Gjøvik University College, Høgskolen i Gjøvik

imt 4951 - Applied Digital Workflow

Final report

creating time-lapse movies

by using free command-line tools

submitted by: Matthias Böhme, exchange student
e-mail: boehmem@fh-brandenburg.de

Supervisor: Kjell Are Refsvik, M.Sc.

Abstract

In 2011, the city of Gjøvik celebrates its 150th birthday. Related to celebration, one of the suggested project ideas in the course Applied Digital Workflow was a movie with scenes of familiar places in Gjøvik. The different locations might be described by taking pictures in time-lapse[1]
Time-lapse is a technique which is slowing down a process, so that continuous projection of the frames gives the illusion that time is passing quickly.
The outcome is a CLI workflow for generating time-lapse movies, based on open source command-line tools. By formalizing the process of making these movies and using free software, more people are able to deal with it.
To demonstrate the applicability, a demo movie can be found on the related webpage

[1] `http://www.ansatt.hig.no/kjellr/imt4951/index.html#ideas`

Contents

1 Background

Time-lapse is a cinematography technique for very slow changing conditions to create a smooth impression of motion. Processes that would normally appear almost imperceptive to the human eye, become very pronounced. Classic subjects are cloudscapes, plants growing or people in the city. Camera movement during the shots is also possible, for example driving through the city would create an imagination of extreme speed.[1]

There is a lot of software which you can use for creating your time-lapse movies. But mostly you have to pay for and it is far too complex for this specific task anyway. Final Cut[2] for Mac or Adobe Premiere[3] for Windows are great GUI tools for video editing, but they are just too powerful and consequently unchallenged for this project. The biggest problem for the proper software is to automate the processes. In most cases you have to add the images manually. Some of these tools may have also a batch processing mode, but it's often hard to understand and needs time to get used in it. To sum up, there is a lot of work before, until you can compile the movie.

This time-lapse workflow automates as much as possible relevant processes to reduce and simplify the user's activity. It provides an environment where you can create a time-lapse movie step by step, without big effort and much background knowledge in this field. Users have the possibility, to build their own individual movies with open source software.

[1] http://en.wikipedia.org/wiki/Timelapse
[2] http://www.macworld.com/article/141826/2009/07/finalcutpro7.html
[3] http://www.trustedreviews.com/software/review/2009/01/18/Adobe-Premiere-Pro-CS4/p1

2 Goals and Objectives

2.1 Review

The main-goal agreed at the start of this project was to make the creation of time-lapse movies easier and quicker using a formalized CLI approach. That included:

- researching the field and look for similar solutions to improve
- keeping straightforwardness
- Providing fundamental advantages over GUI based software

The specific objectives agreed were:

- make it more efficient as common video software
- set focus on the essential elements (merging, mixing, transforming)
- make simple and short scripts to find and fix program errors faster
- utilize open source command line tools

2.2 Changes

These aims and objectives remained always relevant during the development and it's quite difficult to argue, that some of them were probably not satisfied. It's a matter of argument, if a script with 850 lines of codes is still simple and short enough. Actually the user doesn't need to have a look inside the script. Even advanced users have only set the focus on the first 100 lines, where they can change the constants for a more individual look of the time-lapse movie. It's also hard to say, if this workflow is more efficient as other solutions. On the one hand it might be faster when FFmpeg do its job (merging clips, adding music, encoding videos, etc.), on the other hand it takes more time than usual when ImageMagick is working (resizing images, applying filter, etc.). Eventually it kept the most important conditions - having a CLI workflow, which uses free command line tools and doesn't need specific background knowledge in video editing.

3 Methodology

3.1 Overall Approach

The primary aim was to develop and deliver a workflow which is easy to handle and non commercial through using open source software for processing. Considering that no graphical user interface is implemented, users have to navigate through the workflow via prompt.

I utilized 3 external tools inside my script. Exiftool for metadata, ImageMagick for Image editing and FFmpeg for video processing.

3.2 Detail

Exiftool tasks were rather little. I used it to get the current video size and to fit metadata in the video. ImageMagick was employed for resizing and transforming the images which also contained compositing them (necessary for fader and location stamp). I applied FFmpeg for merging images and clips as well as mixing video with audio. It was also used for encoding to proper video formats.

