it converted properly.

## MPEG1 Video creation Example

```
> lav2yuv editlist.eli | mpeg2enc −b 2000 −r 24 −q 6 −o video.m1v
```

mpeg2enc creates an video with an bitrate of 2000kBit/s (or 2048000Bit/s) but the −q flag activates the variable bitrate and a quality factor of 6. It uses a search radius of 24.

**Explanation:**when mpeg2enc is invoked without the 'q' flag it creates "constantbit−rate" MPEG streams. Where (loosely speaking) the strength of compression (and hence picture quality) is adjusted to ensure that on average each frame of video has exactly the specified number of bits. Such constant bit−rate streams are needed for broadcasting and for low−cost hardware like DVD and VCD players which use slow fixed−speed player hardware.

Obviously this is fairly inefficient as it means inactive scenes use up bits that could better be "spent" on rapidly changing scenes. Setting the 'q' flag tells mpeg2enc to generate variable bit−rate streams. For such streams the bit−rate specified is simply the maximum permissible. The 'q' parameter specifies the minimum degree of compression to be applied by specifying how exactly picture information is recorded. Typically, 'q' would be set so that quiet scenes would use less than the specified maximum (around 6 or 8) but fast moving scenes would still be bit−rate limited. For archival purposes setting a maximum bit−rate high enough never to be reached (e.g. 10Mbps) and a q of 2 or 3 are reasonable choices.

Example:

```
> lav2yuv stream.avi | yuvscaler −I ACTIVE_352x240+0+24 | mpeg2enc −b
1152 −r 16 −4 1 −2 1 −o video.m1v
```

Usually there is at the top and at the bottom a nearly black border and a lot of bandwidth is used for something you do not like. The yuvscaler −I ACTIVE option sets everything that is not in the described area to black, but the imagesize (352x288) is not changed. So you have a real black border the encoder only uses a few bits for encoding them. You are still compatible with the VCD's format in this example. To determine the active window extract one frame to the jpeg format:

```
> lavtrans −f i −i 100 −o frame.jpg test.avi
```

Than use your favorite graphic display program to determine the active size. The −4 1 and −2 1 options improves the quality about 10% but conversion is slower.

At the size of 352x288 (1/2 PAL size, created when using the −d 2 option when recording) the needed bitrate is/should be between 1000 − 1500kBit/s. For NTSC it should be about the same, because the image is smaller, but there are more frames per second than in PAL.

Anyways, the major factor is quality of the original and the degree of filtering. Poor quality unfiltered material typically needs a higher rate to avoid visible artifacts. If you want to reduce bit−rate without annoying artifacts when compressing broadcast material you should try one (or more) of the noise filters.

Example:

```
> lav2yuv stream.avi | mpeg2enc −b 1500 −n s −g 6 −G 20 −P −o video.m1v
```

Here the stream.avi will be encoded with:

***−b 1500***
: a Bitrate of 1500kBit/sec

***−n s***
: the input Video norm is forced to SECAM

***−P***

This ensures that 2 B frames appear between adjacent I/P frames. Several common MPEG−1 decoders can't handle streams that do not have 2 B−frames between I/P frames

***−g 6 −G 20***

the encoder can dynamically change the group−of−pictures size to reflect scene changes. This is done by setting a maximum GOP (−G flag) size larger than the minimum (−g flag). For VCDs sensible values might be a minimum of 9 and a maximum of 15. For SVCD 9 and 15 would be good values. If you only want to play it back on SW player you can use other min−max values.

Example

```
> lav2yuv stream*.avi | mpeg2enc −b 1500 −r 16 −4 1 −2 1 −S 630 −B 260 −o
video_n1_1500_r16_41_21_S630_B240.m1v
```

lav2yuv processes all the stream files. Then mpeg2enc is given some options that make the encoded stream look nicer. Using **−S 630** means that mpeg2enc marks the stream so that mplex generates a new stream every 630MB. One important thing is the use of the **−B** option which specifies the non−video (audio and mplex information) bitrate. The **−B** value of 260 should be fine for audio with 224kBit and mplex information. For further information take a look at the encoding scripts in the scripts directory.

### MPEG1 Multiplexing Example

Example

```
>mplex sound.mp2 video.m1v −o my_video.mpg
```

Puts the sound.mp2 and the video.m1v stream together to my_video.mpg. It only works that easy if you have CBR (the −q option was not used with mpeg2enc).

Example

```
mplex −V −r 1740 audio.mp2 video_vbr.m1v −o vbr_stream.mpg
```

Here we multiplex a variable bitrate stream. mplex is now a single pass multiplexer so it can't detect the maximal bitrate and we have to specify it. The data rate for the output stream is: **audio bitrate + peak videobitrate + 1−2% for mplex information**. If audio (−b 224) has 224kBit, video has 1500kBit (was encoded with −b 1500 −q 9) then we have 1724 * 1.01 or about 1740kBit.

## 7.5 Creating MPEG2 Videos

MPEG2 is recommended for sources with a greater picture than 352x240 for NTSC and 352x288 for PAL. MPEG2 can also handle interlaced sources like recording from TV at full resolution.

MPEG2 allows the usage of mpeg layer 3 (mp3) sound. So you can use your favorite mp3encoder for the creation of the sound. However, MP3 audio is not valid for DVDs. It is best to use MP2 (Layer 2) audio. The audio can also be a VBR Stream.

MPEG2 is usually a VBR Stream. MPEG2 creation with optimization requires a lot of CPU power. A film with the double resolution is NOT 4 times larger than an MPEG1 Stream. Depending on your quality settings it will be about 1.5 up to 3 times larger than the MPEG1 Stream at its lower resolution.

## MPEG2 Audio creation Example

```
> lav2wav editlist.eli | mp2enc −o sound.mp2
```

This will fit the MPEG2 quite well. You can save some bits by telling mp2enc to use a lower bitrate (−b option) like 160 or 192 kBit/s. And might want to add −r 44100 so that mpeg2enc generates a 44.1kHz sampling rate audio file. I hope I don't need to explain the usage of an MP3 Encoder. But you should not use all the fancy options that are available.

