Ppmtompeg User Manual

Ppmtompeg

Updated: 23 July 2006
Table Of Contents

NAME

ppmtompeg - encode an MPEG-1 bitstream

SYNOPSIS

ppmtompeg [*options*] *parameter-file*

DESCRIPTION

This program is part of Netpbm.

ppmtompeg produces an MPEG-1 video stream. MPEG-1 is the first great video compression method, and is what is used in Video CDs (VCD). **ppmtompeg** originated in the year 1995. DVD uses a more advanced method, MPEG-2. There is an even newer method called MPEG-4 which is also called Divx. I don't know where one finds that used.

There's technically a difference between a compression method for video and an actual file (stream) format for a movie, and I don't know if it can be validly said that the format of the stream **ppmtompeg** produces is MPEG-1.

Mencoder from the Mplayer package is probably superior for most video format generation needs, if for no other reason than that it is more popular.

The programming library **PM2V** generates MPEG-2 streams.

Use Mplayer (not part of Netpbm) to do the reverse conversion: to create a series of PNM files from an MPEG stream.

param_file is a parameter file which includes a list of input files and other parameters. The file is described in detail below.

To understand this program, you need to understand something about the complex MPEG-1 format. One source of information about this standard format is the section Introduction to MPEG in theCompression FAQ.

OPTIONS

The **-gop**, **-combine_gops**, **-frames**, and **-combine_frames** options are all exclusive.

-stat stat_file This option causes **ppmtompeg** to append the statistics that it write to Standard Output to the file *stat_file* as well. The statistics use the following abbreviations: bits per block (bpb), bits per frame (bpf), seconds per frame (spf), and bits per second (bps).

These statistics include how many I, P, and B frames there were, and information about compression and quality.

-quiet *num_seconds* causes **ppmtompeg** not to report remaining time more often than every *num_seconds* seconds (unless the time estimate rises, which will happen near the beginning of the run). A negative value tells **ppmtompeg** not to report at all. 0 is the default (reports once after each frame). Note that the time remaining is an estimate and does not take into account time to read in frames.

-realquiet causes **ppmtompeg** to run silently,with the only screen output being errors. Particularly useful when reading input from stdin.

This option prevents ppmtompeg from printing a summary line for each frame

-float_dct forces **ppmtompeg** to use a more accurate, yet more computationally expensive version of the DCT.

-gop *gop_num*causes **ppmtompeg** to encode only the numbered GOP (first GOP is 0). The parameter file is the same as for normal usage. The output file will be the normal output file with the suffix **.gop.***gop_num*. **ppmtompeg** does not output any sequence information.

-combine_gops causes **ppmtompeg** simply to combine some GOP files into a single MPEG output stream. **ppmtompeg** inserts a sequence header and trailer. In this case, the parameter file needs only to contain the SIZE value, an output file, and perhaps a list of input GOP files (see below). If you don't supply a list of input GOP files is used, then **ppmtompeg** assumes you're using the same parameter file you used when you created the input (with the **-gop** option) and calculates the corresponding gop filenames itself. If this is not the case, you can specify input GOP files in the same manner as normal input files -- except instead of using INPUT_DIR, INPUT, and END_INPUT, use GOP_INPUT_DIR, GOP_INPUT, and GOP_END_INPUT. If no input GOP files are specified, then the default is to use the output file name with suffix **.gop.***gop_num*, with *gop_num* starting from 0, as the input files.

Thus, unless you're mixing and matching GOP files from different sources, you can simply use the same parameter file for creating the GOP files (**-gop**) and for later turning them into an MPEG stream (**-combine_gops**).

-frames *first_frame last_frame*This option causes **ppmtompeg** to encode only the frames numbered *first_frame* to *last_frame*, inclusive. The parameter file is the same as for normal usage. The output will be placed in separate files, one per frame, with the file names being the normal output file name with the suffix **.frame.***frame_num*. No GOP header information is output. (Thus, the parameter file need not include the GOP_SIZE value)

Use **ppmtompeg -combine_frames** to combine these frames later into an MPEG stream.

-combine_frames This option causes **ppmtompeg** simply to combine some individual MPEG frames (such as you might have created with an earlier run of **ppmtompeg -frames**) into a single MPEG stream. Sequence and GOP headers are inserted appropriately. In this case, the parameter file needs to contain only the SIZE value, the GOP_SIZE value, an output file, and perhaps a list of frame files (see below).

The parameter file may specify input frame files in the same manner as normal input files -- except instead of using INPUT_DIR, INPUT, and END_INPUT, use FRAME_INPUT_DIR, FRAME_INPUT, and FRAME_END_INPUT. If no input frame files are specified, then the default is to use the output file name with suffix **.frame.***frame_num*, with *frame_num* starting from 0, as the input files.

