provides a good initialization of the wall positions. Finally, we initialize the position of the cameras at the center of all wall corners that are seen in each camera's image.

14.5.7 Obtaining wall textures from the panoramic images

After the optimization process has converged, the position of the walls and the cameras are known, and we can generate texture maps for the walls by projecting the panoramic image content onto the wall planes. Since we may have several views of the same walls, recorded by different cameras, we have to decide from which camera image we extract the image data. The following points have to be considered.

- The wall should be visible. Cameras that are located on the back side of the wall, or which are occluded by other walls cannot be used.
- The larger the distance of the camera to the wall, the lower the texture resolution that is obtained.
- If the camera is too close to the wall, parts of it are viewed in an acute angle. As a result, changes of depth that are not reflected in the floor plan model can lead to perspective distortion artifacts. This applies, e.g., to furniture that is not projected orthogonally onto the wall.
- A wall may not be visible completely in any single camera view. In this case, the texture information has to be collected from several camera images.

The walls are processed independently, where we first determine which cameras are located at the front side of a wall. This information is obtained easily using the oriented angle from Eq. (14.7). If $\beta'_{i,j,k} > \pi$, then the camera $\mathbf{p}_k$ is located at the backside of wall $\mathbf{p}_i, \mathbf{p}_j$. Cameras that are at the backside are excluded from the further processing.

To decide from which camera the wall texture should be taken, we evaluate the expected image quality by determining the deviation of the camera position from an ideal camera position. For room corners $\mathbf{p}_1, \mathbf{p}_2$, we define the ideal camera position as $\mathbf{p}_c = \frac{1}{2}(\mathbf{p}_1 + \mathbf{p}_2) + \frac{1}{2}\mathbf{R}_\perp(\mathbf{p}_2 - \mathbf{p}_1)$, where $\mathbf{R}_\perp$ is a rotation by $\pi/2$. This places the ideal camera position on the perpendicular bisection of the wall at a distance that is half of the wall width. All cameras that are not at the backside are ordered according to the distance of this ideal position. In this ordering, the camera which is closest to the ideal position, comes first. For every column of texture pixels, the ray between the first camera and a pixel in the column is checked for

intersection with other walls. If there is an intersection, the second camera is checked for free sight to the pixel, and so on. Note that only one pixel in the column has to be checked since all walls are upright planes.

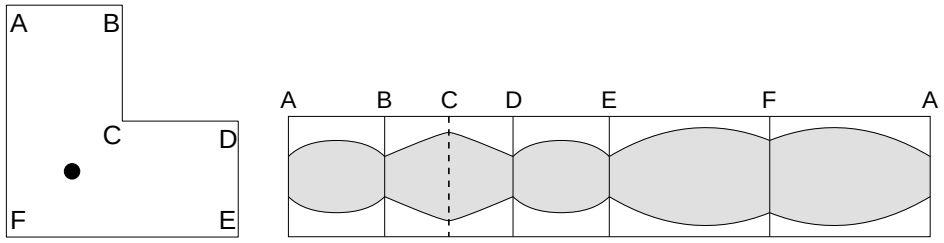
14.6 Experimental Results

Experiments have been carried out for both reconstruction algorithms presented in this chapter. For the rectangular-room reconstruction, the input images were captured with the panoramic video camera described in Section 14.2.2. These are well calibrated and generate undistorted panoramic images. An example reconstruction result is shown in Fig. 14.8.

Example results for the floor plan reconstruction are shown in Figure 14.21 and Figure 14.22. The input images for the floor plan reconstruction were captured with a digital still camera and combined into a panoramic image later. The focal length of the camera had to be estimated, since the EXIF data did not contain this information. During the stitching process, small inaccuracies in the image alignment were observed, which lead to inaccurate angle measurements. The computation time for the reconstruction was clearly below one second in all of our examples. The time for generating the texture maps depends on the required resolution and the number of walls, and was about one second for our most complex model. We evaluated the accuracy of the reconstruction result by comparing the normalized size of the walls in the reconstruction with their real sizes. The average deviation was about 4%, which is probably mainly due to the inaccurate alignment of the input images. Moreover, for simplicity, we assumed that the walls itself have zero depth, which is obviously wrong in reality and which also leads to small deviations in room size. Note that these inaccuracies are not obviously visible in the reconstruction, because the wall textures are stretched by this factor. Corners in the texture image always map exactly to corners in the geometric model.

14.7 Conclusions

In this chapter, we have described techniques to capture panoramic images and videos and we have discussed ways for optimal presentation of these panoramic images to the user. We have proposed a visualization specialized for panoramic images recorded in a rectangular room, which reconstructs the room geometry from the panoramic image and presents the panoramic image as the projection onto the room walls. The reconstruction algorithm requires only minor user support and is guaranteed to find the optimum solution. Furthermore, we generalized the concept to the reconstruction



(a) Room shape. (b) Corresponding cylindrical panoramic image.

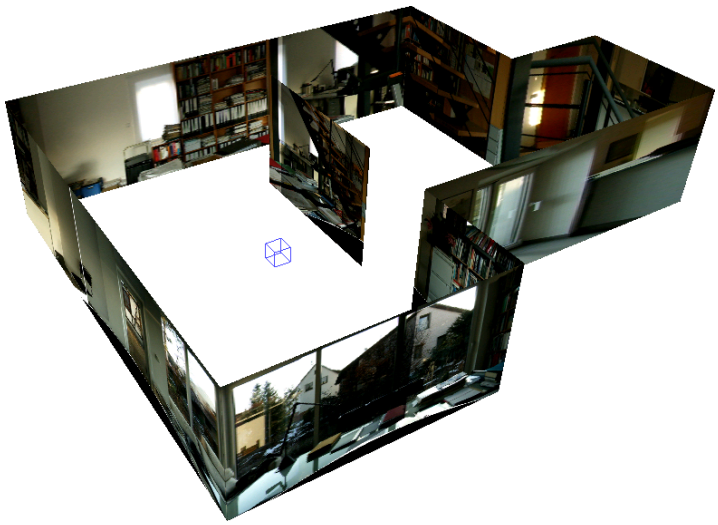
Figure 14.20: *The example room (a) is recorded with a camera located at the black spot. This results in the panoramic image (b). The convex corner C is indicated with a dashed line.*

of floor plans, comprising an arbitrary number of arbitrarily shaped rooms (preferably but not necessarily with perpendicular walls).

