slomoVideo: Slow-motion for video with Optical Flow

Simon A. Eugster

Handed in at *ETH Zürich*
Supervisor: Marc Pollefeys

August 12, 2011

1 Introduction

1.1 How it started

Sometimes ideas evolve over a long time – *slowmoVideo* is one of them. It started with a game that caused me to record and edit video tutorials, and editing caused me to join development of an open-source video editing program.

The game was the first *Assassin's Creed*. The player lives to see the life of an assassin (which try to protect humanity from templars) in medieval times. A big part of this game was fighting with swords, daggers, and throwing knives, as expected from its name; an equally big part however was climbing. Over walls, on rooftops of Damascus' clay houses, up viewpoints in Jerusalem, and even to the top of a huge church in Akka. The player was not limited to the street, but could visit any place in the whole town, no matter how well-guarded, there always was a path to sneak in or to climb up.

I then started working on video tutorials showing basics and tricks about climbing. For this I needed a video editing program. After a while I found *Kdenlive* which is open source, running on Linux, and not abandoned by its developers as some other programs were. Yet it used to crash often and had some limitations, namely in the titler. Since I was looking for a way to force myself to learn C++ anyway, I decided to try working on the titler and report some bugs in Kdenlive. This was 2.5 years ago. Now I still am programming on Kdenlive, together with other good guys, so it seems that joining development was a good decision.

While working on the *Assassin's Creed* videos I once came across a video on youtube which used a speed effect to speed up parts of the video to make fighting scenes look more fluid. Not in an obvious way like playing $2\times$ speed, but much more subtle: Only parts where Altaïr, the main character of the game, seemed to be waiting for the opponent, were accelereated, unnoticeable except for players who knew the fighting rhythm. It was fascinating because the video looked much more dynamic. This is when I wrote a feature request in Kdenlive's bug tracker for a speed effect defined by a curve and not a fixed factor. Later I added more ideas and thoughts to this feature request.

Then, slightly more than half a year ago during *Visual Computing* lectures, Prof Marc Pollefeys presented us a short video clip that was slown down by a factor of 10 and even more – and it still looked smooth due the use of *Optical Flow* which allowed the computation of intermediate frames. This was absolutely fascinating for me. I went searching for an Open-Source program[1] that would do this on Linux, or if necessary on Windows, as I did two years ago with Kdenlive, and found – nothing.

[1] Therer are multiple commercial closed-source programs on the market, sometimes only available as plug-in for expensive video editing programs and not as stand-alone software.

I still needed a bachelor thesis ...

1.2 Goals

After thinking about the whole topic, repeatedly in fact, I came to the conclusion that I wanted to have a program,

- running on Linux,
- available as Open Source,
- capable of making slow-motion videos by the aid of optical flow,
- and in fact not using a constant speed factor, like $0.1\times$ or $2\times$, but a curve.

Even if I had to write it myself. Fortunately Christopher Zach had already written an Optical Flow library[2] which I could use for my project, and ffmpeg[3] would do the video decoding and encoding. What I had to do was therefore the application that should fulfil the above requirements.

1.3 Optical flow

1.3.1 What is optical flow?

Optical flow is what makes it possible to calculate a new frame between two consecutive frames. It tells for each pixel in the first frame to which position in the second frame it *appears to* have moved to. Using these movement vectors, intermediate frames can be interpolated, as in figure 1.1.

Figure 1.1: Interpolating frames with a movement vector field (optical flow)

Optical flow does not necessarily follow reality. Figure 1.2 shows two examples with one interpolated frame in the middle; neither of them is incorrect in terms of optical flow: The pixel's colour at source and target position *matches*. (The second sequence leaves a hole in the tree; Holes are often seen when interpolating with optical flow and are discussed later in chapter 3.2.)

