Animated 3D Creatures from Single-view Video by Skeletal Sketching

Bernhard Reinert* MPI Informatik
Tobias Ritschel† University College London
Hans-Peter Seidel‡ MPI Informatik

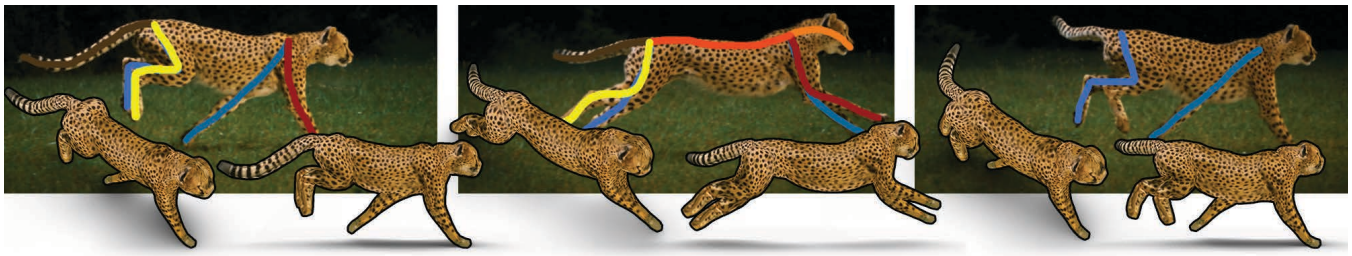

Figure 1: Our approach analyzes 2D sketches over some key frames of a 2D video to produce an animated and textured 3D mesh.

ABSTRACT

Extraction of deformable 3D geometry is not accessible to casual users, as it either requires dedicated hardware or vast manual effort. Inspired by the recent success of semi-automatic 3D reconstruction from a single image, we introduce a sketch-based extraction technique that allows a fast reconstruction of a dynamic articulated shape from a single video. We model the shape as a union of generalized cylinders deformed by an animation of their axes, representing the "limbs" of the articulated shape. The axes are acquired from strokes sketched by the user on top of a few key frames. Our method bypasses the meticulous effort required to establish dense correspondences when applying common structure from motion techniques for shape reconstruction. Instead, we produce a plausible shape from the fusion of silhouettes over multiple frames. Reconstruction is performed at interactive rates, allowing interaction and refinement until the desired quality is achieved.

Index Terms: I.4.8 [Image processing and Computer Vision]: Scene Analysis—Tracking, I.4.6 [Image processing and Computer Vision]: Segmentation—Pixel classification

1 INTRODUCTION

The acquisition of dynamic 3D content requires either dedicated scanning hardware, such as range sensors [21], multi-camera setups [30] or large amounts of manual effort [28] often not accessible to casual users. Simpler means to convert a common monocular video into an animated and textured 3D surface would provide new opportunities in display, manipulation and transmission of visual content.

Multi-view reconstruction, e. g., from a large set of general community images or videos [36], provides an accessible means for the reconstruction of static geometry. Existing approaches reconstruct static models [15, 35] from multiple views using Structure from Motion (SfM) techniques. Extensions to moving or deformable objects [2, 7, 14] were proposed, but the challenge to establish reliable correspondences between the different views remains difficult. While for rigid shapes a single linear transformation suffices, non-rigid content requires complicated space-time regularizations.

Inspired by a recent user-assisted reconstruction technique [12], in this paper we develop a sketch-based acquisition approach for the reconstruction of non-rigid shapes from a single view video. The premise of our approach is that the animated 3D object can be modeled as a union of deforming generalized cylinders. The approach is guided by a user sketching a set of "limb strokes" indicating the axis of a generalized cylinder in a couple of key frames. The core of our method is to propagate this annotation from key frames to all video frames. Our method bypasses the meticulous effort required for specifying dense correspondences used in common SfM techniques. Instead, "limb strokes" are used to fuse the silhouettes of all video frames into a consistent, animated 3D shape. Combining different silhouettes does not only result in animation, but also resolves shape details that are not revealed in a single frame. The procedure can be performed at interactive rates, allowing the user to refine the result until the desired quality is achieved.

The main contributions of this work are therefore:

- Tracking of axis sketches through image sequences
- Consolidation of segmentations over all video frames
- Fitting of 3D generalized cylinders from tracked strokes and segmentation masks

Our results demonstrate extraction of animated 3D shapes such as animals in motion from video with applications including cloning, reposing, novel view synthesis, texture transfer and 3D printing.

2 BACKGROUND

Our system extends the line of work on user-assisted acquisition of static 3D geometry from a single view to animated 3D geometry from multiple video frames. In this section we review previous work on acquisition of static and dynamic geometry, in particular from a single view as well as important user interfaces such as sketching.

3D reconstruction Reconstructing 3D shape from one or multiple images has been an important area of research in the past decades and remains a challenging task. 3D geometry is usually acquired using special hardware, such as depth sensors [21] or multi-camera setups [36]. Sufficiently textured rigid scenes can reliably be acquired using SfM and allow for impressive applications [36] when large collections of images are available. These algorithms however only reconstruct 3D information for a sparse set of reliably tracked features. Using those features in combination with additional

*e-mail: breinert@mpi-inf.mpg.de
†e-mail: t.ritschel@cs.ucl.ac.uk
‡e-mail: hpseidel@mpi-inf.mpg.de

Graphics Interface Conference 2016
1-3 June, Victoria, British Columbia, Canada

constraints provided by the user, such as symmetry or planarity, high-quality 3D models can be constructed [35]. If the class of object is known a-priori, specialized template-based solutions for humans from many 3D scans [3] or animals [11] have been proposed. All approaches require user interaction in some way, such as defining correspondences by clicking [3, 11]. If the video contains a human, for which templates are available, motion can be captured [38] using automatic or semi-automatic template fitting allowing to manipulate images or videos [22]. Our approach goes beyond human shapes, allowing the user to draw and refine arbitrary skeletons unknown a-priori.

Reconstruction of animated, non-rigid 3D models without special hardware remains a challenging problem. Non-rigid SfM is currently addressed by either assuming that the deformation is a combination of rigid transformations of basis shapes [7] or basis trajectories [2]. Even if correspondences are given [18] reconstruction is typically limited to moderately deforming, sphere like objects and requires long computation time, defying interactive use.

Multi-view 3D reconstruction For multiple views, skeletons and template models sophisticated systems exist that estimate both skeletons and shape simultaneously [17]. In contrast to such approaches, we do not rely on any a-priori known model or an explicit understanding of the underlying skeletal structure of the creature. Additionally our algorithm allows for a rich set of deformations, exceeding those of other tracking approaches. While other tracking approaches deform each bone by a single rigid transformation, our limbs commonly aggregate several biological bones allowing for piecewise rigid but also non-rigid motions. This enables tracking of limbs that are otherwise hard to track using a single bone, such as the tail and body of a cheetah or the neck of a giraffe, and abstracts model complexity away. Our system solely relies on the input video in combination with user-defined strokes enabling 3D reconstruction even for creatures with unknown or no skeleton at all. All video sequences used in this paper are taken from online video platforms and do not require any calibration steps beforehand, rendering the system useful also for casual users within the assumptions of our paper.

