Innovative Tools for 3D Cinema Production

Werner Bailer[1], Hannes Fassold[1], Jakub Rosner[2], Georg Thallinger[1]

[1] JOANNEUM RESEARCH
DIGITAL – Institute for Information and Communication Technologies
Steyrergasse 17, 8010 Graz, Austria, {firstName.lastName}@joanneum.at

[2] Silesian University of Technology
PhD Faculty of Data Mining
Ulica Academicka 16, 44-100 Gliwice, Poland, jakub.rosner@joanneum.at

Summary

The current success of immersive 3D experiences in feature films and the trend towards 3D TV require advanced tools and workflows for high-quality capture of multi-view live scenes. The 2020 3D Media project is researching, developing and demonstrating novel forms of compelling entertainment experiences based on new technologies for the capture, production, networked distribution and display of three-dimensional sound and images. In this contribution, we discuss three innovative tools for the production and post-production of high-quality multi-view content. The first is a GPU accelerated feature point tracker, providing crucial input for further processing such as motion analysis and depth estimation. Compared to an already optimised CPU implementation (using Intel IPP) a speedup factor of approximately 10 can be achieved, enabling real-time tracking of several thousand feature points in two Full HD resolution video streams on a single GPU. The second is a suite of automatic analysis tools for multi-view content, and the third is a tool for browsing multi-view content sets, with both a desktop and a Web-based user interface.

1 Introduction

The current success of immersive 3D experiences in feature films and the trend towards 3D TV require advanced tools and workflows for high-quality capture of multi-view live scenes. The 2020 3D Media project[1] is researching, developing and demonstrating novel forms of compelling entertainment experiences based on new technologies for the capture, production, networked distribution and display of three-dimensional sound and images. The users of the resulting technologies will be both media industry professionals across the current film, TV and 'new media' sectors producing programme material as well as the general public.

In this contribution, we discuss three innovative tools for the production and post-production of high-quality multi-view content: A feature point tracker, accelerated by implementation on graphics processors (GPU), providing crucial input for further processing such as motion analysis and depth estimation, a suite of automatic analysis tools for multi-view content, and a tool for browsing multi-view content sets. The components described have been integrated in a 10GigE based capture infrastructure.

In the 3D production process, we are dealing with multi-view content, i.e., at least two views of the scene are recorded. In contrast to audio, where presentation with more than two channels is in widespread use, the target presentation medium for visual content is typically stereoscopic cinema or TV. Nonetheless, there are several reasons for shooting more than two views. Depth information is needed in many applications, and while it can be estimated from a stereo pair, more reliable depth information can be obtained from more views. In post-production, one would like to have the creative freedom to move objects in the scene, and to adjust the depth in order to control the 3D impression experienced by the audience. Objects in the scene occlude each other and the background, so that more views are needed for filling the occlusion areas, when the scene is modified. In addition, there are types of displays, such as autostereoscopic

[1] http://www.20203dmedia.eu

displays, that generate a high number of views and thus need reliable depth and occlusion information as input.

The rest of this paper is organised as follows. Sections 2 through 4 describe the three tools in more detail, and Section 5 provides conclusions.

2 GPU Accelerated Feature Point Tracking

The automatic detection and tracking of (typically corner-like) feature points throughout an image sequence is a necessary prerequisite for many algorithms in computer vision. The gathered information about the feature points and their motion can be used subsequently, e.g., for pose estimation, camera calibration and for tracking various kinds of objects like people and vehicles. For feature point tracking, we employ the widely used KLT algorithm (Lucas & Kanade, 1981). It first detects a sparse set of feature points with sufficient texture in the current image and adds them to the already existing ones. Afterwards, the positions of feature points in the subsequent image are estimated by minimising the dissimilarity measured as sum of squared differences (SSD) in a window around the point positions. In this section, we first describe the detection and tracking step of the KLT algorithm briefly. Afterwards, we show how to implement the KLT algorithm efficiently on the GPU using the NVIDIA CUDA framework[2] and compare the CPU and GPU implementations in terms of quality and runtime. More details on the GPU implementation of the feature point tracker can be found in (Fassold et al., 2009).