-nice This option causes **ppmtompeg** to run any remote processes "nicely," i.e. at low priority. (This is relevant only if you are running **ppmtompeg** in parallel mode. Otherwise, there are no remote processes). See 'man nice.'

-max_machines *num_machines*This option causes **ppmtompeg** to use no more than *num_machines* machines as slaves for use in parallel encoding.

-snr This option causes **ppmtompeg** to include the signal-to-noise ratio in the reported statistics. Prints SNR (Y U V) and peak SNR (Y U V) for each frame. In summary, prints averages of luminance only (Y). SNR is defined as 10*log(variance of original/variance of error). Peak SNR is defined as 20*log(255/RMSE). Note that **ppmtompeg** runs a little slower when you use this option.

-mse This option causes **ppmtompeg** to report the mean squared error per block. It also automatically reports the quality of the images, so there is no need to specify **-snr** then.

-bit_rate_info *rate_file* This option makes **ppmtompeg** write bit rate information into the file *rate_file*. Bit rate information is bits per frame, and also bits per I-frame-to-I-frame.

-mv_histogram This option causes **ppmtompeg** to print a histogram of the motion vectors as part of statistics. There are three histograms -- one for P frame, one for forward B frame, and one for backward B frame motion vectors.

The output is in the form of a matrix, each entry corresponding to one motion vector in the search window. The center of the matrix represents (0,0) motion vectors.

-debug_sockets This option causes **ppmtompeg** to print to Standard Output messages that narrate the communication between the machines when you run **ppmtompeg** in parallel mode.

-debug_machines This option causes **ppmtompeg** to print to Standard Output messages that narrate the progress of the conversion on the various machines when you run **ppmtompeg** in parallel mode.

PARAMETER FILE

The parameter file must contain the following lines (except when using the **-combine_gops** or **-combine_frames** options):

PATTERN *pattern*This statement specifies the pattern (sequence) of I frames, P frames, and B frames. *pattern* is just a sequence of the letters I, P, and B with nothing between. Example:

```
PATTERN IBBPBBPBBPBBPBB
```

See I Frames, P Frames, B Frames.

OUTPUT *output file*This names the file where the output MPEG stream goes.

INPUT_DIR *directory*This statement tells where the input images (frames) come from. If each frame is in a separate file, *directory* is the directory where they all are. You may use **.** to refer to the current directory. A null *directory* refers to the root directory of the system file tree.

To have **ppmtompeg** read all the frames serially from Standard Input, specify

```
INPUT_DIR stdin
```

INPUTThis line must be followed by a list of the input files (in display order) and then the line **END_INPUT**.

There are three types of lines between INPUT and END_INPUT. First, a line may simply be the name of an input file. Second, the line may be of the form *single_star_expr* [*x*-*y*]. *single_star_expr* can have a single * in it. It is replaced by all the numbers between x and y inclusive. So, for example, the line **tennis*.ppm [12-15]** refers to the files tennis12.ppm, tennis13.ppm, tennis14.ppm, tennis15.ppm.

Uniform zero-padding occurs, as well. For example, the line **football.*.ppm [001-130]** refers to the files football.001.ppm, football.002.ppm, ..., football.009.ppm, football.010.ppm, ..., football.130.ppm.

The third type of line is: *single_star_expr* [*x*-*y*+*s*], where the line is treated exactly as above, except that we skip by *s*. Thus, the line **football.*.ppm [001-130+4]** refers to the files football.001.ppm, football.005.ppm, football.009.ppm, football.013.ppm, etc.

Furthermore, a line may specify a shell command to execute to generate lines to be interpreted as described above, as if those lines were in the parameter file instead. Use back ticks, like in the Bourne Shell, like this:

```
`cat myfilelist`
```

If input is from Standard Input (per the **INPUT_DIR** statement), **ppmtompeg** ignores the

INPUT/END_INPUT block, but it still must be present.

BASE_FILE_FORMAT {**PPM** | **PNM** | **YUV** | **JPEG** | **JMOVIE**} **ppmtompeg** must convert all input files to one of the following formats as a first step of processing: PNM, YUV, JPEG(v4), or JMOVIE. (The conversion may be trivial if your input files are already in one of these formats). This line specifies which of the four formats. PPM is actually a subset of PNM. The separate specification is allowed for backward compatibility. Use PNM instead of PPM in new applications.