Single-view 3D reconstruction Creating a 3D model from a single image is an even more challenging task, often addressed using semi-automatic approaches. Research of human perception has found, that the occluding contour/silhouette is a strong cue for the inference of a full shape from its 2D projection [26]. Most systems require the user to interactively segment the object in question unless it has been imaged in front of a simple background.

Sketching The first system to create 3D surfaces from sketches was proposed by Zeleznik et al. [40]. Later, the "Teddy" system [20] introduced intuitive modelling of free-form 3D surfaces from simple sketches. The SmoothSketch system [25] allows the user to draw the silhouette, from which a smooth surface is created similar to the creation of silhouettes from photos [31]. As many real-world objects can be approximated using simple geometric primitives as proxies, like cuboids [41] or generalized cylinders [12] some recent systems snap the user sketches directly to a reference photo with interactive feedback. Modelling objects by sweeping a profile along the main axis e. g., a generalized cylinder has been used as an intuitive modelling metaphor for many objects [13]. If template models are available contour sketches can customize a deformable model [27]. Our approach combines skeleton and surface estimation in a single interaction metaphor and runs at interactive rates for natural image input of detailed natural animated surfaces.

Animation reconstruction The idea to sketch motion is inspired by the observation, that recognizing motion of a human, an animal, or even an unknown object by a set of moving points is an easy task [24] for a human. Bregler et al. [8] capture key frame motion from cartoon characters and transfer it to 3D character key frames. Favreau et al. use segmentation on videos to extract a small set of key frames that represent the principal components animal gait [16]. While their approach allows transferring motion to 3D models, they do not reconstruct the 3D shape. Automatic and semi-automatic creation of animation from examples [4] or annotations [5] has been proposed, but requires a known geometry and skeletal animation hierarchy. Xu et al. [39] have reconstructed animal motion from a single or low number of images that show multiple animated poses of a walk cycle, but their reconstructions are only two-dimensional and require manual segmentation. Reconstructing motion from simple sketches has received less attention than static geometry.

Our approach can in many cases reproduce the complicated gait pattern of an entire limb in an animal walk cycle from a single stroke, including occlusion handling. Our work builds upon user-guided segmentation [1] of animated objects in video, but to the end of reconstruction an animated 3D shape. Optical flow [9, 33, 37] establishes correspondences over time and solves a task similar to our stroke tracking automatically. We build a global motion model from strokes on the fly allowing the user to inspect the resulting 3D shape quality and refine the input. Our comparisons show that this allows for better tracking than off-the-shelf optical flow methods.

3 From skeletal sketches to animated shapes

3.1 Overview

Input to our method is an arbitrary video sequence depicting a moving creature and a set of strokes that the user draws on top of a sparse set of key frames. Each stroke is drawn alongside one body part of this creature, representing a deforming "limb" potentially connected to other limbs. The user is free to draw as many strokes as she wants but typically each non-branching body part can be drawn in a single stroke. Every stroke is associated with a label and the user is required to draw consistently labelled strokes in all key frames. Our limb strokes are conceptual and do not necessarily have a biological meaning: As long as a creature's body part can be represented by a single generalized cylinder, i. e., without a branch, it can be drawn with a single stroke. We assume that the limbs observed in all frames deform but exhibit the same connectivity, although not all limb strokes are necessarily entirely observable in all frames. We shall show this rather simple user input suffices to enable the reconstruction of an animated mesh.

The four main components of our approach are stroke tracking, segmentation, cylinder fitting as well as texturing (Fig. 2). A simple user interface (Sec. 3.2) provides the necessary input.

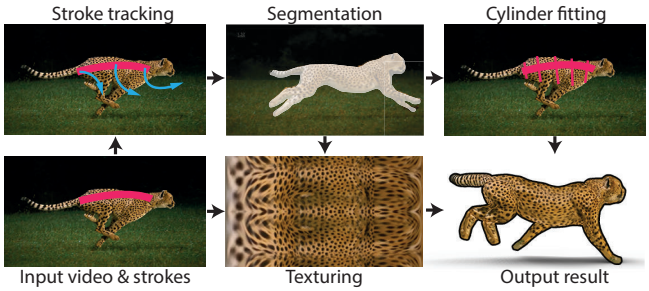

Figure 2: System overview.

Stroke tracking turns the temporally sparse limb strokes defined in a few key frames into temporally dense, coherent strokes for all frames. refSecStrokeTracking describes tracking of strokes that

form a skeleton over a video sequence. We exploit the connectivity and spatio-temporal structure to track complicated structures such as limbs with occlusion and pose changes over long frame ranges, such that for unoccluded parts often a single stroke is sufficient to be tracked over a typical sequence.

Segmentation uses the temporally densified strokes from tracking and the video to consistently segment foreground from background in all video frames, described in Sec. 3.6. Again, we exploit the space-time structure of the 2D strokes to simplify the problem and make a consistent segmentation over many video frames from sparse input.

Cylinder fitting uses the foreground segmentation in all video frames to fit animated 3D cylinder geometry around each limb stroke. Sec. 3.7 explains how to use marching in 2D masks to implement a voting scheme to robustly find cylinder radii and 3D paths.

Texturing finally utilizes the segmentation and video frames to assign texture colors to the animated 3D cylinders (Sec. 3.8).

3.2 User interface

The main user interface elements are the limb strokes suggesting the 2D projection of a non-branching, generalized cylinder-like shape part. Identical limb stroke labels are identified in different frames by the same stroke color. Not all limb strokes need to be drawn in all key frames but the user is required to draw the full skeleton in the first one. The results of the stroke tracking step (Sec. 3.5) are used to complete and display the missing strokes in subsequent frames. The user can review the tracking results at any time using a time slider and insert a new key frame correcting only the inaccurately tracked strokes. Once the user has drawn a stroke in one key frame, for subsequent stroke sketches we display a circle with the maximal limb length found (Fig. 4, limb length hint) to facilitate sketching of consistent stroke lengths, particularly of (partially) occluded limbs. Each limb stroke comes with a direction state that determines if the stroke points towards or away from the viewer which can be altered with a click and is detailed in Sec. 3.4 and Sec. 3.7.2.

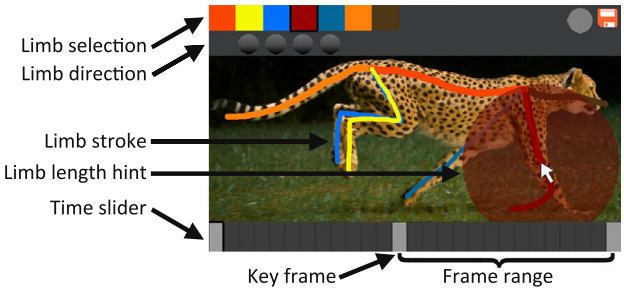

Figure 4: User interface *(Please see text).*

3.3 Preprocessing

Initially, the video sequence is resampled to a fixed resolution of 512×512 pixels for all steps to speed up the subsequent computations and allow identical parameter settings for all video sequences. Cylinder fitting (Sec. 3.7) operates on a filtered variant of the scaled video to facilitate video segmentation: a five-times-repeated bilateral filter with a spatial radius of 10 pixels and a range radius of 0.02 (cf. Fig. 5, a).

