Volume 0 (1981), Number 0 pp. 1–11

Editing Object Behavior in Video Sequences

Volker Scholz[1], Sascha El-Abed[1], Marcus Magnor[2] and Hans-Peter Seidel[1]

[1]MPI Informatik Saarbrücken, Germany
[2]TU Braunschweig, Germany

Abstract

While there are various commercial-strength editing tools available today for still images, object-based manipulation of real-world video footage is still a challenging problem. In this system paper, we present a framework for interactive video editing. Our focus is on footage from a single, conventional video camera. By relying on spatio-temporal editing techniques operating on the video cube, we do not need to recover 3D scene geometry. Our framework is capable of removing and inserting objects, object motion editing, non-rigid object deformations, keyframe interpolation, as well as emulating camera motion. We demonstrate how movie shots with moderate complexity can be persuasively modified during post-processing.

Categories and Subject Descriptors (according to ACM CCS): I.3.4 [Computer Graphics]: Graphics Utilities Graphics Editors I.4.8 [Image Processing and Computer Vision]: Scene Analysis Time-varying imagery

1. Introduction

Digitally retouching photographs has become routine in the publishing industry. For still images, a number of professional editing products such as Photoshop exist, offering a wide variety of different manipulation techniques that leave little to ask for [BC02]. In contrast, retouching video recordings is still a tedious per-frame editing procedure. Existing software tools that modify the content of video footage, like Adobe's After Effects and Apple's Shake, offer tools for matte extraction [CAC*02] and rotoscoping [AHSS04]. Tools for object segmentation do not address the challenges of altering the motion and shape of objects in video footage [WTXC04, WBC*05, LSS05, CCBK06]. Building a practical, interactive system for these advanced retouching operations is still a challenge, since video inpainting techniques are computationally expensive. The potential applications of a powerful video editing framework are, nevertheless, imposing.

In this paper we address the challenge of convincingly altering the content of real-world video footage. A tool to post-process movie sequences enables removing any accidentally visible crew member or equipment. Another common flaw of many movies is missing continuity between film shots. Such annoyances are easily eliminated if scene objects can still be rearranged during post-processing. Finally, a powerful video post-processing framework gives the movie director additional artistic freedom to tell the visual story of a film during editing.

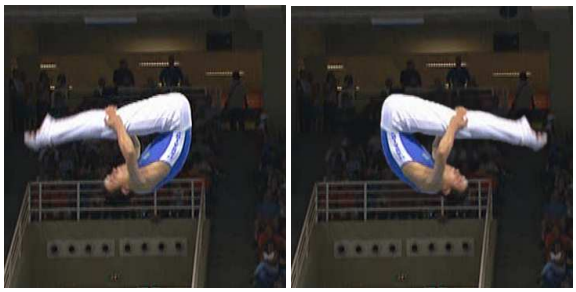

Figure 1: *Trampoline sequence. The rotation of the athlete was reversed and the missing background was reconstructed.*

To enable sophisticated video editing operations on general, real-world footage, our processing framework consists of several modules. Instead of attempting to augment real-world scenes with synthetic 3D models, we opt for editing spatio-temporal video objects. Video segmentation is needed to select the objects which the user wants to alter. To fill in the holes left behind by (re)moved objects, we rely on a video inpainting algorithm. Finally, compositing is applied

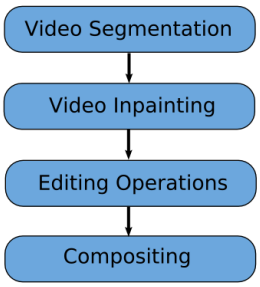

Figure 2: *System overview. The segmentation and editing steps are interactive, while inpainting and compositing are automatic.*

to authentically blend modified objects into inpainted video frames. Object shapes and motions are specified interactively at keyframes. Our video editing framework achieves visually convincing results while requiring only moderate user effort.

We present a practical end-to-end video processing system that offers a wide variety of object transformation effects to the user where missing background inpainting is performed automatically. We address the real-life problem of slightly modifying the behavior of video objects. Our goal is an easy-to-use interactive system that leaves the user in full control of the editing result. Technical contributions of our framework include

1. a complete video editing system, by using modified algorithms to make them practical in an interactive system,
2. a spatio-temporal background hole filling method that can handle fast camera motion,
3. a modified color-based video segmentation algorithm, including a boundary-editing user interface [LSTS04] extended to video,
4. a keyframe-based interpolation method for 2D object animation in video, and
5. various visual effects: editing of object trajectories, non-rigid deformation, object replacement and emulation of camera motion.

2. Related Work

An automatic approach for *video segmentation* is described by Wang et al. [WTXC04]. The mean-shift image segmentation method is extended to video and applied to pixels in $6D$ (x,y,t,r,g,b) space. Adaptive anisotropic kernels allow better feature extraction than previous isotropic kernels. The running time for a video sequence is on the order of several hours. Faster, more interactive systems have been recently proposed. Wang et al. [WBC*05] compute a presegmentation with a 2D mean-shift algorithm. A graph-cut based image segmentation algorithm is extended to video, with running times of a few seconds. Li et al. [LSS05] apply a 3D graph-cut segmentation algorithm to the spatio-temporal video cube. The result is refined with a 2D graph cut algorithm in localized windows around the objects border. As a final step, coherent matting is used to extract an alpha matte for the object. Our system is inspired by this work and proposes several important extensions (steerable presegmentation and a new interactive boundary editing tool). In contrast, automatic, learning-based methods which use color and motion cues produce good quality results [CCBK06] but require ground truth segmentation masks, which is not practical for our general-purpose editing tool.