## MPEG2 Video creation Example

```
> lav2yuv editlist.eli | mpeg2enc −f 3 −b 3000 −q 9 −o video.m2v
```

A very simple example for MPEG2 Video. The most important option is the −f 3. That tells mpeg2enc that it should create a MPEG2 stream. Because it is a generic MPEG2 you have to use the −b bitrate options. And should use the −q option because you usually want a space saving VBR Stream. When using VBR streams the −b option tells mpeg2enc the maximum bitrate that can be used. The −q option tell mpeg2enc what quality the streams should have. The bitrate has an upper bound of the value specified by −b.

```
> lav2yuv editlist.eli | mpeg2enc −f 3 −4 1 −2 1 −q7 −b 4500 −V 300 −P −g
6 −G 18 −I 1 −o video.m2v
```

This will generate a higher quality MPEG2 stream because the −4 1 and −2 1 options were used. With −b 4500 −q 7 you tell mpeg2enc the maximal bitrate and the quality factor. −V is the video buffer size used for decoding the stream. For SW playback it can be much higher than the default. Dynamic GOP is set with −g and −G. A larger GOP size can help reduce the bit−rate required for a given quality but very large sizes can introduce artifacts due to DCT/iDCT accumulated rounding errors. The −P option also ensures that 2 B frames appear between adjacent I/P frames. The −I 1 option tells mpeg2enc that the source is a interlaced material like videos. There is (time consuming) interlaced motion compensation logic present in mpeg2enc. Mpeg2enc will use that logic if the size of the frames you encode is larger than the VCD size for your TV Norm.

If you denoise the images with yuvdenoise and use the deinterlacing (−F) option you should tell mpeg2enc that it does not need to do motion estimation for interlaced material. You have to use the −I 0 option of mpeg2enc to say that the frames are already deinterlaced. This will save a lot of time when encoding. If you don't use −I 0 it will not cause problems, the encoding will just take longer.

You can also use scaling an options that optimize (denoise) the images to get smaller streams. These options are explained in detail in the following sections.

## Which values should be used for VBR Encoding

The −q option controls the minimum quantization of the output stream. Quantization controls the precision with which image information is encoded. The lower the value the better the image quality. Values below 4 are extremes and should only be used if you know what you are doing

Usually you have to set up a maximum bitrate with the −b option. The tricky task is to set a value for the −q option and the −b option that produces a nice movie without using too much bandwidth and does not introduce too many artifacts.

A quality factor should be chosen that way that the mplex output of Peak bit−rate and average bit−rate differ by about 20−25%. If the difference is very small (less than < 10%) it is likely that you will begin to see artifacts in high motion scenes. The most common cause of the average rate being too close (or equal) to the maximum rate is wrong value for the maximal bitrate or a quality factor that is too high.

A combination that will produce more artifacts than you can count is a SVCD with a maximal video bitrate of 2500kBit and a qualitfactor set to 1 or 2. For SVCD with a video limit of 2500kBit a quality factor of 7−11 fits quite good (8 is the default). If you use filter programs or have a very good source like digital TV, DVD like material or rendered pictures you can use a quality factor of 6 when creating SVCDs. If your SVCD/DVD player supports higher bitrates than the official 2788kBit/sec for the video and audio. When using a higher bitrate and quality factor action scenes will look much better but of course the playing time of the disc will be less.

The same (7−11) quality factor for a full size picture and a top bitrate of 3500 to 4000 kBit will produce few artifacts.

For SVCD/DVD you can expect a result like the one described if the maximal bitrate is not set too low:

```
q <= 6 real sharp pictures, and good quality
q <= 8 good quality
q >= 10 average quality
q >= 11 not that good
q >= 13 here even still sequences might look blocky
```

## Encoding destination TV (interlaced) or Monitor (progressive)

MPEG2 supports interlaced data in addition to the progressive format. A MPEG2 movie can be interlaced or progressive. It depends on the source (film or broadcast) and on the viewing device.

If you encode a film both fields should be the same. Deinterlace the stream with yuvdenoise −F, or if you have a high quality source, and don't need to use the denoiser, with yuvcorrect −T NOT_INTERLACED. Also set the mpeg2enc interlace−mode (−I) option to 0. This means that there is no interlacing. We do not really need deinterlacing here because there is no motion between the fields of the frame. We only need to unite the two fields into a single progressive frame.

This movie should play back an any device (TV or Monitor) without problems.

If you have an interlaced source (broadcast) you can encode it as interlaced stream. Or deinterlace the stream and encode it as progressive stream. If you deinterlace it with yuvdenoise −F, you will lose details. But if you plan to play the recorded stream on your DVD player and your TV it would not be wise to perform deinterlacing. If you only want to play it back on the Monitor (progressive display) the picture looks better when playing it back if it is deinterlaced. If the player you use can do deinterlacing it does not matter if your encoded video has interlaced frames or progressive frames.

If you plan to deinterlace the stream you can only do this with yuvdenoise −F, and set the mpeg2enc −I 0. If you do not want to deinterlace the stream you do not need to set any special option (do not use yuvdenoise −F and mpeg2enc −I 0)

If you like to pause the stream and look on the still you should deinterlace. Because then the image is flicker free when pausing.

If you have a film (progressive) with parts from a broadcast (interlaced) mixed together (like in a documentary where some parts from a speaker are recorded interlaced and other parts are filmed) you have to choose between good film sequences with average still images or average looking film sequences with good still images.

For good film with average stills do not deinterlace. For average film sequences with good stills then deinterlace (using −F and −I 0).

### MPEG2 Multiplexing Example

```
> mplex −f 3 −b 300 −r 4750 −V audio.mp3 video.mp3 −o final.mpg
```

Now both streams (a mp3 audio and a mpeg2 video) are multiplex into a single stream (final.mpg). You have to use the −f 3 option to tell mplex the output format. You also have to add the −b decoder buffers size with the same value used when encoding the video. −r is that rate of video + audio +1−2% of mplex information.

The −V option tells that your source for mplexing is a VBR stream. If you don't use this option mplex creates something like a CBR Stream with the bitrate you have told it with the −r option. These streams are usually get BIG.

## 7.6 Creating Video−CD's

VCD is a constrained version of MEPG1 streams. VCD format was defined by Philips. The goal was to use a single speed CD−drive and other cheap hardware (not flexible) to have a cheap HW−Player. Because of that there are some limitations on VCD's. The bitrate for video is 1152kBit and for audio layer 2 audio 224kBit stereo. You are not allowed to use the −q option, dynamic GOP sizes and the video buffer is limited to 46kB. The image size is limited to 352x240 for NTSC, an to 352x288 for PAL.

If you have no VCD (only) player and you plan to use your DVD player then it is quite possible that the DVD player will be flexible enough to allow higher bitrates, dynamic GOP sizes, larger video buffer and so on

### VCD Audio creation Example

```
> lav2wav stream.avi | mp2enc −V −o sound.mp2
```

**−V** force VCD 2.0 compatible output. There the audio samplerate is fixed to 44.1kHz. And you can choose the audio bitrate for mono audio to be 64, 96 or 192kBit/sec. If you have stereo audio you can choose 128, 192, 224 or 384kBit/sec. For hardware players, you should stick to 44.1 224kBps Stereo layer 2 Audio.