In the early stages of the project, most of the effort was placed on studying proper literature and sources on the Internet because prior knowledge in shell programming was little. Information gathering also included conceiving the time-lapse functionality and collecting images. Eventually this was rather having resources for the program than becoming a major stage in this project. As opposed to this, code development and testing was most time-consuming and played the main role. Especially validating user input contained big effort. The process for the application development also considered the functionality on different UNIX-like systems[1] as Mac OS X. Obtaining feedback was the last step in development and testing, to apply the final corrections like fixing bugs and improving the usage. User involvement was also essential to ensure error robustness. The last week in the time interval of the project was scheduled for the presentation. It included the final report, website and time-lapse demo movie as well as the release of the

[1] `http://en.wikipedia.org/wiki/Unix_like`

final product - the time-lapse script. Used material was self-created or freely available. Web pages like Flickr or ccMixter offer a lot of material which include an appropriate creative commons license. Therefore my outputs are also contributed under a creative commons license[2].

It should be noted, that the different stages were not necessarily followed in strict order. Especially code development was in progress during the whole period. Compared to the schedule, which was made in the proposal, it was at last not manageable to satisfy the time frame for the particular tasks. But that issue was expected before and is not considered as failure.

[2] http://creativecommons.org/licenses/by-nc-sa/2.0/

4 Implementation

First of all the planning involved breaking down each work package into separate activities to make it easier to estimate the process. Time was short and needed to be well-conceived. The following weeks were focused in collecting images and code development. Information gathering in shell script programming proceeded simultaneously.

4.1 First shitty draft

Before starting and testing the workflow, some footage was needed, so I decided to take pictures over a period of time. Afterwards I was able to create the first time-lapse movies. It needed only a few lines to realize it. Setting path of image sequence, resizing images with ImageMagick to desired size and merging them together to one single movie with FFmpeg. At this point, there was no menu prompt, therefore it was full automatic. Compared with the goals, it still needed a lot of improvement.

Listing 4.1: collecting and resizing images + merging them to one single movie

```
$directory="image_sequence"
# get all jpg-files in current directory
files=$(ls -1 $directory | grep -i '[.]jpe\?g$')
size="640x480"
# resize all images to declared size, copy them to directory "resized_images"
# and rename them to img001.jpg, img002.jpg, etc.
for fname in $files
do
  index=$((index+1))
  convert -sample $size^ -crop $size+0+0 -gravity "center" \
  "${directory}/${fname}" `printf "resized_images/img%03d.jpg" $index`
done
# merge image to one single movie; doc: http://ffmpeg.org/ffmpeg-doc.html#SEC9
ffmpeg -r 30 -f image2 -i "resized_images/img%03d.jpg" \
-r 30 -qscale 2 -s vga -vcodec mpeg2video "timelapse_movie.mpg"
```

1 2

[1] ImageMagick resize operator: `http://www.imagemagick.org/Usage/resize/`
[2] FFmpeg image sequence to movie: `http://ffmpeg.org/faq.html#SEC14`

4.2 User involvement

Users should be able to set the image path individually. Furthermore they might choose between several video formats. There should also be a possibility to merge several timelapse scenes. Before this stage, you could create only one clip and the process was finished. That's why a file structure was implemented so it is possible, to create a new project, or to continue the old one. Continuing means, you can create more clips within the current project and finally merge them to one single movie. Moreover this part included a lot of validation for the different given inputs.

Listing 4.2: create a new project by valid user input

```
# create new project directory by given input after checking
# if name is valid and available
while :
do
  validateName # go to method "validateName"
  path_project="${HOME}/${name}";
  if [ -d "${path_project}/clip001" ]; then
        echo -e " invalid : this project already exists"
  else
        mkdir -p -v "${path_project}" # create project folder
  fi
done
validateName() {
  while :
  do
    read name # waiting for given input
    # strip whitespaces from beginning and end
    name=‘expr "$name" : '[[:space:]]*\(.*\)[[:space:]]*$'‘
    # not more than one whitespace between the words
    name=$(echo ${name} | tr -s " ")
    case $name in
      .*) echo -e " invalid : no points at beginning\n" ;;
      /*|*/|*//*) echo -e " invalid : slashes\n" ;;
      *[^a-zA-Z0-9-_.,/[:space:]]*) \
      echo -e " invalid : name includes special character\n" ;;
      *) break ;;
    esac
  done
}
```

3

[3] Regular expressions: `http://tldp.org/LDP/Bash-Beginners-Guide/html/chap_04.html`

4.3 Features

Actually the workflow could be finished now. You can specify your image path, create your project or continue your old project and choose your video format. The script will resize the images, merge them to a single clip and even merge all clips to one single movie if wished. But I wanted to provide some features for the clip and the final movie. You have to differentiate between clip- and movie additions. Fade-in, fade-out and an individual location stamp can only be employed on clips. Adding metadata (which is performed by Exiftool), background music and credits are just possible for your final movie as well as choosing the final video settings (e.g. Youtube, Iphone, etc.). There is a reason for splitting the features in clip options and movie options. Fader wouldn't be possible between the scenes if you have this means just for your final movie. Location stamp can be added for each time-lapse clip instead only once. On the other side, you have to think oppositely for the movie options. These features make sense when you apply them not more than once.

