Adaptive Slow Motion For Videos
Semester project

Loïc Barben Boris Neubert Mark Pauly
Computer Graphics and Geometry Laboratory, I&C, EPFL, January 2014

Abstract

This paper describes an approach on how to quantify the motion in the videos and on how to represent and remap the time in videos based on the motion information to preserve the dynamism. A few attempts have been done using different tools to finally come with a custom solution. A modified optical flow between each video frames is computed first, then the displacement vectors inside each optical flow are clustered using an adaptive clustering. The resulting average of the cluster's average of their values gives a hint on the dynamism of the scene between two consecutive frames. The video is then remapped according to this information to preserve the dynamism. A video player has been developed for the purpose of the research with customizable options.

Keywords: optical flow, adaptive, slow motion, spline

1 Introduction

1.1 Background

Slow motion in videos is mainly used to show subtle movements that might go unnoticed while playing at normal speed. However, this has the effect that a viewer looses the feeling for the dynamics in a scene, e.g. while looking at a slow motion replay of a football foul you can clearly see if there was a real impact, but it is hard to tell how strong it was. Another effect is that impressive footage of physical stunts (e.g. snow board or parkour tricks) might become less attractive to watch due to the reduced perceived dynamics. Physical properties such as speed, momentum, force, or acceleration are time dependent; remapping time might change the perceived magnitude of these properties.

The main challenge is to correctly quantify the motion in the videos (avoiding the background motion), that would allow to draw video's profiles. The videos could then be played at different speed according to their profile. Even if the remapping is correct, people would not like the result due to difficulties to understand where is really the motion, which depends strongly on the camera position. Thus, it is important to give the possibility to modify the profile before reencoding the video for example.

1.2 Preview

The section 2 presents the different approaches investigated to compute the motion in videos. The section 3 describes the implementation of the chosen approach and how the motion is shown to the user. Section 4 discusses the results obtained with that implementation, and finally the section 5 introduces possible improvements. The paper is concluded by the Acknowledgments and References.

2 Approaches

Several approaches have been considered for quantifying the motion in the videos. The first approach was to track feature points over the frames, along with their speeds. The second approach is about segmenting objects in each frame, and using the optical flow to find their speed. The last approach uses an adaptive clustering to directly quantify the motion. The following subsections describe why these approaches have been taken and what has been tried and implemented.

2.1 Tracking points

The idea behind tracking points over the frames is to select groups of points that are likely to be part of the same objects. Then, it is possible to determine the motion (in pixels/s) of these groups by dividing the number of pixels inside them by the time elapsed between two frames. From this information it is possible to get an idea of the dynamism of the video by drawing a profile for the video. To track the points, a framework developed at MIT as been used, which is Particle Video [Sand and Teller 2006].

2.1.1 Particle Video

Particle Video [Sand and Teller 2006] uses a reduced set of tracked points that are likely to be part of the same object. They distribute the points to track according to a gray-scaled image with similar color regions. The regions have fewer points in their center than near its border. Then the points are linked together using a constrained Delaunay triangulation. A link has a higher weight if the two particles have the same trajectory, meaning they are likely to be part of the same surface (hence, it also handles occlusions by giving a weight near zero).

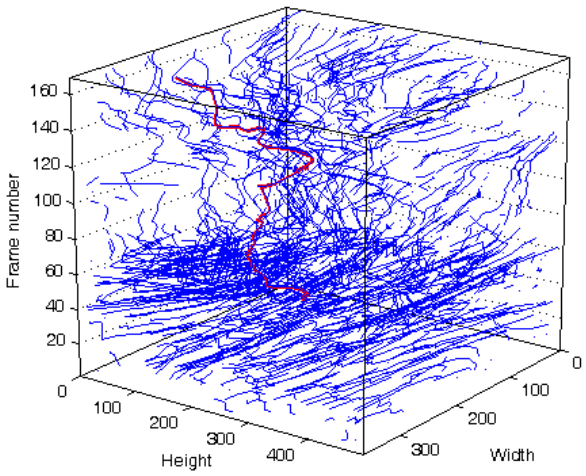

Figure 1: *Set of 1000 curves in a video with 169 frames. One trajectory has been colored in red.*

The input is a video and the output is a file that reports the tracked points in the format $[start\_frame][end\_frame][x, y][x, y]...$, where $[x, y]$ are the width and height image space coordinates.