INPUT_CONVERT *conversion_command*You must specify how to convert a file to the base file format. If no conversion is necessary, then you would just say:

```
INPUT_CONVERT *
```

Otherwise, *conversion_command* is a shell command that causes an image in the format your specified with **BASE_FILE_FORMAT** to be written to Standard Output. **ppmtompeg** executes the command once for each line between **INPUT** and **END_INPUT** (which is normally, but not necessarily, a file name). In the conversion command, **ppmtompeg** replaces each '*' with the contents of that line. If you had a bunch of gif files, you might say:

```
INPUT_CONVERT giftopnm *
```

If you have a bunch of separate a.Y, a.U, and a.V files (where the U and V have already been subsampled), then you might say:

```
INPUT_CONVERT cat *.Y *.U *.V
```

Input conversion is not allowed with input from stdin, so use

```
INPUT_CONVERT *
```

as described above.

SIZE *width***x***height*

width and *height* are the width and height of each frame in pixels.

When **ppmtompeg** can get this information from the input image files, it ignores the **SIZE** parameter and you may omit it.

When the image files are in YUV format, the files don't contain dimension information, so **SIZE** is required.

When **ppmtompeg** is running in parallel mode, not all of the processes in the network have access to the image files, so **SIZE** is required and must give the same dimensions as the input image files.

YUV_SIZE *width***x***height*This is an obsolete synonym of **SIZE**.

YUV_FORMAT {**ABEKAS** | **PHILLIPS** | **UCB** |**EYUV** | *pattern*} This is meaningful only when **BASE_FILE_FORMAT** specifies YUV format, and then it is required. It specifies the sub-format of the YUV class.

GOP_SIZE *nn* is the number of frames in a Group of Pictures. Except that because a GOP must start with an I frame, **ppmtompeg** makes a GOP as much longer than *n* as it has to to make the next GOP start with an I frame.

Normally, it makes sense to make your GOP size a multiple of your pattern length (the latter is determined by the PATTERN parameter file statement).

See Group Of Pictures.

SLICES_PER_FRAME *nn* is roughly the number of slices per frame. Note, at least one MPEG player may complain if slices do not start at the left side of an image. To ensure this does not happen, make sure the number of rows is divisible by SLICES_PER_FRAME.

PIXEL {**FULL** | **HALF**} use half-pixel motion vectors, or just full-pixel ones It is usually important that you use half-pixel motion vectors, because it results in both better quality and better compression.

RANGE *n*Use a search range of *n* pixels in each of the four directions from a subject pixel. (So the search window is a square *n**2 pixels on a side).

PSEARCH_ALG {**EXHAUSTIVE** | **TWOLEVEL** |**SUBSAMPLE** | **LOGARITHMIC**} This statement tells **ppmtompeg** what kind of search technique (algorithm) to use for P frames. You select the desired combination of speed and compression. **EXHAUSTIVE** gives the best compression, but **LOGARITHMIC** is the fastest. **TWOLEVEL** is an exhaustive full-pixel search, followed by a local half- pixel search around the best full-pixel vector (the PIXEL option is ignored for this search technique).

BSEARCH_ALG {**SIMPLE** | **CROSS2** | **EXHAUSTIVE**}This statement tells **ppmtompeg** what kind of search technique (algorithm) to use for B frames. **SIMPLE** means find best forward and backward vectors, then interpolate. **CROSS2** means find those two vectors, then see what backward vector best matches the best forward vector, and vice versa. **EXHAUSTIVE** does an n-squared search and is extremely slow in relation to the others (**CROSS2** is about half as fast as **SIMPLE**).

IQSCALE *n*Use *n* as the qscale for I frames. See Qscale.

PQSCALE *n*Use *n* as the qscale for P frames. See Qscale.

BQSCALE *n*Use *n* as the qscale for B frames. See Qscale.

REFERENCE_FRAME {**ORIGINAL** | **DECODED**} Thisstatement determines whether **ppmtompeg** uses the original images or the decoded images when computing motion vectors. Using decoded images is more accurate and should increase the playback quality of the output, but it makes the encoding take longer and seems to give worse compression. It also causes some complications with parallel encoding. (see the section on parallel encoding). One thing you can do as a trade-off is select **ORIGINAL** here, and lower the qscale (see **QSCALE** if the quality is not good enough.

center ; Original or Decoded? (Normalized) <?makeman r c c c c c. ?> <?makeman _ ?> <tr align="center" bgcolor="#CCCCCC"> <th> Reference</th>

<th> Compression</th>

<th> Speed</th>

<th> Quality I</th>

<th> Quality P</th>

<th> Quality B</th>

</tr> <tr> <td align="right"> Decoded</td>

<td align="center"> 1000</td>

<td align="center"> 1000</td>

<td align="center"> 1000</td>

<td align="center"> 969</td>

<td align="center"> 919</td>

</tr> <tr> <td align="right"> Original</td>

<td align="center"> 885</td>

<td align="center"> 1373</td>

<td align="center"> 1000</td>

<td align="center"> 912</td>

<td align="center"> 884</td>

</tr>

The following lines are optional:

FORCE_ENCODE_LAST_FRAME This statement is obsolete. It does nothing.