In the **feature point detection** step, new features are found based on a cornerness measure in an image and added to the already existing feature points. We denote I as the current image and J as the subsequent image in the sequence. We write the spatial gradient of I as $\nabla I = \partial I / \partial(x, y)$ and a small (e.g. 5×5) rectangular region centred at a point p as $W(p)$. Note that p can have non-integer coordinates, the values are then calculated using bilinear interpolation.

As a measure of the cornerness of a feature centered at a pixel p, first the structure matrix G defined by

$$G = \sum_{x \in W(p)} \nabla I(x) \cdot \nabla I(x)^T$$

is calculated. Its eigenvalues λ_1 and λ_2 (which are ≥ 0, as the matrix is positive-semidefinite) can be used to derive information about the image structure in the neighbourhood of p. The smaller eigenvalue $\lambda = \min(\lambda_1,\lambda_2)$ is employed in the KLT algorithm as cornerness measure of region W. The detection of new points can be summarised in the following steps:

- *Cornerness calculation.* For each pixel in the image I, its structure matrix G and cornerness is calculated. Furthermore, the maximum cornerness λ_{max} which occurs in the image is calculated.
- *Thresholding and non-maxima suppression.* From the image pixels, only the pixels which have a cornerness λ larger than a certain percentage (e.g. 5%) of λ_{max} are kept. Afterwards, non-maxima suppression within the neighbourhood of the remaining points is performed.
- *Minimum-distance enforcement.* From the remaining points new ones are added, starting with the points with the highest cornerness values. To avoid points concentrated in some area of the image, newly added points must have a specific minimum distance d (e.g. 5 or 10 pixels) to points already added.

In the **feature point tracking step**, we want to calculate for each feature point p in image I its corresponding motion vector v so that its tracked position in image J is $p + v$. As 'goodness' criterion of v we take the SSD error function

$$\varepsilon(v) = \sum_{x \in W(p)} (J(x+v) - I(x))^2 .$$

[2] http://www.nvidia.com/object/cuda_home_new.html

It measures the image intensity deviation between a neighbourhood of the feature point position in I and its potential position in J. Setting the first derivative of $\varepsilon(v)$ to zero and approximating $J(x+v)$ by its first order Taylor expansion around $v = 0$ results in a better estimate. By repeating this multiple times, we obtain an iterative update scheme for v. Typically, v converges after 4 - 7 iterations, for static feature points the convergence is even faster.

Note that the Taylor expansion around zero is only valid for small motion vectors v. In order to handle large motion (e.g. fast moving objects), we apply the scheme in a multi-resolution representation. For this, we first generate an image pyramid, by repeatedly convolving the image with a small Gaussian kernel and subsampling it. Then we apply the scheme in each pyramid level (for all feature points), going from the highest levels to lower ones. In each level, the properly scaled motion vector v of the previous level is used as the initial guess in the current level. In this way, the estimated motion vector v is refined in each level.

In the following, we discuss some aspects of the **GPU implementation**, using the CUDA programming environment by NVIDIA and compare the GPU implementation with a highly optimized CPU implementation from the OpenCV[3] library.

Feature point detection. The cornerness calculation is easy to parallelise, as it is done independently for each pixel. In contrast, the parallel calculation of the maximum cornerness λ_{max} is more involved, as many threads would have to modify the same variable simultaneously, leading to many read-write hazards. This issue can be solved efficiently by using the CUDPP[4] library. In the thresholding and non-maxima suppression steps, we mark features that do not meet the respective conditions as invalid. The last step, the minimum distance enforcement, cannot be implemented efficiently on the GPU as it is inherently serial. So we first transfer all features, which have not been marked as invalid, back to the CPU memory. Once again we use the CUDPP library for filtering out the invalid features before transferring. For the minimum-distance enforcement, we employ an alternative algorithm (using a mask image) which is more efficient than the OpenCV method, if many feature points (> 1000) are to be handled. It adds a point only if its position is not already marked in the mask image. If the point is added, a circular area around it with radius d is marked in the mask image, indicating that area as occupied.

Feature point tracking. In contrast to feature point detection, the whole feature point tracking (for a specific pyramid level) is done only by one CUDA kernel to get the most benefit of shared memory. Each feature point corresponds to exactly one GPU thread. To take advantage of the hardware bilinear interpolation, the image pyramids are stored as texture images. Additionally, we take advantage of the most recent features of the NVIDIA Fermi GPU architecture by configuring the CUDA kernel so that it prefers the L1 cache over shared memory.