The output of Particle Video is visualized as a set of curves, stretched out over a period of time (figure 1). The curves are then

grouped together based on their similarity. The groups of curves represent ideally objects with the same trajectory and speed. These groups define a profile for the video. The clustering part is achieved with a tool for Matlab called CCToolbox [Gaffney 2004].

2.1.2 CCToolbox

The curves clustering using CCToolbox [Gaffney 2004] needs at least two parameters : the number of clusters K wanted and the clustering method. For the experimentation, the value of K has been set to 3, which was roughly corresponding to the subject that is doing the action, the background and everything that is in-between. The method for clustering chosen is a linear regression. One of the results obtained is shown in figure 2.

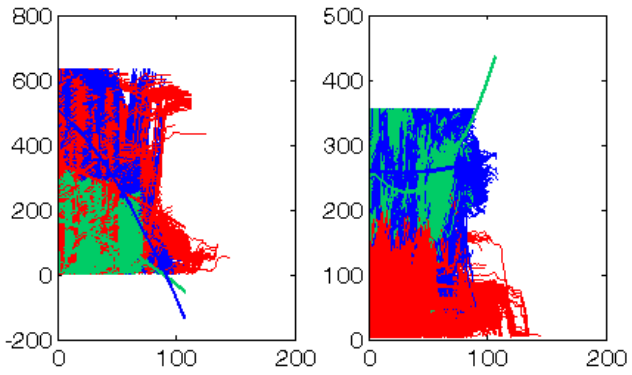

Figure 2: *A bold line is the cluster mean curve. Left : Width vs Time, right : Height vs Time*

The figure 2 shows that the green curves corresponds to the subject and correctly models the action (skier doing a front flip, then going straight). The blue curve is the background.

However further experimentations on this approach have not been pursued. The cause is that Particle Video is slow at computing the trajectories. The correct K for the clustering part is part and it is difficult to estimate it; especially in an automated way. However, this approach could have been continued if the time at disposal was enough.

2.2 Objects Segmentation

In a more direct way, the second approach tempted to quantify the motion frame by frame (and not by looking at long range trajectories). The first step is to segment the objects, texture-based, in each frame. The areas in the image corresponding to similar objects group the corresponding values contained in the underlying optical flow. The average of the optical flow values in the area give an approximation of the motion of an object. This approach would then give enough information to build a motion profile for the video. After the profiling, a good idea would be to add the result to a small database with the number of objects detected, for which texture, and their speed. This database could then be used to enhance the correctness of a profile.

The object segmentation has been done with a hierarchical graph-based video segmentation [Grundmann et al. 2010] technic.

2.2.1 Graph-based Segmentation

The graph-based segmentation tool is available as an online web service where the video is uploaded. Once the computation done, it produces a XML-like file that contains a hierarchy of segmentation levels found. An additional tool that uses the protobuffer library from Google[1] is helpful to visualize the levels (see figure 3). The next step would be to find a reliable method to determine the right degree of highest hierarchy levels to set, for each frame. This approach could have been the approach taken and could be investigated in a future work.

Figure 3: *Top : source image, Bottom : 50% of highest hierarchy level*

2.3 Adaptive Clustering

The third approach is simpler than the other two. The advantages are : a simpler design means a better guarantee to have results and, if the results are satisfying, they can be improved. Also, it is worthy to mention that the computation time is greatly reduced. The main idea is to compute directly from the optical flow a quantifier of the motion between two frames. These quantifiers, seen together in a spline, inform of the dynamism of the video. The first step is to read the optical flow values and cluster them using an adaptive clustering (see the section about the implementation for more details). The optical flow used as a source is a modified optical flow, called ω-flow [Jain et al. 2013].

2.3.1 Omega-Flow

The omega flow (ω-flow) [Jain et al. 2013] is obtained by subtracting to the original optical flow the affine optical flow. Let F_t be the optical flow between frame f_t at time t and frame f_{t+1} at time $t+1$. Let A_t be the affine optical flow defined between the same frames, which corresponds to the curves satisfying the equation $ax + b$, for a and b fixed. Then the ω-flow is simply defined as

$$\omega flow_t = F_t - A_t. \quad (1)$$

The omega flow is better for isolating the action in the videos by canceling the dominant movement of the scenes, which usually is the movement of the camera.