Listing 4.3: code snippet for one of the movie options (background music)

```
# add background music to your movie by given user input
# http://ffmpeg.org/ffmpeg-doc.html#SEC11
audio_settings="-vol 80 -acodec libmp3lame -ac 2 -ab 160k"
while :
do
  echo -e " file name: \c"
  read music # waiting for user input
  # returns 1 if input is audio file
  check_audio=‘file "$music" | grep -c -i ’Audio’‘
  if [ $check_audio -eq 1 ]; then
        # mixing audio and video together and finish encoding
        # when the shortest input stream ends.
    ffmpeg -shortest -i "${path_project}/movie.mpg" -vcodec copy \
    -i "$music" $audio_settings "${path_project}/movie_sound.mpg"
    rm "${path_project}/movie.mpg"
    mv "${path_project}/movie_sound.mpg" "${path_project}/${movie}"
  else
    echo " couldn’t find the audio file / file wasn’t a valid audio file"
    echo -e " try again? [y,n]: \c"
    read confirm; echo ""; [[ $confirm != "y" ]] && break
  fi
done
```

4

[4] FFmpeg resize operator: `http://www.imagemagick.org/Usage/resize/`

For the advanced user I specified a couple of constants. You can modify the available video settings, fade color, label- and credits settings, etc. . You are also free to change the default ranges for the frame rate, image amount and name length.

Listing 4.4: extract of changeable values for the advanced user

```
# some listed examples for constants,
# which should only be changed by advanced users

min_amount=60; max_amount=600 # min and max image amount
min_name=3; max_name=50 # min and max length of project name

# video settings
VIDEO_SETTINGS_HD1080="-r 30 -qscale 2 -s hd1080 -vcodec mpeg2video"
VIDEO_SETTINGS_DVD="-target pal-dvd"
VIDEO_SETTINGS_MOBILE="-r 30 -qscale 2 -s 480x320 -vcodec mpeg2video"

# only for final movie
VIDEO_SETTINGS_YOUTUBE_HD="-s hd720 -b 2000k -vcodec mpeg4"
VIDEO_SETTINGS_FLV="-ar 22050 -acodec libmp3lame -ab 32K -r 30 -s 320x240
    -vcodec flv"
VIDEO_SETTINGS_XVID="-vcodec libxvid -qscale 3"

# http://www.imagemagick.org/script/color.php
fade_color="#000000"

# label settings
label_maxChar=50 # max length for label
label_textColor="#EEEEEE"
label_backgroundColor="#111111"
label_backgroundOpacity=75 # 100=opaque, 0=transparent
```

Guidelines were implemented for every input to make it as easy as possible to use. To maintain this quality the workflow is divided in several steps which are all well-explained. The program will also check at the beginning, if all necessary tools are installed.

To get a better overview about the different program stages and how it is constructed, see A.1

4.4 Issues

The main problems and issues during the implementation were especially the validation of user input and the development of additional features. When you have no GUI based tool, the user is principally able to type in whatever he wants. But directories can't contain any desired character and empty folders for the image path are not really helpful. You have to ensure, the program will not crash, no matter what the user is entering on the prompt. Error messages for explaining why the input is invalid, are also quite helpful. Adding Metadata was only succesful, when the final video format was an AVI file. I got no error for other file formats, but metadata wasn't visible then.

FFmpeg is powerful - and sometimes hard to understand. Especially if you want to use contained video filters, which was eventually not manageable for me. Tutorials in this field are rare and if you find one, it's obviously not working anymore concerning your updated FFmpeg version. Because of that, I needed to implement some of these features in ImageMagick which is rather slow in video processing[5].

The suggestion, to take pictures by GPS-enabled camera with remote control gadget and tripod was canceled regarding several reasons. These cameras are inventory of the university and can be borrowed by considering the related schedule. That meant a limited flexibility which is quite inconvenient for getting time-lapse footage. Weather conditions change fast, especially in November. Therefore I needed to be more spontaneous in taking pictures and decided to borrow the camera from Martin Olsen, a Master student at Gjøvik University. This camera, a Canon 450D, is not able to take GPS-tagged pictures without any external gadget. However providing a workflow which is not dependent on GPS-data, admits the usage for more users, even if they haven't cameras with this functionality. For remote control I used my laptop and the software DSL Remote Pro[6] as trial.

