Roto++: Accelerating Professional Rotoscoping using Shape Manifolds

Wenbin Li [1] Fabio Viola [2,*] Jonathan Starck [2] Gabriel J. Brostow [1] Neill D. F. Campbell [3]

University College London [1] The Foundry [2] University of Bath [3]

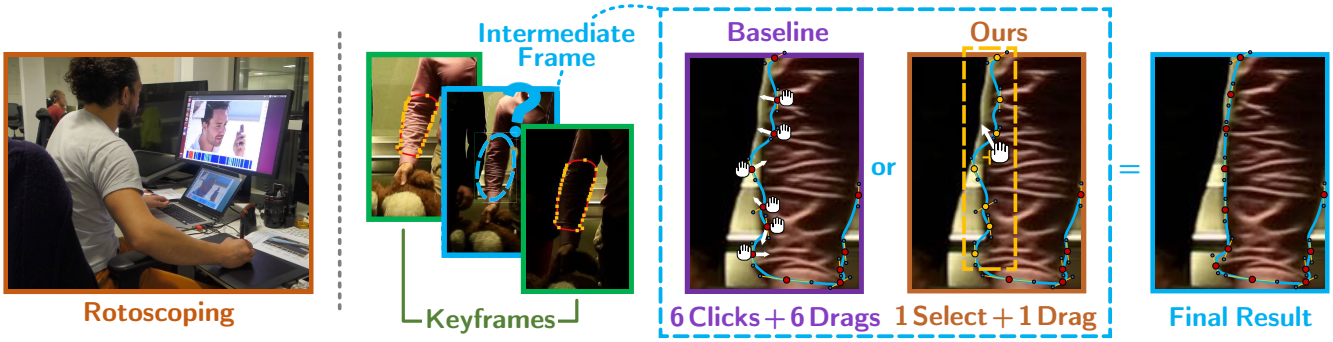

Figure 1: *An example of our new Roto++ tool working with a professional artist to increase productivity. The artist has already specified a number of keyframes but is not satisfied with one of the intermediate frames. Under standard baselines, correcting the erroneous curve requires moving the individual control points of the spline. Using our new shape model we are able to provide an Intelligent Drag Tool that can generate likely shapes given the other keyframes. In our new interaction, the user simply selects the incorrect points and drags them all towards the correct shape. Our shape model then correctly proposes the new control point locations, allowing the correction to be performed in a single operation.*

Abstract

Rotoscoping (cutting out different characters/objects/layers in raw video footage) is a ubiquitous task in modern post-production and represents a significant investment in person-hours. In this work, we study the particular task of professional rotoscoping for high-end, live action movies and propose a new framework that works with roto-artists to accelerate the workflow and improve their productivity.

Working with the existing keyframing paradigm, our first contribution is the development of a shape model that is updated as artists add successive keyframes. This model is used to improve the output of traditional interpolation and tracking techniques, reducing the number of keyframes that need to be specified by the artist. Our second contribution is to use the same shape model to provide a new interactive tool that allows an artist to reduce the time spent editing each keyframe. The more keyframes that are edited, the better the interactive tool becomes, accelerating the process and making the artist more efficient without compromising their control. Finally, we also provide a new, professionally rotoscoped dataset that enables truly representative, real-world evaluation of rotoscoping methods. We used this dataset to perform a number of experiments, including an expert study with professional roto-artists, to show, quantitatively, the advantages of our approach.

CR Categories: I.3.7 [Computer Graphics]: Methodology and Techniques—Interaction Techniques; I.4.1 [Image Processing and Computer Vision]: Scene Analysis—Shape

Keywords: Rotoscoping, Video Segmentation, Manifold Models, Shape Models

1 Introduction

Visual effects (VFX) in film, television, and even games are created through a process called compositing in the post-production industry. Individual shots are broken down into visual elements that are then modified and combined, or seamlessly integrated with computer generated (CG) imagery. This process of assembling each shot from different sources requires teams of highly skilled artists working on a range of problems from removing unwanted objects such as rigging; modifying appearance to create a specific look; combining 2D live action elements from different clips, to integrating 3D elements to augment a set; or adding digital effects. A fundamental requirement of this work is that it is procedural: each effect has to be shared, reviewed, and modified with multiple iterations between artists, before being reproduced when rendering the final composite.