3.4 Stroke processing

Strokes are sequences of vertices defined at 2D image locations. Each stroke of label $l \in \mathbb{N}^+$ drawn by the user in frame $f \in \mathbb{N}^+$ initially may consist of a varying number of vertices, defined by a set $\hat{X}^{l,f} = \{\hat{\mathbf{x}}_1^{l,f}, \hat{\mathbf{x}}_2^{l,f}, \ldots\}$ with $\hat{\mathbf{x}}_i^{l,f} \in \mathbb{R}^2$. As our subsequent steps require consistent vertex counts in all key frames, the strokes need to be resampled with an approximately equal inter-vertex distance along the original stroke. To this end we employ a linear resampling per label l with a vertex count $n_l \in \mathbb{N}^+$ depending on the maximum stroke length in all frames. A vertex sequence is resampled to a set $X^{l,f} = \{\mathbf{x}_1^{l,f}, \mathbf{x}_2^{l,f}, \ldots, \mathbf{x}_{n_l}^{l,f} \in \mathbb{R}^2\}$ such that $d_{\hat{X}^{l,f}}(\mathbf{x}_i, \mathbf{x}_{i+1}) \approx c, \forall i \in [1, n_l - 1]$ with $d_{\hat{X}^{l,f}}$ as the length along stroke $\hat{X}^{l,f}$. Stroke tracking (Sec. 3.5) uses an approximate inter-vertex distance of $c = 10$ pixels while segmentation (Sec. 3.6) and cylinder fitting (Sec. 3.7) operate on vertices of approximately $c = 1$ pixel distance. Ideally stroke resampling should take texture features into account; due to limb deformations and occlusions matching those features however is not feasible in this step.

As we require all limb strokes to be connected to other limb strokes, they share common start- and/or end vertices, resulting in a connection graph of strokes. We arbitrarily define the first vertex of the first limb stroke to be the root vertex, turning the connection graph into a connection tree.

We use a simple, heuristic symmetry detection on the strokes drawn by the user to bootstrap some of the subsequent steps. Two limb strokes drawn in the same key frame that share a connecting stroke vertex are defined to be symmetric if their stroke lengths differ by less than 10% and if the y-coordinates of their outer stroke vertex differ by less than 30 pixels. The limb associated with the stroke of higher label number is heuristically chosen to be the outside limb, further away from the viewer than the inside one. This decision can be flipped for both limbs using the limb direction buttons of the user interface.

3.5 Stroke tracking

Input are temporally sparse 2D strokes in some key frames. Output are dense "space-time strokes", i. e., 2D image locations for every limb stroke vertex in every frame. Space-time strokes are tracked on the *range* between key frames forward and backward independently (cf. Fig. 4). This potentially contradicting information of two overlapping ranges for each point in time is combined in a blending step.

Without loss of generality, we will next describe the forward tracking of all stroke vertices for one range. Starting from a key frame k (Fig. 3, a) every frame in the range is tracked sequentially. The current frame is denoted by f whereas the previous frame in forward or backward tracking direction is indexed by $f \pm 1$. For every frame f, new candidate positions $\mathbf{x}^{f,l}$ for every vertex of every stroke l are generated and combined into stroke candidates $\overline{X}^{f,l}$. For each of the stroke candidates we compute a unary cost $u(\mathbf{x}) : \mathbb{R}^2 \to \mathbb{R}$ encoding appearance constancy and a binary cost $b(\mathbf{x}, \mathbf{y}) : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$ encoding kinematic constraints between neighboring vertices of a stroke. Hence we can compute the cost of a candidate stroke $\overline{X}^{l,f}$ as

$$c(\overline{X}^{l,f}) = \sum_{i=1}^{n_l} u(\overline{\mathbf{x}}_i^{l,f}) + \lambda \sum_{i=1}^{n_l - 1} b(\overline{\mathbf{x}}_i^{l,f}, \overline{\mathbf{x}}_{i+1}^{l,f}) \qquad (1)$$

where $\lambda \in \mathbb{R}$ constitutes a relative weight between unary and binary terms. Finally the best sequence of candidates that globally minimizes Eq. 1 is selected. We will detail every step in the following.

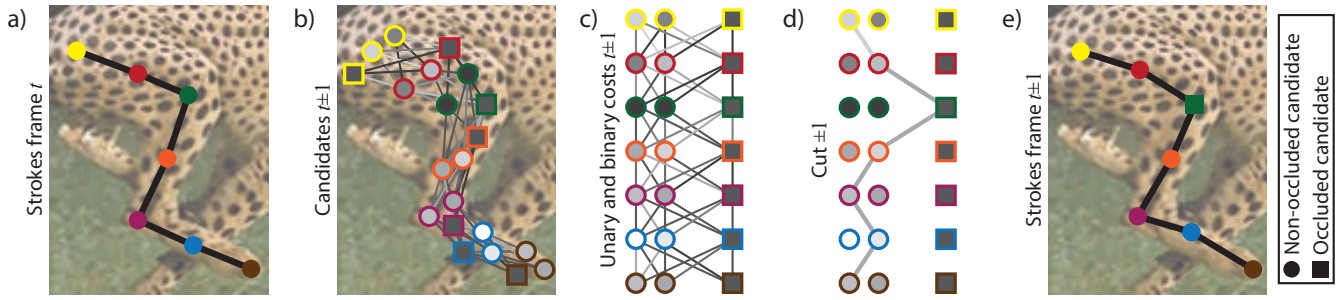

Figure 3: Stroke tracking of a stroke from frame t to $t \pm 1$: The solution of frame t initializes candidates for frame $t \pm 1$ (a and b). Non-occluded *(circles)* and occluded *(squares)* candidates are generated for frame $t \pm 1$ (b and c). Costs are visualized as grey values on candidates and on edges for an abstract stroke domain (b - d). The path with lowest cumulative cost is found (d) and yields the solution for frame $t \pm 1$ (e). Note that, although being unoccluded, the green marker is chosen from the set of occluded candidates as all non-occluded candidates lead to higher costs.

Candidate generation Candidate vertex positions $\overline{\mathbf{x}}^{l,f}$ for the current frame are generated in a small window around the vertex position $\mathbf{x}^{l,f\pm1}$ of the previous frame (Fig. 3, b). Two types of candidates are produced: Occluded and non-occluded ones. Non-occluded candidates are placed for every pixel in a search window of 64×64 pixels around the vertex position of the previous frame. Additionally, 100 occluded candidate vertex position are placed on a regular grid in the identical search window, allowing occluded and non-occluded candidates to potentially use the same position. Non-occluded candidates that fall outside of the image are removed, whereas occluded candidates are allowed to fall outside the image, enabling tracking of strokes partially outside of the image. Each candidate stores its occlusion type for later stages.