[5] ImageMagick video handling: `http://www.imagemagick.org/Usage/video/`
[6] `http://www.breezesys.com/DSLRRemotePro/`

5 Outputs and Results

The project output comprise a workflow to enable users to create time lapse clips, modifying and merging them to one single movie which can include background music, credits and metadata. Everything is based on open source command line tools and requires a UNIX-like system. All resources, including shell script, project plan and final report, can be found on my project web page

5.1 Process

The output can be divided in two main parts.

The first part is to create and adapt the clip. Here you need your image sequence. It also determines the video size for all following clips in the current project and provides the different features. This part is actually not dependent on part 2, which means, you already have a time-lapse movie without a need to apply the second part.

Part two is especially for connecting different time-lapse scenes as a whole. That also included background music, credits and metadata as optional features. Besides you can change your video format. Contrary to part 1 it is dependent. There already has to be a project and at least one clip inside to run this part.

The Documentation associated with the workflow can be found in the script itself. The manual is not realized as single document, but integrated in the workflow. By doing this, the user is able to look up the explanations during the process without switching between workflow and manual.

Screenshots from the workflow can be found in B.3

5.2 Finding

Automation in this workflow is interrupted periodically because of needed user input. That means, it is not possible to execute the script, wait for a while and get the final result. That's not how it works and it wasn't the purpose as well. Resizing images, applying effects, merging and adding data - that is still going to happen automatically, but it always needs some sort of confirmation before.

6 Conclusions

6.1 Differences

On the scale of things the end result doesn't much differ to the plan. As stated in section 2.2, it's a matter of opinion if some of the goals and objectives failed. It is still for free (open source command line tools), it works on Mac OS X as well as on Ubuntu (portability), partly it is more efficient than common video software (automation) and the focus has been on the essential elements during the whole development (merging, mixing, transforming).

6.2 Problems

However, this workflow might have one handicap - that's ImageMagick, which is a great tool, as long as you use it for single or just a few images. It's not very suitable, if you need to transform a large amount of images. The GUI based Tool IrfanView[1] is also able to do batch processing. I claim, there is no advantage in speed when I use the command line tool ImageMagick. As already mentioned in section 2.2, Ffmpeg is fortunately able to equalize this disadvantage in performance.
Nevertheless, the time-lapse workflow has succeeded in producing time-lapse movies easily and for free. It's no longer necessary, to have commercial software for this task.

For bigger projects such as cutting videos including transitions or multi-track audio editing I wouldn't recommend it, but for small tasks like time-lapse videos it's advisable, although it might be uncommon to edit your videos on the command line most likely.

6.3 Future Work

The impact of this tool may be small, but potential exists. Users who already tested the script were satisfied. It's easy to handle, and if you background knowledge in programming, also easy to modify.

[1] `http://www.irfanview.de/`

Even though it was a nice experience to work on a command line interface, I would still recommend a GUI for this workflow. I spent much time on validating user input, more than on purpose. That's also one of the main causes why this script is eventually larger than expected. If you have a GUI based workflow, you don't need most of this validation. User input is limited from the beginning. Wrong characters or ignoring the range isn't even possible. Aside from that, most users prefer to browse to the requested directory, which is not possible on a CLI. An ideal solution might be using the GUI for usability advantage and the CLI for automation advantage. My first approach was a sort of shell script GUI. Corresponding utilities like dialog[2] and zenity[3] allows you to create menus or lists.

Additional work in the future could be the implementation of further features (e.g. location stamp based on GPS data, sort of connection to Google Maps, etc.). I would also suggest, to find a better alternative than ImageMagick, to apply the appropriate effects. FFmpeg is obviously able to adopt this part[4]. Unfortunately the documentation for it is missing or hard to follow.

[2] `http://linuxgazette.net/101/sunil.html`
[3] `http://bashcurescancer.com/man/cmd/zenity`
[4] `http://article.gmane.org/gmane.comp.video.ffmpeg.devel/85981`

7 Recommendations

Before getting started you need to make some arrangements. There are a few things which you should consider.