[2] Link: `http://www.inf.ethz.ch/personal/chzach/opensource.html`
[3] ffmpeg: `http://ffmpeg.org/`

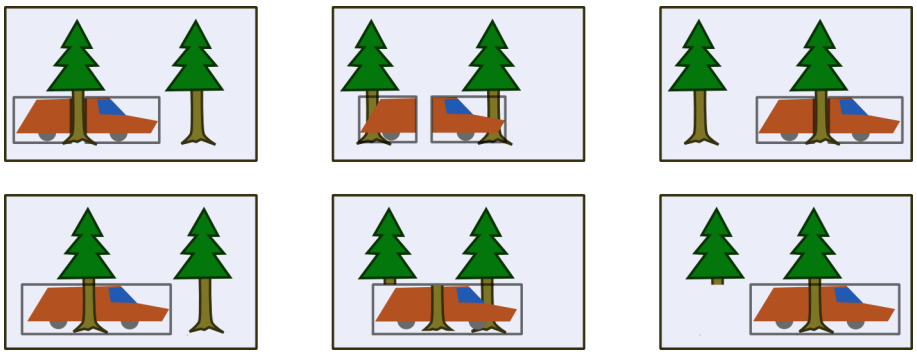


Figure 1.2: Which option is correct here?

For a more in-depth description please refer to existing literature like *Optical Flow Estimation*[4].

1.3.2 Other fields of application

Optical flow is also used in (randomly selected):

Video encoding Current codecs like H.264 use motion estimation for pixel blocks (of variable size from 4×4 to 16×16) to predict frames. The amount of data required for correcting the errors that occur during interpolation is relatively small, comparing to the amount of data that would be required without motion prediction. There even is a program called yuvmotionfps[5] that uses the MPEG motion field for slow-motion interpolation.

Camera interpolation Already back in 1999 the first Matrix video used optical flow, for the shooting scenes.[6] Since the smooth camera dolly shot had to be accomplished using multiple *static* cameras (each shooting at 600 fps), jumps between the single cameras would have been necessary, which denies slow movement of the camera(s). The missing positions between two cameras lined up next to each other were interpolated using optical flow.

Robots use optical flow data for detecting obstacles in their way.[7]

[4] Optical Flow Estimation by David J. Fleet, Yair Weiss: `http://www.cs.toronto.edu/~fleet/research/Papers/flowChapter05.pdf`
[5] yuvmotionfps: `http://jcornet.free.fr/linux/yuvmotionfps.html`
[6] Art of Optical Flow (1999, The Matrix): `http://www.fxguide.com/featured/art_of_optical_flow/`
[7] Numerous papers can be found about this topic when searching in the web.

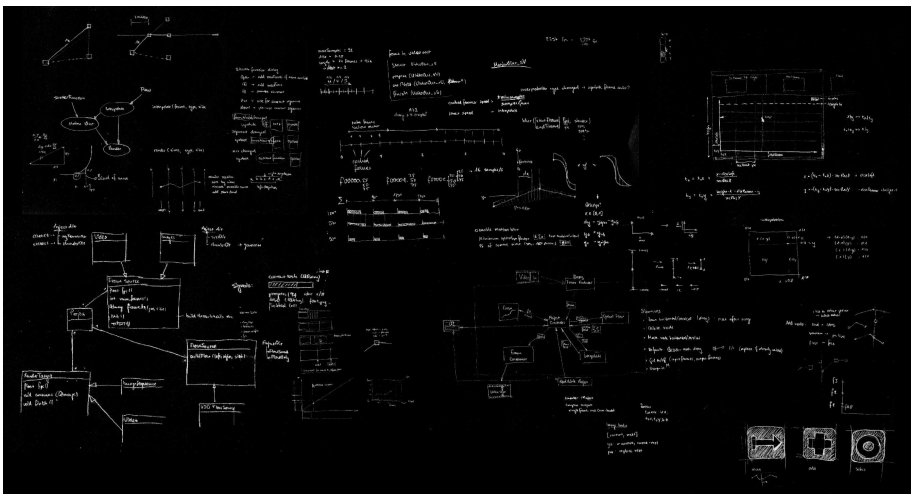
Face validation to distinguish between real 3-dimensional faces and faces printed on paper.[8] There actually is a funny story about this: 3 years ago, in 2008, Japan built new cigarette vending machines which verified the age of the buyer by examining his or her face, and would only sell cigarettes if the age was estimated 20+. Unfortunately those machines did not mind if the face used for age verification was printed on the banknote it was later fed with, and the elder men on the banknotes were always permitted to buy cigarettes.[9]

Kinect Microsoft's Kinect actually does not use optical flow but an extended, 3-dimensional, version of it (and therefore uses 2 cameras for a stereo image).