### VCD Video creation Example

```
> lav2yuv stream.avi | yuvscaler −O VCD | mpeg2enc −f 1 −r 16 −o
video.mpg
```

For a VCD compatible output the −f 1 sets all options in mpeg2enc as needed. It seems that many VCD players (Avex for example) are not able to play MPEG streams that are encoded with a search radius greater than 16 so do not use the −r option to override the default of 16.

```
> lav2yuv streams.eli | mpeg2enc −f 1 −4 1 −2 1 −S 630 −B 260 −P −o
```

```
video.m1v
```

Using **'−S 630'** means that mpeg2enc marks the stream so that mplex generates a new stream every 630MB. One important thing is the use of the **−B** option which specifies the non−video (audio and mplex information) bitrate. The −B value of 260 should be fine for audio with 224kBit and mplex information. For further information take a look at the encoding scripts in the scripts directory. So the multiplexed streams should easily fit on a CD with 650MB.

The default value (−B) is 700MB for the video. mpeg2enc marks automatically every stream at that size if the −B option is not used to set a different value. If you have a CD where you can write more data (perhaps as much as 800MB), you have to set the −S option or otherwise mpeg2enc will mark the stream at 700 MB, and mplex will split the stream there. Which is almost certainly not what you want.

## VCD Multiplexing Example

```
> mplex −f 1 sound.mp2 video.mpg −o vcd_out.mpg
```

The −f 1 option turns on a lot of weird stuff that otherwise has no place in a respectable multiplexer!

## Creating the CD

The multiplexed streams have to be converted to an VCD compatible. This is done by vcdimager http://www.vcdimager.org/

```
> vcdimager testvideo.mpg
```

Creates a **videocd.bin**, the data file, and a **videocd.cue** which is used as control file for cdrdao.

You use cdrdao to burn the image. Cdrdao is yet another fine Sourceforge project which is found at: http://cdrdao.sourceforge.net/

## Notes

For MPEG−1 encoding a typical (45 minute running time) show or 90 odd minute movie from an analog broadcast a constant bit−rate of around 1800 kBit/sec should be ideal. The resulting files are around 700M for 45 minutes which fits nicely as a raw XA MODE2 data track on a CD−R. For pure digital sources (DTV or DVD streams and similar) VCD 1152 works fine.

**Note:** If you encode VBR MPEG1 (−q) remember the Hardware was probably not designed to do the playback because it is not in the specifications. If it works be very happy. I've noticed that it helps when you have an MPEG1 Stream to tell vcdimager that it is an SVCD. vcdimager complains (but only with a warning and not a fatal error) but you should be able to burn it. This could convince the player to use different routines in its firmware and play it back correct, but there is no guarantee of that.

## Storing MPEGs

If you record the data as XA mode 2 tracks you can fit appreciably more on a CD (at the expense of error correction/detection). You can use vcdimager to do this and vcdxrip (part of the vcdimager package) to extract ("rip") the resulting files. For better Quality there are SVCD and XVCD and DVD.

Currently SVCD is fully supported with a pre−set format in mplex and tools to create disks. MPEG streams that can be played by DVD player hardware and software can readily produced using mpeg2enc/mplex

If your player doesn't support SVCD you may well find it can handle VCD streams that have much higher than standard bit−rates. Often as much as 2500kBit/sec is possible. The several brands of DVD players can also play wilding out of spec SVCD and VCD discs. With higher bit−rates and good quality source material it is worth trying mpeg2enc's −h flag which produce a stream that is as sharp as the limits of the VCD standard permits.

However, if your player supports it and you have the patience for the longer encoding times SVCD is a much better alternative. Using a more efficient MPEG format SVCD more than doubles VCD's resolution while typically producing files that are less than twice as big.

# 7.7 Creating SVCD

Super Video CD (SVCD) is an enhancement to Video CD that was developed by a Chinese government−backed committee of manufacturers and researchers. The final SVCD spec was announced in September 1998. A good explanation of the SVCD format from Philips can be found here: http://licensing.philips.com/information/cd/video/documents575.html.

Record at full TV resolution (means: −d 1 for PAL this is 720x576) The resolution is for NTSC is 480x480 of PAL 480x576, so you know why you should record at full size.

## SVCD Audio creation Example

```
> lav2wav stream.avi | mp2enc −V −e −o sound.mp2
```

The SVCD specifications permit a much wider choice of audio rates, it is not necessary to use 224 kBit/sec. Any audio rate between 32 and 384 kBit/sec is permitted. The audio may be VBR (Variable Bit Rate). The −e enables the CRC error protection for the audio. The CRC has to be enabled to be SVCD conform. But it seems that most players don't pay attention to the CRC information. The CRC information need 2 bytes per Audio frame

The approximate frame length formula for MPEG−1 layer−II is:

```
(frame length in bytes) = 144 * (byte rate) / (sample rate)
```

If you have the typical VCD settings the CRC data needs about 0,27% of the whole data. In the worst case, where you have a MONO 32k Bitrate stream, the CRC data needs 1,92%.

## SVCD Video creation Example

```
> lav2yuv stream.avi | yuvscaler −O SVCD | mpeg2enc −f 4 −q 7 −I 1 −V 200
−o video.m2v
```

***−f 4***
: sets the options for mpeg2enc to SVCD

***−q 7***
: tell mpeg2enc to generate a variable bitrate stream

***−I 1***

tell mpeg2enc to assume that the original signal is field interlaced video where the odd rows of pixels are sampled a half frame interval after the even ones in each frame. The −I 0 (progressive output (no field pictures)) option will also work for PAL

You can use lower bitrates, but the SVCD standard limits **total bit−rate (audio and video) to 2788800 Bit/sec**. So with 224Kbps audio and overhead 2550 may already be marginally too tight. Since the SVCD format permits any audio rate between 32 and 224 kBit/sec you can save a few bits/sec by using 192k audio (or for non−musical material 160k).

SVCD supports variable bitrate (VBR), because MPEG2 is usually VBR, but with the top video bitrate limit of 2500kBit/sec. With the −f 4 flag the encoder also sets dynamic GOP with a low limit of −g 6 and a high limit of −G 18. This saves a few bits/sec and improves the picture quality during scene changes. When encoding with −f 4 mpeg2enc ignores the video bitrate (−b) and search radius (−r) options. If you use −f 5 you have to specify the bitrate and other options to mpeg2enc.

Another possibility for movies in PAL (European style 25 frames/50 fields per sec) video is:

```
> lav2yuv stream.avi | yuvscaler −O SVCD | mpeg2enc −f 4 −I 0 −V 300 −o
video.m2v
```

Movies are shot on film at 24 frames/sec. For PAL broadcast the film is simply shown slightly "too fast" at 25 frame/sec (much to the pain of people with an absolute pitch sense of pitch). The −I 0 flag turns off the tedious calculations needed to compensate for field interlacing giving much faster encoding.

Unfortunately, movies broadcast in NTSC (US style 30 frames/60 fields sec) video this will produce very poor compression. The "pulldown" sampling used to produce 60 fields a second from a 24 frame a second movie means half the frames in an NTSC *are* field interlaced.

Don't forget the −S and −B options mentioned above. You want the stream to fit on a CD don't you ?