Our conclusion is that the proposed visualization can provide a better understanding of the scene to the user than a flattened panoramic image or a projection onto a cylinder, where the information about the room geometry is lost. Applications of our proposal, especially for the floor plan reconstruction, are also the advertisement of appartments or hotel rooms, for which virtual tours could be made available online. Another application could be the reconstruction of scenes in surveillance systems, in which the objects are extracted from the video and inserted into the 3-D model at their corresponding real-world position. It should be noted that both reconstruction algorithms can be used directly with panoramic video instead of single images, providing video textures on the walls of the 3-D model. Therefore, the geometry model only has to be computed once if the camera positions are kept fixed.

Future research

In future research, the reconstruction could be extended to a completely automatic process. Note that in a cylindrical panoramic image of a room, the vertical lines of the room corners remain straight, while the horizontal lines at ceiling and the ground become bent (see Fig. 14.8(a)). Tracing along the bent horizontal lines, it is easy to find the room corners, because these corners are always located at sudden changes of the line direction. Depending on the angle in which these lines meet in the corner, it is even possible to distinguish between *concave* and *convex* corners, corresponding to an inwards (90 degrees) or outwards (-90 degrees) corner (Fig. 14.20). Furthermore, corners at occluding walls show as discontinuities between the bent horizontal lines. If several panoramic images from the same room



(a)

Figure 14.21: *Example reconstruction of a single, non-rectangular room from only one panoramic image.*

are available, corresponding corners could be identified by comparing the wall texture.

Figure 14.22: *Example reconstruction for a complete appartment.*

Plaudite, amici, comedia finita est.
– Applaud, my friends, the comedy is over.
(Ludwig van Beethoven, on his deathbed)

Chapter 15

Conclusions

This thesis has presented various techniques for video-object segmentation. An automatic segmentation system for rotating cameras was presented and extended with object-model controlled segmentation and camera autocalibration. In this chapter, the achievements are summarized and it is discussed how these techniques could be enhanced in future research. Finally, interesting research directions are highlighted that may be promising approaches for future video-segmentation systems.

15.1 Discussion on the individual chapters

In the sequel, the achievements of this thesis and ideas for future research are discussed separately for each of the chapters of this thesis.

15.1.1 Chapter 3 and 4: camera-motion estimation

The first step in the proposed segmentation system is the alignment of all input frames into a background image. To compensate for camera motion, a combination of a feature-based motion estimator (Chapter 3 and 4) and a direct motion estimator (Section 5.2) is proposed. This combination couples the high accuracy of the direct motion estimation with the robustness to fast motion because of the feature-based estimator. The separation of object motion from the camera motion is carried out with a robust-estimation algorithm (RANSAC). This algorithm has been modified since the original algorithm does not reach the theoretically predicted performance.

Feature-based vs. direct estimation

In our experiments, the computation speed of the feature-based estimator reaches real-time execution, while the direct motion estimation is approximately a factor of ten slower. Hence, it would be interesting to explore if a comparable estimation accuracy can be obtained without the direct motion estimator. The reader should recall that while the frame-to-frame parameters of the feature-based estimator can be computed with good accuracy, the error accumulation, when concatenating the motion parameters to long-term frame-to-sprite parameters, reduces the final accuracy. A possible approach to prevent this drift could be to compute the feature-based motion parameters directly between the input frames and the sprite image. Our conjecture is that this approach would reach comparable accurate motion parameters at a computation speed comparable to the current feature-based estimator, i.e., real-time.

Robustness in difficult scenes

In our experiments, we have occasionally observed sequences, for which the feature-points are concentrated in one corner of the image. In this case, the estimation of motion parameters is of low numerical stability and usually results in skewing motion ($k \neq 0$ in Table 2.1).

Although this kind of motion is included in the projective motion model, it represents a physically impossible motion. In fact, compared to the eight free parameters of the projective motion model, there are only four varying physical parameters (camera rotation and focal length). Another

two parameters (principal point) are constant throughout the sequence. One approach to increase the robustness of the motion estimation may be to impose constraints or a parameterization that only permits physically possible motions.

15.1.2 Chapter 5: background estimation

In Chapter 5, a new algorithm for synthesizing a pure background image without foreground objects has been proposed. In contrast to previous algorithms, which usually estimate a statistical background model, we classify the content of each frame explicitly into foreground and background classes. This approach yields better results particularly in short sequences. For video segmentation, this is particularly important because the input is often a short scene from a longer movie for which the amount of input frames is fixed.

The proposed background-estimation algorithm divides the image into small blocks and determines for each block the periods of time, in which the image content is stable. Additionally, the pre-knowledge that background content is visible in comparable periods as in neighboring blocks is exploited to select those periods during which the background is visible. The central data-structure in this algorithm is the *similarity matrix* which collects information if the block content at two time instants is similar.

Alternative implementation with block-diagonal matrices

While our algorithm uses a combinatorial hill-climbing optimization to find the periods with high similarity of content, this problem can also be understood as resorting the matrix into block-diagonal form. An algorithm to bring a matrix to block-diagonal form has been described in [32] for the application of multi-object 3-D reconstruction. It will be interesting to compare both optimization approaches with respect to efficiency.

When the matrix is resorted to block-diagonal form, each block can be subsumed into one state. Exploiting again the spatial coherency between blocks, a Markov random field can be applied to determine the background label for each block.