[8] Biometrie: Tot oder lebendig? `http://www.heise.de/tr/artikel/Biometrie-Tot-oder-lebendig-280153.html`

[9] Japan: Gesichtserkennung mit Portraits auf Geldscheinen ausgetrickst `http://www.heise.de/newsticker/meldung/Japan-Gesichtserkennung-mit-Portraits-auf-Geldscheinen-ausgetrickst-185511.html`

2 Programming



2.1 Program structure

slowmoVideo is separated into multiple sub-projects (better visible in the class diagram, figure 2.1): The *user interface* (slowmo UI), which uses the *project libraries* which are responsible for reading and writing project files, rendering a project, etc., and some *basic libraries* which contain interpolation code and a class for reading and writing flow files. An additional small project, *visualizeFlow*, creates "human-readable" images from optical flow files. Projects can be rendered either via the user interface or with *slowmoRenderer* which is a separate command-line application only made for rendering (e.g. with `./slowmoRenderer myProject.sVproj -target video rendered.avi libxvid -fps 24`). *slowmoFlowEditor* is a small program for correcting optical flow files.

The optical flow itself is generated with Christopher Zach's *GPU-KLT+FLOW* library (resulted in flowBuilder) which is a stand-alone project and therefore a separate executable that is called by slowmoVideo. It also uses the basic libraries to write the optical flow to a file. flowBuilder is a stand-alone project to avoid having the dependencies (glut, glew, OpenCV, sdl, Cg and more) in slowmoVideo as well, also because flowBuilder only works

for nVidia cards but slowmoVideo could be extended to use other optical flow libraries as well.

slowmoVideo uses abstractions for flow sources, frame sources, and render targets. A flow source builds the optical flow between two frames and returns it. Frame sources and render targets are input and output. A project's frame source has a certain number of frames and can deliver these frames (which are then, for example, used for calculating the optical flow); Supported are image sequences and videos. Images are read with the Qt libraries directly, videos with ffmpeg. Render targets are the counterpart, the name self-explaining; supported are again images and videos[1].

2.2 Software quality

Comfortable building also is a criteria of quality, at least from a programmer's perspective. slowmoVideo and flowBuilder use CMake[2] for building.

Another quality measurement for coding is documentation. Doxygen[3] will put source code comments together in a web page which is a handy quick reference. I paid attention on describing passed parameters (e.g. if a pointer is allowed to be NULL), possible side effects, and what the function exactly does and where it is used.

To ensure that I do not break important parts of the code (like the library for reading and writing optical flow files) and to spot regressions I use unit tests. To find bugs in other (less central) parts of the software I often use assertions: If compiled in debug mode, the program simply exits when an assertion is false. This is something that for sure does not get lost in the debug output as it is a more obvious message, and it forces me to fix a bug (or implement a yet unimplemented feature). I found this technique very useful for keeping the bug count low in the code. Exceptional cases that were not caused by a wrong function call or similar but by a user error or exceptions of the kind *file not found* were replaced by thrown exceptions which could then for example notify the user.

2.3 Features

2.3.1 Main features

Slow motion or, more general, *time manipulation* is certainly the main feature of slowmoVideo. Screenshot 2.3.1 shows how time is manipulated – with the curve ($c : T_{\text{out}} \rightarrow T_{\text{source}}$). Input frames are on the left (frame 247 at mouse position), the current frame is shown in the input monitor, and the frame that will be rendered at mouse position is shown

[1] Video encoding was actually quite a hard part.

[2] CMake, see `http://www.cmake.org/`, is a tool that helps compiling source code. It searches for available libraries (and shows an error message if one is missing) and creates a makefile for comiling and installing the software.

[3] Doxygen (see `http://www.stack.nl/~dimitri/doxygen/`) extracts documentation from various programming languages and can build, for example, web pages from it.