## SVCD Multiplexing Example

```
> mplex −f 4 −b 300 −r 2750 sound.mp2 video.m2v −o svcd_out.mpg
```

***−f 4***
: tells mplex to mplex a SVCD

***−r 2750***
: is the calculated Audio + Video Bitrate + 1−2% multiplex information

***−b 300***
: is the buffer available on the playback device (the same value as used for the video encoding (mpeg2enc's −V option).

## SVCD Creating the CD

Example:

```
> vcdimager −t svcd testvideo.mpg
```

Creates a **videocd.bin**, the data file, and a **videocd.cue** which is used as control file for cdrdao.

Use cdrdao to burn the image as mentioned earlier.

**NOTE:**If you want to build "custom" VCD/SVCD you will need to use the mplex −f 2 and −f 5 switches.

**NOTE:**The VCD and SVCD stuff may work on your HW player or not. There are many reports that it works quite well. Don't be worried if it does not work. Nor am I responsible for unusable CDs. ("coasters")

# 7.8 Creating DVD's

Everything in this section is very new. So it can change every day. The limitations I mention here might not exist in the current version.

You need obviously a DVD writer. I own a Ricoh DVD+RW that works, and I know of a DVD−RAM writer that is able to to burn DVD−R. That disks also work with a DVD−Player. Which programs you use for burning depends on the DVD writer drive.

For the creation and writing of the VOB, IFO and BUP files we use dvdauthor. Aviable from Sourceforge (you might have guessed it) http://dvdauthor.sourceforge.net.

## DVD Audio creation Example

```
> lav2wav stream.eli | mp2enc −o sound.mp2
```

The sample rate has to be 48kHz. The mp2enc does create by default a sample rate of 48kHz. If it is not a 48kHz mp2enc will resample the audio to get the sample rate. If the audio is recorded at 48kHz then no resampling is needed and toolame can be used for the encoding (it is faster than mp2enc).

## DVD Video creation Example

```
> lav2yuv stream.eli | mpeg2enc −f 8 −o video.m2v
```

***−f 8***

This sets the options correctly for a MPEG−2 video that is compliant with the DVD standard. The maximum bitrate is set to 7500kBps and the video buffer size is set to 230KB. The default quality factor is set to 8. mpeg2enc sets currenty no automatic sequence length as it does for VCD/SVCD.

The other options to get a low bitrate and high quality stream can also be used to override the default settings mentioned above. You can also use yuvdenoise to increase the picture quality if the input data is noisy (from a VHS tape for example). A typical command will look like this:

```
lav2yuv moby.eli | yuvdenoise −F −l 2 | mpeg2enc −f 8 −q 7 −4 1 −2 1 −P
−I 0 −N −o video_DVD.m2v
```

## DVD Mplex Example

```
> mplex −f 8 sound.mp2 video.m2v −o my_dvdlikestream.mpg
```

***−f 8***

Here again we specify that we want to have DVD like MPEG stream. mplex cannot do all the fancy

things allowed for a DVD, but it is close enough that the HW−DVD players accept it.

***−o***

there we specify the output filename.

### DVD creation Example

This topic will be covered by the documentation of the dvdauthor program. For questions please see dvdauthor In general it will work like this:

```
> makedvd stream1.mpg stream2.mpg ... my_dvdlikestream.mpg
```

You will get a directory with AUDIO_TS and VIDEO_TS directories. Burning the data from the disk to a DVD+R/+RW writer would be done like this:

```
growisofs −Z /dev/scd2 −dvd−video mydvd/
```

If you own a DVD+RW/+R drive a good place for more information is: DVD+RW/+R for Linux page. You also need a version of the cdrtools with **dvd−video** support. The cdrtools 1.11a27 is known to work but newer versions already exist.

For other writers the commands to write a DVD will be different. You can get some more information in the dvdauthor package. There is no guarantee that it will work at all !!!

## 7.9 Creating DIVX Videos

### lav2avi.sh

Another way of creating DIVX is the program **mencoder** which is from the mplayer project. http://www.mplayer.hu/. For more information about mencoder please read mencoder/mplayer help and documents. A first and a second pass give at the end of pass hints for bitrate which can be used for encoding to specific size (650 MB, 700 MB and 800 MB). The script lav2avi.sh uses this information if provided (for short streams it is omitted by mencoder). Look for parameter *preferedSize* in the script. You can also specify other parameters used for encoding with *encoderParam* option in the script. For a description of the usable parameters take a look in the mplayer/mencoder manual.

The outputfilename is that name of your input file (first option) but with the extension avi. If the size of file is less then specified by *preferedSize* it's because the source was of very high quality (no noise) and the specified bitrate was higher than required. You usually get 700MB for 1.5 hour film at half image size with bitrate around 900 that means for divx good quality (assuming good quality source material of course).

The script does a 3 step encoding:

- 1st step − audio encoding
- 2nd step − first video pass
- 3rd step − second video pass

The mplayer/mencoder documentation deprecates the use of the 3 pass encoding method (it can cause A/V sync problems) and recommends the use of the 2 pass method. The mencoder/mplayer documentation is extensive and has many helpful hints (and a bitrate calculator in the TOOLS/ directory).

For encoding use the fast ffmpeg (lavc) codec. It gives nice results together with high good performance. For audio encoding mp3 is used. For encoding of all parts it uses unix pipes. This mean that you DO NOT need additional space on your hard drive where all glav manipulations will be done. For audio encoding the script uses a FIFO queue.

If you want to tweak the script for your own needs, use these hints:

- Output of 1st step is file called **frameno.avi** with encoded audio
- 2nd step is using **frameno.avi** and output is text file called *lavc_stats.txt* with timing informations
- 3rd step is using **frameno.avi** and *lavc_stats.txt* for encoding the stream to the output file **movie2.avi**
- If you want change only video bitrate, keep the file **frameno.avi** comment out the 1st step encoding and repeate 2nd and 3rd step. Dont forget to remove the line where the **frameno.avi** is removed.

# 8. Optimizing the stream

Using filters helps to increase the image quality of constant bitrate (CBR) video streams. With VBR (variable bit rate) video the filesize is reduced.

Example:

```
> lav2yuv stream.avi | yuvmedianfilter | mpeg2enc −o video.m1v
```

Here the yuvmedianfilter program is used to improve the image. This removes some of low frequence noise in the images. It also softens the image a little. It takes a center pointer and averages the pixels around it that fall within the specified threshold. It then replaces the center pixel with this new value. You can also use the −r (radius) option for an other search radius.