There exists a large body of work on *texture synthesis* [WL00, EL99, EF01] and the closely related *image inpainting* problem [BSCB00]. Image inpainting propagates linear image structures (called isophotes) from a hole's circumference into the hole region by using a PDE-based method. It works well for small, smooth and low-textured regions. For larger missing regions or textured regions it may generate blurring artifacts. Exemplar-based texture synthesis fills unknown regions by copying image patches from the hole border. It is aimed at reproducing textural patterns but has problems with macrostructure in the image. Approaches that generate texture on a per-pixel basis [WL00, EL99] are computationally more expensive than patch-based methods [EF01]. Criminisi et al. [CPT03] showed how continuations of strong edges can be propagated inwards, which preserves simple structures. Our algorithm builds on the work by Criminisi et al.. We extend it to video and contribute two valuable improvements: weighted matching and patch blending. We also focus on a time-efficient implementation. Compared to the global optimization approach proposed by Wexler et al. [WSI04], our method is significantly faster. We can also handle fast camera motion with our method, while Patwardhan et al. [PSB05] and [WSI04] present results for a static camera only. In [PSB07], inpainting results for moderate camera motion and for moving objects that slightly change size are presented. Shiratori et al. [SMTK06] propose a method where color information is propagated into the hole area by using local motion fields. Periodic motion in the background can be recovered. To composite the edited object back into the video, a *matting* algorithm is needed to compute alpha masks. Techniques for image matting [LLW06, WAC07] are not directly applicable to video because temporal consistency is not taken care of. For video matting, various techniques exist [CAC*02, CCBK06]. We choose the robust border matting method for images [RKB04] and propose two modifications (a different color model and thin-plate spline regularization). *Keyframe animation* is a well-known technique from production systems like Autodesk's Maya. It offers the animator excellent control over the final motion and is used in high end production. For our general purpose video editor it is the ideal tool to specify object motion without having to consider motion laws from physics. Much work has been

done on interpolating keyframes. Relevant to our approach, Kochanek et al. introduced interpolating splines with local tension, continuity and bias control [KB84]. This technique gives the user much control over the final result. To reduce the number of parameters we use cubic spline interpolation.

Several *video editing* systems have been proposed. The Proscenium system [BM03] uses the video cube paradigm. The user can slice the cube with a cutting plane and distort or warp the video to facilitate alignment. Object-based operations are not part of their system. Wang et al. [WDAC06] present the cartoon animation filter, which takes an arbitrary input motion signal and modulates it in such a way that the output motion is more alive or animated. The goal of our system is to give the user more editing possibilities. Acha et al. [RAPLP05] manipulate the time flow of a video sequence by sweeping an evolving 'time front' surface through the video's aligned space-time volume. A variety of interesting video operations such as timing changes are demonstrated. The motion magnification approach was proposed by Liu et al. [LTF*05]. The input is a sequence of images from a stationary camera, the output is a re-rendered video sequence where the subtle motion of selected layers is amplified. Bhat et al. [BSHK04] propose a method for editing flow-like phenomena (water, smoke etc.) by user-specified flow lines. The Videoshop system proposed by Wang et al. [WXRA07] uses a gradient domain editing framework which allows e.g. seamless compositing operations by solving a spatiotemporal Poisson equation. Many of these approaches address specific editing problems and are only applicable to certain video sequences. Our system, in contrast, solves the more general problem of editing video object shape and motion in various ways also for the case when the camera is moving.

3. Overview

Our framework consists of several processing components (Fig. 2), which we describe in the following sections. The first task is interactive video segmentation for simple object selection (Sect. 4). As for still images, this interactive step is the key to any object editing operation. The second task is automatic video inpainting (Sect. 5), to fill in the holes in the video background that are left behind when the segmented objects are cut out and edited. In Sect. 6, we present the interactive editing operations of our framework, before in Sect. 7 we finally describe the compositing step. We show how our system can be applied to edit camera motion, to modify object trajectories, and to apply non-rigid deformations (Sect. 8), before we conclude with a brief outlook on future extensions.

4. Video Segmentation

In order to be able to separately edit individual objects in a video, the recorded video frames must first be segmented.

Figure 3: *Result of segmentation during preprocessing. Left: input image, right: segmentation result, each colored region is a superpixel.*

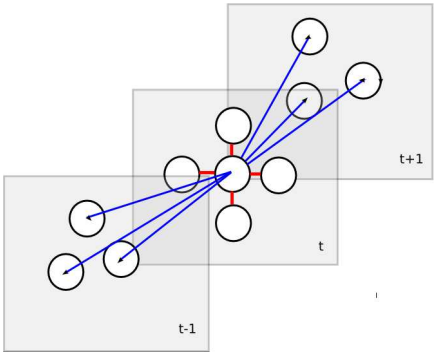

Figure 4: *3D graph for graphcut minimization. Every region is connected to neighbor regions within a frame (red edges E_1) and to neighbor regions in adjacent frames (blue edges E_2).*

Preprocessing

In the preprocessing step we use the image segmentation algorithm by [FH04] in a batch procedure to create regions of homogeneous color, so-called superpixels, for each video frame (Fig. 3). The main idea of this algorithm is to merge the most similar pixels first in a pixel graph data structure. As merging criterion the current graph edge which connects two regions is compared with the internal color variation of the regions. The merging threshold can be steered by a parameter, which controls the coarseness of the segmentation result. A detailed description can be found in [FH04]. This preprocessing reduces the amount of data for the subsequent steps and makes the following graphcut video segmentation feasible. Segmenting an image at VGA resolution takes a few seconds. Previous work [LSS05] used the watershed algorithm, which generates a larger amount of superpixels, or mean-shift segmentation [WBC*05], which requires more computational effort.