Figure 2.1: Class diagram of slowmoVideo

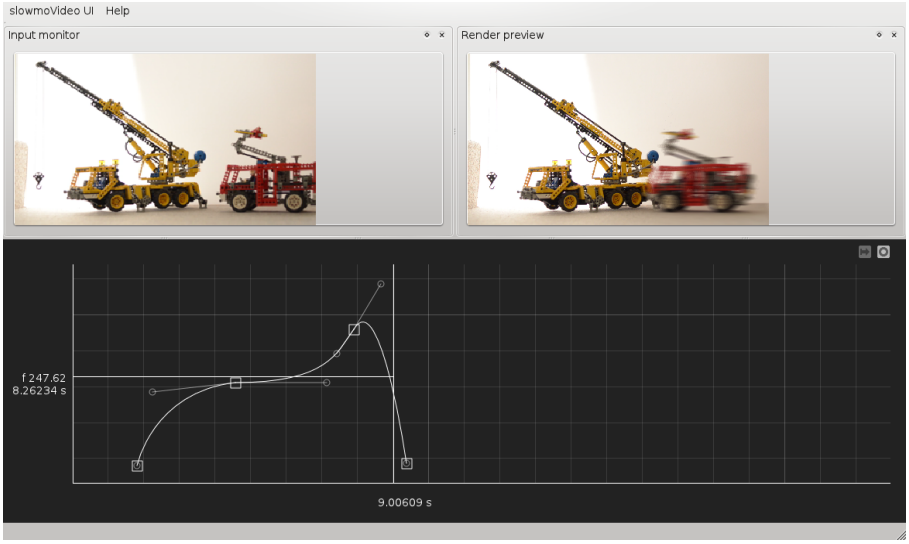


Figure 2.2: slowmoUI with a curve, input monitor and render preview.

in the Curve monitor (not in this screenshot). Output frames are on the bottom. The curve assigns an input frame to an output frame. Horizontal parts in the curve mean that the time does not change, lines with low inclination indicate slow motion, 45 ° lines replay the video in real-time, and steeper lines speed the video up. Curves can be drawn by using linear or Bézier segments. The range of the Bézier handles is limited since only one input frame is allowed for every output frame.

Motion blur is the counterpart of slow motion. If a video is re-played at a higher speed factor (e.g. 2×) it does not "feel right" to the viewer – instinctively. This is also related to the shutter speed: Cinema often uses a *180° shutter* which is a term from analog film and essentially means that the film was exposed half of the recording time. If it was shot at 24 frames per second (which is a traditional frame rate for cinema), each frame would be exposed for 1/48 s. When replaying this video with a speedup factor of 2 by simply dropping each second frame, the exposure time is halved (1/96 s). slowmoVideo allows to define a custom shutter function on each segment, as seen in screenshot 2.3.1. In this timespan additional frames are interpolated, if necessary, until a certain (user-defined) number of frames are reached, which then are added to simulate a longer shutter. (This can also be taken to extreme values, for example to create the impression of very high speed.)

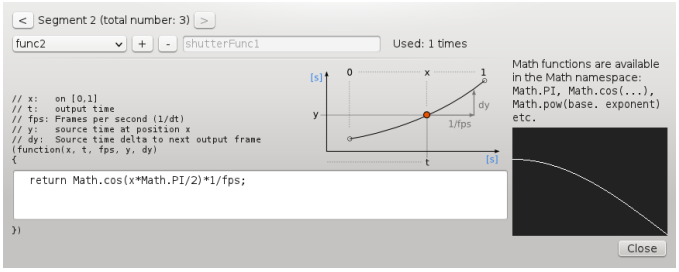


Figure 2.3: Custom shutter functions

2.3.2 Selected features

In this section I highlight some additional features of slowmoVideo and buildFlow. On the one hand for allowing the reader to gain a better over-all picture of slowmoVideo (and the work behind :)), on the other hand for pointing out details which I think are important.

buildFlow uses several shader files. Originally the environment variable `V3D_SHADER_DIR` had to be set, which is not ideal for distributing a binary package. I modified the shader files such that they are included directly in the binary file (using the C preprocessing directive `#`) unless CMake is given the flag `-DDISABLE_INCLUDE_SOURCES`.

slowmoVideo uses project files in XML format for saving and loading projects. The project file includes the nodes that were defined, the input file(s), shutter functions, but also settings[4] like previously chosen directories, rendering settings and the current viewport. It also includes a version number which allows slowmoVideo to update old project files or notify the user if they attempt to open a project file that has been created with a newer version of slowmoVideo.

slowmoUI uses different kinds of shortcuts: The usual ones with modifiers (`Ctrl+O`), and alternatively timed ones without modifiers (`o`, `d-n`); For the combined shortcuts there is a time limit within which the second key has to be pressed, otherwise it times out.