Rotoscoping is the technique used in live-action movies to separate the elements in each shot and allow artists to perform these compositing tasks. Imagine a set augmentation where a reflection in a window is changed or a set extension where futuristic buildings are added in the background; these tasks both require separation of all the foreground layers in the shot so that the background can be updated.

Professional Rotoscoping Rotoscoping itself is a time-consuming, manual task. It requires the set up of complex shapes to decompose a shot into the different elements and then tracking or animating the shapes to match the shot. An experienced artist can rotoscope on average 15 frames per day, depending on the complexity of the scene. In big-budget movies, effect-rich shots are routinely rotoscoped to separate every element ready for compositing, and in 2D to 3D conversion all shots must be rotoscoped to allocate each item a specific depth in the scene. This rotoscoping work is often outsourced to dedicated artists, a lengthy process that creates a bottleneck and dependency in post-production pipeline. These artists are highly trained and very demanding of the tools they use; they

*Fabio Viola is currently affiliated with Google DeepMind

SIGGRAPH 2016 Technical Paper, July 24 - 28, 2016, Anaheim, CA
ISBN: 978-1-4503-4279-7/16/07
DOI: http://dx.doi.org/10.1145/2897824.2925973

require complete control over the shapes and contours they produce since they will often have to draw-in details that are obscured in the shot by complications such as motion blur or smoke.

Why not Green Screen? The simplest way to achieve the separation of foreground elements in live-action footage is by using a green (or blue) screen background so that the foreground elements can be extracted through chroma-keying, for example using the work of Smith and Blinn [1996]. This, however, requires the placement of a screen on-set which interrupts production; it is much simpler to shoot in a natural environment. Also, there is risk that a screen, if poorly located or badly lit, will result in blue or green color spill on the foreground. Furthermore, a screen cannot be used to separate multiple foreground elements or layers which may be present. Thus, there is a trade-off between the creative impact of shooting against a green screen and the cost of rotoscoping in post-production. There is a clear need to be able to shoot in natural scenes without incurring large post-production costs for rotoscoping.

Why not Brush-Based Matting? Image matting is a well studied problem in computer vision. Many techniques have been proposed based on either a *trimap* that separates a shot into a known foreground and known background region with an uncertain region for the matte, or through a scribble interface where a user can annotate a shot to define the foreground and background regions. The Video SnapCut method of Bai *et al.* [2009] is an example of the state-of-the-art in this approach. While there are many use cases where these approaches perform well, brush or scribble based methods are universally not used by artists in our target area of *high-end VFX post-production*. This is due to the lack of procedural control of the generated alpha matte that this interaction paradigm offers. The workflow quickly degrades into an iterative process of defining more detailed constraints for the optimization to improve the quality of the matte, at which point it is simply faster to paint the alpha matte by hand.

Keyframe Paradigm The standard interface paradigm used by professional roto-artists is that of keyframing. In this framework, the artist picks a subset of the video frames in a shot to be keyframes. In each of these frames they create a set of close spline curves around the same visual element. This is a laborious task since it often involves moving manually the many individual control points and tangent vectors that define each spline in each keyframe. The control points of the splines are then interpolated for the interim frames between the keyframes [Bratt 2011].

This is time consuming since a number of keyframes may need to be drawn to achieve the required quality of segmentation. In some circumstances, the number of keyframes required can be reduced by using tracking to try to improve the interpolation estimates between keyframes, for example the work of Agarwala *et al.* [2004].

Areas for Improvement This leaves us with two main areas to improve efficiency while still maintaining the workflow that the trained artists are used to and which affords them the control and precision they require. The first is to reduce the number of keyframes required to achieve a high accuracy across a wide range of shot complexities. The second is to reduce the artist editing time for each keyframe.