In order to group the computed 2D regions into 3D regions for video segmentation, we apply the same merging algorithm to a newly constructed 3D graph in a second step. Every superpixel region is connected with an edge to its

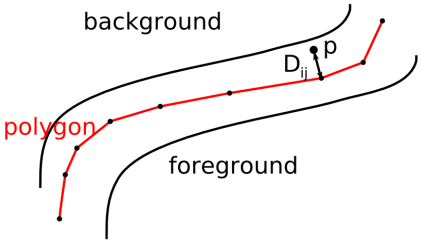

Figure 5: *User-defined polygon (red) in unknown region and distance D_{ij} of pixel p to polygon.*

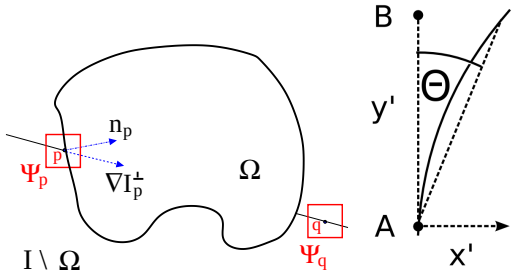

Figure 6: *Left: Schematic overview of inpainting terminology, see text for details. Right: bending parameters and the result obtained from bending AB.*

superpixel neighbors in the same frame. We determine region adjacency by a contour following algorithm [SA85] for every superpixel and the spatially overlapping regions in the adjacent frames (Fig. 4, blue edges between different frames). We use a search radius of typically 25 pixels for all examples to connect a region in frame t to its neighbors in frames $t-1$ and $t+1$. A graph edge for every overlapping pair, weighted with the squared RGB color difference of the two regions is added to the graph. Finally the region merging algorithm computes 3D regions from the 2D regions.

Min-cut Segmentation on Superpixels

After this data reduction step we apply a min-cut minimization [BJ01, LSS05] to the following energy function for all 3D regions $r_i \in V$ in the graph $G = (V, E_1 \cup E_2)$ (Fig. 4):

$$E = \sum_{r_i \in V} E_D(x_i) + \lambda \sum_{(r_i,r_j)\in E_1} E_N(x_i,x_j) + \mu \sum_{(r_i,r_j)\in E_2} E_N(x_i,x_j) \quad (1)$$

$E_D(x_i)$ is the color likelihood for region r_i, a measure of conformity with foreground/background color models and E_N the neighborhood energy. $x_i \in \{0,1\}$ denotes the background/foreground label of the region. Typical values for the neighborhood energy weights are $\lambda = 20$ and $\mu = 10$. This energy function is similar to [LSS05, WBC*05], but we use a different color model. We get color samples for foreground and background regions from user input and build two color models from them by clustering them with the k-means algorithm. When k is large enough (a typical value is $k = 64$) we get more robust classification results than with standard Gaussian mixture models (GMM), where often a fixed number of mixture components is used (like in [LSS05, WBC*05]). We also avoid convergence problems of the EM algorithm which is normally used to fit the GMM. The cluster centers K_k^F and K_k^B from the k-means clustering are then used to compute the minimum distance from a region color c_i to foreground clusters $d_i^F = min_k \|c_i - K_k^F\|$ and similarly for the background $d_i^B = min_k \|c_i - K_k^B\|$.

$$E_D(x_i = 0) = \frac{d_i^F}{d_i^F + d_i^B} \quad (2)$$

$$E_D(x_i = 1) = \frac{d_i^B}{d_i^F + d_i^B} \quad (3)$$

is the color likelihood E_D [LSTS04], a proximity measure to cluster centers normalized by the sum of distances to foreground and background clusters. $E_N(x_i,x_j) = g(C_{ij})$ is the neighborhood energy E_N between neighboring regions [LSTS04], where $C_{ij} = \|c_i - c_j\|^2$ is the squared RGB color difference of the regions r_i and r_j, and $g(x) = \frac{1}{1+x}$ is a weighting function. The user marks foreground and background regions with some paint strokes (markers) in keyframes which are treated as hard constraints, i.e. for these regions $E_D \in \{0, \infty\}$ [BJ01]. The marked pixels are also used as color samples for computing the color models. The min-cut algorithm [BK04] computes the labels x_i which minimize Eq. 1. The user can add additional strokes to refine the result, where only the color likelihood E_D of the affected regions is updated. By defining spatio-temporal slices of the video cube (x,y,t) [WBC*05] the user can also put markers in several video frames simultaneously which makes interaction more efficient.

Min-cut Refinement

To obtain fine segmentation on the pixel level, we employ the same graphcut technique in a corridor around the segmentation boundary obtained in the first step. A typical corridor width is 12 pixels. The min-cut graph for the corridor pixels is constructed with a 10-neighborhood (8 spatial and 2 temporal neighbors) and edge weights according to Eq. 1. The diagonal edge weights are multiplied with a factor $\frac{1}{\sqrt{2}}$ which gives smoother segmentation boundaries [BJ01]. Typical parameter values for this step are $\lambda = \mu = 0.1$.

Boundary Editing Tool

The color-based segmentation algorithm fails at low contrast edges and the min-cut minimization does not preserve

thin structures well. We provide two boundary editing tools for the user to correct these errors. First, the segmentation result is converted into boundary polygons with a contour following algorithm [SA85]. With the overriding brush as described in [LSTS04] the user can specify a corridor for the boundary with a paint stroke and min-cut optimization is used to find the actual boundary for one video frame. The neighborhood energy E_N is now defined differently. In addition to the color difference, it also uses the paint stroke (polygon) as a soft constraint in order to deal with low contrast edges. Similar to [LSTS04], the neighborhood energy for two pixels p_i, p_j with labels x_i, x_j is

$$E_N(x_i,x_j) = g((1-\beta)\cdot C_{ij} + \beta\cdot\eta\cdot g(D_{ij}^2)) \qquad (4)$$

g is the weight function introduced earlier, D_{ij} is the distance of the center of the edge (p_i, p_j) to the polygon (Fig. 5) and η a scaling factor (typical value $\eta = 10$). By varying β the user can control the influence of D_{ij}, a typical value is $\beta = 0.5$. We drop the color likelihood term E_D in Eq. 1 as it is often misleading if the automatic segmentation fails and reduce the energy equation to

$$E = \sum_{(x_i,x_j)\in E_1} E_N(x_i,x_j) \qquad (5)$$

which we minimize per video frame. The min-cut optimization is performed in a corridor around the whole segmentation boundary and only the paint stroke region is updated. We extend this image editing method to video. The user can put paint strokes at two different keyframes and the system interpolates the strokes linearly for the frames in between. We use a nearest neighbor matching algorithm to obtain vertex correspondences for the two polylines. The boundary refinement is then computed for all frames so that the user only needs to edit keyframes. The segmentation result is temporally smooth for the interpolated frames.