The nodes in the canvas can be moved inside the bounds given by the length of the video (vertically or horizontally only if the `Ctrl` key is pressed); a second tool moves all nodes right to the cursor. The curve type (linear or Bézier) and the shutter function can be changed by right-clicking on a segment. Nodes can be deleted by right-clicking or by selecting them and using the shortcut `d-n`. When right-clicking, a list of objects below (or near to) the cursor position is created and used for determining which options to show in the context menu. The same happens when dragging a node or a handle. Nodes have priority over handles, so if a handle is at the same position as a node, only the node can be moved – this because

[4] I personnally cannot stand it if an application constantly forgets my choices. Therefore every setting will eventually be remembered – maybe not directly after implementation, but at the latest when it has become clear that the setting will stay.

moving nodes is a more basic action than moving a handle. The Shift key disables nodes in the list mentioned before, they are therefore ignored and handles can be dragged. – One word about selection: In the first versions of the slowmo UI I used exactly the node's area for selecting; when clicked aside, nothing got dragged. The current version adds a quite large padding (currently a circle of 24 pixels diameter), which I find much more[5] comfortable to handle.

An additional element that is available for the canvas is the *tag* which marks a position in the source (for user reference) or in the output. Tags in the latter can be used when rendering only parts of the whole project.

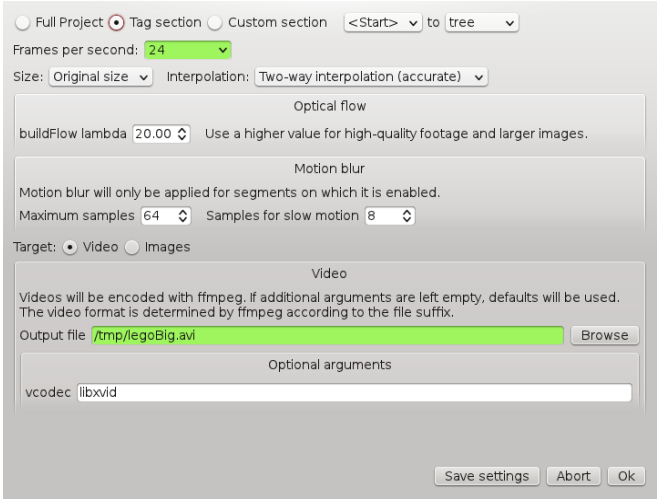


Figure 2.4: Rendering dialog

Flow fields are cached (saving via the FlowRW class, see the class diagram, figure 2.1) and, secondly, only built when requested (i.e. not generally for the whole frame source). slowmoVideo renders to image sequences and to videos. Images are maybe the preferred way for larger projects since it is possible to interrupt rendering and continue it later on (using tags or directly in/out time; both will snap to a frame time, so there will always be exactly 1/fps s between two consecutive images). Images can afterwards be combined using ffmpeg and a command like `ffmpeg -i renderedFrame-%05d.png -vocdec libxvid -vb 5M renderedVideo.avi`. Videos are ideal for short tests and for users which prefer not to work on the command line; slowmoVideo is using the ffmpeg libraries (libavformat, libavcodec, libswscale) directly.

[5] Really! A huge lot more. Just a hint, should the reader ever have the pleasure of also writing an application where objects can be selected.

slowmoFlowEditor switches to the next or previous flow file with with `Ctrl+Left/Right` (or, alternatively, `j/k`). If necessary, it skips several files (e.g. jumps from `forward--lambda20.00_14453-14454.sVflow` to `forward-lambda20.00_14492-14493.sVflow`). The visualized flow image can be amplified, colours are then boosted (visible in figure 3.5) and errors are easily visible.

3 Issues

This chapter discusses two kinds of issues, which belong to the category optical flow or frame interpolation.