Unary cost The unary cost (cf. Eq. 1) models appearance consistency between frames (filled circles and squares in Fig. 3). To this end, we compare a neighborhood patch $\mathcal{N}$ of size 32×32 pixels around each non-occluded candidate position to a patch of the same size around the vertex position in previous frames using a sum of squared pixel color distances (SSD), i. e.,

$$u(\mathbf{x}) = \min_{T \in \mathcal{T} \times i \in [1,3]} \sum_{\mathbf{o} \in \mathcal{N}} \| a^{f \pm i}(T(\mathbf{x}+\mathbf{o})) - a^{f}(\mathbf{x}+\mathbf{o}) \|_2^2,$$

where $a^f(\mathbf{x})$ gives the pixel values at position $\mathbf{x}$ in frame f, $T \in \mathcal{T}$ is single transformation of a set of transformations $\mathcal{T}$ and i is a frame distance detailed below. SSD is used instead of normalized cross correlation (NCC) as the differences in appearance are small between frames and mainly due to deformations rather than illumination changes. Further SSD can be trivially parallelized whereas NCC requires additional computations.

In particular, we consider *variants* in template and orientation when taking this difference, defined as a set of rigid transformations $\mathcal{T}$. First, the difference to the patch around the three last non-occluded vertex positions is enumerated. Second, we allow for a 10 degrees positive or negative rotation in seven steps. The unary cost of the candidate is then the minimal difference over all 21 possible variants. These rigid transformations are applied per vertex and their combination allows for a wide variety of non-linear deformations.

Occluded candidates do not have an apparent unary cost as the appearance constancy for occlusions cannot be evaluated directly from the video frames. In order to select an occluded candidate the unary cost should reflect a certain occlusion penalty cost that allows these candidates to be chosen over non-occluded ones that match poorly. To this end we assign a unary cost of 150% of the previous best non-occluded matching cost to each occluded candidate. Combining occlusion states with template comparison makes the system robust against occlusions, that occur frequently e. g., on legs, and offers a solution to the well-known "template update"-problem [29].

Binary cost The binary cost (cf. Eq. 1) consists of two components: *distance* and *direction* preservation (lines in Fig. 3, b – d), defined as

$$b(\mathbf{x},\mathbf{y}) = \phi(\mathbf{x},\mathbf{y}) + \theta(\mathbf{x},\mathbf{y}).$$

Distance preservation measures the change in length for each candidate pair relative to the original distance in the key frame as $\phi(\mathbf{x},\mathbf{y}) = \left| \|\mathbf{x}-\mathbf{y}\|_2 - \|\mathbf{x}^k - \mathbf{y}^k\|_2 \right|$ with $\mathbf{x}^k, \mathbf{y}^k$ as the respective vertex position in key frame k. It encodes topology and isometry knowledge put into the system through user interaction and adds user-given semantics not available, e. g., to optical flow. Orientation preservation measures the change in orientation and distance of each candidate pair from the current to the previous frame as $\theta(\mathbf{x},\mathbf{y}) = \left\| (\mathbf{x}-\mathbf{y}) - (\mathbf{x}^{f\pm1} - \mathbf{y}^{f\pm1}) \right\|_2$ where $\mathbf{x}^{f\pm1}, \mathbf{y}^{f\pm1}$ are the respective vertex positions of the previous frame. It encodes velocity constraints and enforces smooth movements but also penalizes large distance changes between frames.

Candidate selection Candidate selection chooses one candidate out of the occluded and non-occluded candidates for every stroke vertex that globally minimizes the cumulative unary and binary costs of Eq. 1, i. e., it computes $X^{l,f} = \arg\min_{\overline{X}^{l,f} \in C} c(\overline{X}^{l,f})$ where C is the set of all possible stroke candidates. Ideally all combinations of all candidate positions should be evaluated but the sheer amount of candidates makes the optimization prohibitively expensive as the complexity is quadratic in the number of candidates. To enable an efficient optimization we prune the non-occluded candidates down to a count of five constituting the dominant minima of the unary costs. We found that although being an approximation this pruning strategy does not noticeably degrade our result. As distance and direction preservation pose important constraints on our optimization we weight them with a factor of 10 relative to the unary and binary distance cost, i. e., $\lambda = 10$. The optimized result for each frame becomes the initial result governing the candidate generation of the next frame (Fig. 3, d). Globally optimizing a one-dimensional sequence of labels with unary and binary costs can efficiently be solved using dynamic programming as done e. g., by Buchanan et al. [10]. In contrast to their approach that regularizes feature tracks over time, we employ dynamic programming to regularize in space, i. e., along our strokes.

In general we do not only want to select candidates for a single stroke but for all strokes of the connection tree in a single frame at once. Without loss of generality we can employ the connection tree to run a global candidate selection on all strokes simultaneously starting from the tree's leaf vertices up to the root vertex to get the globally optimal candidate choice.

We apply special treatment to those vertices that are likely to be occluded by other parts. Symmetric limbs flagged to be further away from the viewer (Sec. 3.4) are presumably (partially) occluded by

their symmetric counterpart in many frames. Hence our optimization is allowed to only choose from occluded candidates for stroke vertices of these strokes that are closer than 20 pixels away from any other stroke vertex. In some cases these limbs can however be fully occluded in all frames and for such we add the vertex position offsets of the symmetric inward facing limb also to the vertex positions of the outward limb.

The spatial connections of the stroke vertices form a Markov chain. Including temporal connections to other frames could allow for global optimization in both spatial and temporal direction resulting in a Markov random field model. As stroke tracking is based on a cost depending not only on the last but all previous frames up to the last key frame candidate selection would become prohibitively expensive. Taking into account all previous frames leads to an excessive number of temporal connections between frames making an optimization of our non-submodular costs too expensive to be optimized with interactive performance.

Blending After all ranges have been tracked, they need to be combined into one consistent sequence. This happens for all forward-backward pairs of ranges between two key frames in isolation. Input to this step are two ranges that overlap in some frames, one from the forward and one backward in time. Output is a single, consistent track. To preserve motion and intrinsic distances, we blend the orientations of each limb stroke segment relative to its parent using spherical linear interpolation (SLERP) [34] with a linear interpolation of the root vertex position. The weight given to a range at each vertex at each point in time decreases with distance to the respective key frame. An alternative way of blending between the two frame ranges could incorporate texture information, i. e., using the unary costs of Eq. 1. However, the resulting weights would not necessarily be smooth over time and could lead to noticeable jumps in the motion. Computing matching-based weights in a temporally smooth way remains future work.

3.6 Segmentation

As now all strokes are known in all frames, we next segment the foreground of the filtered input video and, in a second step, label the foreground such that every pixel belongs to exactly one stroke.