Results. We compare the quality and speed of the GPU implementation with respect to a CPU implementation from the OpenCV library (which uses internally the multi-threaded, performance-optimized Intel Performance Primitives library[5]). The runtime measurements were done on a 3.0 GHz Intel Xeon Quad-Core, equipped with a NVIDIA Geforce GTX 480 GPU. A quality comparison of the feature points delivered by both implementations shows us that most of the detected and tracked feature points are similar. Minor differences are likely due to slightly different floating-point implementation in the GPU and CPU. The GPU implementation is roughly an order of magnitude faster than the CPU implementation (see Figure 1) and allows us to track several thousand points in multiple Full HD video streams in real-time with only one GPU.

[3] The OpenCV library (http://opencv.willowgarage.com/) uses SSE and OpenMP internally.

[4] CUDPP (http://code.google.com/p/cudpp/) provides some useful functions for efficient compaction, sorting etc.

[5] http://software.intel.com/en-us/articles/intel-ipp/

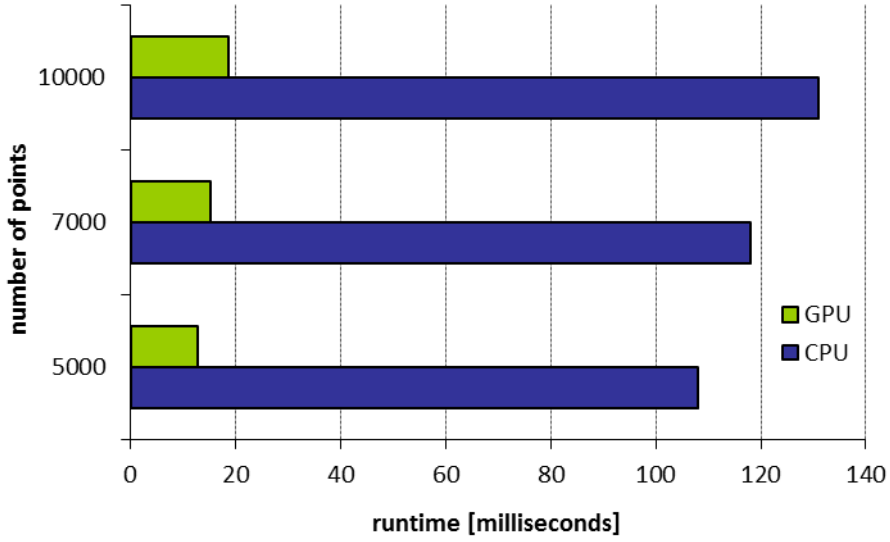

Figure 1: Comparison of the runtime of the GPU and CPU implementation of the KLT algorithm for Full HD material and different maximum numbers of feature points (5000, 7000, 10000).

3 Automatic Content Analysis for Multi-view Content

As manual annotation of audiovisual content is a time consuming and thus costly process, automatic extraction of metadata from audiovisual content is often the only feasible option. For some simple features, such as key frame extraction, the available algorithms are reliable. However, existing content analysis tools only work on single video streams, ignoring the relation of different streams in multi-view content. This requires a late fusion of information extracted from the individual streams, which might be at least redundant, or in some cases even contradicting.

For more challenging analysis tasks, such as segmentation of objects, the robustness of algorithms can benefit from fusing the information from multiple views. In a few cases it may be possible to achieve this improvement in robustness in a late fusion step, however, even if this is possible it is often much more costly in terms of computation. It thus makes sense to treat multi-view content specifically throughout the content analysis process.

The automatic content analysis framework is mainly used for pre-processing content when ingesting into the browsing tool described in Section 4. The experimental application of the naïve approach of treating the streams of the different views independently and discussions with users have revealed the following requirements: The workflow must be streamlined so that a single tool handling both single- and multi-view content is necessary. Such a tool shall be able to perform temporal alignment of the views if necessary and must produce a single metadata document containing both analysis results valid across views as well as specific to one view. The most relevant information to be described synchronously across views is temporal segmentation information as well as representations of these segments such as synchronous key frames. In addition, low-level visual features useful for retrieval and browsing, e.g., motion activity or colour descriptors, shall be described jointly for all views.