Thin structures are difficult to obtain by min-cut minimization, so we also provide a UI tool where the user can add vertices to the boundary polygons. The result of the video segmentation step is a binary alpha mask for each video frame.

5. Video Inpainting

We use two approaches for the hole filling problem: a texture synthesis algorithm and a mosaicking technique.

Image Inpainting Revisited

We start with the patch-based image inpainting algorithm proposed by [CPT03]. Given an image I, the pixels of the hole Ω to be filled are given by a binary mask. Let p be a pixel located on the hole's boundary, and Ψ_p be a squared patch centered at p. To guide the filling order, a priority value $P(p)$ is computed for each pixel p on the hole's boundary:

$$P(p) = C(p)\cdot D(p), \qquad (6)$$

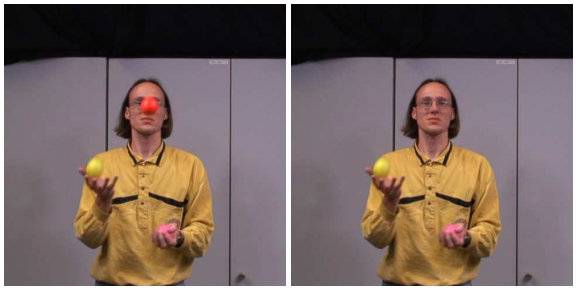

Figure 7: *Video inpainting results. Left: Original frame. Right: inpainted frame. The missing facial texture was reconstructed from other frames.*

where the *confidence* term $C(p)$ and the *data* term $D(p)$ are computed by

$$C(p) = \frac{\sum_{q\in\Psi_p\cap\bar{\Omega}} C(q)}{|\Psi_p|} \quad \text{and} \quad D(p) = \frac{|\nabla I_p^{\perp}\cdot n_p|}{\alpha}. \qquad (7)$$

$|\Psi_p|$ is the area of Ψ_p, α is a normalization factor, n_p is the unit vector orthogonal to the hole boundary at p and $\nabla I_p^{\perp}$ is the isophote direction at p, see also Fig. 6 (left). $C(p)$ is initialized to $C(p) = 0$ for $p \in \Omega$ and $C(p) = 1$ for $p \in I\backslash\Omega$. This priority-based filling aims at propagating linear structures in the texture into the hole region. For the pixel with the highest priority, the region $\bar{\Omega} = I\backslash\Omega$ is searched for a patch Ψ_q that is most similar to Ψ_p in terms of a simple sum-of-squared differences (SSD) error measure. Finally, the missing pixels of Ψ_p are replaced with pixels from Ψ_q; this is repeated until all pixels of the hole are filled in.

Video Inpainting with Spatio-temporal Patches

To solve the problem of inpainting holes in video, we extend this idea by using 3D spatio-temporal patches. The main motivation behind this is the temporal coherence of the inpainting result, a typical patch size is $9x9x3$. Analogously to the pixels in the 2D case, a priority value is computed for each voxel $p = (x,y,t)$ on the spatio-temporal hole's boundary as in Eq. 6. For the *data* term, we compute

$$D(p) = \frac{|\nabla I_p \times n_p|}{\alpha} \qquad (8)$$

where α is a normalization constant, ∇I_p is the spatio-temporal gradient vector of the video cube $I(x,y,t)$ and n_p is the normal vector of the spatio-temporal hole at p. We compute n_p using the 3D structure tensor J:

$$J = \begin{pmatrix} M_x^2 & M_xM_y & M_xM_t \\ M_xM_y & M_y^2 & M_yM_t \\ M_xM_t & M_yM_t & M_t^2 \end{pmatrix}, \qquad (9)$$

where $M \in \{0,1\}$ is a binary representation of the spatio-temporal hole with M_x, M_y and M_t being the spatial and tem-

poral derivatives, respectively. The eigenvector corresponding to the largest eigenvalue of J gives us the direction of the highest variance within the neighborhood of the considered voxel. This eigenvector equals the normal vector direction at this voxel, which is equivalent to the normal vector of the surface formed by the spatio-temporal hole.

The inpainting procedure remains the same as in the 2D case: taking the boundary voxel p with highest priority, we search for a spatio-temporal patch Ψ_q in the neighborhood being most similar to the spatio-temporal patch Ψ_p centered at p in terms of an SSD error measure, and transfer only those voxel locations of Ψ_q being empty in Ψ_p. This is repeated until all voxels are filled in. The search for a matching patch Ψ_q for the patch Ψ_p is restricted to the spatial and temporal vicinity of the location of Ψ_p in order to speed up the search process. For the results shown here, we use a spatial range of 15 voxels and a temporal range of 8 frames. We perform a grid search on the video cube with spatial mesh size Δx and temporal mesh size Δt. From these samples, the n best matches are determined and refined by a second pixelwise search in the local neighborhood. The best matching result is taken for inpainting. Although this might not necessarily find the optimal match, the huge speed-up justifies the small loss of quality. Typical parameter values are $\Delta x = 3$, $\Delta t = 2$ and $n = 10$. However, this search procedure relies on the assumption that the neighborhood of Ψ_p contains texture which should be used for inpainting. The user can also mark the search area and influence the quality of the inpainting result.