Contributions We provide following three main contributions:

1. **Shape Manifold Framework:** Working in the existing keyframe paradigm, we develop a *shape model* as the artists add successive keyframes. This model is used to improve the output of traditional interpolation and tracking techniques, reducing the number of keyframes specified by the artist. Additionally, our model allows us to analyze the shot in realtime and suggest which keyframes the artists should edit next. This feedback process further reduces the number of keyframes needed across a variety of different shots.
2. **Intelligent Drag Tool:** After consultation with professional roto-artists we have developed a new interactive tool for editing individual keyframes. Traditionally, global models have been avoided in the editing process since artists need to maintain control and do not like local edits to have non-local effects. Through careful design, we have produced a new tool that exploits the "global" shape information across the shot, from our shape model, with a locally constrained editing process. This new tool reduces the amount of time the artists needs to spend editing each keyframe. The more keyframes that are edited, the better the interactive tool becomes, accelerating the process and making the artist more efficient without compromising their control and precision.
3. **Professional Rotoscoping Dataset:** We provide a new, professionally rotoscoped dataset that enables truly representative, real-world evaluation of rotoscoping methods. It comprises a large number of shots over a range of complexity ratings representative of a high-end, live-action movie. The ground truth data was generated professional artists and comprises over 700 person-days of rotoscoping effort. We used this dataset to perform a number of experiments, including an expert study with professional roto-artists to show, quantitatively, the advantages of our approach.

2 Background

Both the interfaces and algorithms underpinning rotoscoping build from research in several domains. We now outline the main connections to color segmentation, contour tracking, and shape models. In examining these works, one should keep in mind the variety of everyday rotoscoping challenges, including non-rigid objects, occlusion and out-of-plane rotation, textured surfaces, changing illumination/shadows, and blur caused by motion or defocus. In practice, these situations regularly challenge automatic algorithms, so roto-artists are distrustful of automation, and insist on override interfaces [Bratt 2011].

2.1 Related Work

Specifying a Region in 2D or 3D Rotoscoping can be viewed as semi-supervised segmentation: the user indicates which region they wish to isolate, in one or more frames of the image sequence. Ideally, many unlabeled (*i.e.* untouched) frames would get segmented to the user's satisfaction, on the basis of a few frames that they did label. In practice, the user keeps labeling further frames, until the region is correctly segmented throughout. This whole process benefits from giving the user a quick interface for labeling. The target region is labeled in most interfaces either by having the user trace the region's *boundary*, *e.g.* [Mortensen and Barrett 1995; Rzeszutek et al. 2009], or by painting brush-strokes on the region *interior* [Rother et al. 2004].

Complex boundaries can be tedious to trace, so brush-based methods are appealing for their interface: a user must only tag enough pixels as foreground or background to build up the respective color models. Subr *et al.* [2013] even do this with some robustness to mis-labeled pixels. Soft Scissors by Wang *et al.* [2007] gives users a hybrid option, where the user's tracing of the region can be approximate, and the fast segmentation feedback allows for quick corrections, compared to the drawing of a trimap that is optimized later [Levin et al. 2008]. In turn, trimap-driven segmentation is its own subfield, which strives to segment and model transparency. The Alpha Matting Evaluation benchmark [Rhemann et al. 2009] has an evolving leaderboard of top algorithms for image matting, and [Erofeev et al. 2015] is similar for video matting, evaluated on artificially messy

trimaps derived from green screen sequences.

The work of Boykov and Jolly [2001] is one of the cornerstone methods for treating semi-supervised image and video segmentation as a single graph regularization problem. The assumptions is that the inferred labels of neighboring pixels should be similar, unless they differ in intensity. Other graph-cut based methods such as [Kwatra et al. 2003; Wang et al. 2005] and geodesic distance optimizing methods such as [Bai and Sapiro 2007] and [Criminisi et al. 2010] have explored other models and pairwise term priors.

These single-optimization methods lead to video segmentations that are adequate for simple video effects. Possibly surprising, the more work-intensive direct boundary-tracing is almost the only modality employed by professional rotoscoping artists [SilhouetteFX]. This is probably because object boundaries are often camouflaged (*e.g.* no visible boundary between the lower and upper arm), and many strong image gradients can appear as object boundaries, but are in fact lighting/shadow boundaries or patterned textures. We give our users such an unassisted boundary-tracing interface for making Bezier spline outlines, akin to the market-dominant software interfaces. Post-production pipelines completely depend on roto-curves being in spline form, but much of our algorithm could proceed unchanged if the boundary-tracing was assisted.