Foreground segmentation We employ cost-volume filtering [32] using the previously generated space-time strokes as input. To this end we build two (soft) foreground RGB histogram with 50^3 bins (Fig. 5, b): one of all pixel colors of the stroke vertices in all frames of the filtered video (colored strokes in Fig. 5, a) and one build from a set of background vertices in all frames (white stroke in Fig. 5, a). The background stroke is found as vertices of a slightly offset bounding box of the limb strokes. Softness is achieved using a Gaussian blur with a σ of 2. The cost volume is blurred (we typically use 2×2 pixels for our resampled video) in space and a binary foreground mask (Fig. 5, c) is extracted with a threshold at 0.75. We use this slightly higher threshold to allow for larger segmentations that can later be consolidated by the cylinder fitting step. To improve segmentation stability for tracking errors, the last 20 % of the stroke vertices on each terminal limb stroke are skipped. A terminal limb stroke is a stroke with no further child strokes in the connection tree. This helps, as such strokes are susceptible to be drawn too long or are occluded near the ground, e. g., by the grass in Fig. 1 or can be difficult to segment due to contact shadows. As the user is required to draw the full skeleton in the first key frame the initial histograms resemble a valid color distribution of the object. Additional segmentation results can be found in the supplemental material.

Stroke segmentation Each pixel of the foreground segmentation does not belong to all limbs and drawing radius samples from this segmentation directly prevalently leads to overestimation. Hence in a second step the foreground is partitioned such that every foreground pixel belongs to exactly one stroke. This second segmentation step is done independently in all frames. For every stroke vertex we march the image orthogonal to the stroke until we leave the foreground mask or the image. For every pixel visited during the walk we keep a reference to the closest stroke and draw a line from the stroke to the radius sample with a depth value corresponding to this distance (Fig. 5, d). This can be done efficiently on a GPU using splatting of sufficiently thickened stroke IDs and z-buffering of distances. Note that this is slightly different from a generalized Voronoi decomposition of the skeleton in terms of Voronoi sites, as distances are computed strictly orthogonal to the limb strokes, the marching direction of the cylinders.

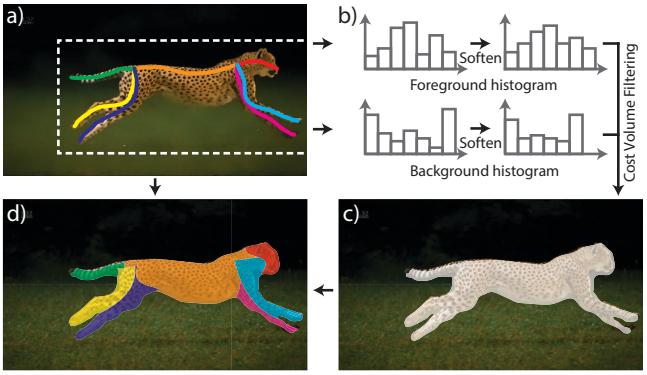

Figure 5: Segmentation *(Please see text).*

3.7 Cylinder fitting

Cylinder fitting takes as input the dense 2D space-time strokes as well as the stroke segmentation. Output is one deforming generalized 3D cylinder for each limb stroke represented as a 1D table of 3D positions (the *path function*), as well as a 2D table of radii for every direction at all stroke positions (the *radius function*). The radius function (Sec. 3.7.1) and 3D direction (Sec. 3.7.2) are found independently (Fig. 6). Finding the correct radius in a single frame can fail due to self-occlusions, especially in the vicinity of limb joints, and segmentation errors. Hence, consolidation of segmentation results of all frames is used to compute consistent radii.

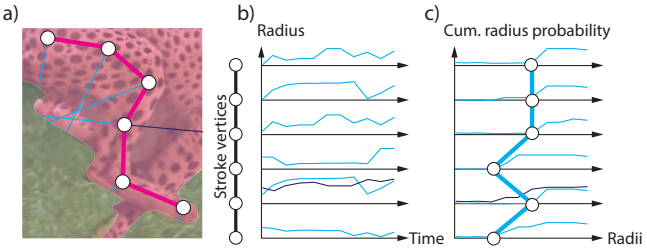

Figure 6: Cylinder fitting: For every stroke vertex *(circle)* in each frame orthogonal marching finds the closest background pixel which is then counted as a vote for a radius *(blue line).* Each frame produces a different radius vote over time, here shown as a radius function of each stroke vertex. Taking a robust minimum finds the effective radius for every stroke vertex, producing a radius function along the stroke *(thick blue vertical line).* In practice, two radius functions are computed for each vertex *(only shown for one vertex).*

3.7.1 Radius fitting

To account for uncarefully drawn terminal limbs of creatures, such as legs or heads, we initially extend every stroke along its start

and end direction in 2D by an additional 10 % before all further processing. This approach potentially creates samples outside of the segmentation, but the final validation step eliminates invalid samples.

Next, the radius function at every vertex is found independently (Fig. 6, a and b) before combining them into consistent radius samples along the stroke (Fig. 6, c). We will now describe this procedure for a single vertex. In the work of Chen et al. [12] the user has to specify the radii explicitly; in contrast our method automatically combines segmentation results of all video frames to fit consolidated radii. While our approach could benefit from explicitly drawn radii it would make the interface more complicated and especially at limb joints specifying the correct radius is fairly difficult and requires careful drawing. Hence, correcting false radii using additional strokes remains future work.

Marching The full radius function is computed from a *radius sample pair*: One radius in the normal direction of the limb stroke and one in the opposite one. In a second step, these many sample pairs are converted into a full 360-degree radius function. We will describe this procedure for the radius samples drawn from the normal direction without loss of generality, both are performed equally.

Starting from a stroke vertex position we march orthogonal to the stroke direction [12] until we reach a pixel not belonging to the foreground. Each such walk results in a vote for a radius. If on the way we encounter a part belonging to a different stroke we reject this sample. Additionally radius samples at image boundaries or smaller than a threshold of 3 pixels are rejected. Drawing radii in all frames results in a distribution of different 2D radii votes: one, potentially rejected, vote per frame. To obtain a robust radius estimation for every vertex from this distribution, we use the robust 25 %-percentile of the accepted radius samples for the vertex itself, its four neighbor vertices along the stroke and vertices of the symmetric limb, if applicable. Percentiles help to resolve false radius samples due to overlapping limbs or erroneous segmentation. If too many radius samples are rejected per vertex (more than 90%), we flag the radius as being invalid. This happens if a limb stroke is stuck inside the body of a creature or if segmentation permanently fails. Assigning a radius for these vertices is left to filtering and in-painting along the stroke.

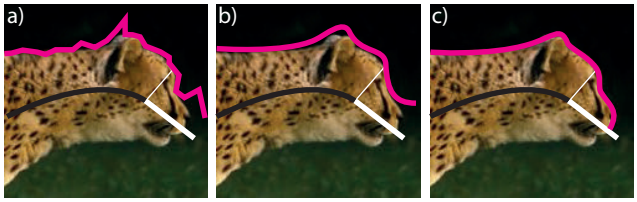

Figure 7: Estimating the radius pair at every stroke vertex can be noisy and has errors such as at the ends, which are not part of the limb strokes, but added automatically *(white part)*. Smoothing is used to reduce noise. Outliers are detected and in-painted along the stroke. Finally, the radius function is fit to the mask in one reference frame to refine the shape, in particular at the end.

Filtering and in-painting Next, invalid radius samples are in-painted from nearby valid ones. Also, the radius function is slightly blurred with a σ of 10 % of the total stroke length accounting for the noisy percentile segmentation along the stroke (Fig. 7, middle).

Capping All cylinder ends that are connections between limbs are capped, i. e., the radii at the start and end are blended to 0 in an interval of 20 samples around the ends.