15.1.3 Chapter 6: multi-sprites

The projective motion model, which is commonly used to describe rotational camera motion, assumes that the scene can be projected on a flat plane. As shown in Chapter 6, this does not work for large rotation angles. The proposed multi-sprite technique makes it possible for the first time to

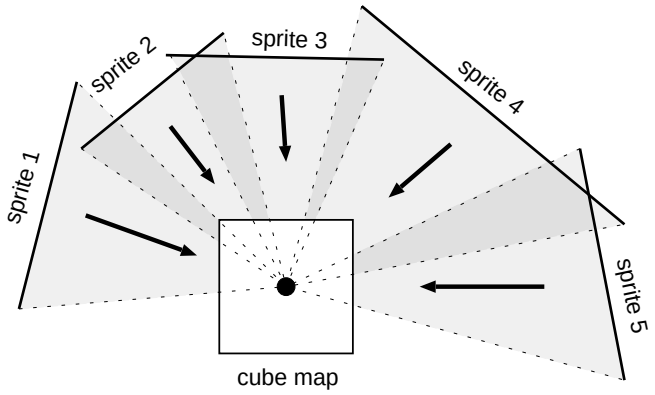


Figure 15.1: *When the 3-D position of the sprite planes is known, the texture can be projected onto a cube-map.*

cover scene backgrounds recorded by arbitrary rotational camera motion. Rather than of projecting the background scene onto a single plane, the scene is now covered by a collection of planes. Note that although we know the homography between these planes, their 3-D positions in space are unknown. However, these 3-D positions can be computed when the multi-sprite technique is combined with the camera autocalibration described in Chapter 12.

Cube maps

A further possible model for scene backgrounds, which is popular in computer graphics, is the *cube map* model. The cube map is a virtual cube that encloses the camera. The background scene is projected onto the faces of the cube, resulting in six square texture images. Compared to a spherical model, the cube map has the advantage that the homography transform can be reused and no transcendental functions are required. Compared to the multi-sprite representation, it has the advantage that it is easier to work with, since the number of planes and their position is *a-priori* known, and because simple homographies can be used as the transforms. On the other hand, unlike the multi-sprite model, the cube map cannot adapt to different resolutions that may be required to prevent a loss of detail (e.g., the resolution varies during a camera zoom). Moreover, a cube map has always a constant size, covering every viewing direction, independent of the actual camera motion, which can also lead to inefficient storage.

Note that it is not possible to estimate the textures of a cube map directly from the input sequence, because it requires that the camera pa-

rameters are available in absolute physical units. However, we can start with a multi-sprite model onto which the camera autocalibration is carried out to determine the position of the multi-sprite planes in 3-D space. After this, the multi-sprite representation can be converted to cube map textures (see Fig. 15.1).

15.1.4 Chapter 7: background subtraction

The proposed video-object segmentation is based on an extended background-subtraction algorithm. This algorithm compares the input image with the background image and identifies the changed areas as foreground. Compared to previously proposed background-subtraction algorithms, we have added the concept of risk maps to prevent segmentation errors that are due to misalignment of the input image to the background.

Null-hypothesis vs. object models

In future research, the foreground-object segmentation can be enhanced in several simple or more fundamental ways. Currently, the segmentation is carried out independently for each input frame, but improvements might be obtained by considering several frames at once. This can be modeled, e.g., with a three-dimensional Markov random field that enforces temporal stability. This concept can even be extended with a local object-motion compensation to connect corresponding pixels along the time axis.

A more fundamental change is to redefine the definition of changed pixels. In the current system, pixels are compared only to a null-hypothesis for an unchanged pixel. An alternative formulation would be to also have a model for the foreground, so that the choice for the best-fitting hypothesis could be made. However, the main difficulty of the two-hypotheses approach is to obtain an object model, especially for non-rigid objects.

15.1.5 Chapter 9 and 10: graph-based object models

An essential problem in segmentation is to define the object that is intended to be extracted. We have approached this problem by employing user-defined object models to identify specific objects in the input. A major design decision is the choice of representation that should be used for the object models. We decided to use a graph-based model in which nodes represent image regions and edges represent spatial proximity. The advantage of this model is that it allows articulated object motion. Moreover, since we restricted the graph to tree structures, an efficient implementation of the object detection is possible with a dynamic-programming approach.

Object-model matching

A principal difficulty in applying object models is the definition of a robust cost function for object localization. Our cost function is a combination of region-color differences, region shapes, object deformation, and other terms (see Eqs. (9.4), (9.5) or (10.9)). Note that an optimal weighting of these cost contributions cannot easily be derived. Even the sensible definition of each single cost contribution is difficult. Hence, the definition of a suitable model is always a compromise between a robust detection and accurate placement. More insight into this problem can probably be obtained with a further exploration how the human visual system perceives its environment.

Object models in the segmentation process

Because the adopted object model allows for some variability, the detected object position does not cover the real object area accurately. In order to extract detailed object boundaries, the object detection should be combined with other techniques that operate at the pixel level. In our two proposals in Chapter 9 and 10, we combined the object detection with a color segmentation. The primary difference between our two proposals is that for the object detection in cartoon sequences, color segmentation is carried out prior to object detection, whereas for natural video sequences, color segmentation follows the object detection.

In the object-detection system for natural video, we have observed in our experiments that for textured objects, the object boundaries cannot be accurately found. For example, in the results shown in Figures 10.11 and 10.12, it is visible that segmentation borders along uniformly-colored regions are acceptable, whereas the borders at textured regions are poorly segmented (they appear fringy). The reason is that the color segmentation splits textured areas into a multitude of small regions. Many of them are not covered by the object model and consequently, they are excluded from the object mask. More accurate object boundaries have been obtained with change-detection masks (CDMs), which work particularly well if the object is textured. The disadvantage of CDMs is that they can only detect objects if there is a significant difference to the background or to a preceding frame (see Fig. 8.14).

Considering these properties of the algorithms, we can propose a promising object-detection system for further research. The framework of this system is depicted in Figure 15.2. This segmentation system combines the following two principal techniques.