3.1 Optical flow issues

The issues listed here refer to the GPU-KLT+FLOW I'm using, but may also occur in other optical flow libraries. The paper *Secrets of optical flow estimation and their principles*[1] describes several problems and how they are solved with current methods.

Lighting changes One problem are *lighting changes*. This can be seen on shadows, but also in light beams, as figure 3.1 shows. The light beam moves from right to left, and so does the optical flow as it assumes the brightness to stay constant – but the constant brightness moves with the light beam. The books on the shelves get distorted. The optical flow can be improved in such cases using texture-based approaches: The image's texture usually does not change a lot if only lighting changes.

Figure 3.1: Moving light beam. Interpolated frame on the left, optical flow on the right. The books are distorted near the beam's center.

Disocclusion A different issue is *layering* that includes (dis-)occlusion. A person walking in front of a wall will constantly occlude or disocclude parts of the wall since they are in a layer closer to the viewer. Proper handling of boundaries between foreground and background

[1] *Secrets of optical flow estimation and their principles* by Sun, Roth, Black can be downloaded from: `http://www.cs.brown.edu/~black/code.html`

can be tackled with image segmentation by separating the image into patches on layers with different depth, as shown in *Consistent Segmentation for Optical Flow Estimation*[2]. Depth ordering is also examined in *Layered Image Motion with Explicit Occlusions, Temporal Consistency, and Depth Ordering* from 2010;[3]; their Layer++ algorithm still holds top ranks in the Middlebury ranking[4].

Transparency Also related to layers are *transparent objects*. They add additional difficulties, as shown in figure 3.2. The colour filters block part of the colour spectrum. Similar on glass where the objects behind are visible, but the reflections as well. Two layers move in different directions, but additionally an understanding of colour filters (the filter's layer blocks or attenuates part of the spectrum, i.e. performs a multiplication on the background) or of reflections (in this case the two layers are added on top of each other; but how to segment reflection and background?).

Figure 3.2: Colour filters (transparent layers)

Blurring When shooting at low shutter speed like 1/30 s for 24p footage, moving objects get blurred (a typical example are legs of people as seen in figure 3.3). At the blurred borders the background as well as the foreground is visible (this is therefore related to the

[2] *Consistent Segmentation for Optical Flow Estimation* by Zitnick, Jojic, Kang. Paper and example videos are available on this web page: `http://research.microsoft.com/en-us/um/people/larryz/opticalflow.htm`

[3] Download link: `http://www.cs.brown.edu/~black/Papers/nips10paperplus.pdf`

[4] Middlebury ranking: `http://vision.middlebury.edu/flow/eval/`

transparency issue). Zitnick et al. use alpha blending to solve this. Even better would be to shoot at a faster shutter speed to avoid blurring altogether and add motion blur in post-production, which slowmoVideo is capable of too.

Figure 3.3: Blurred legs at a shutter speed of 1/30 s

Large movements Objects that cover a large distance between two consecutive frames cannot be tracked. Usually this issue is answered by using image pyramids and starting with a low resolution image; the coarse optical flow received is then refined in the following iterations with images of higher resolution. However if a structure is too small to show up in the low-resolution image its movement will not be detected. Figure 3.4 shows such an example; the chain is completely pulverized. So is the medallion, possibly due to the noisy background, since V3D does use image pyramids. A shooting workaround for this is to use higher shutter speeds; The frames in this figure have been shot at 24 fps, most camera can easily shoot at twice this frame rate.

3.1.1 Optical flow workarounds

For still being able to produce high-quality optical flow it makes sense to simply avoid problematic situations when shooting – which is done for other effects as well –, as also suggested on fxguide.com:

> *Optical flow code is written with some assumptions or rules – when your shot sticks closer to the rules – you get a better result. This is natural, – we all understand green or blue screen keying these days – which in turns means we*