3 Implementation

The ω-flow is first computed between each consecutive frames. For each ω-flow, the squared norm of the displacement vectors

[1] Protocol Buffer : code.google.com/p/protobuf/

are sorted. During this phase, values that are below a threshold, the *_minThreshold*, are skipped. This threshold eliminates the background values. The highest values are also eliminated if they are larger than the half of the length of the diagonal of the frame, squared (*_maxTheshold*). If the time between two frames is 40ms, an object that appears at the top-left, then in the middle and finally at the bottom-right can barely be seen. Furthermore, the optical flow is likely to be of very bad quality, that's why this threshold could even be further decreased. The sorted values are then read from the lowest value to the highest one. A value v_i is put in the same cluster as value v_{i+1} if $v_i - v_{i+1} < \_maxDistInsideCluster$. This adaptive distance takes into account the difference in speed of the pixels that are part of the same object (like in figure 4).

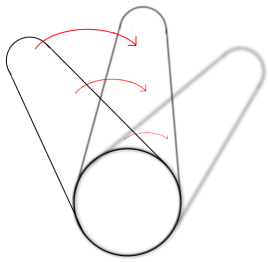

Figure 4: *Relative movement of pixels of the same object.*

A cluster is deleted if the number of elements inside it are below the threshold $\_minClusterSize$. Then, the values inside a cluster c_i are averaged. The motion estimation of two consecutive frames, f_t and f_{t+1}, is computed by the average of all the n cluster's averages, represented by this formula :

$$Motion_t = \frac{1}{n}\sum_{i=1}^{n} c_i.average \tag{2}$$

The adaptive clustering also reduces the number of the clusters that could be find with the standard clustering method using a fixed distance between the two extreme values. It thus avoids overestimating the final value.

Once all the $Motion_t$ are computed, they are smoothed using two running averages. This smoothing is important as the ω-flow values can be shifted up or down by a small amount between each frames, depending on the quality of the video, its encoding method and its resolution. To obtain a good smoothing while preserving the characteristics of the motion, a double running average is applied :

$$pass_1 = moving\_avg(values, window\_size) \tag{3}$$
$$pass_2 = moving\_avg(pass_1, window\_size - 1) \tag{4}$$

This double average gives good result in practice (see figure 5. The $window\_size$ (or ws) parameter can be customized. A lower value is best suited for videos of good quality. Videos with very little dynamism should also use a small window size so that small moments with high motion are not flatten.

Typically, a good value for the window size is around 10. The window size is adjusted for data points that cannot accommodate the specified number of neighbors on either side, and the end points are not smoothed because a span cannot be defined.

The profile is then loaded into a timeline, each key representing the t^{th} motion estimation for frame f_t (or the value of $Motion_t$). The intermediate values are reconstructed by drawing a spline through all these keys. The spline can then be adjusted in different useful ways (see the next subsection Spline). From the spline, the interpolation factors are computed to render the final remapped video.

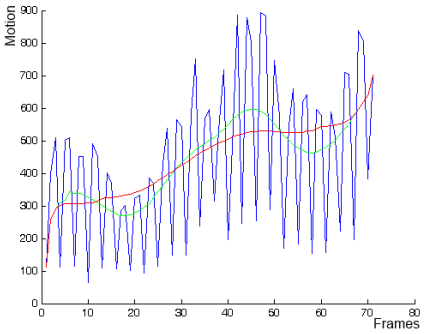

Figure 5: *Blue : raw $motion_t$, Red : ws=24, Green : ws=10*

3.1 Spline

The spline represents the time remapping for the video. The transition from the profile vector that contains all the $Motion_t$ values to the two dimensional graph containing the control keys for the spline is computed like this : the profile's values are read and the average, μ, of all the motion estimators is computed. The average μ is hence the average motion intensity of the video. Then, for each value in the profile, its position in the graph is : the x position is the timestamp in milliseconds for the frame at time t, and the y position is computed by

$$y = -1 + \begin{cases} -\frac{Motion_t}{\mu} & \text{if } Motion_t > \mu \\ \frac{\mu}{Motion_t} & \text{otherwise} \end{cases} \tag{5}$$

A value of 0 for y means that the motion is exactly equal to the average. A value of -1 means that the motion is two times faster than the average, and a value of 2 means that the motion is three times slower than the average. The reason of the shift by 1 is to center the spline in the middle of the graph. Now it is easy to read that a negative value means that the video have to be slowed down by the appropriate factor : if y is equal to -1, then the frame at original time should be played two times slower.