- Depicted at the left side is a change-detection algorithm, similar to the

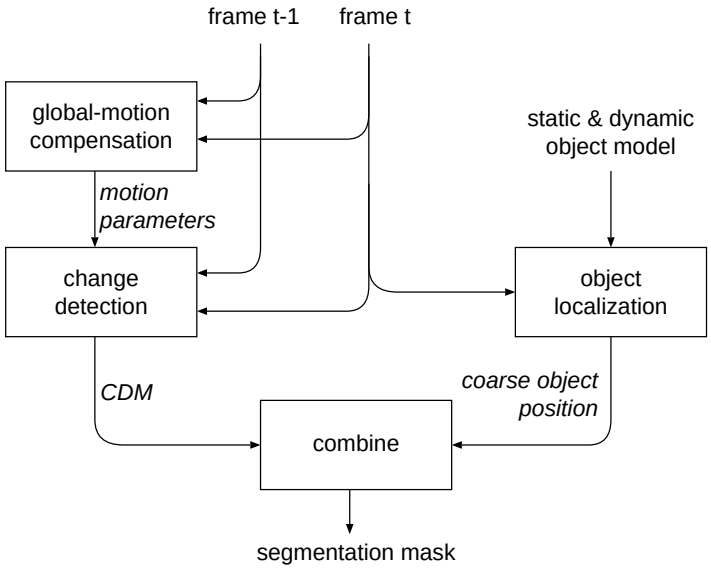


Figure 15.2: *Proposal for a low-latency real-time segmentation system.*

system described in Part I of the thesis. However, a major difference is that the change detection is not based on the comparison to a scene-background image, but to the previous frame. Note that this means that all steps of the background synthetization can be omitted, but the global-motion estimation is still required to compensate the camera motion of successive frames.

- The remaining part of the system comprises an object-detection algorithm as described in Chapter 9 and 10. The results of both techniques should subsequently be combined such that the object boundary is derived from the CDM and the interior of the object is filled from the detected object position.

This new alternative segmentation system would have the advantage that it enables a real-time segmentation with very low delay. Because no background-sprite is generated, the long latency introduced by this step vanishes. On the other hand, the segmentation relies to a large extent on the object-model detection, since the CDM is only computed between successive frames, leading to incomplete segmentation masks. To achieve a real-time execution speed and a high temporal consistency, we suppose that the object model should be further extended with a dynamic model, describing its possible motions.

15.1.6 Chapter 11: Corridor Scissors and circular paths

Chapter 11 has approached the segmentation problem with a semi-automatic segmentation technique. The Intelligent Scissors algorithm has been extended to a Corridor Scissors tool, in which the user coarsely marks the outline of the object with a broad corridor. Afterwards, the algorithm extracts the accurate object contour within the corridor and tracks this contour through the successive images. The elegancy of the approach is that the manual segmentation and the tracking step are both based on the same core algorithm, computing shortest circular paths in graphs. Both computation processes differ only in the cost definition for the path computation. In the manual segmentation, a cost based on the image gradient is used. For the tracking, the texture along a previously-segmented contour is compared with the current image.

The core of the Corrisor Scissors tool is a new algorithm for computing shortest circular paths in planar graphs. Even though our shortest circular-path algorithm only guarantees to find the optimum for planar graphs, it yields a good approximation even for *almost* planar graphs like grid graphs with nodes connected to their eight-neighborhood. The approximation is better when the width of the corridor is small compared to the cycle length. In fact, we use the approximation for non-planar graphs in the tracking step. Since the typical cycle length is about 100 times longer than the corridor width, the optimal solution is usually obtained for the non-planar graph in the tracking step.

15.1.7 Chapter 12: physical camera-parameter extraction

Part III of the thesis commenced with Chapter 12 by describing an algorithm to convert the projective motion parameters into the physically meaningful absolute camera rotation-angles and the focal length. The difference to previous camera-autocalibration algorithms is that the input data comprises the projective motion parameters as they are stored for example in the MPEG-4 sprite-motion parameters or the MPEG-7 camera-motion descriptors. This enables to obtain the physical parameters of coded motion data without access to the original image data. Note that this is not possible with previous autocalibration algorithms since they are usually based on a bundle-adjustment technique, applied to the detected feature-points. Another speciality of our algorithm is that it also applies the multi-sprite technique to enable the processing of unrestricted rotational camera motion.

The current algorithm applies a global optimization over all available frames. This leads to a high-accuracy calibration, but it introduces long

processing delays. An interesting future research topic is to find an online algorithm that does not introduce large delays. The main problem is that a significant change of rotation angle is required to obtain the parameters with sufficient accuracy. Especially the estimation of the focal length is numerically unstable for small rotation angles.

15.1.8 Chapter 13: camera calibration for sport videos

A different kind of physical-parameter estimation is described in Chapter 13. For applications like sport-video analysis, the position of the players on the screen has to be translated into coordinates in the real-world in order to derive a semantic meaning. For this special application, we have proposed a new algorithm that employs a model of the arrangement of lines on the playing field to establish a correspondence between playing field lines in the image and the position of lines in the real-world. The advantage of our algorithm is that it is not affected by secondary features, such as the court color, varying illumination, or even significant occlusions.

Calibration with insufficient visual markers

A remaining limitation of the calibration is that, similar to the global-motion estimation, at least four line-correspondences have to be established. In situations where only a small part of the playing field is visible (e.g., only a T-junction of two lines on a soccer field), insufficient information is available for a calibration. This is a fundamental problem of any calibration algorithm and it can only be solved by further restricting the camera model. A full set of camera parameters involves seven parameters: three rotation angles, three for the camera position, and the focal length. However, the cameras are usually mounted at a fixed position, such that only four parameters remain per frame. Additionally, three parameters for the position are unknown but constant during the sequence. It is an interesting topic to investigate if a camera calibration can be developed that uses the restricted set of parameters when insufficient are available. In fact, Hayet *et al.* recently proposed an extension to our calibration algorithm that further restricts the number of camera parameters [88]. Additionally, a dynamic model for camera motion could be employed to extrapolate the camera motion, in case that insufficient calibration information can be obtained from the image.

15.1.9 Chapter 14: floor plans from panoramic images

Panoramic background images from rotating cameras are usually visualized in cylindrical panoramic images. This is inconvenient for the viewer since the image shows the scene background from any direction around the camera at once. To provide a better orientation for the viewer, a new algorithm was described in Chapter 14. This algorithm reconstructs the shape of rooms or even complete floor plans from the panoramic images, in which the user has previously marked the room corners.

Note that the floor-plan reconstruction is conceptually comparable to the camera calibration for sport sequences. In both approaches, a model of the observed object is used to connect the real-world geometry with the observed images. The difference is that for the analysis of sport videos, the model was fixed, whereas for the floor-plan reconstruction, the camera positions and the shape of the model are both estimated simultaneously.