**NOTE:**a radius greater than the default value of 2 is horrendously slow!

yuvmedianfilter has separate settings for luma and chroma. You can control the search radius and the trigger threshold independently. If you use a threshold of 0 then filtering is disabled (−t 0 disables luma filtering, −T 0 disables chroma filtering).

```
> lav2yuv stream.avi | yuvmedianfilter −r 3 −t 4 −T 0 | mpeg2enc −o
video.m1v
```

This example uses a search radius of 3 pixels for the luma, a threshold of 4 (the default is 2), and disables filtering for the chroma components. Sometimes, depending on the source material, median filtering of the chroma can cause a slight color shift towards green. Filtering on the luma component (disabling the chroma filtering) is the solution to that problem.

Example:

```
> lav2yuv stream.avi | yuvdenoise | mpeg2enc −o video.m1v
```

Now we are using yuvdenoise to improve the image. The filter mainly reduces color and luminance−noise and flickering due to phase errors but is also effective at removing speckles. If you want yuvdenoise to deinterlace the stream use the −F option.

By default yuvdenoise denoises interlaced if the input is interlaced. You can of course change the denoiser threshold (−t), as well as the different search radius (−r). You can also set a **active area** where everything outside that area is set to real black. Creating a black border can lower the bitrate of the encoded stream because pure black areas compress much better than noise (captures from analog sources such as VHS and 8mm usually have several lines at the time and bottom that are very noisy).

yuvdenoise uses a different approach to filter the noise. More information about how yuvdenoise works as well as descriptions of its options are found in the manpage.

If you have a high quality source you can only use **fast mode**. In the fast mode yuvdenoise only uses the bitnoise reduction. You might also use the mpeg2enc **−h/−−keep−hf** option. That option tells mpeg2enc to keep as much high frequency information as possible. Using −h will greatly increase the bitrate (filesize). If the bitrate is too close to the maximum (set with −b) the encoder will have to decrease the quality to avoid exceeding the maximum bitrate.

A builtin filter in mpeg2enc is the −N/−−reduce−HF option. This option is not really filter in the usual sense. Rather it changes how exactly the high frequency information is encoded. Often the high frequency is noise. You also have high frequencies on sharp borders or transitions. The −N option can have values between 0.0 and 2.0 where 0.0 does nothing (disables the high frequency quantizer boost) and 2.0 gives the maximum quantization boost. The value to use depends on the desired output quality and filesize. Values of −N less than 0.5 are very subtle while a value of 1.0 will achieve a good balance between bitrate reduction and output quality. Using −N values above 1.5 will noticeably reduce the sharpness of the output picture and are normally used only for poor quality sources (VHS tapes for example).

Using yuvmedianfilter's capability to only filter the chroma (−T) is moderately effective at reducing noise in dark scenes without softening the image during normal (brighter) scenes. Median filtering of the luma (−t) will produce a lower bitrate but can cause loss of detail (softening). Chroma only medianfiltering is less agressive and is a good choice to use in combination with yuvdenoise.

Combining the filters yuvdenoise, yuvmedianfilter and the mpeg2enc −N option gives a very fine degree of control over the bitrate (filesize). The reduction (or increase) in the bitrate depends on the source material and the exact encoding/filter options used. So we can give no exact numbers how much each option and combination will reduce the filesize, only guidelines.

Usually you should use the −N option in a range from 0.5 to 1.5. Below 0.5 it does not reduce the bitrate very much (but does preserve sharpness). At 1.5 and higher you will notice a softening in the video and possibly artifacts (halo/ringing) around edges of objects (text/subtitles especially). If you combine the filters you should use yuvdenoise and maybe afterwards yuvmedianfilter. Maybe yuvmedianfilter even after scaling. Having yuvmedianfilter in the chain does not reduce the bitrate that much. Often the use of yuvdenoise is enough. The yuvmedianfilter helps much if you have low quality sources, and not that much if you already have a rather good quality. When you combine the filter and option you will very likely reduce the filesize to about the half of the filesize without using the options and programs.

In general aggressive filtering will produce smaller files (lower bitrate) but reduce the quality (details) of the picture. Less aggressive filtering/processing will preserve more detail but result in larger files.

Example:

```
> lav2yuv stream.avi | yuvkineco −F 1 | mpeg2enc −o video.m1v
```

yuvkineco is used for NTSC sources. It does the conversation from 30000.0/1001.0 (about 29.97) fps to 24000.0/1001.0 (about 23.976) fps, you can call it "reverse 2−3 pulldown" more info about this in the README.2−3pulldown. yuvkineco does only remove NTSC specific problems.

If you want to improve the image you should also use yuvdenoise:

```
> lav2yuv stream.avi | yuvkineco | yuvdenoise | mpeg2enc −o video.m1v
```

Example

```
> lav2yuv stream.avi | yuvycsnoise | mpeg2enc −o video.m1v
```

yuvycsnoise is also used for NTSC and is specialized for NTSC Y/C separation noise. If video capture hardware has only a poor Y/C separator then at vertical stripes (especially red/blue) noises appear which seem checker flag and bright/dark invert per 1 frame. yuvycsnoise reduces noises of this type. You can also use different thresholds for luma/chroma and the optimizing method. This filter is not needed with working with DV (Digital Video) data.

yuvycsnoise works only correct when we have NTSC with:

- full height (480 lines)
- full motion captured (29.97 fps)
- captured with poor Y/C separator hardware

For more information about the yuvkineco and yuvycsnoise read the README in the yuvfilters directory.

If you want to experiment to determine the optimal settings for the denoiser, scaler and so on replace the mpeg2enc with yuvplay. yuvplay plays back the yuv frames so you can see if the options you have chosen are making the thing better or worse.

A command would look like this:

```
> lav2yuv stream.eli | yuvdenoise −options | yuvscaler −options | yuvplay
```

If you are looking for a hardware device that can make the video look better before you record it we currently know about two firms that produce such boxes.

One produced by SIMA: http://www.simacorp.com/scc.htm, that device will work with NTSC. And a one other produced by ELV (german distributor): http://www.elv.de there you find in the SHOP area, a section where you can take a look at their Video − Audio devices. Most of the ELV devices work only with PAL.

If you want to know how much each tool lowers the average bitrate. You can use this table to see what you can expect if you have a full size video and want to create a DVD with a qality factor of 5 and the allowed maximal bitrate of 8500kb/sec.

- no denoising : 8300 kb/s (mostly hitting the upper bound)
- yuvenoise : 7700 kb/s
- mpeg2enc −−reduce−hf : 7400 kb/s
- yuvdenoise + yuvmedianfilter : 6000 kb/s
- yuvdenoise + mpeg2enc −−reduce−hf : 4900 kb/s
- all of the above : 3600 kb/s

While **−N|−−reduce−hf** or yuvdenoise alone is only a modest improvement, together they reduce the bitrate substantially. There is not really much visible difference between using yuvdenoise alone and yuvdenoise with mpeg2enc −−reduce−hf. The usefull values are between 0.0 and 1.5. Where you can say that the higher the quality factor you want, the less this option improves. At a quality factor 4 you save using −N 1.0 about 1%. If you want a quality factor of 9 and use the −N 1.0 you might save up to 40%. But you might save less, that depends on the video you encode!!!