Validation In-painting, blurring and capping modifies the radius samples, resulting in cylinders that potentially fall outside of the segmentation. Also the robust percentile is prone to a slight overestimation of the correct radius. However, we can be more sure about radii in the key frames. Therefore, we iterate over all key frame foreground masks, and make sure no vertex falls outside this mask in any key frame (Fig. 7, right).

Densification Finally, the radius pair is converted into a dense radius sample function for all directions at every stroke vertex by fitting an ellipse (Fig. 8, a). Simply using the average of the normal and counter-normal radii (pink and blue) for both, major and minor ellipse radius (green and red), yields a circular cross-section (Fig. 8, b). For many shapes such as an animal torso, where the spine is located rather at border of the cylinders, a compressed ellipse provides a better shape approximation. Thus, we multiply the minor ellipse radius with the ratio of the smaller of the two normal radii divided by the larger one (Fig. 8, c). As this leads to flat cross sections in case of close to zero minimum normal radii, we average both strategies to obtain our final minor ellipse radius: half the average and half the ratio (Fig. 8, d).

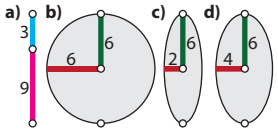

Figure 8: Ellipse fitting.

3.7.2 Path fitting

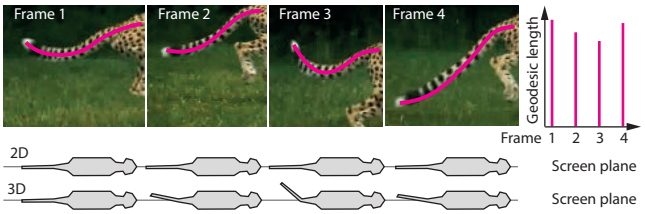

Figure 9: The length of each stroke in image space varies, for strokes that change 3D direction such as the tail. To the right we plot the geodesic length for all four frames. We use this information to add 3D out-of-plane rotation of strokes to the 2D shape *(Bottom)*.

The x- and y-coordinate of the generalized cylinder path function result from the stroke tracking step (Sec. 3.5). Assuming approximately constant object-to-camera distance throughout the entire video sequence these coordinates immediately constitute the x- and y-coordinates for each stroke vertex. The missing z-component is found using the kinematic information as follows (cf. Fig. 9): We detect limb strokes that leave the image plane with the aid of their projected lengths. A limb stroke with a substantially shorter projected length (50 % of its maximum length) likely has rotated outwards of the image plane. For such frames we use inverse perspective of known 3D length to compute its out-of-plane rotation. To obtain the final z-values we use a polynomial of degree two that interpolates the start position as well as its gradient and exhibits the correct 3D length. To determine if we have an increasing or decreasing z-component we use the user-defined limb direction flag. A temporal smoothness assumption allows propagating this flag from the sparse set of key frames to all frames: When a limb moves smoothly from an in-plane direction to an out-of plane direction facing the viewer, it is assumed to continue facing the viewer. Finally limbs that are found to be symmetric get a z-offset depending on their thickness. For this we calculate the average thickness of the lower 50 % of the limbs and use this offset with a factor of 4 to move the limbs in positive resp. negative z-direction. The previously mentioned limb direction decides on the sign of this offset. This reliably offsets the limbs of animals at hip- or shoulder-type joints and appears a suitable heuristic for the animals used in this paper.

3.8 Texturing

Texturing uses the generalized cylinders as well the video sequence. It outputs a cylindrical texture for each limb cylinder. Blending textures from different frames has shown to be counter-productive due to slight but apparent shading and deformation changes. Hence, we chose a single frame of the video to obtain the texture for all limbs. We heuristically always pick the first key frame as it contains all limbs by construction. To obtain the texture for each cylinder we project the cylinders vertices into the video frame and calculate a binary reliability. Unreliable pixels are those that fall outside of the foreground segmentation, pixels at grazing angles of the cylinders, and pixels marked as being occluded. These unreliable pixels are filled with the closest reliable pixels using push-pull in-painting [19]. To allow for wrap-around of the in-painting it is performed on an unwrapped cylinder that respects a toroidal parametrization. Occluded limbs use the texture of their symmetric, non-occluded counterpart for texturing. Texturing uses frames from the original video in full resolution to produce textures with maximum detail.

3.9 Implementation

To ensure interactive performance, most steps are implemented using parallel graphics hardware. Candidate selection of the stroke tracking governs the complexity of our algorithm with a running time of 20 to 400 ms per frame depending on the skeleton complexity. All other steps can be parallelized over all stroke vertices and strokes in all frames resulting in a running time of less than 5 ms. The initial tracking of a typical skeleton in a video sequence of 64 frames can be processed in 20 30 seconds, whereas the number of frames to be tracked decreases for subsequent key frames. Each video was processed in under 5 minutes. The resulting mesh can be rendered and animated at several hundred frames per second.

4 Results

We report qualitative results in form of 3D mesh animations from videos that allow for certain applications as well as quantitative results in terms of a user study, measurement of reconstruction error for synthetic scenes and a comparison to previous work. On average 4.39% of all strokes in all frames were drawn to reconstruct the sequences; further stroke statistics can be found in the supplemental material.

Figure 10: Cloning of creatures (Fig. 1, Fig. 14) in new views and new poses.

Figure 11: Transfer of a texture from frame *a)* to frame *b)*. Please note the occlusion handling. *c,d)* Transfers of entirely different textures.

Qualitative results Our primary results are 3D mesh animations from skeleton sketches (Fig. 14). This allows for typical applications such as cloning (Fig. 10), texture transfer (Fig. 11) and re-posing (Fig. 12). Our output meshes allow to be fed directly to 3D printers such as the MakerBot Replicator 2 (Fig. 13). While the quality of our animated meshes does not meet high-quality production needs,

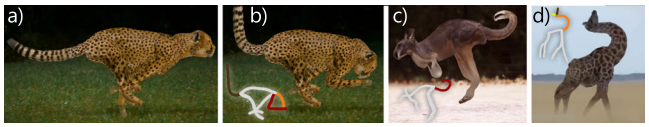

Figure 12: *a)* Original frame. *b,c,d)* Posed results using the shown colored strokes. Please not the Kangaroo's tail in *c)* leaving the image plane as it was drawn shorter.

Figure 13: 3D printing results for the models of Fig. 1 and Fig. 15.

we think that they suffice for most casual applications. To obtain high-quality meshes our results can be used as an initial solution for further automatic and manual mesh processing, offering a valuable source for which the vast efforts of tracking as well as shape fitting have already been carried out and only small corrections and shape details need to be added. The mesthod of Ji et al. [23] or Borosan et al. [6]. Our parametrization is flexible enough to propagate changes on the mesh in one frame to all other frames immediately.