15.2 Explicit vs. implicit models

In this thesis, several approaches for video-object segmentation have been presented. Common to most of these techniques is that they apply explicit models for describing objects or the camera. Even in the generic segmentation system described in the first part of the thesis, the objects are effectively defined with a *negative model*: everything that differs from the background should be foreground. Hence, the background synthetization defines the foreground objects. Since the background is derived from the video itself, the approach works only if there is enough video data available to construct a correct background model. Switching from a background model that is derived from the sequence itself to an explicitly-defined model, like the graph-based object models, enables a successful segmentation on much shorter videos or even single frames, since no model information has to be derived from the input video.

The choice for a particular approach depends on the typical video content. For a surveillance-type input, the segmentation system proposed in Part I proves already very robust. In more general video sequences like movies, this approach is not satisfactory since single scenes are too limited in length to derive suitable background models. Moreover, movie sequences often comprise non-rotational camera motion, such that a background sprite cannot be created.

To enable video-object segmentation for general video sequences, we consider two techniques to be of principal importance. First, we require a generalized background model that also supports translatorial camera motion. Probably, this means that a true 3-D model of the background is

required. Second, we need a good workable definition of an object model in order to extract objects in scenes with incomplete background information. The problem of defining a suitable object model is probably the most ambitious in segmentation.

15.3 Future of segmentation

As elaborated in the introduction of the thesis, there are numerous applications that can be enhanced with segmentation. On the other hand, segmentation will remain a challenging problem for the foreseeable future, and it is not clear yet whether it can ever be considered solved. Currently, it appears more probable that instead of a general solution to the segmentation problem, a variety of specialized algorithms will be developed for specific applications. It should also be considered that many applications do not require exact segmentation masks, which might be too difficult to compute. Instead, the detection of video-objects and extraction of some of their features can often be sufficient (e.g., the motion path of players and the ball in a sport game is usually sufficient for the analysis). The most important application requiring accurate segmentation masks are video-editing applications, but for these applications, the requirements are so high that segmentation will probably stay a semi-automatic tool to ease the editing (Intelligent Scissors tools, alpha-channel estimation [26], or reconstruction of 3-D models from still pictures, like for the preservation of cultural heritage).

Many applications, like video-archive databases and compression systems, have to deal with general video sequences, for which an automatic segmentation does not seem tractable. However, for these systems, an accurate segmentation is also less important, as it is only a tool to support high-level database queries or enable higher compression factors. In this sense, segmentation-like techniques can be applied for the analysis, but to yield a high robustness, the system should not depend on an error-free segmentation. For example, in 3-D compression systems, the separation into independently-coded objects can help in interpolating new views, but there are too many special cases (e.g., reflecting surfaces) that should be processed with content non-adaptive techniques, as one can never rely on a successful segmentation. In video databases, special object-detection algorithms for specific frequently-occuring queries (e.g., face-recognition) might replace a generic segmentation-based analysis.

For applications concentrating on sequences of a specific domain, like specialized video databases, content analysis, or medical image-processing, more detailed object models will become increasingly important. In the

near future, there will also be an increasing merge between image-analysis techniques and computer-generated graphics or visualizations. Early applications are appearing in the movie industry (e.g., motion capture) and sports analysis for augmented event-visualization.

Part IV

Appendices

There's no sense in being precise when you don't even know what you're talking about. (John von Neumann)

Appendix A

Video-Summarization with Scene Preknowledge

A.1 Introduction

For long video sequences, it is not easy to quickly get an overview of the content. Usually, this means browsing through the video in a fast forward mode to save time. To relieve the user from this searching task, algorithms [115, 104, 78] have been developed to automatically generate video abstracts, which are short videos extracted from the original to give a good overview about its content. However, often we desire to have a static overview of the video either to print it or to present a list of video sequences that are available in a data-base. In this case, the video content can be summarized with a collection of representative key-frames.

One approach to extract key-frames is to apply a cut detection to separate the video into a large number of shots. Key-frames are obtained by choosing a representative frame of each shot (see [22]). The disadvantage of this approach is that errors in the cut detection are propagated into the key-frame selection process. Furthermore, the number of key-frames is directly coupled to the number of shots and cannot be adjusted by the user. Even if the content changes much within a single shot (consider a camera pan), only a single key-frame is extracted for the shot.

A second technique to find a good set of key-frames is to extract a feature-vector that describes the image content in a low dimensional space.

The features should be chosen such that images from the same scene are mapped to similar feature-vectors. Usually, some kind of color histogram is used to describe the rough image content. Feature-vectors belonging to the same scene can be grouped with clustering algorithms. Finally, some of the feature-vectors are selected as key-frames.

Various work has been carried out to find key-frames which show especially important scenes, or which are very charateristic for the input video. However, the evaluation, which scenes are representative is very subjective and on a high semantic level, that cannot easily be automatized. In fact, our experiments have shown that it is difficult to distinguish summaries that were generated with sophisticated algorithms from summaries that simply take pictures randomly or at regular intervals.

However, there are four kinds of key-frames that should be avoided, since they show an obviously bad selection of images:

- **Repetitions.** Several images that show almost similar input images. These do not provide any new information. Usually, repetitions can be avoided by a well designed clustering process.

- **Cross-fades, wipes.** The transition between scenes is usually editied as a short cross-fade between the two scenes. Pictures from these transitions show a mixture of two scenes and are not meaningful when viewed statically. Unfortunately, these frames are often selected in a clustering process, because their feature-vector is clearly different from both scenes and as such, the algorithm treats the transition like a scene of its own.

- **Black or white frames.** Sometimes there are ranges where the video is simply black or white to have a pause between two scenes. These kind of pictures with no information should be excluded from the summary. This class of undesired images also includes pictures that are too bright because of some flashlights.

- **Uninteresting scenes.** For some applications, it may be known beforehand that some of the scenes are not essential and they should be excluded from the summary. One example are scenes showing the news-speaker or the weather chart in news broadcasts, while we are mainly interested in the reports in between.