Our SSD error measure is summed over all color channels in Lab color space and is evaluated only for those voxels of the patch Ψ_p that are already filled in. To improve the matching quality, each voxel Ψ_p^i of the patch is assigned a weight depending on its location within the spatio-temporal patch. Assigning a high weight in the center of Ψ_p helps to find matching patches Ψ_q that have a similar color in the patch's center. On the other hand, high weights on the patch's corners penalize huge color deviations on the corners. To this end, we combine both approaches and construct a weight function which is the maximum of five Gaussian kernel functions centered at each patch corner and the patch's center. For simplicity, these Gaussians have an extent reaching over the whole patch. Finally, we normalize by dividing by the sum of weights and the number of voxels used for matching:

$$SSD(\Psi_p, \Psi_q) = \frac{\sum_{i \in F} w_i \cdot (\Psi_p^i - \Psi_q^i)^2}{|F| \cdot \sum_{i \in F} w_i} \qquad (10)$$

Here, $F = \Psi_p \cap \bar{\Omega}$ is the set of all voxels in patch Ψ_p that are already filled in. To achieve temporal smoothness of the inpainted region Ω, we blend between old voxels that have already been filled and the voxels from the new patch. For each new patch, the blending is done by averaging between the old and the new voxel values with equal weights (0.5).

Figure 8: *Editing results from our bending operator. Left: original frame. Right: the forearm was edited by applying the bending operator.*

The blending step is performed individually for each copied patch. Fig. 7 shows a video inpainting result. The face behind the removed ball is reconstructed faithfully. Observe that merely copying the affected region from neighboring video frames would result in temporal artifacts, since the person is moving. To get a visually pleasing result, the filled hole region must respect the complete hole boundary.

Camera Motion

Filling spatio-temporal holes in a video sequence with fast camera motion can also be handled with the approach described in the previous section, but this might introduce artifacts if the holes are large. In this case, for example in the *trampoline* sequence, we fill holes in the background using a background mosaic. For general camera motion we make the assumption that the background is planar and static. This is a good approximation if the distance between camera and background is large enough as in our example. In the first step, we use the RANSAC algorithm [HZ00] to estimate homographies H_n between all pairs of subsequent frames $n, n+1$ from Harris corner feature correspondences ($\mathbf{x_{n+1}} = H_n \cdot \mathbf{x_n}$ for image coordinates $\mathbf{x_n}, \mathbf{x_{n+1}}$). The homographies are estimated with the normalized Direct Linear Transform (DLT) algorithm [HZ00]. The products $\Pi_{i=0}^{n} H_i^{-1}$ are used to project all frames onto the reference frame 0. We filter out moving objects by employing median filtering in the temporal domain. To obtain a globally accurately aligned mosaic, the homographies are refined by computing feature correspondences between each frame n and the common mosaic plane. The mosaic is then recomputed. If parts of the background are occluded throughout the whole sequence, unknown regions remain. These empty mosaic pixels are filled in using the image inpainting method of [CPT03]. Finally, we assign to each pixel of the spatio-temporal hole the color value of its corresponding pixel in the background mosaic.

6. Editing Operations

We provide a novel *keyframe-based interpolation* method for video object animation. The user specifies a rigid transform (translation, rotation and scale) of the video object at keyframes interactively and the system interpolates the transformations of the object pixels for the remaining video frames. To preserve the characteristics of the original motion, we warp the motion by a smooth offset function. For the case of *translation*, the object trajectory $\mathbf{p}(t) = (p_x, p_y)$ is warped to $\mathbf{p}'(t) = \mathbf{p}(t) + \mathbf{d}(t)$. $\mathbf{d}(t)$ is a 2D natural cubic spline [BBB98], a C^2 continuous curve which interpolates the user constraints $\mathbf{d}(t_0), \ldots, \mathbf{d}(t_k)$ at keyframes. In our experience, linear interpolation does not give satisfactory results in most cases.

For uniform *2D scaling and rotation* we have to interpolate one transformation parameter (scaling factor and rotation angle, respectively). We use again a $2D$ spline, where the first coordinate corresponds to frame time t and the second coordinate to the transformation parameter. Also, the scaling and rotation center is user-defined and interpolated in the same way. For *mirroring* objects vertically or horizontally a negative scale factor is used.

As an example for *non-rigid deformation*, our system contains a bending operator. It can be used to bend an articulated structure, e.g. the forearm in the *waving* example (Fig. 8). Our use of this image editing operation [BC02] for video is novel. The bending transform contains three user-specified parameters, two points A and B defining the y-axis of a local coordinate system (x', y') centered at A and a bending angle θ (Fig. 6, right). A and B are interpolated with a separate 2D cubic spline between keyframes. For interpolating θ we augment it again with the time t. The bending operation essentially rotates the object, but with a linear decreasing attenuation factor $a(y') = c \cdot y'$ [BC02] towards the pivot point A. The rotation is applied to all pixels of the selected object with $y' \geq 0$:

$$(x'', y'') = \begin{pmatrix} cos(a \cdot \theta) & -sin(a \cdot \theta) \\ sin(a \cdot \theta) & cos(a \cdot \theta) \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix} \qquad (11)$$

As an alternative deformation method we have extended the moving least squares image deformation method by Schaefer et al. [SMW06] to video. The user specifies the deformation by moving control points at keyframes. These are interpolated to the remaining frames with cubic splines as described above. The moving least squares algorithm computes a deformation field which is locally as rigid as possible (Fig. 13, right, *face* example).

To emulate *camera motion*, we use the alpha mask from segmentation to construct foreground and background depth layers. We simulate two different camera motion effects. Since image disparity is inversely proportional to object depth (parallax effect), we generate a *freeze-and-translate* effect by shifting foreground and background layers by different amounts. We apply this operation to a frame of the sequence and interpolate user-specified shift vectors $\mathbf{dx_B}$ of the background at keyframes with a cubic spline. The foreground shift is then $\mathbf{dx_F} = \alpha \cdot \mathbf{dx_B}$ where the α parameter controls the parallax effect. The second effect, camera motion along the z-axis of the camera is depth-dependent scaling according to the pinhole equation $x' = f \cdot \frac{x}{z}$. We simulate this effect by linear interpolation of z, which results in different scaling factors for the foreground and background layers.