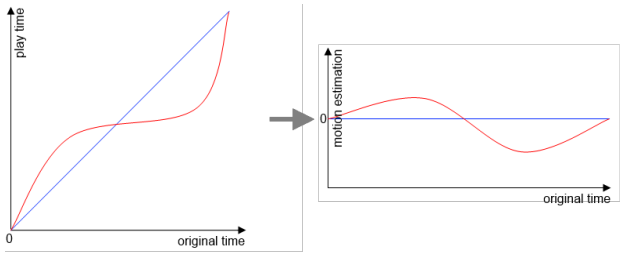

Figure 6: *Comparison between the remapping graph and the motion estimation representation.*

Compared to a simple graph that would directly relate the original time to the remapped time, this graph is interesting for intuitively see when we have a speed up or a slow down, and it is easier to

draw it like a flat rectangle instead of a square (especially for long video). Another advantage is that the computation of the factor for interpolating the frames does not require finding any inverse function as we already have a representation of the ratios (see next subsection about Interpolation).

The spline could have some outliers. To remove them, a filter can be applied that uses the Chauvenet's criterion to determine if a point should be rejected or not :

$$2 * CDF((\bar{y} - y)/\sigma) * n < cutoff \tag{6}$$

where CDF is the cumulative distribution function of the Gaussian distribution, $\bar{y}$ is the mean of the points, σ is the standard deviation of the points and n is the number of points. The $cutoff$ value is ideally set to 0.5, but can be changed between 0 and 1 to relax or increase the effect.

One problem that can arise is that the control points are too numerous. A feature permits to reduce their number by using a spline curve fitter[2] from the *GeometricTools*[3] library. It finds a curve with the desired number of control points that minimizes the least square distances with the original curve.

3.2 Point of interest

Once the spline is loaded, it is possible to select a key moment in the video where we want to slow down the video. It would break the adaptive remapping, but it is useful if the video is instead very uniform in dynamics and we are not happy with remapping. The resulting video would then be progressively slowed down until the point of interest is reached, then accelerated again. The values of the profile are decreased (equivalent to increasing the slow motion effect) stronger when the x position is near the point of interest's x position. If the profile overfits the filter at some points, it is possible to cut these values down. A possible filter is shown in the following pictures :

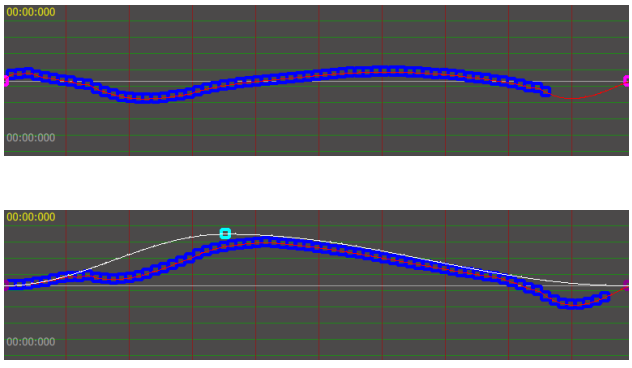

Figure 7: *The point of interest is in cyan.*

3.3 Optical Flow

The optical flow is needed to compute the image between two frames (interpolation). Given a frame f_t and a frame f_{t+1}, the optical flow between these two frames is composed by the forward flow Ff_t and the backward flow Fb_t. A frame fi_t can be computed in between by defining a factor $\alpha \in [0, 1]$:

$$fi_t(\alpha) = \frac{(f_t + \alpha * Ff_t + f_{t+1} + (1 - \alpha) * Fb_t)}{2} \tag{7}$$

For the experiment, the optical flow is computed using two methods from OpenCV : Farneback (on the CPU) and Brox (on the GPU). The result is better with Brox but takes more time to compute. The first attempt at computing the optical flow was to use a state of the art library [Liu]. The Matlab code was working fine, but C++ implementation was broken and too much time has been spent trying to fix it.