This appendix presents a new clustering-based algorithm, providing a solution to these problems. The problem of frames selected out of transitions is solved by a two-stage clustering process. The first stage provides a soft form of shot separation, determining periods of stable image content

with good key-frame candidates. These candidates are then used in the second clustering stage to select the final set of key-frames. Image content which is not desired in the summary can be explicitly excluded by providing domain-knowledge in form of sample images of shots to exclude. This aspect is solved by modifying the clustering step to circumvent the building of clusters for these shots.

A.2 Summarization algorithm

Our algorithm is composed of three steps which operate from low-level features to semantically more meaningful data structures. As a first step, a small set of features is extracted from each input frame. These feature vectors are subsequently used to determine similarity between frames. The second step then groups time consecutive feature vectors to small *segments*. We define a segment as a short period in the video sequence (usually even smaller than a shot) such that the content in the segment is as static as possible. More specifically, no cut should be present in a segment. The third step combines the segments to clusters such that as much as possible of the input video content is covered by the clusters. Domain-knowledge is integrated by inhibiting the building of clusters in areas of the feature-space which are known to be irrelevant.

A.2.1 Feature extraction

For each input frame n, a feature vector f_n is extracted. The subsequent steps of our algorithm work on arbitrary feature vectors. Thus, a variety of features can be used, provided that an appropriate distance measure $||f_a; f_b||$ can be defined which corresponds to visual similarity.

For our implementation, we have chosen to use quantized luminance histograms as feature vectors $f_n = (h_1, ..., h_m)$. We are using two different distance measures for the segment positioning and clustering steps. In segment positioning, the *sum of absolute difference* measure (SAD) is used, which is defined as

$$||f_a; f_b||_{SAD} = \sum_{i=1}^{m} |f_a(i) - f_b(i)|. \tag{A.1}$$

For the clustering step, the *Earth-Mover's Distance* (EMD) is used. This measure has been used by several authors in the context of image retrieval from large databases; see [160] for an in-depth description. In the one-

dimensional case, the EMD can be determined efficiently as

$$||f_a; f_b||_{EMD} = \sum_{x=1}^{m} \Big| \sum_{i=1}^{x} f_a(i) - f_b(i) \Big|. \tag{A.2}$$

The reason for using two different distance measures is that the two steps operate on different time-scales. In short periods of time, the image contents varies less. Hence, the more sensitive SAD measure is used to accurately segment slow transitions. From a global perspective, shots with large distances in time can show large differences even when the semantic content is comparable. Therefore, the more liberal EMD is more appropriate for the high-level clustering step.

A.2.2 Determining segment boundaries

Video sequences may contain gradual transition effects like fades and wipes between shots. It is desired that video frames from these transitions are not present in the final video summary. However, when two subsequent shots are grouped into the same cluster, clustering algorithms usually tend to select the transition frames as cluster centers because these features are mixtures of the features from both shots. To avoid the selection of those frames, we split the input video into short segments of about 4 seconds length. The objective of the first clustering step is to position the segment boundaries such that they are favourably positioned at cuts and within transitions. The intention is to obtain small segments of video with almost homogeneous content (see Fig. A.1). Consequently, the frame in the middle of each segment will be a good key-frame candidate.

Our algorithm for positioning the segment boundaries is motivated by the time-constrained clustering technique described in [153]. However, that paper used the clustering technique to generate hierarchical summaries of existing key-frames, whereas we are using it for the low-level placement of segment boundaries.

Let p_i be the first frame of segment i. To determine the positions p_i, the total sum of inhomogeneity over all N segments is minimized. The

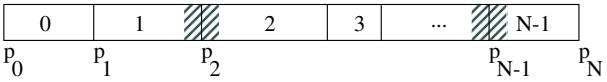


Figure A.1: *The input video is divided into a large number of segments (N). Segment boundaries are positioned such that they coincide with cuts or that they are placed in the middle of transitions between shots (shown as shaded areas).*

inhomogeneity of a segment i is determined by summing up the distances between all frames in the segment and the mean feature vector s_i of the segment with

$$s_i = \frac{1}{p_{i+1} - p_i} \sum_{n \in [p_i \ ; \ p_{i+1})} f_n. \tag{A.3}$$

The positions of the segment boundaries p_i are chosen to minimize the total segment inhomogeneities:

$$\min_{p_1,\ldots,p_{N-1}} \underbrace{\sum_{i \in [0;N)} \underbrace{\sum_{n \in [p_i \ ; \ p_{i+1})} ||s_i; f_n||_{SAD}}_{\text{inhomogeneity of segment } i}}_{\text{minimized over all segments}}. \tag{A.4}$$

If there are enough segments available, the above optimization will place segment boundaries into transitions between shots. In the usual case in which many more segments are available than shots, long shots will be split into several segments. According to the optimization criterion, the segment length will depend on the amount of change in the video. Static parts will be assigned longer segments, while fast changing parts will be split into shorter segments.

Since the computational complexity of an exact optimization would be too high for practical implementations, we are using a time-continuous variant of the k-means algorithm for optimization. The algorithm approaches the global optimum by performing many local optimizations as follows (see Fig. A.2):

1. Distribute p_i equally spaced over the full length of the video;

2. for all pairs of adjacent segments $[p_{i-1} \ ; \ p_i)$ and $[p_i \ ; \ p_{i+1})$ set p_i to

$$\operatorname*{argmin}_{p_i} \underbrace{\sum_{n \in [p_{i-1} \ ; \ p_i)} ||s_{i-1}; f_n||_{SAD}}_{\text{left segment inhomogeneity}} + \underbrace{\sum_{n \in [p_i \ ; \ p_{i+1})} ||s_i; f_n||_{SAD}}_{\text{right segment inhomogeneity}};$$

3. repeat step 2 until convergence is reached.

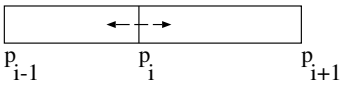


Figure A.2: *Local optimization step. The boundary between two adjacent segments is moved to a position such that the homogeneity of the segments on both sides is maximized.*

Usually, the solution found does not exactly correspond to the global optimum, however at cuts, the segment boundaries are positioned reliably between shots. This property makes the solution sufficient for later steps of the algorithm. From each segment i, the frame at the middle of the segment (at position $m_i = (p_i + p_{i+1})/2$) is taken as the representative frame of that segment and as a later key-frame candidate. Further processing steps operate only on the feature vectors $K = \{k_i = f_{m_i}\}$, leading to a significant reduction of computation time compared to clustering algorithms using all input frames.