If you ask yourself why not alyways use all of the above filters? Hmmm, hard question. The image softens, and the encoding time increases. Each filter needs about the same amount of time as mpeg2enc needs for encoding the video.

If you have very high quality material and want to keep every detail you should try to use the mpeg2enc −−keep−hf|−h on the other hand

**Note:** The bitrate reduction you have depends on the material and on the noise of the images.

A other interresting mpeg2enc option is the **−E|−−unit−coeff−elim** option. This option is disabled by default. If you enable it, a special "unit coefficient elimination" algorithm, is applied to the encoded picture blocks. Basically this proceedure forces blocks of a type that do not carry much information (but use many bits to encode) to be skipped. A negative value examines the base (DC) as well as the AC coefficients. A positive value means that only texture (AC) coefficients are examined and possibly zeroed. The recommended values lies between −20 and +20. You usually can expect that you have a 5% decreased filesize. The amount the bitrate is reduced can vary considerably, the range spans from not really noticable up to 20%.

If you think a other quantization matrice will help use the **−K|−−custom−quant−matrices** option. You can try out your own quanitsation matrice or use another builtin than the default. You can choose between kvcd, tmpgenc, hi−res, and your own. Using −K usually makes the file smaller except the hi−res option (that makes files considerably larger). Exact guidelines are hard to give, sometime a other quanitsation matrix saves almost nothing, and the next time up to 20%. More than 20% is very unlikely, 10−15% at a moderate qualityfactor (−q 8−10) are likely. The higher the qualiy the less it saves, at a quality factor of 4−6 the reduction in bitrate may only be 5%

One thing to keep in mind is that the unit coefficient elimination and the quantization matrix option are decreasing the bitrate while maintaining the same visual quality. At this point you can chose to use the smaller file to increase the amount of video that will fit on the disc media or you could chose to increase the quality even more by lowering the −q value by 1 and make a larger (but higher quality) file.

# 8.1 Scaling and offset correction

The basic scaling is described in the **Converting video** section

The scaling, takes a part of the picture, and scales it to a larger or smaler size. The scaling is done by yuvscaler:

```
lav2yuv test.eli | yuvscaler −I USE_400x400+50+100 | yuvplay
```

Here we only take part of the picture and scale it up to the size of the original frame. But yuvscaler also changes the pixel aspect ratio. That means when you look at the stream using yuvplay it looks like a square in our example. After scaling, if the sample (pixel) aspect ratio were not changed, the video would not display with the proper aspect ratio. Yuvscaler compensates by adjusting the sample aspect ratio. If you have a

interlaced video, the height and HeightOffset have to be a multiple by 4 if the video is interlaced. Else the values (width, height, widthoffset, heightoffset) have to be a multiple of 2.

A problem that cannot be solved easily with scaling is when the picture is not centered horizontal. On one side you have no black pixels and on the other you have 30 for example. Scaling is here is the wrong solution. y4mshift is the perfect solution because it can shift the image to the left or right.

```
lav2yuv test.eli | y4mshift −n 20 | mpeg2enc −f 3 −b 4000 −q 10 −o
video.m2v
```

That will shift the image 20 pixels to the right. If you use a negative the image is shift to the left. You have to use a even number. The inserted pixels are set to black.

Some might wonder why the image is not centered and there is a black border around the image when you view what you have recorded. The reason for the black border is the TV (CRT = Catode Ray Tube) and a really interresting storry about how the TV standart was definden. But tha topic is described in other books.

The TV does not show the full picture. A part of the picture is not shown because the TV sets overscan (sometimes as much as 10% but more common today is 5%). But when you capture the video with a card you see the whole image including the border that TVs lose due to overscanning. A horizontal offset is usually not a problem of the capture card. It is a problem when the film is broadcast and not well synchronized with the image. This means that the scan of the source not exactly synchronized with the carrier signal, you wont see that on TV.

## 8.2 Frame rate conversion

Ever needed to convert the framerate from PAL to NTSC or the other direction around ? Or something much simpler like converting the framerate from 24FPS to 24000:1001 for conversation from a film frame rate to a valid NTSC frame rate.

Than **yuvfps** is your program. It can lower the framerate by dropping frames, or create a higher framerate by replicating frames. If you have a wrong framerate in the header you can only change the header of the YUV stream and not modify the stream.

Because the frames are only replicated (copied) you should denoise first and then change the framerate and scale at als last step. If you have a interlaced source you should also deinterlace before changeing the framerate. If you create a higher frame rate it is very likely that you will have weird flickers when you play it back. If you convert PAL to NTSC (30000:1001 FPS about 29,97 FPS) the frame rate will lower by about the factor 480/576 (NTSC lines / PAL lines). If you lower the frame rate from PAL to NTSC (at 24000:1001) or NTSC FILM (24FPS) the bitrate will be about (480 Lines * 24 FPS) / (576 Lines * 25FPS). If you change the frame rate before denoising the yuvdenoise will have problems finding the noise across the frames, so the needed bandwith will slightly increase.

Example

```
> lav2yuv video.eli | yuvfps −r 30000:1001 | yuvscaler −O SVCD | mpeg2enc
−f 4 −o video_ntsc_svcd.m2v
```

This is a example to convert the source video to a NTSC video running at 30000:1001 FPS (or about 29,97FPS) at SVCD size.

Example

```
> lav2yuv video.eli | yuvdenoise −F | yuvfps −r 24000:1001 | yuvscaler −O
SIZE_720x480 | mpeg2enc −f 3 −b 4000 −q 7 −o video_ntsc.m2v
```

This example shows how you should use the tools. Denoise first and than change the framerate and in the last step change the image size.

It can happen that yuvscaler or mpeg2enc do not detect the TV norm correct. If that happens you have to add the norm option **−n n/p/s** to the program that chooses the wrong norm.

If you know that the header tells the wrong framerate, you can simply change the framerate of the yuv header this way:

```
> lav2yuv video.eli | yuvfps −r 25:1 −c | mpeg2enc −f 3 −b 4000 −q 7 −o
video_pal.m2v
```

You need the **−c** option. To tell yuvfps that it only should change the header of the stream. With the **−r 25:1** you tell yuvfps the frame rate it should write into the header. In your example the PAL frame rate of 25 FPS. You always have to use the fractional form.

If you know that the header is wrong, and you need a different output bitrate you can do this in a single step:

```
> lav2yuv video.eli | yuvfps −s 24:1 −r 25:1 | mpeg2enc −o video.m1v
```

# 9. Transcoding of existing MPEG−2

For transcoding existing MPEG−2 streams from digital TV cards or DVD a lower data−rate than for broadcast will give good results. Standard VCD 1152 kbps typically works just fine for MPEG1. The difference is in the Signal/Noise ratio of the original. The noise in the analog stuff makes it much harder to compress.