Before Netpbm 10.26 (January 2005), **ppmtompeg** would drop trailing B frames from your movie, since a movie can't end with a B frame. (See I Frames, P Frames, B Frames. You would have to specify **FORCE_ENCODE_LAST_FRAME** to stop that from happening and get the same function that **ppmtompeg** has today.

NIQTABLEThis statement specifies a custom non-intra quantization table. If you don't specify this statement, **ppmtompeg** uses a default non-intra quantization table.

The 8 lines immediately following **NIQTABLE** specify the quantization table. Each line defines a table row and consists of 8 integers, whitespace-delimited, which define the table columns.

IQTABLEThis is analogous to NIQTABLE, but for the intra quantization table.

ASPECT_RATIO *ratio*This statement specifies the aspect ratio for **ppmtompeg** to specify in the MPEG output. I'm not sure what this is used for.

ratio must be 1.0, 0.6735, 0.7031, 0.7615, 0.8055, 0.8437, 0.8935, 0.9157, 0.9815, 1.0255, 1.0695, 1.0950, 1.1575, or 1.2015.

FRAME_RATE *rate*This specifies the frame rate for **ppmtompeg** to specify in the MPEG output. Some players use this value to determine the playback rate.

rate must be 23.976, 24, 25, 29.97, 30, 50, 59.94, or 60.

BIT_RATE *rate*This specifies the bit rate for Constant Bit Rate (CBR) encoding.

rate must be an integer.

BUFFER_SIZE *size* This specifies the value **ppmtompeg** is to specify in the MPEG output for the Video Buffering Verifier (VBV) buffer size needed to decode the sequence.

A Video Verifying Buffer is a buffer in which a decoder keeps the decoded bits in order to match the uneven speed of the decoding with the required constant playback speed.

As **ppmtompeg** encodes the image, it simulates the decoding process in terms of how many bits would be in the VBV as each frame gets decoded, assuming a VBV of the size you indicate.

If you specify the **WARN_VBV_UNDERFLOW** statement, **ppmtompeg** issues a warning each time the simulation underflows the buffer, which suggests that an underflow would occur on playback, which suggests the buffer is too small.

If you specify the **WARN_VBV_OVERFLOW** statement, **ppmtompeg** issues a warning each time the simulation overflows the buffer, which suggests that an overflow would occur on playback, which suggests the buffer is too small.

WARN_VBV_UNDERFLOW

WARN_VBV_OVERFLOW See **BUFFER_SIZE**.

These options were new in Netpbm 10.26 (January 2005). Before that, **ppmtompeg** issued the warnings always.

The following statements apply only to parallel operation:

PARALLELThis statement, paired with **END PARALLEL**, is what causes **ppmtompeg** to operate in parallel mode. See Parallel Operation.

END PARALLEL This goes with **PARALLEL**.

PARALLEL_TEST_FRAMES *n* The master starts off by measuring each slave's speed. It does this by giving each slave *n* frames to encode and noting how long the slave takes to finish. These are not just test frames, though -- they're real frames and the results become part of the output. **ppmtompeg** is old and measures time in undivided seconds, so to get useful timings, specify enough frames that it will take at least 5 seconds to process them. The default is 10.

If you specify **FORCE_I_ALIGN**, **ppmtompeg** will increase the test frames value enough to maintain the alignment.

If there aren't enough frames for every slave to have the indicated number of test frames, **ppmtompeg** will give some slaves fewer.

PARALLEL_TIME_CHUNKS *t*When you specify this statement, the master attempts to feed work to the slaves in chunks that take *t* seconds to process. It uses the speed measurement it made when it started up (see PARALLEL_TEST_FRAMES) to decide how many frames to put in the chunk. This statement obviously doesn't affect the first batch of work sent to each slave, which is the one used to measure the slave's speed.

Smaller values of *t* increase communication, but improve load balancing. The default is 30 seconds.

You may specify only one of PARALLEL_TIME_CHUNKS, PARALLEL_CHUNK_TAPER, and PARALLEL_PERFECT. PARALLEL_CHUNK_TAPER is usually best.

PARALLEL_CHUNK_TAPER When you specify this statement, the master distributes work like with PARALLEL_TIME_CHUNKS, except that the master chooses the number of seconds for the chunks. It starts with a large number and, as it gets closer to finishing the job, reduces it. That way, it reduces scheduling overhead when precise scheduling isn't helpful, but still prevents a slave from finishing early after all the work has already been handed out to the other slaves, and then sitting idle while there's still work to do.