Reconstruction error We rendered animated 3D meshes of a camel and a horse into a video (Fig. 15), painted strokes on it, let the system reconstruct the 3D mesh animation and compared the result to the input in terms of time-averaged intersection-over-union (IoU) in 2D with a resolution of 1150×1000 and in 3D using a voxelization of size 64^3. The 2D-IoU and 3D-IoU are 63.53% and 85.85% for the horse as well as 58.31% and 83.95% for the camel, respectively. Additionally we visualized the accumulated error for every vertex of the reconstructed mesh by averaging the distance transform of the rasterization for the 2D and the voxelization for the 3D case. It can be seen that the 2D error is relatively low except for one part of the camel where the radius was wrongly estimated due to consistent segmentation errors and at the occluded limbs due to imperfect tracking. The 3D error at all limbs is relatively high due to our limb z-offset-heuristic. We found the resulting 3D object to be very similar to the input mesh, even in new views. Some geometric details such as the ears of the camel or the animals' feet details could not be reproduced, as they constitute geometry outside the scope of generalized cylinders.

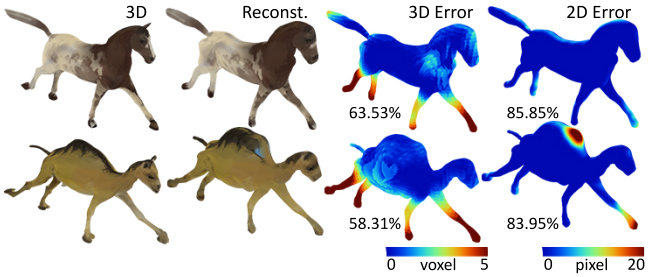

Figure 15: We rendered an animated 3D mesh of a horse *(top)* and a camel *(bottom)* to a video, reconstructed the 3D mesh with our system *(middle)* and compared the outcome to the input using different metrics *(right)*.

User study To evaluate the usefulness of our system we asked nine novice users with previous experience in computer graphics and media production on different levels to use our system and reproduce the results of the first sequence in Fig. 14. After a short introduction to the system using the sequence in Fig. 1 the users had at most five

Figure 14: Results for different video sequences *(rows)* and their key frame strokes (colored strokes). We always show the animated 3D surface at the frame of the video from one view that is similar to the video and from a different one. Note, how the video footage has compression artifacts and low resolution as it is downloaded from community video collections. The total sketching time for such input is below 5 minutes.

minutes to reconstruct the sequence, but most users were satisfied before (3:43 minutes on average). The users used on average 9.5 strokes to reconstruct the sequence which is in agreement to the stroke count used by the authors. We would argue it does not need a formal control group to see that without substantial training users are not capable to produce models of this quality using any available modelling and animation software such as Blender. Users were able to produce animations of a quality similar to the animation produced by the authors, indicating the system can be used by non-experts.

Comparison Our stroke tracking approach relies on appearance and regularization terms commonly used in extensively-investigated optical flow methods. An interesting question could be if methods designed to solve a much more general problem already suffice to solve the tracking problem at hand. To this end, we compared our limb stroke tracking method on a running cheetah sequence of 20 frames to state-of-the-art optical flow methods: SimpleFlow [37], Brox et al. [9], TVL1 [33] and a ground truth for which all strokes were painted manually for all frames. While the construction of a self-made reference could be considered unfair the supplemental video shows that it likely matches what is expected to happen in this scene. We optimized the parameters for each method by hand and only report the best results obtained. The strokes are tracked back and forth starting from the same single key frame for all methods. We use the optical flow offset vectors to track the strokes by accumulating them for every vertex position in every frame. As optical flow does not produce offset vectors for occlusions, only strokes that were completely visible in all frames are used for tracking. An error measure is computed as the sum of absolute distances to the ground truth vertex positions. The accumulated error over all frames is 77.5 for Tao et al., 21.4 for Brox et al.,12.6 for Sanchez Perez et al., and 0.189 for our approach. Optical flow works well for many vertices in many frames, but the sheer number of vertices and frames will quickly result in a disintegrating skeleton. We believe that the main reason for the limited performance of the investigated optical flow methods is the accumulation of small matching errors. As these methods only compute optical flow between neighboring frames this error accumulates rapidly over longer frame ranges. One way to overcome this issue could be to compute optical flow not only between neighboring frames but over longer frame ranges. In practice however it is unclear how to choose these ranges to stabilize the tracking on the one hand and allow for limb deformations on the other hand. Optical flow could be used as a prior in our stroke tracking algorithm but would require additional, expensive preprocessing steps and hence remains future work. We conclude that optical flow, which successfully performs a much more general task, might not be the best tool for solving the skeletal correspondence challenge we addressed using our user interface. Additionally our algorithm allows for interactive performance while the optical flow methods employed use up seconds to minutes to finish per frame.

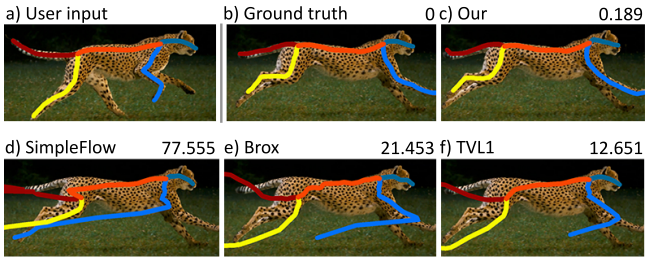

Figure 16: Stroke tracking using different approaches.

5 Limitations and Conclusion

Our system is putting the user in the loop to solve an otherwise heavily under-constrained and hard vision problem. We apply several assumptions that might not hold for certain inputs. Stroke tracking

fails if there are not enough features to be tracked on the object or in the presence of severe occlusions. Occluded parts can be mistaken to be non-occluded, in which case tracking starts using erroneous templates for matching. The regularization terms in Eq. 1 however lead to graceful failures that can be resolved by additional user interaction. Next, segmentation fails if the fore- and background histograms are too similar and salient edges separating the object from its background are missing. Our method is particularly designed to allow tracking of creatures with arbitrary skeletons for which template models might not be available. While our method might also be applicable to humans, the large body of research on this specific topic led to results that likely outperform our method. Our model allows to track limbs of arbitrary complexity and abstracts model knowledge, but a more rigid model could be beneficial and allow for more realistic motion extraction. We have demonstrated reproduction of sequences where the creature as a whole exhibits approximately image-plane-parallel motion. While our approach is capable of handling out-of-plane motion for isolated limbs, such as tails or trunks, as well as limited camera motion and zooming, support for arbitrary camera motion is missing and constitutes the strongest limitation of our algorithm. Surprisingly however, many videos found in online video platforms depicting animal locomotion satisfied our assumptions, enabling the reconstruction of a wide variety of creatures. Handling arbitrary camera motion would not only enable support for a larger variety of videos but the extra views add additional knowledge about the true 3D shape of the object to the system. Exploiting these cases in our algorithms remains the most promising direction for future work. Shape reconstruction using generalized cylinders is a valid assumption in many cases, but fails in case of certain cylindrical cross sections, such as for bodies and fins of fish, or complicated branching structures. Again in these cases, multiple views can help to find the true shape of the limbs. Nevertheless, many creatures look plausible in views substantially different from the side view (Fig. 10). Videos courtesy of National Geographic and BBC Worldwide.