3.4 Interpolation

In the previous subsection the α term has been defined. This term must be found for each possible in between frames fi_t needed. For this purpose, the terms are found by asking the spline what is the interpolation factor at position x until the sum of the factors is above 1. The x position is incremented by a fraction of the time per frame (tpf) that is usually 40ms (for 25 images per seconds).

```
List list; sumTime(0); ratio(0); sumRatio(0);
do {
    y = splineValue(sumTime);
    if (ratio < 0) {
        //slow down : 0 < ratio < 1
        ratio = 1 / (1 - y + 1);
        sumTime += tpf * ratio;
    } else {
        //acceleration : ratio >= 1
        ratio = y + 1;
    }
    list.push_back(ratio);
} while ((sumRatio += ratio) <= 1)
```

For example, if the ratios returned are $0.3, 0.4, 0.5$ for $x = 0$, the sum is 1.2. Using equation 7, the following frames are computed : $f0_t(0.3)$, $f1_t(0.7)$, $f0_{t+1}(0.2)$. The next time, the factors will be asked for $x = 0 + tpf * 1.2 = 48$. The newly created frames are then read in order at an intervall of 40ms, forming the remapped video.

3.5 Video Player

The video player has been implemented in C++ with Qt 5.1 and its development has taken a major part of the time at disposal (as it was also needed to have a robust test system). It uses FFMPEG to decode and write each valid frame of a video to the disk. The frames are then used later to compute the optical flows between each consecutive frames. The player has a size-customizable buffer for playing the video very fast or slowly. It also has a timeline, of the length of the video, with a spline and control points that represents the time remapping. Different user buttons start the computation of the optical flows, the motion estimation and the interpolation. Each button brings a dialog with customizable parameters. There are other controls that affect the spline, like the application of a slow motion amplification filter, or global amplification filter, outliers filters, duration adjustment filters and rescaling.

The main development platform has been Windows (7 and 8.1). The port to Linux and MacOS should not be too long, but requires the compilation of OpenCV, Boost, GeometricTools and FFMPEG, any recent versions.

[2] LibMathematics/CurvesSurfacesVolumes/Wm5BSplineCurveFit.h
[3] www.geometrictools.com

4 Results

The tests have been done using two different machines. The first is using Windows 7, has 4 GB of RAM and a T9500 processor @ 2.6GHz. The second machine is using Windows 8.1, has 8 GB of RAM and an i7-4800MQ @ 2.7GHz based clock. The videos are selected for they different kind of dynamism. The set of videos used is composed of (Length/Res : video's length and resolution, CTime : computation time for the clustering and spline part (average between the two test machines), Summary : video content description) :

Length/Res	CTime	Summary
3sec(640x360)	9s	Counter-intuitive remapping
5sec(640x360)	15s	Already slow motioned.
10sec(480x270)	28s	Three snowboard jumps.
13sec(608x336)	38s	Long, interspersed movements
19sec(640x360)	62s	Long, repetitive movements

The first point to notice is that even if the ω-flows are noisy, the resulting motion profiles correspond to the majority of the videos tested. In the table above, the video with interspersed movements was the only one that couldn't obtain a satisfying profile. Changing one parameter a bit would change the shape of the spline consequently. This is mainly due to the length of the video. The action occurs all the time, but some of them far of the camera, others close. Then the scene cut with less movements, and continue again with high dynamics. The ω-flows values are clustered locally, so the different angles of the camera could not be uniformized.

The videos with repetitive movements are nicely profiled and remapped. The part where a snowboarder is in the air is slowed down, and when he lands the video is sped up. The video that have been already slow-motioned is remapped by accelerating the slow-motioned part and by decelerating the normal speed part. The effect is interesting. The last video shows that some videos containing parts we would like to slow-motion do not always correspond to what we expected. The dynamic is closely related to the position of the camera regarding the action. If a skateboard trick performed in the air is farther from the camera than the end of the trick, then the end of the trick would have more dynamics and would be slowed-down.

Regarding the profile display, the borders of the profile however have sometimes outliers due to the first/last frames decoded that are not of good quality (when a video is cut manually for example). These outliers can be corrected by using the Chauvenet's criterion, and the profile can be reshaped easily by diminishing the number of control points and adjusting the control keys near the border with the mouse if the shape is not satisfying (example in figure 8).