One other very good guide that helps you transcoding videos can be found at:
http://www.bunkus.org/dvdripping4linux/index.html

You will also need to manually adjust the audio delay offset relative to video when multiplexing. Very often around 150ms delay seems to do the trick.

You have to download the ac3dec and mpeg2dec packages. You can find them at mjpeg homepage ( http://sourceforge.net/projects/mjpeg ). You also need sox and toolame.

In the scripts directory there is a **mpegtranscode** script that does most of the work.

So transcoding looks like this:

```
> mjpegtranscode −V −o vcd_stream mpeg2src.mpg
```

***−V***
: set's the options so that a VCD compatible stream is generated

***−o vcd_stream***
: a vcd_stream.m1v (video) and vcd_stream.mp2 (audio) is created

***mpeg2src.mpg***
: specifies the source stream

The script prints also something like this:

```
> SYNC 234 mSec
```

You will need to adjust the audio/video startup delays when multiplexing to ensure audio and video are synchronized. The exact delay (in milliseconds) that you need to pass to mplex to synchronize audio and video using the "−v"" is printed by the extract_ac3 tool labeled "SYNC" when run with the "s" flag. This is the value th mjpegtranscode script prints out after the **SYNC** word.

Then you need to multiplex them like this:

```
> mplex −f 1 −O 234 vcd_stream.mp2 vcd_stream.m1v −o lowrate.mpg
```

***−f 1***
: Mux format is set to VCD

***−O 234***
: Video timestamp offset in mSec, generated by the mjpegtranscoding script, there negative values are allowed

***vcd_stream.mp2 & vcd_stream.m1v***
: generated files by the script

***lowrate.mpg***
: the VCD compatible output stream

Here we have a SVCD (MPEG−2 video) example:

```
> mjpegtranscode −S −o svcd_stream mpeg2src.mpg
```

You have to multiplex it with:

```
> mplex −f 4 −O 234 svcd_stream.mp2 svcd_stream.m2v −o lowrate.mpg
```

**Problem:** There is sometimes a problem with NTSC and VCD playback because movies may be recoded with 3:2 pulldown NTSC with 60 fields/sec. mpeg2dec is designed for playback on computers and generates the original 24frames/sec bitrate. If you encode the video now 30frames/sec video is created. This video is now much too short for the encoded audio.

The transcoding can be made to work but it must be done manually:

```
> cat mpeg2src.mpg | mpeg2dec −s −o YUVs | mpeg2enc −I 0 −f 4 −q 9 −V 230
−p −P −o svcd_stream.m2v
```

The −p tells mpeg2enc to generate header flags for 3:2 pull down of 24fps movie. It may also work if you do not add the −p flag. You do not need the −p flag when transcoding to VCD format because it is not supported in mpeg1.

## 9.1 If you want to do every step on your own it will look something like this

Extracting Audio:

```
> cat test2.mpg | extract_ac3 − −s | ac3dec −o wav −p sound.wav
2>/dev/null
```

One of the first lines showed contains the label "SYNC" you have to use this time later when multiplexing. The 2>/dev/null redirects the output of ac3dec to /dev/null. In the next step you generate the mpeg audio file:

```
> cat sound.wav | mp2enc −V −v 2 −o audio.mp2
```

***−V***
: forces VCD format, the sampling rate is converted to 44.1kHz from 48kHz

***−v 2***
: unnecessary but if you use it mp2enc tells you how many seconds of the audio file are already encoded.

***−o***
: Specifies the output file.

cat test2.mpg | extract_ac3 − −s | ac3dec −o wav | sox −t wav /dev/stdin −t wav −r 44100 /dev/stdout | toolame −p 2 −b 224 /dev/stdin audio.mp2

One of the first lines again output contains the label "SYNC". You have to use this time (referred to as "SYNC_value" below) when doing the multiplexing.

For VCD creation use:

```
> cat test2.mpg " mpeg2dec −s −o YUVh " mpeg2enc −s −o video_vcd.m1v
```

: mpeg2dec:

***−s***
: tells mpeg2dec to use program stream demultiplexer

***−o YUVh***
: the output size of the extracted frames.

There are other output modes, try "mpeg2dec −−help" but the most important here are:

***YUV***
: is the full image size, unscaled

***YUVs***
: is SVCD size, it can only scale down to 2/3 of the original size

***YUVh***
: is VCD size, or about the half of the original size

Mplex with:

```
> mplex −f 1 −O SYNC_value audio.mp2 video_vcd.m1v −o vcd_stream.mpg
```

***−f 1***

generates an VCD stream

***−O SYNC_value***

the value mentioned above

For SVCD creation use:

```
> cat test2.mpg | mpeg2dec −s −o YUVs | mpeg2enc −f 4 −q 9 −V 230 −o
video_svcd.mpg
```

***−q 9***

Quality factor for the stream (VBR stream) (default q: 12)

***−V 230***

Target video buffer size in KB

***−o***

Output file

Mplex with:

```
> mplex −f 4 −b 230 audio.mp2 video_svcd −o svcd_stream.mpg
```

***−f 4***

generate an SVCD stream

***−b 200***

Specify the video buffer size by the playback device.

For other video output formats this might work:

```
> cat test2.mpg | mpeg2dec −s −o YUV | yuvscaler −O SIZE_320x200 −O
NOT_INTERLACED | mpeg2enc −o strange_video.m1v
```

If you want to edit mpeg streams, this also works, but in a slightly different way. For demultiplexing you can use bbdmux, from the bbtools package. Splits out either video or audio very cleanly. You can't get it any more from the homepage from Brent Beyler, it can still be found when you search for it using that keywords " bbtools linux −suse −blackbox". Currenty it can be found at: http://www.nop.org/inkling/

First run:

```
> bbdmux myvideo.mpg
```

You should get something like this:

```
Found stream id 0xE0 = Video Stream 0
Found stream id 0xC0 = MPEG Audio Stream 0
Found stream id 0xBE = Padding Stream
```

Extract audio with:

```
> bbdmux myvideo.mpg 0xC0 audio.mp1
```

Convert it to wav:

```
> mpg123 −w audio.wav audio.m1v
```

Extract video with:

```
> bbdmux myvideo.mpg 0xE0 video.m1v
```

Converting video to an mjpeg avi stream:

```
> cat video.m1v | mpeg2dec −o YUV | yuv2lav −f a −o test.avi
```

Then adding the sound to the avi:

```
> lavaddwav test.avi audio.wav final.avi
```

If the source video has already the size of the target video use −o YUV. Using YUVh makes the video the half size! The rest can be done just like editing and encoding other streams. If you have videos with ac3 sound you only have to adapt the commands above.