You may specify only one of PARALLEL_TIME_CHUNKS, PARALLEL_CHUNK_TAPER, and PARALLEL_PERFECT. PARALLEL_CHUNK_TAPER is usually best.

PARALLEL_PERFECT If this statement is present, **ppmtompeg** schedules on the assumption that each machine is about the same speed. The master will simply divide up the frames evenly between the slaves -- each slave gets the same number of frames. If some slaves are faster than others, they will finish first and remain idle while the slower slaves continue.

This has the advantage of minimal scheduling overhead. Where slaves have different speeds, though, it makes inefficient use of the fast ones. Where slaves are the same speed, it also has the disadvantage that they all finish at the same time and feed their output to the single Combine Server in a burst, which makes less efficient use of the Combine Server and thus can increase the total elapsed time.

You may specify only one of PARALLEL_TIME_CHUNKS, PARALLEL_CHUNK_TAPER, and PARALLEL_PERFECT. PARALLEL_CHUNK_TAPER is usually best.

RSH *remote_shell_command***ppmtompeg** executes the shell command *remote_shell_command* to start a process on another machine. The default command is **rsh**, and whatever command you specify must have compatible semantics. **ssh** is usually compatible. The command **ppmtompeg** uses is one like this: **ssh remote.host.com -l username shellcommand**.

Be sure to set up **.rhosts** files or SSH key authorizations where needed. Otherwise, you'll have to type in passwords.

On some HP machines, **rsh** is the restricted shell, and you want to specify **remsh**.

FORCE_I_ALIGN This statement forces each slave to encode a chunk of frames which is a multiple of the pattern length (see **PATTERN**). Since the first frame in any pattern is an I frame, this forces each chunk encoded by a slave to begin with an I frame.

This document used to say there was an argument to **FORCE_I_ALIGN** which was the number of frames **ppmtompeg** would use (and was required to be a multiple of the pattern length). But **ppmtompeg** has apparently always ignored that argument, and it does now.

KEEP_TEMP_FILES This statement causes **ppmtompeg** not to delete the temporary files it uses to transmit encoded frames to the combine server. This means you will be left with a file for each frame, the same as you would get with the **-frames** option.

This is mostly useful for debugging.

This works only if you're using a shared filesystem to communicate between the servers.

This option was new in Netpbm 10.26 (January 2005).

Parameter File Notes

If you use the **-combine_gops** option, then you need to specify only the SIZE and OUTPUT values in the parameter file. In addition, the parameter file may specify input GOP files in the same manner as normal input files -- except instead of using INPUT_DIR, INPUT, and END_INPUT, use GOP_INPUT_DIR, GOP_INPUT, and GOP_END_INPUT. If you specify no input GOP files, then **ppmtompeg** uses by default the output file name with suffix **.gop.***gop_num*, with *gop_num* starting from 0, as the input files.

If you use the **-combine_frames** option, then you need to specify only the SIZE, GOP_SIZE, and OUTPUT values in the parameter file. In addition, the parameter file may specify input frame files in the same manner as normal input files -- except instead of using INPUT_DIR, INPUT, and END_INPUT, use FRAME_INPUT_DIR, FRAME_INPUT, and FRAME_END_INPUT. If no input frame files are specified, then the default is to use the output file name with suffix **as the input files.**

Any number of spaces and tabs may come between each option and value. Lines beginning with **#** are ignored. Any other lines are ignored except for those between INPUT and END_INPUT. This allows you to use the same parameter file for normal usage and for **-combine_gops** and **-combine_frames**.

The file format is case-sensitive so all keywords should be in upper case.

The statements may appear in any order, except that the order within a block statement (such as INPUT ... END INPUT) is significant.

ppmtompeg is prepared to handle up to 16 B frames between reference frames when encoding with input from stdin. (To build a modified **ppmtompeg** with a higher limit, change the constant B_FRAME_RUN in frame.c and recompile).

GENERAL USAGE INFORMATION

Qscale

The quantization scale values (qscale) give a trade-off between quality and compression. Using different Qscale values has very little effect on speed. The qscale values can be set separately for I, P, and B frames.

You select the qscale values with the **IQSCALE**, **PQSCALE**, and **BSCALE** parameter file statements.

A qscale value is an integer from 1 to 31. Larger numbers give better compression, but worse quality. In the following, the quality numbers are peak signal-to-noise ratio, defined as: where MSE is the mean squared error.