Equipment:
A camera with timelapse functionality is essential. You have to ensure, your camera can be programmed to automatically shoot a number of frames over a period of time or with a certain time interval between each frame. Some cameras feature a built-in time lapse mode. Others allow you to set up time lapse as part of a Remote Capture application. This requires the camera to be connected to a computer[1]. A tripod is also advisable

Camera settings:
Use applicable settings for your timelapse shooting. That includes proper shutter speed, exposure time, etc. hypertargettutTutorials for time-lapse shooting exist in great quantities.

Software:
You need an UNIX-like system or at least Cygwin[2] for Windows. Required tools for this workflow are Ffmpeg, Exiftool and ImageMagick. You can install the tools manually or just start the script install_tl_tools.sh (recommended).

This tool can't perform miracles. If you have poor footage, you can't expect a nice looking time-lapse movie. It took me a while until i figured out, how to make suitable time-lapse scenes. Therefore you also need to be able to have patience. The problem isn't the creation of the movie, it's more the interaction between camera settings, weather conditions and your own talent in photographing.

[1] `http://www.dpreview.com/learn/?/Glossary/Exposure/Time_Lapse_01.htm`
[2] http://www.cygwin.com/

8 References

Project output (script, demo movie, material, reports):
`http://www.stud.hig.no/~091328/timelapse/`

External tools

- Exiftool: `http://owl.phy.queensu.ca/~phil/exiftool/exiftool_pod.html`
- ImageMagick: `http://www.imagemagick.org/script/command-line-tools.php`
- FFmpeg: `http://ffmpeg.org/ffmpeg-doc.html`

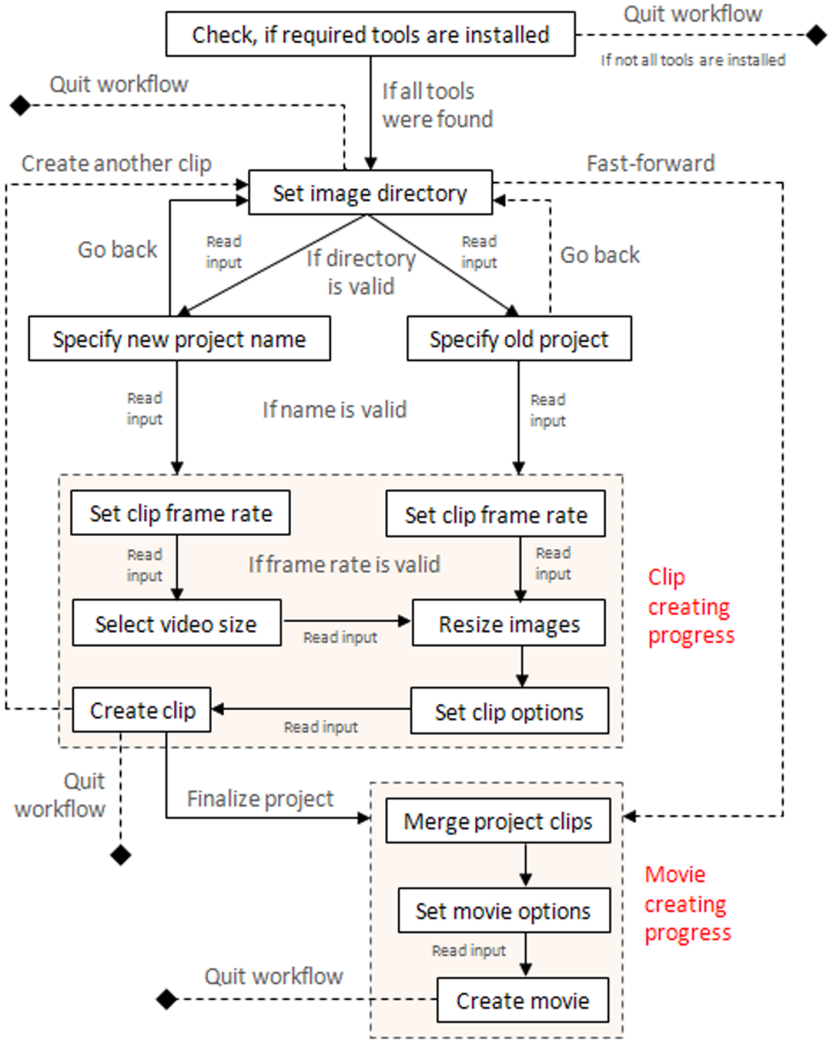
Time-lapse tutorial:
`http://digitalartwork.net/2007/01/30/time-lapse-tutorial/`