Tracking a Region It is also possible to split region-specification and region-tracking into two distinct steps. However, if the region's boundary was specified, tracking can dictate how the boundary evolves over time. For example, [Grundmann et al. 2010] and [Wang and Collomosse 2012] automatically construct hierarchical graphs in the space-time volume. On the other hand, segmentations based on the superpixels are constrained to follow superpixel boundaries, which frequently does not coincide with the object boundary. Furthermore, editing is limited to selecting and merging existing superpixels. Boundaries can also be propagated using optical flow estimates [Tsai et al. 2012; Santosa et al. 2013; Li et al. 2013], or tracked keypoints [Lebeda et al. 2016] (combined with color models in [Li et al. 2005]). Trimap propagation using optical flow has featured in video matting [Chuang et al. 2002], and Wang and Cohen [2007] provide a good overview of methods to that point.

Nonrigid structure-from-motion techniques can also be used to help with tracking. The work of Torresani *et al.* [2008] makes use of a different representation to recover a low-dimensional model for 2D non-rigid point tracks. These models have also been used to improve tracking results [Torresani et al. 2001].

Tracking Shape Boundaries Video SnapCut [Bai et al. 2009] propagates a user's sketched boundary frame-to-frame using both interest-point tracking and optical flow. They train localized appearance/shape classifiers that hug the region's boundary. There, Bai *et al.* observed that users preferred when their edits changed segmentation only in nearby frames, without affecting a global color-model, because that required repeatedly revisiting already-approved frames in the sequence. In contrast, we update the global model (though it is a shape model instead of a color model), without requiring the user to revisit previous frames. SnapCut was released as RotoBrush in Adobe After Effects CS5, and we discuss its usage as part of our evaluation.

Among academic efforts toward rotoscoping, [Agarwala et al. 2004] bears the most resemblance to modern production tools, and to our own approach. Two key features of that work were its interpolation of user-specified keyframes, and that the interpolation was guided by shape and image terms. The image term is based on [Lucas and Kanade 1981], and allows the user to specify, within the search window, which side of the curve contains the foreground region that should respect color constancy. This automatic tracking must be initialized, and then reinitialized by the user, when it falls off the region boundary. The shape term acts to smooth out the abrupt changes in time: it prefers for the curve to preserve its length over time, and for both the changes in curvature and the velocity of points on the curve to be small.

These are generic objectives, but they help the system interpolate between keyframes, without too much parameter tuning. Similarly, Blender's and Nuke's rotoscoping functionality leverages a planar tracker as the image term, often used to stabilize parts of the shot while the user edits roto-curves to handle remaining motions. Our approach integrates the best available tracker as the image term, and uses a global (per-shot) shape model that learns progressively as keyframes are added. The Blender [Blender Online Community 2015] planar tracker that we use in our tool is an extended version of the Benhimane and Malis [2004] tracker using the fast ceres solver [Agarwal et al. 2015].

Shape Manifolds There is a large literature on subspace models for modeling shapes. In the 3D domain active shape models [Cootes et al. 1995] fit a low dimensional linear model to a set of fiducial points to capture a subspace of deformations. This model has been applied to object tracking in video [Baumberg and Hogg 1994]. Similarly, tracking and reconstruction using a learned 3D shape model has recently been used in motion capture [Loper et al. 2014]. Particularly related to our approach are methods that use the Gaussian Process Latent Variable Model (GP-LVM) (see Section 4) to build manifolds of shape. The GP-LVM is an unsupervised learning method that creates generative models and has been shown to be effective in graphics tasks, for example style generation in inverse kinematics [Grochow et al. 2004].

The use of the GP-LVM in shape modeling can be categorized by the choice of representation. Prisacariu and Reid [2011] used shape manifold models for 2D tracking of curves using elliptical Fourier components; they focused on real-time automatic tracking and have no mechanism for user guidance or interaction making the leap to rotoscoping unclear. The same representation was used by Turmukhambetov *et al.* [2015] to build a joint manifold of object contour and ellipses to assist users with sketching. Object outlines can also be encoded as signed distance transforms as in the work of Dame *et al.* [2013] which uses shape models in 3D as surface priors. Finally, Campbell and Kautz [2014] demonstrated a shape model that represented the curve directly as a poly-line although they considered the space of font outlines which are originally composed of Bézier curves. This is closest to our model, where we identify that the roto-curves are typically manipulated as a set of closed Bézier curves. We show that we can use the GP-LVM to model the curves directly in Bézier space resulting in a very efficient learning and synthesis algorithm that allows our framework to update and respond to user events in realtime.