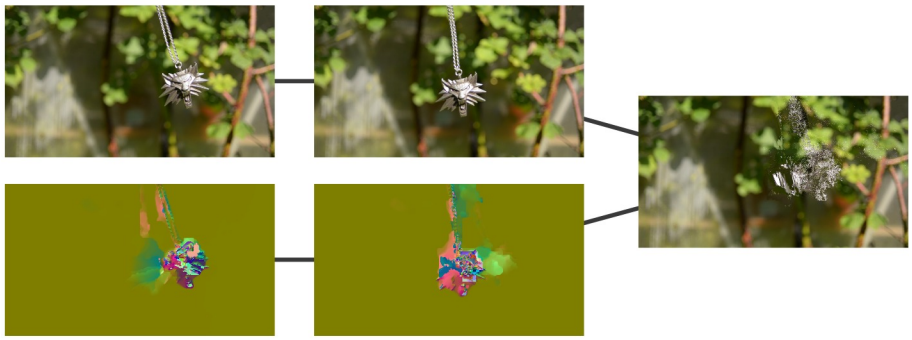


Figure 3.4: Too large movement, also combined with the chain's small structure, is not detected.

> *all have a bunch of "rules" for shooting green screen material ... such as don't wear a green shirt in front the green screen, or don't use promist filters or avoid having a green screen too close to the talent ... You can choose to violate these rules anytime you like, but you may create more work for yourself and achieving great results may be much much harder to obtain.*
>
> from: `http://www.fxguide.com/featured/art_of_optical_flow/`

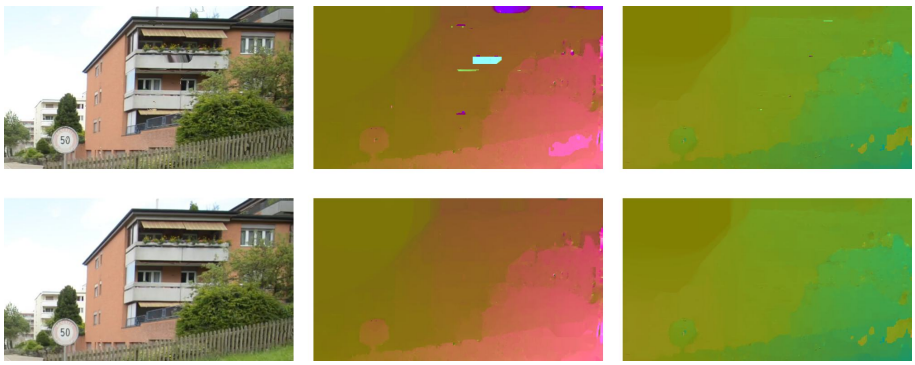


Figure 3.5: Top: Uncorrected optical flow and interpolated image. Bottom: Corrected versions; the balcony is not torn apart anymore.

Another option is manual fixing, either before or after generating the optical flow. Former has been done e.g. for panorama stitching software where the user had to insert point pairs

on identical objects in the left and right image, and I believe that optical flow in critical scenes would benefit from user-based hints since we still understand movement better than the computer. I have not found any description of such an approach though.

Correcting optical flow files after generating them is the way I have chosen for getting rid of outliers, i.e. sections in the image which move around although they should not. The user can draw a rectangle about such areas in *slowmoFlowEditor*, it is then cleared and filled with the data from the surrounding pixels. I have successfully used this for one of my videos.

3.2 Interpolation issues

Further difficulties arise when interpolating an intermediate frame from the source frames and the optical flow. One that is easy to understand is objects that do not move on a straight path between two frames. The knot mounted on the wheel in figure 3.6 would then not rotate but only move up and down. This extreme example cannot be solved, but if more information is available – like four images per revolution –, then an image sequence of 3 or more images can be used to smooth the path a pixel takes using splines, for example. I tried to use Bézier splines in slomoVideo.[5]

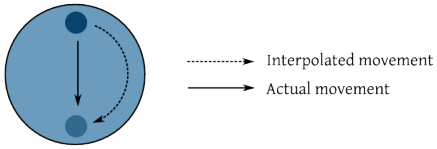
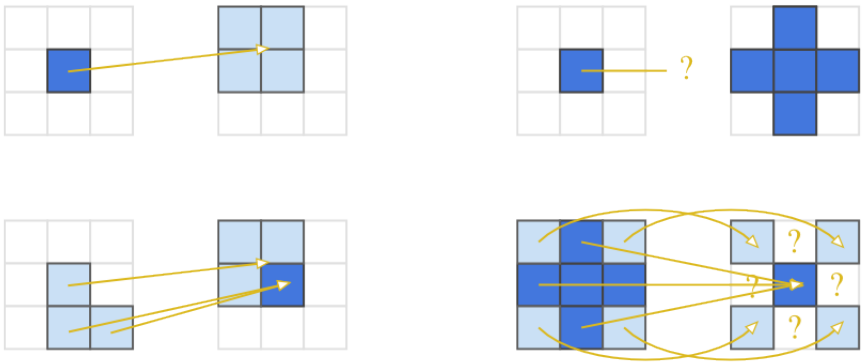