Flower garden tests:

center ; Qscale vs Quality <?makeman r r r r. ?> <?makeman _ ?> <tr align="center"> <th> Qscale</th>

<th> I Frames</th>

<th> P Frames</th>

<th> B Frames</th>

</tr> <tr> <td align="right"> 1</td>

<td align="right"> 43.2</td>

<td align="right"> 46.3</td>

<td align="right"> 46.5</td>

</tr> <tr> <td align="right"> 6</td>

<td align="right"> 32.6</td>

<td align="right"> 34.6</td>

<td align="right"> 34.3</td>

</tr> <tr> <td align="right"> 11</td>

<td align="right"> 28.6</td>

<td align="right"> 29.5</td>

<td align="right"> 30.0</td>

</tr> <tr> <td align="right"> 16</td>

<td align="right"> 26.3</td>

<td align="right"> 26.8</td>

<td align="right"> 28.6</td>

</tr> <tr> <td align="right"> 21</td>

<td align="right"> 24.7</td>

<td align="right"> 25.0</td>

<td align="right"> 27.9</td>

</tr> <tr> <td align="right"> 26</td>

<td align="right"> 23.5</td>

<td align="right"> 23.9</td>

<td align="right"> 27.5</td>

</tr> <tr> <td align="right"> 31</td>

<td align="right"> 22.6</td>

<td align="right"> 23.0</td>

<td align="right"> 27.3</td>

</tr>

center ; Qscale vs Compression <?makeman r r r r. ?> <?makeman _ ?> <tr align="center"> <th> Qscale</th>

<th> I Frames</th>

<th> P Frames</th>

<th> B Frames</th>

</tr> <tr> <td align="right"> 1</td>

<td align="right"> 2</td>

<td align="right"> 2</td>

<td align="right"> 2</td>

</tr> <tr> <td align="right"> 6</td>

<td align="right"> 7</td>

<td align="right"> 10</td>

<td align="right"> 15</td>

</tr> <tr> <td align="right"> 11</td>

<td align="right"> 11</td>

<td align="right"> 18</td>

<td align="right"> 43</td>

</tr> <tr> <td align="right"> 16</td>

<td align="right"> 15</td>

<td align="right"> 29</td>

<td align="right"> 97</td>

</tr> <tr> <td align="right"> 21</td>

<td align="right"> 19</td>

<td align="right"> 41</td>

<td align="right"> 173</td>

</tr> <tr> <td align="right"> 26</td>

<td align="right"> 24</td>

<td align="right"> 56</td>

<td align="right"> 256</td>

</tr> <tr> <td align="right"> 31</td>

<td align="right"> 28</td>

<td align="right"> 73</td>

<td align="right"> 330</td>

</tr>

Search Techniques

There are several different motion vector search techniques available. There are different techniques available for P frame search and B frame search. Using different search techniques present little difference in quality, but a large difference in compression and speed.

There are 4 types of P frame search: Exhaustive, TwoLevel, SubSample, and Logarithmic.

There are 3 types of B frame search: Exhaustive, Cross2, and Simple. The recommended search techniques are TwoLevel and Logarithmic for P frame search, and Cross2 and Simple for B frame search. Here are some numbers comparing the different search methods:

center ; P frame Motion Vector Search (Normalized) <?makeman r c c c. ?> <?makeman _ ?> <tr align="center"> <th> Technique</th>

<th> Compression¹ </th>

<th> Speed ² </th>

<th> Quality ³ </th>

</tr> <tr> <td align="right"> Exhaustive</td>

<td align="center"> 1000</td>

<td align="center"> 1000</td>

<td align="center"> 1000</td>

</tr> <tr> <td align="right"> SubSample</td>

<td align="center"> 1008</td>

<td align="center"> 2456</td>

<td align="center"> 1000</td>

</tr> <tr> <td align="right"> TwoLevel</td>

<td align="center"> 1009</td>

<td align="center"> 3237</td>

<td align="center"> 1000</td>

</tr> <tr> <td align="right"> Logarithmic</td>

<td align="center"> 1085</td>

<td align="center"> 8229</td>

<td align="center"> 998</td>

</tr>

center ; B frame Motion Vector Search (Normalized) <?makeman r c c c. ?> <?makeman _ ?> <tr align="center"> <th> Technique</th>

<th> Compression¹ </th>

<th> Speed² </th>

<th> Quality³ </th>

</tr> <tr> <td align="right"> Exhaustive</td>

<td align="center"> 1000</td>

<td align="center"> 1000</td>

<td align="center"> 1000</td>

</tr> <tr> <td align="right"> Cross2</td>

<td align="center"> 975</td>

<td align="center"> 1000</td>

<td align="center"> 996</td>

</tr> <tr> <td align="right"> Simple</td>

<td align="center"> 938</td>

<td align="center"> 1765</td>

<td align="center"> 991</td>

</tr>

¹ Smaller numbers are better compression. ² Larger numbers mean faster execution. ³ Larger numbers mean better quality.