Figure 9: *Results from our border matting algorithm. Left column: raw alpha mattes. Right column: regularized alpha mattes.*

7. Compositing

Our segmentation results in binary alpha masks. To obtain a spatio-temporally coherent *alpha matte*, we first parameterize the boundary with a contour following algorithm. We then run a new border matting algorithm to generate smooth alpha mattes. We modify border matting [RKB04] in two important aspects, which leads to a method containing only one smoothness parameter that can controlled more intuitively. Border matting constructs a local foreground and background color model for each contour point by fitting a Gaussian to each of the distributions. We use the color model from Section 4 to also handle the case when multiple colors are present. Then, we compute a raw alpha matte α_0 by using the right-hand side of Eq. 3 which is depicted in Fig. 9 (left). The original border matting algorithm defines a soft α profile function along each normal of the contour which is fitted by dynamic programming. We use a different regularization approach by casting this as a surface fitting problem. An approximating thin-plate surface [Ter88] is fitted to the raw alpha matte in a corridor Ω (typically 10 pixels wide) around the contour. The sparse linear system

$$\alpha - \alpha_0 + \lambda \cdot \Delta^2 \alpha = 0 \qquad (12)$$

is solved on Ω, where α is the resulting alpha matte, Δ^2 denotes the Bi-Laplacian operator ($5x5$ stencil) and λ is a regularization parameter. We use boundary conditions $\alpha = 0$ near the background and $\alpha = 1$ near the object on $\partial\Omega$ to constrain the solution and use the solver CHOLMOD [DH05] for sparse Cholesky factorization. The result of this regularization is shown in Fig. 9 (right). Border matting is lim-

ited to object silhouettes of moderate complexity, but the robustness of this approach is advantageous for video. The foreground object color can be estimated with the Bayesian matting algorithm [CAC*02]. The transforms obtained from keyframe interpolation (Section 6) are then applied to the original video object. We use bilinear interpolation for the pixel lookup. The transformed object is composited into the video inpainting result with alpha blending.

8. Results

The accompanying video shows video editing results for six sequences, *juggling*, *waving*, *girl*, *face*, *waterjump* and *trampoline*, which exhibit moderate background complexity. For the first three sequences we use a Sony HDR-HC3E camcorder recording in HDV format 1080i (frame size 1440x1080 interlaced, frame rate 25 fps). Each sequence contains 100 frames. To avoid artifacts from interlaced recording we downsample the images by dropping every second image row and resize the frames to VGA resolution (640x480 pixels). The *face* sequence was recorded at VGA resolution and the *trampoline* and *waterjump* sequences are MPEG-2 compressed at 720x576 pixels and 50 frames. We present six different applications of our video editing system to demonstrate the capabilities of our system (Fig. 10-14). The quality of the editing results can best be assessed from the accompanying video.

Simulated camera motion is shown in the *girl* sequence (Fig. 10). To emphasize this effect, we freeze the video sequence from a handheld camera and apply a parallax effect in the middle of the sequence. At the end of the sequence we simulate a camera push motion in z-direction towards the actor.

Motion editing is shown in the *juggling* sequence (Fig. 12 left). The trajectory of the pink ball has been scaled. Only a few keyframes are used to generate the results. We place more keyframes at the upper part of the ball path to preserve acceleration effects.

Non-rigid deformation is applied to the *waving* sequence (Fig. 13 left). The bending operator is applied to the forearm of the person so that the amplitude of the waving motion is exaggerated. Keyframes specifying the bending parameters at the turning points of the motion are sufficient. To avoid inpainting artifacts, the shirt region was removed from the search region by the user. As second non-rigid deformation example we apply the moving least squares algorithm to the *face* sequence (Fig. 13 right). By setting control points at keyframes, the smiling motion was amplified. The resulting locally rigid motion is a result of the used image deformation method.

The *trampoline* sequence contains considerable *camera motion*, motion blur, and additionally also exhibits strong MPEG-2 compression artifacts. Nevertheless, our inpainting algorithm is able to accurately fill in the background texture from the background mosaic. We apply a horizontal mirroring transform to reverse the rotation of the athlete (Fig. 1 and Fig. 14 left). Note that the background is not mirrored. To increase the accuracy of the motion estimation algorithm, the segmentation result is used to mask the foreground object with a uniform color to eliminate foreground motion. We show *object replacement* by inpainting the athlete in the *waterjump* sequence. The subject from the *trampoline* sequence is then copied into the background frames (Fig. 14 right).

All computations are done on a Pentium IV 3.2 GHz with 2 GB RAM. Depending on the amount of user interaction, total processing time for video segmentation takes between 5 and 20 minutes for the presented examples (Tables 1 and 2). After segmentation, video inpainting is the only off-line processing step which currently still needs considerable computation time. For the *trampoline* and *waterjump* sequence, mosaic construction and inpainting takes between 7-13 minutes.

One current limitation of our system is the color-based min-cut segmentation which does not perform well if foreground and background have similar color distributions. Also, thin structures are smoothed and must be corrected by the user. The goal of our boundary editing tool is to make user interaction as efficient as possible. Both inpainting techniques assume a static background and cannot preserve large background motion. In Fig. 11 we show some failure cases. First, it can be clearly seen that using a spatiotemporal 3D inpainting method is benificial to a mere 2D image inpainting method to avoid temporal artifacts. Second, the segmentation and inpainting algorithms can fail in cases of little color contrast and the reconstruction of fine structures. The response times of our user interface for the segmentation and editing steps can be evaluated from the accompanying video which shows that it is fast enough for interactive editing.

9. Conclusions

We have presented a system for video object segmentation, inpainting and keyframe animation of video objects. With our framework, a number of video editing options become possible using conventional camcorder footage as input. Further editing options in the temporal domain are possible extensions of the system (temporal translation and scaling/retiming of video objects). Computing long-term pixel correspondences would enable the user to modify pixel regions in one frame which are automatically tracked throughout the video sequence. Furthermore, a stereo camera will be able to provide additional depth information, offering even more editing possibilities. This work was supported in part by the German Science Foundation DFG, project MA2555/4-1 Computational Video.