Material under creative commons license

- Images: `http://www.flickr.com/creativecommons/`
- Sound: `http://ccmixter.org/`

A Workflow structure

Fig. A.1: Process in time-lapse workflow



B Workflow screenshots

Fig. B.1: Step 1: path of image sequence

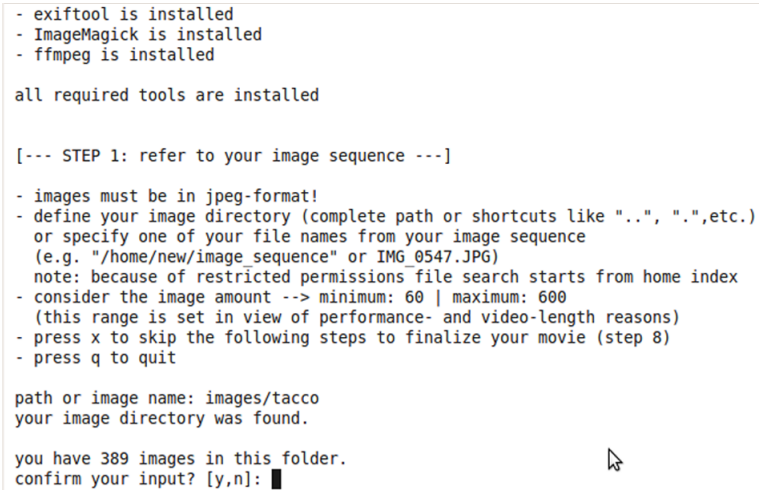


Fig. B.2: Step 3: create new project

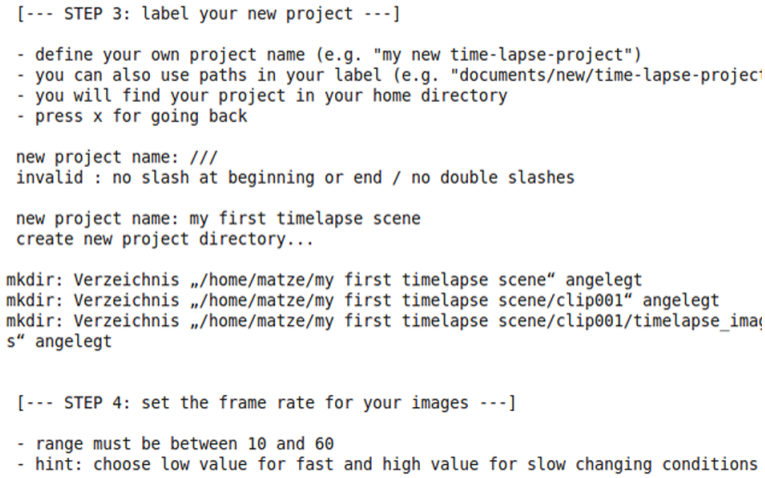


Fig. B.3: Step 7: clip options

```
resizing 97 images to selected video format
---------------------------------------
0%                                  100%
 |=================================|

resizing progress took 32 seconds.


[--- STEP 7: choose final options for your clip ---]

- you can add fader and individual location stamp (suited text) to your clip
- fader are 1 sec long and optional as well as location stamp
- if you don't want to change anything, just press return during the prompt

fade-in  effect [y,n]: y
fade-out effect [y,n]: y
location stamp: location: Gjovik
invalid : name includes special character

location stamp: location - Gjovik
---------------------------------------
0%                                  100%
 |======================
```

Fig. B.4: Step 9: final video format

```
104857 kb/s, 90k tbn, 25 tbc
    Stream #0.1: Audio: libmp3lame, 44100 Hz, stereo, s16, 160 kb/s
Stream mapping:
  Stream #0.0 -> #0.0
  Stream #0.1 -> #0.1
Press [q] to stop encoding
frame=  366 fps=  0 q=-1.0 Lsize=    8125kB time=14.60 bitrate=4559.2kbits/s
video:7804kB audio:289kB global headers:0kB muxing overhead 0.399826%

 ----------------------------------------------------
 metadata is added now.


 to complete your project you can choose between common video formats
 ---------------------------------------------------------------------
  1. Youtube HD  (1280x720, 16:9, MP4)
  2. IPHONE      (480x320, 3:2, MP4)
  3. FLV         (320x240, 4:3, FLV)
  4. Xvid        (no size changing, AVI)
  5. H264        (no size changing, MP4)
  6. Quicktime   (no size changing, MOV)
  x  maintain current video format
 ---------------------------------------------------------------------
 choice: 2
```