For some reason, Simple seems to give better compression, but it depends on the image sequence.

Select the search techniques with the **PSEARCH_ALG** and **BSEARCH_ALG** parameter file statements.

Group Of Pictures (GOP)

A Group of Pictures (GOP) is a roughly independently decodable sequence of frames. An MPEG video stream is made of one or more GOP's. You may specify how many frames should be in each GOP with the **GOP_SIZE** parameter file statement. A GOP always starts with an I frame.

Instead of encoding an entire sequence, you can encode a single GOP. To do this, use the **-gop** command option. You can later join the resulting GOP files at any time by running **ppmtompeg** with the **-combine_gops** command option.

Slices

A slice is an independently decodable unit in a frame. It can be as small as one macroblock, or it can be as big as the entire frame. Barring transmission error, adding slices does not change quality or speed; the only effect is slightly worse compression. More slices are used for noisy transmission so that errors are more recoverable. Since usually errors are not such a problem, we usually just use one slice per frame.

Control the slice size with the **SLICES_PER_FRAME** parameter file statement.

Some MPEG playback systems require that each slice consist of whole rows of macroblocks. If you are encoding for this kind of player, if the height of the image is H pixels, then you should set the SLICES_PER_FRAME to some number which divides H/16. For example, if the image is 240 pixels (15 macroblocks) high, then you should use only 15, 5, 3, or 1 slices per frame.

Note: these MPEG playback systems are really wrong, since the MPEG standard says this doesn't have to be so.

Search Window

The search window is the window in which **ppmtompeg** searches for motion vectors. The window is a square. You can specify the size of the square, and whether to allow half-pixel motion vectors or not, with the **RANGE** and **PIXEL** parameter file statements.

I Frames, P Frames, B Frames

In MPEG-1, a movie is represented as a sequence of MPEG frames, each of which is an I Frame, a P Frame, or a B Frame. Each represents an actual frame of the movie (don't get confused by the dual use of the word "frame." A movie frame is a graphical image. An MPEG frame is a set of data that describes a movie frame).

An I frame ("intra" frame) describes a movie frame in isolation -- without respect to any other frame in the movie. A P frame ("predictive" frame) describes a movie frame by describing how it differs from the movie frame described by the latest preceding I or P frame. A B frame ("bidirectional" frame) describes a movie frame by describing how it differs from the the movie frames described by the nearest I or P frame before and after it.

Note that the first frame of a movie must be described by an I frame (because there is no previous movie frame) and the last movie frame must be described by an I or P frame (because there is no subsequent movie frame).

Beyond that, you can choose which frames are represented by which types. You specify a pattern, such as IBPBP and **ppmtompeg** simply repeats it over and over throughout the movie. The pattern affects speed, quality, and stream size. Here is a chart which shows some of the trade-offs:

center ; Comparison of I/P/B Frames (Normalized) <?makeman r c c c. ?> <?makeman _ ?> <tr align="center"> <th> Frame Type</th>

<th> Size</th>

<th> Speed</th>

<th> Quality</th>

</tr> <tr> <td align="right"> I frames</td>

<td align="center"> 1000</td>

<td align="center"> 1000</td>

<td align="center"> 1000</td>

</tr> <tr> <td align="right"> P frames</td>

<td align="center"> 409</td>

<td align="center"> 609</td>

<td align="center"> 969</td>

</tr> <tr> <td align="right"> B frames</td>

<td align="center"> 72</td>

<td align="center"> 260</td>

<td align="center"> 919</td>

</tr>

(this is with constant qscale)

A standard sequence is IBBPBBPBBPBBPBB.

Select the sequence with the **PATTERN** parameter file statement.

Since the last MPEG frame cannot be a B frame (see above), if the pattern you specify indicates B frames for the last movie frames, **ppmtompeg** makes it an I frame instead.

Before Netpbm 10.26 (January 2005), **ppmtompeg** instead drops the trailing B frames by default, and you need the **FORCE_ENCODE_LAST_FRAME** parameter file statement to make it do this.

The MPEG frames don't appear in the MPEG-1 stream in the same order that the corresponding movie frames appear in the movie -- the B frames come after the I and P frames on which they are based. For example, if the movie is 4 frames that you will represent with the pattern IBBP, the MPEG-1 stream will start with an I frame describing movie frame 0. The next frame in the MPEG-1 stream is a P frame describing movie frame 3. The last two frames in the MPEG-1 stream are B frames describing movie frames 1 and 2, respectively.

Specifying Input and Output Files

Specify the input frame images with the **INPUT_DIR**, **INPUT**, **END_INPUT**, **BASE_FILE_FORMAT**, **SIZE**, **YUV_FORMAT** and **INPUT_CONVERT** parameter file statements.