REFERENCES

[1] A. Agarwala, A. Hertzmann, D. H. Salesin, and S. M. Seitz. Keyframe-based tracking for rotoscoping and animation. *ACM Trans. Graph.*, 23(3):584–91, 2004.

[2] I. Akhter, Y. Sheikh, S. Khan, and T. Kanade. Nonrigid structure from motion in trajectory space. In *Proc. NIPS*, pages 41–48, 2008.

[3] B. Allen, B. Curless, and Z. Popović. The space of human body shapes: reconstruction and parameterization from range scans. *ACM Trans. Graph. (TOG)*, 22(3):587–94, 2003.

[4] O. Arikan and D. A. Forsyth. Interactive motion generation from examples. In *ACM Trans. Graph.*, volume 21, pages 483–490, 2002.

[5] O. Arikan, D. A. Forsyth, and J. F. O'Brien. Motion synthesis from annotations. *ACM Trans. Graph.*, 22(3):402–8, 2003.

[6] P. Borosán, M. Jin, D. DeCarlo, Y. Gingold, and A. Nealen. Rigmesh: Automatic rigging for part-based shape modeling and deformation. *ACM Trans. Graph.*, 31(6):198:1–198:9, Nov. 2012.

[7] C. Bregler, A. Hertzmann, and H. Biermann. Recovering non-rigid 3D shape from image streams. In *Proc. CVPR*, volume 2, 2000.

[8] C. Bregler, L. Loeb, E. Chuang, and H. Deshpande. Turning to the masters: motion capturing cartoons. *ACM Trans. Graph.*, 2002.

[9] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. High accuracy optical flow estimation based on a theory for warping. In *ECCV*. 2004.

[10] A. Buchanan and A. W. Fitzgibbon. Interactive feature tracking using k-d trees and dynamic programming. In *Proc. CVPR*, 2006.

[11] T. Cashman and A. Fitzgibbon. What shape are dolphins? Building 3D morphable models from 2D images. *IEEE PAMI*, 35(1):232–44, 2013.

[12] T. Chen, Z. Zhu, A. Shamir, S.-M. Hu, and D. Cohen-Or. 3Sweep: Extracting editable objects from a single photo. *ACM Trans. Graph.*, 32(6):195:1–195:10, 2013.

[13] B. Choi and C. Lee. Sweep surfaces modelling via coordinate transformation and blending. *CAD*, 22(2):87–96, 1990.

[14] T. F. Cootes, D. H. Cooper, C. J. Taylor, and J. Graham. Trainable method of parametric shape description. *Image Vision Comput.*, 10(5):289–94, 1992.

[15] P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *Proc. SIGGRAPH*, pages 11–20, 1996.

[16] L. Favreau, L. Reveret, C. Depraz, and M.-P. Cani. Animal gaits from video. In R. Boulic and D. K. Pai, editors, *Symposium on Computer Animation*. The Eurographics Association, 2004.

[17] J. Gall, C. Stoll, E. De Aguiar, C. Theobalt, B. Rosenhahn, and H.-P. Seidel. Motion capture using joint skeleton tracking and surface estimation. In *CVPR 2009*, pages 1746–1753, June 2009.

[18] R. Garg, A. Roussos, and L. Agapito. Dense variational reconstruction of non-rigid surfaces from monocular video. In *Proc. CVPR*, pages 1272–9, 2013.

[19] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. *Proc. SIGGRAPH*, pages 43–54, 1996.

[20] T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy: A sketching interface for 3D freeform design. In *Proc. SIGGRAPH*, pages 409–16, 1999.

[21] S. Izadi et al. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In *Proc. UIST*, pages 559–568, 2011.

[22] A. Jain, T. Thormählen, H.-P. Seidel, and C. Theobalt. Moviereshape: Tracking and reshaping of humans in videos. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 29(6):148, 2010.

[23] Z. Ji, L. Liu, and Y. Wang. B-mesh: A modeling system for base meshes of 3d articulated shapes. *Computer Graphics Forum*, 2010.

[24] G. Johansson. Visual perception of biological motion and a model for its analysis. *Perception & psychophysics*, 14(2):201–11, 1973.

[25] O. A. Karpenko and J. F. Hughes. SmoothSketch: 3D free-form shapes from complex sketches. In *ACM Trans. Graph. (Proc. SIGGRAPH)*, volume 25, pages 589–98, 2006.

[26] J. J. Koenderink. What does the occluding contour tell us about solid shape. *Perception*, 13(3):321–30, 1984.

[27] V. Kraevoy, A. Sheffer, and M. van de Panne. Modeling from contour drawings. In *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, SBIM '09, pages 37–44, New York, NY, USA, 2009. ACM.

[28] G. Maestri. *Digital character animation 3*. New Riders, 2006.

[29] I. Matthews, T. Ishikawa, and S. Baker. The template update problem. *IEEE PAMI*, 26(6):810–5, 2004.

[30] W. Matusik, C. Buehler, R. Raskar, S. J. Gortler, and L. McMillan. Image-based visual hulls. In *Proc. SIGGRAPH*, pages 369–374, 2000.

[31] M. Prasad and A. Fitzgibbon. Single view reconstruction of curved surfaces. In *Proc. CVPR*, volume 2, pages 1345–54, 2006.

[32] C. Rhemann, A. Hosni, M. Bleyer, C. Rother, and M. Gelautz. Fast cost-volume filtering for visual correspondence and beyond. In *Proc. CVPR*, pages 3017–24, 2011.

[33] J. Sanchez Perez, E. Meinhardt-Llopis, and G. Facciolo. TV-L1 Optical Flow Estimation. *Image Processing On Line*, 3:137–150, 2013.

[34] K. Shoemake. Animating rotation with quaternion curves. In *ACM SIGGRAPH Computer Graphics*, volume 19, pages 245–54, 1985.

[35] S. N. Sinha, D. Steedly, R. Szeliski, M. Agrawala, and M. Pollefeys. Interactive 3D architectural modeling from unordered photo collections. *ACM Trans. Graph.*, 27(5):159, 2008.

[36] N. Snavely, S. M. Seitz, and R. Szeliski. Photo Tourism: Exploring photo collections in 3D. *ACM Trans. Graph.*, 25(3):835–846, 2006.

[37] M. Tao, J. Bai, P. Kohli, and S. Paris. SimpleFlow: A non-iterative, sublinear optical flow algorithm. In *Comp. Graph. Forum (Proc. EG)*, volume 31, pages 345–53, 2012.

[38] X. Wei and J. Chai. VideoMocap: Modeling physically realistic human motion from monocular video sequences. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 29(4):42:1–42:10, 2010.

[39] X. Xu, L. Wan, X. Liu, T.-T. Wong, L. Wang, and C.-S. Leung. Animating animal motion from still. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 27(5):117:1–117:8, 2008.

[40] R. C. Zeleznik, K. P. Herndon, and J. F. Hughes. SKETCH: an interface for sketching 3D scenes. In *Proc. SIGGRAPH*, pages 163–170, 1996.

[41] Y. Zheng, X. Chen, M.-M. Cheng, K. Zhou, S.-M. Hu, and N. J. Mitra. Interactive images: Cuboid proxies for smart image manipulation. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 31(4):99, 2012.