References

[AHSS04] AGARWALA A., HERTZMANN A., SALESIN D., SEITZ S. M.: Keyframe-based tracking for rotoscop-

Sequence	Size	Presegmentation	Superpixel Min-cut	Pixel Min-cut	User Interaction
juggling	640x480x100	≈3 min	≈60 sec	≈30 sec	≈4 min
girl	640x480x2	≈4 sec	≈6 sec	≈2 sec	≈5 min
waving	640x480x100	≈3 min	≈60 sec	≈50 sec	≈15 min

Table 1: *Timings for the video segmentation algorithm. The min-cut times include the time needed to build the graph. User interaction comprises the time needed to set markers and to edit boundaries.*

Sequence	Size	Inpainting	User Interaction	Compositing	Total
juggling	640x480x100	≈70 min	≈3 min	≈10 sec	≈82 min
girl	640x480x2	≈5 min	≈1 min	≈1 min	≈13 min
waving	640x480x100	≈128 min	≈2 min	≈15 sec	≈130 min

Table 2: *Timings for inpainting and editing operations. User interaction is here the specification of object transforms at keyframes and compositing the time the user has to wait for the editing result.*

ing and animation. *ACM Trans. on Graphics (Proc. of ACM SIGGRAPH 2004) 23*, 3 (2004), 584–591.

[BBB98] BARTELS R. H., BEATTY J. C., BARSKY B. A.: *An Introduction to Splines for Use in Computer Graphics and Geometric Modelling*. Morgan Kaufmann, 1998.

[BC02] BARRETT W. A., CHENEY A. S.: Object-based image editing. In *Proc. of ACM SIGGRAPH 2002* (2002), pp. 777–784.

[BJ01] BOYKOV Y., JOLLY M.-P.: Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images. In *Proc. ICCV* (2001), pp. 105–112.

[BK04] BOYKOV Y., KOLMOGOROV V.: An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. *IEEE Trans. Pattern Anal. Mach. Intell. 26*, 9 (2004), 1124–1137.

[BM03] BENNETT E. P., MCMILLAN L.: Proscenium: a framework for spatio-temporal video editing. In *MULTIMEDIA '03: Proceedings of the eleventh ACM international conference on Multimedia* (2003), pp. 177–184.

[BSCB00] BERTALMIO M., SAPIRO G., CASELLES V., BALLESTER C.: Image inpainting. In *Proc. of ACM SIGGRAPH 2000* (2000), pp. 417–424.

[BSHK04] BHAT K. S., SEITZ S. M., HODGINS J. K., KHOSLA P. K.: Flow-based video synthesis and editing. *ACM Trans. on Graphics (Proc. of ACM SIGGRAPH 2004) 23*, 3 (2004), 360–363.

[CAC*02] CHUANG Y.-Y., AGARWALA A., CURLESS B., SALESIN D., SZELISKI R.: Video matting of complex scenes. In *Proc. of ACM SIGGRAPH* (2002), pp. 243–248.

[CCBK06] CRIMINISI A., CROSS G., BLAKE A., KOLMOGOROV V.: Bilayer Segmentation of Live Video. In *Proc. CVPR* (2006), pp. 53–60.

[CPT03] CRIMINISI A., PÉREZ P., TOYAMA K.: Object removal by exemplar-based inpainting. In *Proc. CVPR (2)* (2003), pp. 721–728.

[DH05] DAVIS T. A., HAGER W. W.: Row modifications of a sparse cholesky factorization. *SIAM Journal on Matrix Analysis and Applications 26*, 3 (2005), 621–639.

[EF01] EFROS A. A., FREEMAN W. T.: Image quilting for texture synthesis and transfer. In *Proc. of ACM SIGGRAPH 2001* (2001), pp. 341–346.

[EL99] EFROS A. A., LEUNG T. K.: Texture synthesis by non-parametric sampling. In *Proc. ICCV (2)* (1999), pp. 1033–1038.

[FH04] FELZENSZWALB P. F., HUTTENLOCHER D. P.: Efficient graph-based image segmentation. *International Journal of Computer Vision 59*, 2 (2004), 167–181.

[HZ00] HARTLEY R., ZISSERMAN A.: *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[KB84] KOCHANEK D. H. U., BARTELS R. H.: Interpolating splines with local tension, continuity, and bias control. In *Proc. SIGGRAPH '84* (1984), pp. 33–41.

[LLW06] LEVIN A., LISCHINSKI D., WEISS Y.: A closed form solution to natural image matting. *Proc. of CVPR 1* (2006), 61–68.

[LSS05] LI Y., SUN J., SHUM H.-Y.: Video object cut and paste. *ACM Trans. on Graphics (Proc. of ACM SIGGRAPH 2005) 24*, 3 (2005), 595–600.

[LSTS04] LI Y., SUN J., TANG C.-K., SHUM H.-Y.: Lazy snapping. *ACM Trans. on Graphics (Proc. of ACM SIGGRAPH 2004) 23*, 3 (2004), 303–308.

[LTF*05] LIU C., TORRALBA A., FREEMAN W. T., DURAND F., ADELSON E. H.: Motion magnification. In *ACM Trans. on Graphics (Proc. of ACM SIGGRAPH 2005)* (2005), pp. 519–526.

[PSB05] PATWARDHAN K., SAPIRO G., BERTALMIO

M.: Video inpainting of occluding and occluded objects. In *Proc. ICIP (2)* (2005), pp. 69–72.

[PSB07] PATWARDHAN K. A., SAPIRO G., BERTALMIO M.: Video inpainting under constrained camera motion. *IEEE Transactions On Image Processing 16*, 2 (Feb 2007), 545–553.

[RAPLP05] RAV-ACHA A., PRITCH Y., LISCHINSKI D., PELEG S.: Evolving Time Fronts: Spatio-Temporal Video Warping. Technical Report, The Hebrew University of Jerusalem, 2005.