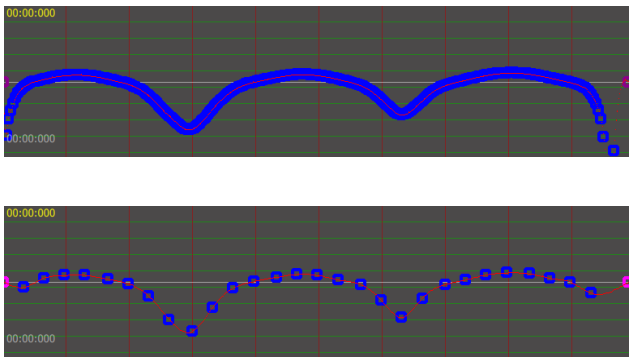

Figure 8: *Top to bottom : original spline, control keys simplification + removed outliers*

The adaptive clustering has proved its utility compared to a fixed clustering. For example, the figure 9 is the same profile as figure 8, and shows that the standard clustering completely fails at finding a valid motion profile for the same parameters.

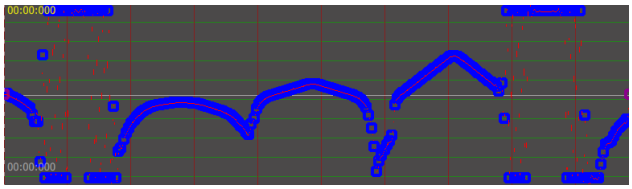

Figure 9: *Non-adaptive clustering method.*

5 Future work

The parameters for the clustering part (*_maxDistInsideCluster, _minThreshold, _minClusterSize*) could be determined automatically. This step requires investigations of the different kind of profiles generated for a set of videos with specific parameters. Videos with a few movements should have a lower *_minClusterSize* to be more accurate. Also, videos with a lot of different movements should have *_maxDistInsideCluster* that is bigger, and *_minClusterSize* increased a bit.

An important part should be the ability to automatically find parts where the camera is filming the scenes the same manner, by comparing the affine flow. Then these parts are separated and would form sequences where subprofiles can be computed.

Another important point is that the intensity of the remapping should automatically be decreased and smoothed a bit when strong accelerations or decelerations appear (big bumps in the spline). Even if the remapping is correct, the visual effect is sometimes too abrupt and should be smoothed.

The code needs some optimizations, like avoiding to compute two times the optical flow, once for the interpolation and once for the ω-flow. The optical flow could be ideally reused in some way between the stages of the runtime. Also, the frames are written to the disk for computing the optical flow for interpolation, and could be avoided. Now, there exist applications that are able to compute a good quality optical flow and interpolate frames in real time (like the Smooth Video Project [4]). Enhancing the application to be able to see a remapped video in real time would have useful applications.

Acknowledgments

Thanks to Boris Neubert for the great advices and help.

References

GAFFNEY, S. J. 2004. Probabilistic curve-aligned clustering and prediction with regression mixture models. Tech. rep., Ph.D. Dissertation, 2004. Laboratoire MAS.

GRUNDMANN, M., KWATRA, V., HAN, M., AND ESSA, I. A. 2010. Efficient hierarchical graph-based video segmentation. In *IEEE CVPR*, 2141–2148.

JAIN, M., JEGOU, H., AND BOUTHEMY, P. 2013. Better exploiting motion for better action recognition. In *CVPR*, IEEE, 2555–2562.

[4] www.svp-team.com

LIU, C. Beyond pixels: Exploring new representations and applications for motion analysis. Tech. rep., Doctoral Thesis. Massachusetts Institute of Technology.

P.OCHS, AND T.BROX, 2011. Object segmentation in video: a hierarchical variational approach for turning point trajectories into dense regions.

SAND, P., AND TELLER, S. J. 2006. Particle video: Long-range motion estimation using point trajectories. In *CVPR (2)*, IEEE Computer Society, 2195–2202.