2.2 Existing Rotoscoping Approaches

We now turn our attention to the keyframe and spline based workflow mentioned in the introduction. This is the framework used currently by professional roto-artists in film production [Bratt 2011]. As part of our research for this work, we conducted interviews with a number of professional roto-artists from different post-production houses. This survey identified the actual techniques used in the real world and the artists' requirements for rotoscoping tool they would use.

Standard Workflow To rotoscope an element in a shot, an artist creates a set of closed splines that define the component shapes for the element. To rotoscope a person, for example, roto-shapes are created for each articulated element such as the upper and lower arms, hands, fingers and face. The artist then manipulates these shapes independently to match the movement in the shot. The rotoscoping task starts with an analysis of the shot to understand the movement and how to break up the scene into individual shapes. A

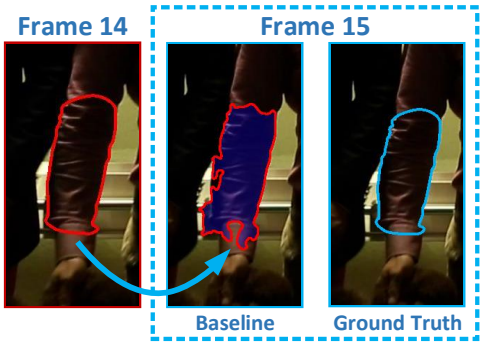

Figure 2: *The difficulties of appearance based approaches: an example of the state-of-the-art mask propagation [Fan et al. 2015]. We propagate the GT curve from the reference frame (Frame 14) to the adjacent frame (Frame 15). Comparing to the ground truth on the right. The method gives 16.7 average pixel error on a full HD image, and takes 23.45 seconds for computation.*

key pose is then selected that defines the outline of each shape, the artist then works outwards to animate the shape.

Keyframe Interpolation Initially the animation is performed using a rigid body transformation using handles to define the translation, rotation, scale and skew for the shape as a whole. Non-rigid deformation is then created by selecting and transforming individual control points. Where possible this may be done by modifying groups of points. There are two standard approaches to creating the animation, either by matching key poses over the shot then successively subdividing and adding intermediate keyframes where necessary, or by proceeding sequentially through a shot creating a keyframe every n^{th} frame. The remaining frames in the shot are then linearly interpolated from their neighboring keyframes.

Acceleration with Tracking Efficiencies in the workflow can be obtained if the shot allows some form of tracking to be performed. In the ideal case of a rigid scene observed by a match-moved camera, geometric constraints can be exploited to correctly move the roto-shapes. Such scenes are designated as easy shots and are not considered further here. A more realistic scenario is that of a moving element which is distinct in the shot. In this case, it might be possible to use 2D to drive groups of control points or planar tracking could be used to define a rigid body animation.

Tracking usually uses a gradient-descent based optimization to match image appearance across image frames [Lucas and Kanade 1981]. Planar tracking uses a similar framework, where pixel regions are tracked based on a parameterised warp between frames for pixels inside a mask. These approaches can extend the search range and accelerate tracking by matching pixels from coarse-to-fine in image pyramids [Baker and Matthews 2004]. If clearly defined edges are available in the video, direct curve tracking is an effective option, for example [Agarwala et al. 2004].

Limitations Although tracking approaches help under some circumstances, often shots are too complex for them to be used reliably and artists are forced to fall back on keyframe interpolation [Bratt 2011]. For example, point tracking on individual control points is often brittle; the control point on a contour may not have a stable appearance and the tracker will be prone to drift. On the other hand, planar tracking is often over constrained and real-world shots containing deformable elements, such as faces, violate their constraints. Direct curve tracking requires edge fitting but edges are not necessarily distinct which forces the use of strong regularization that loses details.