Based on these requirements we have designed a multi-step process for performing content analysis on multi-view content (summarised in Figure 2). In the first content analysis step the mean visual activity of the scene is determined from the visual activities of the individual streams. The visual activity is calculated as the pixel difference of two consecutive frames and averaged over *n* frames. Shot boundaries are detected and fused across the streams and shots are associated. Only hard cut detection is performed, as we are dealing with raw material and can assume that there are no gradual transitions in the material. A

linear SVM is used taking frame differences of three consecutive frames as input. The SVM is implemented through LIBSVM (Chang & Lin, 2001). To train the SVM classifier, ground truth from the TRECVID 2006 shot boundary detection task (Smeaton & Over, 2006) has been used.

This first analysis step also extracts a regularly sampled sequence of colour, texture and visual activity descriptors. This information can be used to perform automatic temporal alignment of the video streams from different views in order to determine their temporal offset. We use the variant of the Longest Common Subsequence (LCSS) proposed in (Bailer et al., 2009) for the identification of repeated takes of the same scene. For the view analysis problem we do not permit gaps in the matches and give 80% of the weight for similarity calculation to visual activity and 20% to colour features. This makes the approach robust against the differences in visual content and colour calibration of the different views.

The second content analysis step uses the fusion results as input. Key frames are a representative subset of the original video. Selecting the first and uniformly every i^{th} frame from each video shot is a simple way to define key frames. But this method produces often a large number of redundant frames showing nearly identical content. Thus key frames are extracted based on the visual activity in the shot, using a threshold determined dynamically within a sliding window. For these key frames, the MPEG-7 descriptors ColorLayout and EdgeHistogram (MPEG-7, 2001) are extracted. From the ColorLayout descriptor, the DC and the first two AC coefficients of each channel are used.

The evaluation of the results on a data set from the 2020 3D Media project shows that 86% of the shot boundaries have a temporal offset between zero and five frames between the views. Only about 2% have an offset between 6 and 50 frames, for the others a shot boundary is simply not present in one of the views. Thus using a five frames time window for determining synchronous shot boundaries across multiple views is able to correctly associate nearly 93% of the shot boundaries that actually exist across all views. For key frames extracted based on the visual activity from single views, about 41% have been extracted at identical time points across all views and 65% within an 11 frame time window, but almost 20% of the key frame time points have been detected only in one or some views due to differences in visual activity.

4 Browsing Collections of Multi-view Content

In the post-production phase, users typically deal with large amounts of audiovisual material with a high degree of redundancy and need to select a small subset for use in a production. We thus propose a tool for navigating and organising collections of raw video material.

Newly shot material is typically sparsely annotated, thus the browsing tool has to rely on automatically extracted features. When content is ingested into the tool, automatic content analysis as described in the previous section is performed. Currently, camera motion estimation, visual activity estimation, extraction of global colour features and estimation of object trajectories are implemented. The extracted features are represented in MPEG-7 and indexed in an SQLite database.

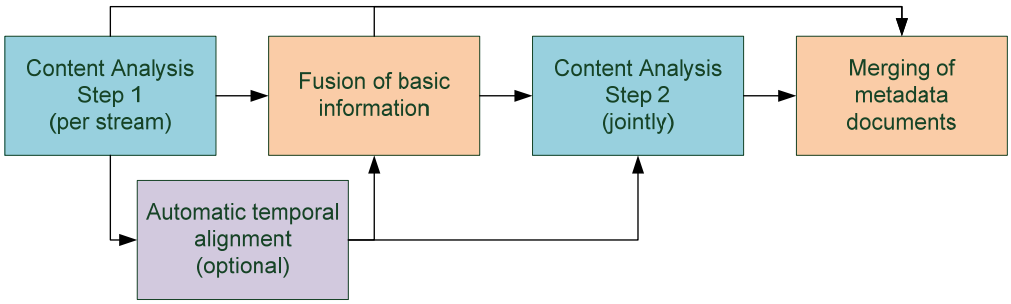

Figure 2: Multi-view content analysis workflow.