A Software

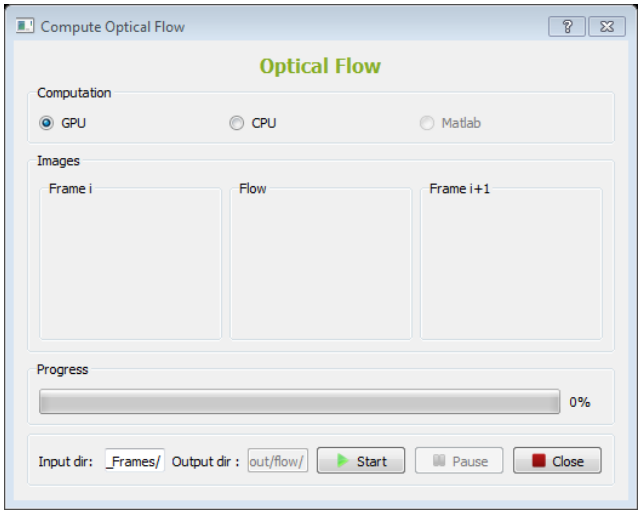

Figure 10: *Optical Flow computation interface*

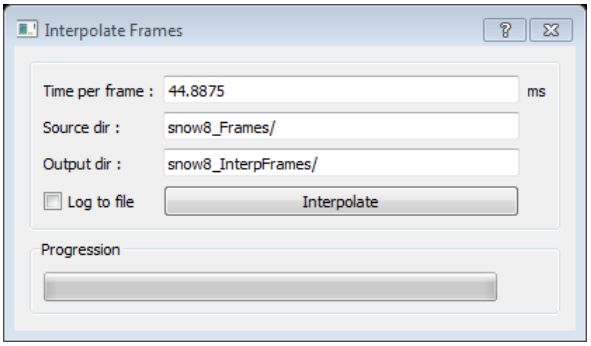

Figure 11: *Interpolation computation interface*

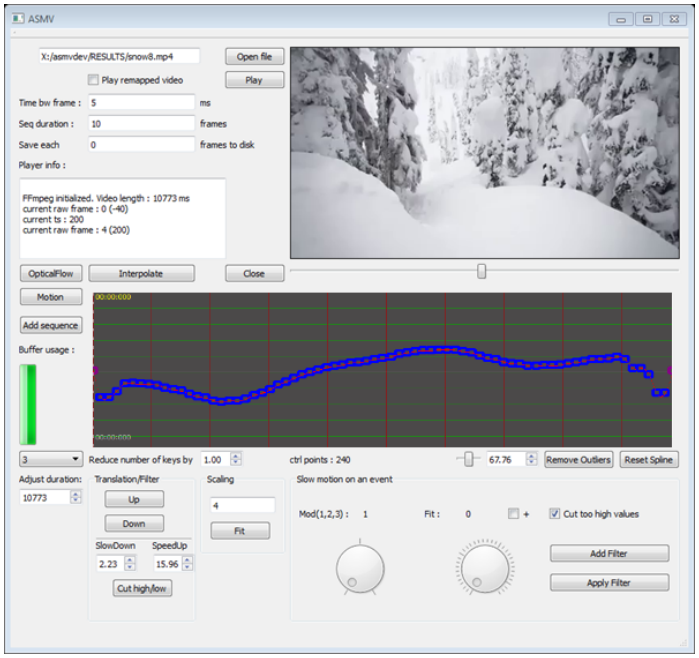

Figure 12: *Main Window interface*

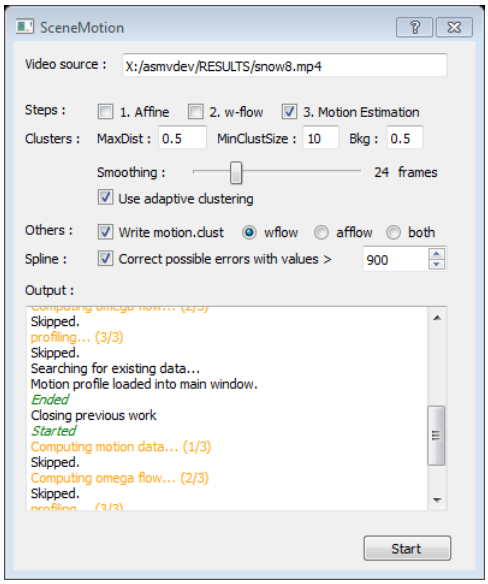

Figure 13: *Motion profile computation interface*