Figure 2 provides an example of the difficulties of relying on appearance with the complex roto shots. It shows a mask propagation result using the state-of-the-art method of Fan *et al.* [2015]. The errors arise because the real-world shot contains a lack of features, illumination changes and non-rigid deformation that make appearance unreliable. These common difficulties may affect the interpolation results in the rotoscoping.

Our Approach To overcome these limitations, we identify that appearance alone is often unreliable and therefore place a strong emphasis on modelling shape. We take appearance information from real-time tracking and combine it with a novel manifold shape regularisation to produce a new *generative* model of rotoscoping. Unlike previous approaches, our single, unified model does not simply smooth or interpolate between keyframes, but is capable of extrapolating and generating new shapes; these are driven by the tracker but still representative of the overall shape statistics to prevent drift. Our results counter the conventional interaction adage that one cannot use global models (from multiple keyframes) since artists want edits to apply locally; in fact, the probabilistic model allows us to provide novel suggestions and interactions that save artists' time.

2.3 Key Artist Requirements

There are several important requirements, put forward by artists, to support the creation of roto-shapes for VFX work in film post-production.

1. The tools need to retain the existing workflow that artists are familiar with; they need to augment and accelerate the rotoscoping process.
2. The workflow should be fast and intuitive; artists are accustomed to working with an interactive tools with instantaneous feedback.
3. The result needs to be procedural and reproducible so that they can be shared, reviewed, and iterated between artists.
4. The results need to be predictable and controllable; when an artist works through a shot there must be no recomputation that changes a previously finalized frame.

2.4 Baselines

Finally, we summarize our findings on existing approaches by proposing the baselines to which we will compare our new rotoscoping tool:

1. **Keyframe Interpolation:** This straight forward approach is still commonly used since it is predictable and can be used when other approaches fail.
2. **Planar Tracking:** This is the most commonly used tracker for a range of shots and artists are usually able to predict when tracking will break down and take appropriate steps.

In addition to these two baselines we consider the most relevant previous academic work to ours, the curve based tracker of Agarwala *et al.* [2004]. To compare against the other dominant paradigm for rotoscoping, we also investigate the interaction of Video SnapCut [Bai et al. 2009], in particular as the RotoBrush tool in Adobe After Effects.

3 Our Roto++ Tool

3.1 Design

We designed a new tool to work with artists to accelerate the rotoscoping workflow while also overcoming some of the main limitations (Section 2.2) and respecting the artists' requirements (Section 2.3).

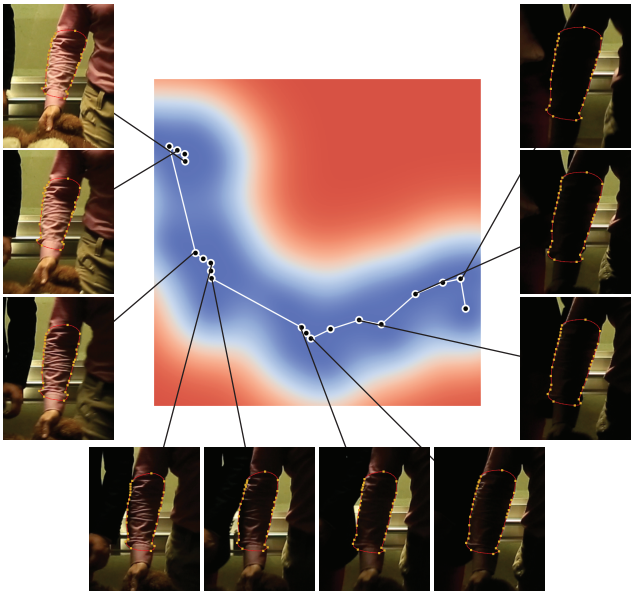

Figure 3: *An example manifold for the arm roto sequence. As we move across the manifold the shape of the arm changes smoothly. Even though the roto-curve contains 87 control points (to account for the ripples in the shirt) the sequence can be perfectly recovered from a 2D shape manifold. (Note: we observe a complete change in object appearance as the character moves from light to shadow; this sort of sequence represents a significant challenge to techniques that track edges or make use of color models).*

Tracker Drift Tracking is known to reduce required keyframe count when it works well but the most common failure mode is for the tracker to drift in difficult situations. This yields roto-shapes which depart significantly in shape from the edited keyframes. The strong regularization of Agarwala *et al.* [2004] can help to prevent drift but limits the output to smooth interpolations of the keyframes.