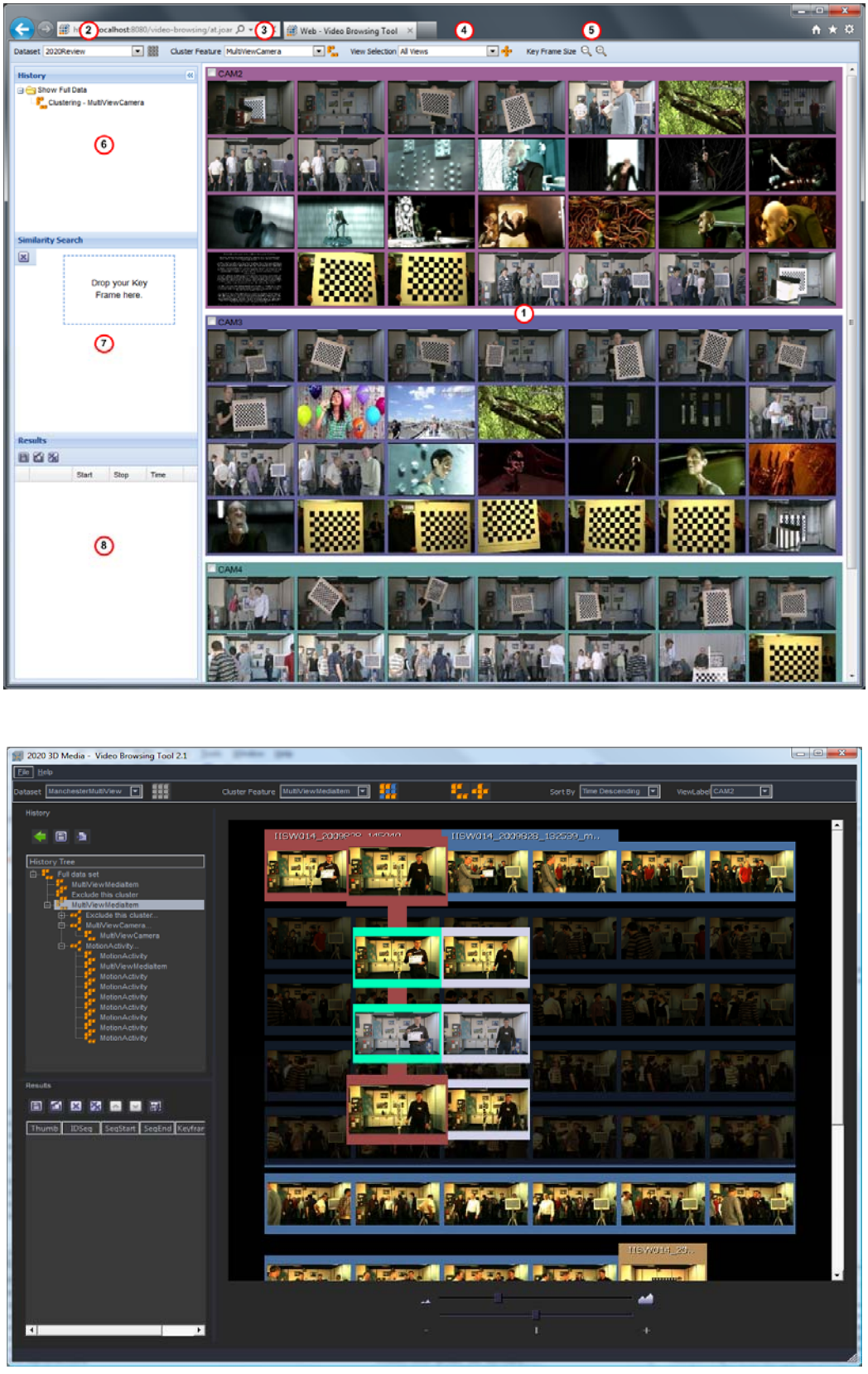

Figure 3: User interface of the Web-based (top) and desktop (bottom) client of the video browsing tool.

The browsing tool implements an iterative content selection process, where users cluster content by one of the automatically extracted features and can then select relevant clusters to reduce the content set. Further clustering by the same or other features can then be applied to the reduced set. A detailed description of the tool can be found in (Bailer et al., 2010).