Extracting Audio:

```
> cat test2.mpg | extract_ac3 − −s | ac3dec −o wav 2>dev/null >sound.wav
```

Extract video:

```
> cat test2.mpg | mpeg2dec −s −o YUVh | yuv2lav −f a −q 85 −o test.avi
```

Adding the sound:

```
> lavaddwav test.avi sound.wav fullvideo.avi
```

**NOTE:**You need much disk space. 1GB of video has a size of about 2GB at SVCD format and of course disk space is needed for some temp files. Converting the video to mjpeg also takes some time. On my Athlon 500 I never get more than 6−7 Frames a second. You loose quality each time you convert a stream into an other format!

# 10. Trading Quality/Speed

If absolute quality is your objective a modest improvement can be achieved using the −4 and −2 flags. These control how ruthlessly mpeg2enc discards bad looking matches between sections of adjacent frames during the early stages of the search when it is working with 4*4 and 2*2 clusters of pixels rather than individual pixels. Setting −4 1 −2 1 maximizes quality. −4 4 −2 4 maximizes speed. Note that because the statistical criteria mpeg2enc uses for discarding bad looking matches are usually fairly reliable the increase/decrease in quality is modest (but noticeable).

Reducing the radius of the search for matching sections of images also increases speed. However due to the way the search algorithm works the search radius is in effect rounded to the nearest multiple of 8. Furthermore, on modern CPU's the speed gained by reducing the radius below 16 is not large enough to make the marked quality reduction worthwhile for most applications.

## 10.1 Creating streams to be played from disk using Software players

Usually MPEG player software is much more flexible than the hardware built into DVD and VCD players. This flexibility allows for significantly better compression to be achieved for the same quality. The trick is to generate video streams that use big video buffers (500KB or more) and variable bitrate encoding (the −f, −q flag to mpeg2enc). Software players will often also correctly play back the more efficient MPEG layer 3 (yes, "MP3" audio format. A good MP3 encoder like lame will produce results comparable to layer 2 at 224Kbps at 128Kbps or 160Kbps.

# 11. SMP and distributed Encoding

The degree to which mpeg2enc tries to split work between concurrently executing threads is controlled by the −M or −−multi−thread [0..32] option. This optimizes mpeg2enc for the specified number of CPUs. By default (−M 1), mpeg2enc runs with just a little multi−threading: reading of frames happens concurrently with compression. This is done to allow encoding pipelines that are split across several machines (see below) to work efficiently without the need for special buffering programs. If you are encoding on a single−CPU machine where RAM is tight you may find turning off multithreading altogether by setting −M 0 works slightly more efficiently.

For SMP machines with two ore more processors you can speed up mpeg2enc by setting the number of concurrently executing encoding threads's you wish to utilize (e.g. −M 2). Setting −M 2 or −M 3 on a 2−way machine should allow you to speed up encoding by around 80%. Values above 3 are accepted but have very little effect even on 4 cpu systems.

If you have a real fast SMP machine (currently 1.Aug.03) like a dual Athlon MP 2600 or something similar the −M 2 and the filtering might not keep both (or more) CPU's busy. The use of the buffer or bfr program with a 10−20MB buffer helps to keep both CPUs busy.

Obviously if your encoding pipeline contains several filtering stages it is likely that you can keep two or more CPU's busy simultaneously even without using −M. Denoising using yuvdenoise or yuvmedianfilter is particular demanding and uses almost as much processing power as MPEG encoding.

It you more than one computer you can also split the encoding pipeline between computers using the standard 'rsh' or 'rcmd' remote shell execution commands. For example, if you have two computers:

```
> rsh machine1 lav2yuv "mycapture.eli | yuvscaler −O SVCD | yuvdenoise" |
mpeg2enc −f 4 −o mycapture.m2vi
```

Here the computer where you execute the command is doing the MPEG encoding and "machine1" is the machine that is decoding scaling and denoising the captured video.

Obviously, for this to work "machine1" has to be able to access the video and the computer where the command is executed has to have space for the encoded video. In practice, it is usually well worth setting up network file−storage using "NFS" or other packages if you are going to do stuff like this. If you have three computers you can take this a stage further, one computer could do the decoding and scaling, the next could do denoising and the third could do MPEG encoding:

```
> rsh machine1 "lav2yuv mycapture.eli | yuvscaler −O SVCD" | yuvdenoise |
rsh machine3 mpeg2enc −f 4 −o mycapture.m2v
```

**NOTE:**How the remote command executions are set up so that the data is sent direct from the machine that produces it to the machine that consumes it.

In practice for this to be worthwhile the network you are using must be fast enough to avoid becoming a bottleneck. For Pentium−III class machines or above you will need a 100Mbps Ethernet.

For really fast machines a switched 100MBps Ethernet (or better!) may be needed.Setting up the rshd ("Remote Shell Daemon" needed for rsh to do its work and configuring "rsh" is beyond the scope of this document, but its a standard package and should be easily installed and activated on any Linux or BSD distribution.

Be aware that this is potentially a security issue so use with care on machines that are visible to outside networks!

# 12. Interoperability

Quicktime files capturing using lavrec can be edited using Broadcast2000. But Broadcast2000 is not available any more on heroinewarrior. mjpeg AVI files captured using the streamer tool from the xawtv package can be edited and compressed and played back using software. Hardware playback is not possible for such files due to limitations in the Zoran hardware currently supported. Videos recorded with NuppelVideo can also be processed with the mjpeg tools.

If you have a Macintosh (MAC) and want to use the mjpeg tools look there: http://www.sjoki.uta.fi/~shmhav/SVCD_on_a_Macintosh.txt and http://homepage.mac.com/rnc/

If you want to compile the mjpeg−tools on your MAC, our just want mpeg2enc and mplex compiled for the MAC take a look here: http://mjpeg.sf.net/MacOS/

Another guide can be found here: Video from an Analogue Camcorder. It covers the process of setting up the applications, recording, encoding and burning.

MPEG files produced using the tools are know to play back correctly on:

- dxr2 (hardware decoder card)
- xine http://xine.sourceforge.net/
- oms http://www.linuxvideo.org/
- dvdview http://rachmaninoff.informatik.uni−mannheim.de/dvdview/
- xmovie http://heroinewarrior.com/xmovie.php3
- mplayer http://mplayer.sourceforge.net/
- vlc http://www.videolan.org/
- MPEG1 only: mtv http://www.mpegtv.com/
- MPEG1 only: gtv http://packages.debian.org/stable/graphics/smpeg−gtv.html
- MS Media player version 6 and 7
- SW DVD Player

To find out what you HW−player (most of the time DVD player) can do take a look at: http://www.vcdhelp.com

It seems that the MS Media player likes MPEG−1 streams more if you have used −f 1 when multiplexing.

If you have any problems or suggestions feel free to mail me (Bernhard Praschinger): waldviertler@users.sourceforge.net There is a lot of stuff added from the HINTS which Andrew Stevens created. Wolfgang Göller and Steven M. Schultz checked the document for bugs and spelling mistakes.

And to the people who have helped me with program descriptions and hints, **thanks**