Figure 3.6: Interpolation of curves

A slightly more complicated problem is that pixels (usually) do not move by an integer value, so interpolation is necessary. The top left graphics in figure 3.2 shows a pixel that moves 0.5 pixels top left and gets distributed over all adjacent ones. The colour becomes lighter, otherwise the size of the dot would change. If two pixels did the same thing, the colour between each other would be added up and be the original dark blue again. But what about the bottom left one? Light blue cannot become dark blue simply because two pixels moved to the same location. The solution I have taken to work around this problem is to revert the arrows and translate them such that they start at the center of an output pixel. It would then end maybe between some pixels in the source image, but this is easy to interpolate!

The right column in figure 3.2 shows two more problems. The upper image is a general problem with the representation of flow fields: A pixel can only move to one single location, yet if an object enlarges, this cannot be expressed anymore. Same if an object shrinks. The result is a hole in the output image. These holes can be filled with inpainting algorithms, for

[5] Bézier spline interpolation still needs some work in order to be ready for production.



example by mixing the colours of the neighbours, or by using more sophisticated inpainting algorithms.

In slowmoVideo I do not use inpainting on the image but already in the *source field* which is a product of the optical flow field and the current position between two frames. The source field contains the reverted vectors, as described above, for the current position (e.g. position 0.73 between frames 0 and 1); it tells for each pixel where it *came from* – which is the opposite of the optical flow field which tells where pixels *went to*. Due to the reasons discussed before, and also if the scene moves out of or into the frame, holes will be created. The inpainting algorithm then fills those holes by averaging the neighbours' vectors. Border issues are additionally solved by using forward and backward flow; with one of them the source pixel lies outside of the frame: In this case it is ignored and the pixel from the flow towards the other direction is used instead.

4 Final words

The goals are achieved – slowmoVideo runs on Linux (and can easily be ported to OSX or Windows), and not only linear curves, but also Bézier curves can be drawn for describing the rendering speed. I could also extend the program and add motion blur which can be defined with functions that can use variables like the current output time, the relative time on the segment, and the source time delta to the next frame that is going to be rendered. The idea of using a function also evolved when I saw the motion blur video on Zitnick's page (2), to hand on the credits.

slowmoVideo is extensible, support for other optical flow libraries (which do not depend on nVidia GPUs, for example) and for 9+ bits per image channel (like 32-bit float) can be added easily.

While I was working on slowmoVideo some small projects arose as well:

- A *web page* for slowmoVideo[1] with a short introduction. I use wiki2xhtml[2] which I also had to extend at certain parts (like improved templating support known from MediaWiki).
- *Example videos* on vimeo (linked on the web page). For one video I used blender to create a kinetic typography animation; this was the opportunity to finally learn it.
- The public *git repository* is currently at `git://granjow.net/slowmoVideo.git`; read-only access is provided by `git-daemon`.
- A cronjob generates the *Doygen documentation* of slowmoVideo at `http://slowmovideo.granjow.net/docs/`. (This also made me learn about cron jobs.)
- *Support* – I use Google+ as platform. slowmoVideo is small enough to not require a bug tracker, and Google+ already provides a service for communicating, therefore maintenance of yet another system falls away.

It is time to say thank you to: The patient #qt IRC channel on freenode (especially thiago, cbreak, and wongk) which helped me solve several Qt problems, cellstorm (founder of the openArtist distribution) and Pander for their testing and suggestions, and Christopher Zach for releasing his library under LGPL and giving tips.

—Simon (Granjow), 2011

[1] slowmoVideo's web page: `http://slowmovideo.granjow.net`
[2] `http://wiki2xhtml.sf.net`, an earlier project of mine