Both a desktop client and a Web-based client have been implemented. The goal was to provide the user a uniform behaviour, look and feel of the different implementations. As illustrated in Figure 3, the central component of the browsing tool's user interface is a light table (1). It shows the current content set and cluster structure using a number of representative frames for each of the clusters. The clusters are visualised by coloured areas around the images. By clicking on an image in the light table view, a video player is opened and plays the video represented by that image starting at the segment's beginning time point. In the application the temporal context of a key frame is shown by a time line of temporally adjacent key frames including possibly available multi-view content which is shown when the user moves the mouse over a key frame. The size of the images in the light table view can be changed dynamically (5) so that the user can choose between the level of detail and the number of visible images without scrolling.

The workflow in the browsing tool is as follows: the user starts with selecting a dataset (2). By selecting one of the available features (3) the content will be clustered according to this feature (e.g., camera motion, visual activity, object trajectory, or global colour similarity). Depending on the current size of the content set and the available space in the browser window, a fraction of the segments (mostly a few percent or even less) is selected to represent a cluster. The user can then decide to select a subset of clusters that seems to be relevant and discard the others, or repeat clustering on the current content set using another feature (3). In the first case, the reduced content set is the input to the clustering step in the next iteration. (4) allows the user to focus on a specific view in a multi view content. If no key frame of the selected view is available then an alternative view is selected where the user gets informed by displaying the name of the alternative view.

On the left side of the application window the history (6), the similarity search (7) and the result list (8) are arranged. The history feature automatically records all clustering and selection actions done by the user. By clicking on one of the entries in the history, the user can set the state of the summarizer (i.e., the content set) back to this point. The user can then branch the browsing path and explore the content using alternative cluster features. To execute the similarity search (7), the user drags the desired key frame into the marked area for the similarity search. The application then queries all available similarity search options, displays them where the user can further proceed with the search task. The result list (8) can be used to memorize video segments and to extract segments of videos for further video editing, for example, as edit decision list (EDL). The user can drag relevant key frames into the result list at any time, thus adding the corresponding segment of the content to it.

Specific clustering features and further options are offered to the user via a context menu of the key frames where the users can select and discard a whole cluster or select and discard a single video. Furthermore, the metadata of the selected key frame can be displayed by selecting the corresponding context menu item.

5 Conclusion and Future Work

In this paper, we have presented three tools for processing multi-view content, which help improving different stages of the 3D cinema production workflow. A GPU-accelerated feature point tracker has been presented which achieves a speedup of roughly an order of magnitude when compared with the CPU implementation, allowing us to track several thousand points on two Full HD video streams simultaneously in real-time with only one GPU. The speedup is crucial for a component like this, which serves as a basic processing step for a number of subsequent algorithms.

For handling content in post-production, an interactive video browsing solution has been presented, deployed both as desktop application and as a light-weight Web-based client. The tools supports an iterative content selection process, which in the end generates a result list that can be transferred to a non-

linear video editing software. The tool relies on the results of automatic content analysis tools and thus enables indexing of production-size sets of audiovisual content without the need for manual intervention.

Acknowledgement

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 215475, "2020 3D Media – Spatial Sound and Vision". The work of Jakub Rosner has been partially supported by the European Social Fund.

6 References

Bailer W., Lee F., Thallinger G. (2009). A distance measure for repeated takes of one scene. The Visual Computer, vol. 25, no. 1, pp. 53–68.

Bailer W., Weiss W., Kienast G., Thallinger G., Haas W. (2010). A Video Browsing Tool for Content Management in Post-production. International Journal of Digital Multimedia Broadcasting.

Chang C.-C., Lin C.-J. (2001). LIBSVM: a library for support vector machines, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

Fassold H., Rosner J., Schallauer P., Bailer W. (2009). Realtime KLT Feature Point Tracking for High Definition Video. GravisMa workshop, Plzen.

Lucas B.D., Kanade T. (1981). An Iterative Image Registration Technique with an Application To Stereo Vision, Joint Conference on Artificial Intelligence.

MPEG-7 (2001). Information Technology—Multimedia Content Description Interface: Part 3: Visual. ISO/IEC 15938-3.

Smeaton A. F., Over P. (2006). TRECVID 2006: Shot boundary detection task overview. In Proceedings of the TRECVID Workshop.