A.2.3 Clustering

Our clustering algorithm is based on the approach described in [134, 38], which is outlined in the following. Differing from the algorithm described in the literature, we are using the EMD distance as clustering criterion which results in perceptually more reasonable clusters. Moreover, we observed that an arbitrary initialization of cluster centers sometimes results in bad convergence. Hence, we perform a gradual increase of the number of cluster centers and initiate new centers into areas where new clusters are most likely to be found.

The basic principle is to find a predetermined number of clusters (corresponding to the number of key-frames) such that the dissimilarity of the frames in each cluster is minimized. To define this more formally, let c_i be the set of M cluster centers and let $K = \{k_i\}$ be the set of key-frame candidates extracted in the last step of the algorithm. For each cluster center, we define its neighbourhood N_{c_i} as:

$$N_{c_i} = \left\{ k \in K \;\middle|\; \forall j : \; ||k; c_i||_{EMD} \leq ||k; c_j||_{EMD} \right\}, \tag{A.5}$$

meaning that each feature vector is assigned to the neighbourhood of the nearest cluster-center (according to the EMD-distance).

We say that a set of cluster centers is optimal iff they fulfill

$$\min_{c_0,\ldots,c_{M-1}} \underbrace{\sum_{i\in[0;M)} \underbrace{\sum_{s\in N_{c_i}} ||s\ ;\ c_i||_{EMD}}_{\text{dissimilarity of cluster } i}}_{\text{summed over all clusters}}. \tag{A.6}$$

The clustering is carried out by a k-means algorithm without the time-consecutiveness constraint used in the last step. Our experiments have shown that the k-means algorithm works best when the initial cluster centers are not chosen randomly, but rather added one at a time. Each new cluster center is initialized to the k_i with the largest distance to any existing cluster center. The overall clustering algorithm can be summarized as:

1. Set c_1 to a random k_i (e.g. k_1), set $n = 1$;
2. determine the neighbourhood N_{c_i} for all clusters;
3. reassign c_i to $c_i := \frac{1}{|N_{c_i}|} \sum_{k\in N_{c_i}} k$;
4. continue at step 2 until convergence is reached;
5. if $n = M$ stop the algorithm, else set $n := n + 1$, set $c_n = \text{argmax}_{k_i} \min_{c_j} ||k_i; c_j||_{EMD}$ and continue at step 2.

For each cluster obtained from the last step, the k_i which is nearest to the cluster center is selected as a key-frame. The selected k_i are sorted to the correct temporal order, and the input frames corresponding to the k_i are composed to the final summary.

A.2.4 Integration of domain-knowledge

In this section, we modify the clustering step from the last section to insert domain-knowledge about irrelevant video scenes. To prevent these scenes from occuring in the summary, we compute the feature vectors for all scenes to be excluded. These feature vectors can also be provided by the user if he detects an image in the summary that he wants to exclude. After feeding this information back into the algorithm, a new summary can be computed without the undesired scenes. After a period of interactivity with the user, the classes of scenes to be excluded are known to the system, and summaries will only contain the desired scenes.

Exclusion of the scenes is accomplished by introducing the feature vectors u_i of uninteresting scenes as additional cluster centers (see Figure A.3).

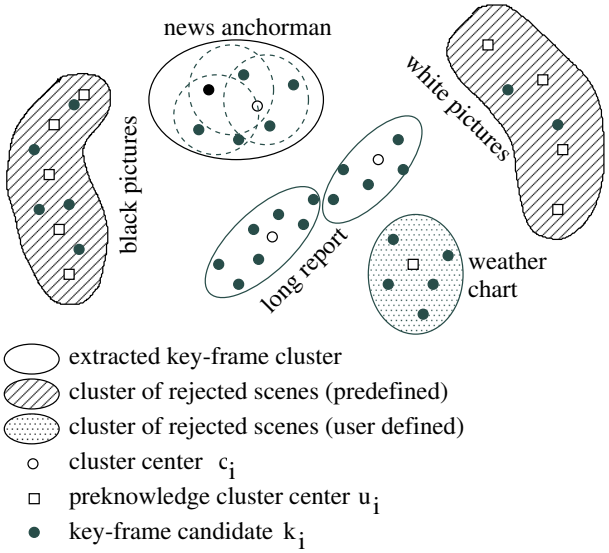
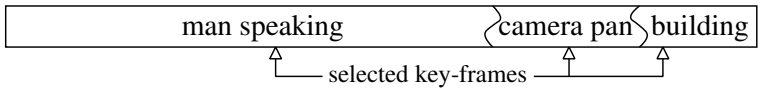


Figure A.3: *Schematic example of the clustering process with integrated domain-knowledge. The lengthy report scene is divided into two separate clusters while the repeated anchorman scenes are combined into a single cluster. Black and white frames are removed by the predefined clustering centers. User defined domain-knowledge has been applied by adding a cluster-center to remove the weather chart scene from the summary.*

They are treated the same as the c_i with the exception that the position of u_i is fixed and that no key-frames will be generated for their clusters. The consequence in the clustering process is that the u_i centers grab the feature vectors of scenes near the u_i vectors. These vectors will have no influence on the clustering because they are contained in the neighbourhood of a vector u_i. The total number of generated key-frames will remain the same.

A.3 Evaluation

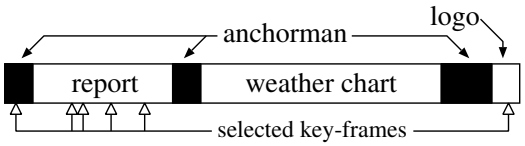
We demonstrate the behaviour of our algorithm with two test sequences. The first is the well-known *Foreman* sequence. This sequence is 400 frames long and contains no cuts. Its plot is depicted in Figure A.4(a). After showing the speaking man for over half of the sequence, the camera pans to the right and shows a building. Note that even though there are no cuts, our algorithm finds the three most important parts in the video. Algorithms that are based on cut detection fail on this sequence.