[RKB04] ROTHER C., KOLMOGOROV V., BLAKE A.: "GrabCut": interactive foreground extraction using iterated graph cuts. *ACM Trans. on Graphics (Proc. of ACM SIGGRAPH 2004) 23*, 3 (2004), 309–314.

[SA85] SUZUKI S., ABE K.: Topological structural analysis of digital images by border following. *CVGIP 30*, 1 (1985), 32–46.

[SMTK06] SHIRATORI T., MATSUSHITA Y., TANG X., KANG S. B.: Video completion by motion field transfer. *Proc. of CVPR (1) 1* (2006), 411–418.

[SMW06] SCHAEFER S., MCPHAIL T., WARREN J.: Image deformation using moving least squares. In *SIGGRAPH '06: ACM SIGGRAPH 2006 papers* (2006), pp. 533–540.

[Ter88] TERZOPOULOS D.: The computation of visible-surface representations. *IEEE Trans. Pattern Anal. Mach. Intell. 10*, 4 (1988), 417–438.

[WAC07] WANG J., AGRAWALA M., COHEN M. F.: Soft scissors: an interactive tool for realtime high quality matting. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers* (2007), p. 9.

[WBC*05] WANG J., BHAT P., COLBURN R. A., AGRAWALA M., COHEN M. F.: Interactive video cutout. *ACM Trans. on Graphics (Proc. of ACM SIGGRAPH 2005) 24*, 3 (2005), 585–594.

[WDAC06] WANG J., DRUCKER S. M., AGRAWALA M., COHEN M. F.: The cartoon animation filter. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers* (2006), pp. 1169–1173.

[WL00] WEI L.-Y., LEVOY M.: Fast texture synthesis using tree-structured vector quantization. In *Proc. of ACM SIGGRAPH 2000* (2000), pp. 479–488.

[WSI04] WEXLER Y., SHECHTMAN E., IRANI M.: Space-time video completion. *Proc. CVPR (1) 01* (2004), 120–127.

[WTXC04] WANG J., THIESSON B., XU Y., COHEN M.: Image and video segmentation by anisotropic kernel mean shift. In *Proc. ECCV (2)* (2004), pp. 238–249.

[WXRA07] WANG H., XU N., RASKAR R., AHUJA N.: Videoshop: A new framework for spatio-temporal video editing in gradient domain. *Graphical Models 69*, 1 (2007), 57–70.

Figure 10: *Simulating camera motion. Top: Camera motion from left to right. Bottom: Camera push motion towards the scene.*

Figure 11: *Failure cases. Top: Comparison between 3D spatiotemporal inpainting (left) and 2D image inpainting (right), where 3D inpainting leads to better results. Bottom left: example for a failed segmentation due to similar foreground and background colors (horse legs). Bottom right: example where the inpainting algorithm cannot reconstruct fine structures in the hole region (goal net).*

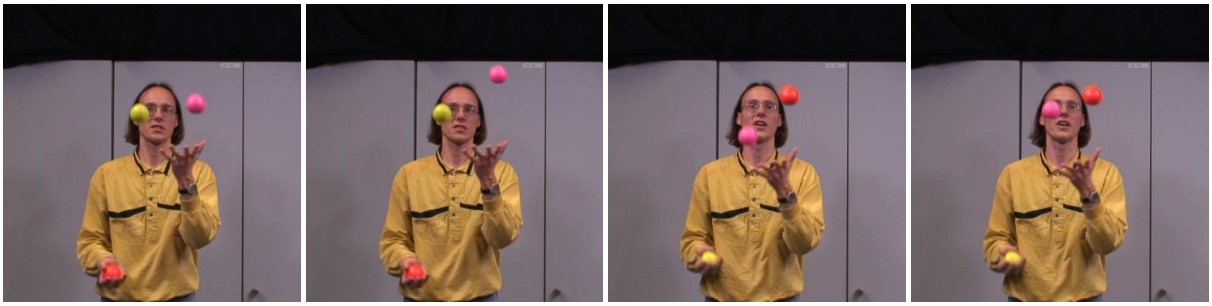

Figure 12: *Juggling sequence. The position of the **pink** ball was altered. From left to right: original, edited, original, edited frame.*

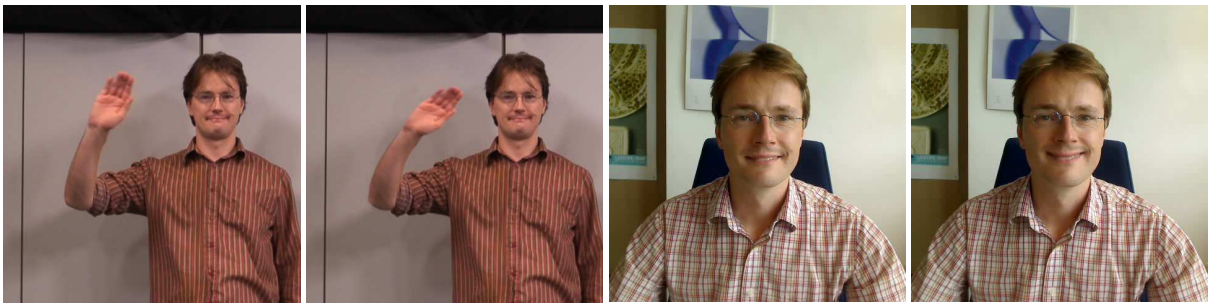

Figure 13: *Non-rigid deformation. Left: waving sequence. The original motion was magnified by applying a non-rigid bending deformation to the forearm. Right: face sequence. The smile was amplified by moving least squares deformation. From left to right: original, edited, original, edited frame.*

Figure 14: *Trampoline and waterjump sequence. The rotation of the athlete was reversed and the missing background was reconstructed (left). The waterjump athlete was replaced by the trampoline athlete (right).*