Specify the output file with the **OUTPUT** parameter file statement.

Statistics

ppmtompeg can generate a variety of statistics about the encoding. See the **-stat**, **-snr**, **-mv_histogram**, **-quiet**, **-no_frame_summary**, and **-bit_rate_info** options.

PARALLEL OPERATION

You can run **ppmtompeg** on multiple machines at once, encoding the same MPEG stream. When you do, the machines are used as shown in the following diagram. We call this "parallel mode."

To do parallel processing, put the statement

PARALLEL

in the parameter file, followed by a listing of the machines, one machine per line, then

END_PARALLEL

Each of the machine lines must be in one of two forms. If the machine has filesystem access to the input files, then the line is:

machine user executable

The executable is normally **ppmtompeg** (you may need to give the complete path if you've built for different architectures). If the machine does not have filesystem access to the input files, the line is:

REMOTE *machine user executable parameter file*

The **-max_machines** command option limits the number of machines **ppmtompeg** will use. If you specify more machines in the parameter file than **-max_machines** allows, **ppmtompeg** uses only the machines listed first. This is handy if you want to experiment with different amounts of parallelism.

In general, you should use full path file names when describing executables and parameter files. This includes the parameter file argument on the original invocation of **ppmtompeg**.

All file names must be the same on all systems (so if e.g. you're using an NFS filesystem, you must make sure it is mounted at the same mountpoint on all systems).

Because not all of the processes involved in parallel operation have easy access to the input files, you must specify the **SIZE** parameter file statement when you do parallel operation.

The machine on which you originally invoke **ppmtompeg** is the master machine. It hosts a "combine server,", a "decode server," and a number of "i/o servers," all as separate processes. The other machines in the network (listed in the parameter file) are slave machines. Each hosts a single process that continuously requests work from the master and does it. The slave process does the computation to encode MPEG frames. It processes frames in batches identified by the master.

The master uses a remote shell command to start a process on a slave machine. By default, it uses an **rsh** shell command to do this. But use the **RSH** parameter file statement to control this. The shell command the master executes remotely is **ppmtompeg**, but with options to indicate that it is to perform slave functions.

The various machines talk to each other over TCP connections. Each machine finds and binds to a free TCP port number and tells its partners the port number. These port numbers are at least 2048.

Use the PARALLEL_TEST_FRAMES, PARALLEL_TIME_CHUNKS, and PARALLEL_PERFECT parameter file statements to control the way the master divides up work among the slaves.

Use the **-nice** command option to cause all slave processes to run "nicely," i.e. as low priority processes. That way, this substantial and long-running CPU load will have minimal impact on other, possibly interactive, users of the systems.

SPEED

Here is a look at **ppmtompeg** speed, in single-node (not parallel) operation:

center ; Compression Speed <?makeman r c. ?> <?makeman _ ?> <tr align="center"> <th> Machine Type</th>

<th> Macroblocks per second¹ </th>

</tr> <tr> <td align="right"> HP 9000/755</td>

<td align="center"> 280</td>

</tr> <tr> <td align="right"> DEC 3000/400</td>

<td align="center"> 247</td>

</tr> <tr> <td align="right"> HP 9000/750</td>

<td align="center"> 191</td>

</tr> <tr> <td align="right"> Sparc 10</td>

<td align="center"> 104</td>

</tr> <tr> <td align="right"> DEC 5000</td>

<td align="center"> 68</td>

</tr>

¹ A macroblock is a 16x16 pixel square

The measurements in the table are with inputs and outputs via a conventional locally attached filesystem. If you are using a network filesystem over a single 10 MB/s Ethernet, that constrains your speed more than your CPU speed. In that case, don't expect to get better than 4 or 5 frames per second no matter how fast your CPUs are.

Network speed is even more of a bottleneck when the slaves do not have filesystem access to the input files -- i.e. you declare them REMOTE.

Where I/O is the bottleneck, size of the input frames can make a big difference. So YUV input is better than PPM, and JPEG is better than both.

When you're first trying to get parallel mode working, be sure to use the **-debug_machines** option so you can see what's going on. Also, **-debug_sockets** can help you diagnose communication problems.

AUTHORS

Kevin Gong - University of California, Berkeley, keving@cs.berkeley.edu

Ketan Patel - University of California, Berkeley, kpatel@cs.berkeley.edu

Dan Wallach - University of California, Berkeley, dwallach@cs.berkeley.edu

Darryl Brown - University of California, Berkeley, darryl@cs.berkeley.edu

Eugene Hung - University of California, Berkeley, eyhung@cs.berkeley.edu

Steve Smoot - University of California, Berkeley, smoot@cs.berkeley.edu

Table Of Contents