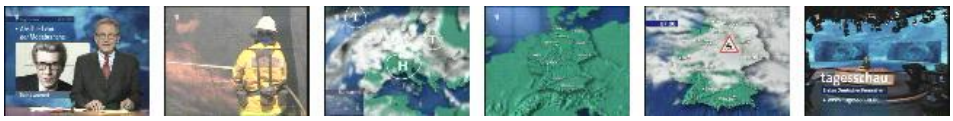
(a) Test sequence plot.

(b) Summary (three key-frames).

Figure A.4: *Summary of the Foreman sequence.*



(a) Test sequence plot.



(b) Without domain-knowledge: 1× anchorman, 1× report, 3× weather chart, 1× logo.

(c) With knowledge to ignore weather chart: 1× anchorman, 4× report, 1× logo.

Figure A.5: *Summary of the last two minutes of a news broadcast.*

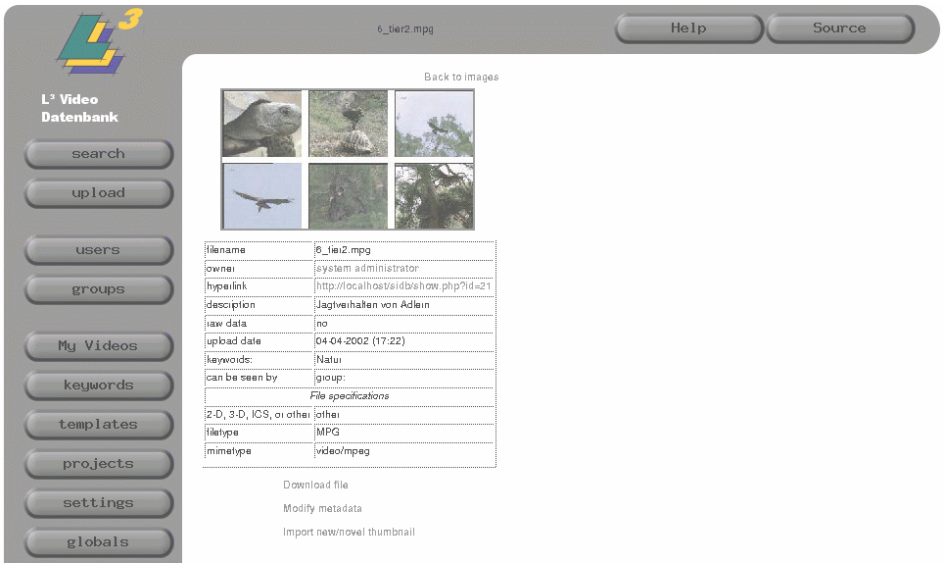
The second video contains the last two minutes of a news broadcast. Again, the video plot is shown in Figure A.5(a). First, we started our algorithm without any domain-knowledge (Figure A.5(b)). Note that the news anchorman only appears once in the summary even though he appears three times in the input sequence. Since the weather chart contains very different image content, the summary contains three pictures of it but only one picture of the preceding report. Let us now suppose that we are not interested in the weather chart. So we provided some pictures of the chart as domain-knowledge in a succeeding experiment and restarted the algorithm (Fig. A.5(c)). All key-frames of the weather-chart were removed, and additionally, more meaningful key-frames of the news report were generated.

A.4 Conclusions

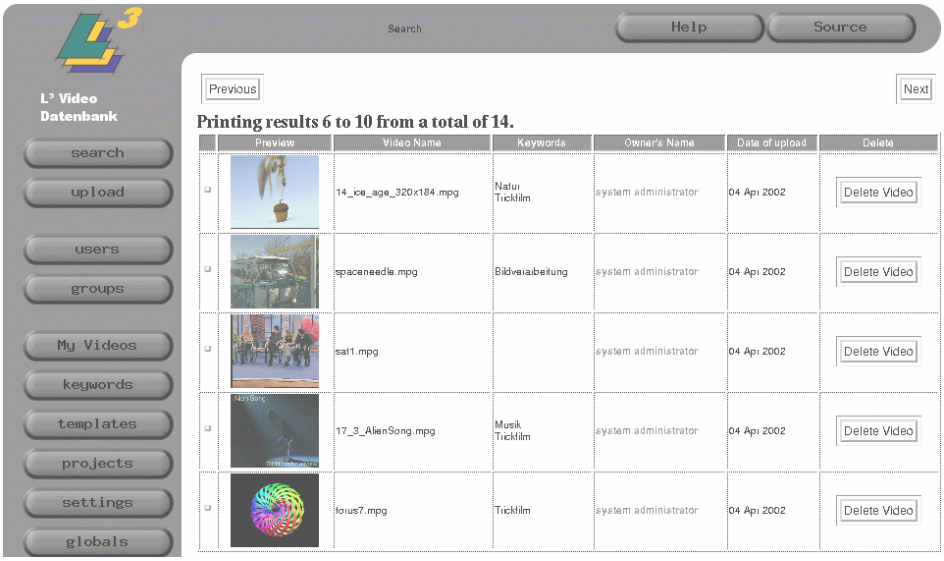
We have described a new algorithm for automatic generation of video summaries. User domain-knowledge about the video-content can be provided to improve the quality of the generated summary. We consider the approach of a modified clustering step superior to specialized filters for excluding undesired frames because our generic approach can be adapted to new application areas by simple user interaction. A topic of further research may be to integrate an algorithm for automatic detection of irrelevant feature-vectors.

Finally, the fact that our algorithm does not depend on an accurate cut detection algorithm (known to have difficulties with soft cuts) increases robustness and enables the summarization of video material without scene changes.

Our algorithm has been integrated into the video-database of the L^3 project (LifeLong Learning) for learning-videos (see Figure A.6) and the ECHO project (European CHronicles On-line) to generate abstracts for four major national audio-visual archives (Italy, France, the Netherlands, Switzerland). The L^3 video-database application is based on the open-source *Scientific Image Data-Base* (SIDB) project, which we extended for video-sequences.



(a) Summary display of a video sequence.



(b) List of available sequences in the data-base.

Figure A.6: *Integration of the video abstracting algorithm in a web-based video database application.*