Shape Manifold To overcome this limitation, we propose to combine the output of the tracker with a strong prior on the possible roto-shapes. We learn a statistical shape model from the existing keyframes the artist has created. We then constrain the intermediate frames to be as close to the tracker output as possible while still being valid shapes from the shape model. This means that if the tracker output drifts away from the realm of reasonable roto-shapes, the output presented to the user will remain reasonable. As the user adds more keyframes, our shape model becomes more and more accurate, resulting in even better estimates for the intermediate roto-shapes.

This model takes the form of a generative, low-dimensional manifold. The original keyframes are points in this space and our hypothesis is that other regions of the manifold will generate distinct but similar shapes that are likely to include the correct shapes for the intermediate frames. Figure 3 provides an illustrative example of our shape manifold. The roto-curve defining the lower arm has been embedded in a 2D manifold with the highlighted points corresponding to the locations that generate the different roto-shape in each frame. The line connecting the points together denotes the passage of time from frame to frame; we note that the roto-shapes lie on a smooth trajectory.

Choice of Manifold Model We use a Gaussian Process Latent Variable Model (GP-LVM) [Lawrence 2005] as part of our global model of the joint probability between the control points both within and between frames since it is generative, Bayesian and non-linear.

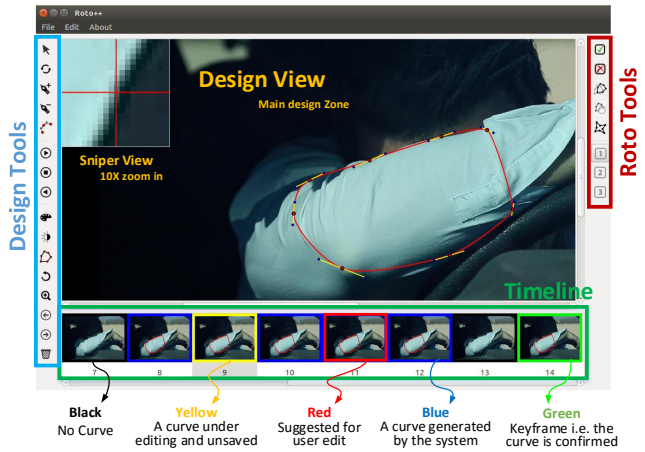

Figure 4: *A screenshot of our interactive tool **Roto++**. This user interface consists of a design view, design tools, roto tools and a timeline. This timeline encodes our frame selection feedback by using colored borders. A full user guide to our tool is included in the supplemental material.*

Previous approaches only regularise trackers with local proxies such as smoothing over neighboring keypoints in time and space. Linear subspace models, such as ASMs, are trained with a large dataset. Our approach is targeted towards few training examples (the keyframes), benefiting from being Bayesian, and the non-linearity is known to capture more variance in fewer dimensions [Prisacariu and Reid 2011]. Our model actually subsumes linear models and is more general. Other models, such as [Agarwala et al. 2004], are neither Bayesian nor generative and therefore cannot be used to provide user suggestions or the intelligent drag tool.

Keyframe Recommendation Another advantage of constraining our predicted roto-shapes to come from our shape model is that we can identify frames when the tracking result departs heavily from our shape model. This could mean one of two things, either the tracker has drifted and needs to be reinitialized, or the shape model is not sufficiently well-defined to include all the valid shapes. Thankfully, both of these situations are remedied by the user labelling the frame as a keyframe. To make use of this result, we provide the artist with helpful feedback. We suggest which keyframe will most help the most to improve the tracker, the shape model, or both to produce better interim shapes.

To update the shape manifold it is necessary to know when a new keyframe is added. We therefore asked the user to formally specify when they have finished editing a keyframe. When a new keyframe is added, the shape model is updated and the tracker is recalculated, all in realtime. This improves the unedited curves and updates the frame recommendation system. Once a frame is labeled as a keyframe it will never be changed by the system ensuring that future editing will never corrupt a checked result.

3.2 User Interface

We implemented an interactive tool, *Roto++*, to evaluate our approach, as shown in Figure 4. The interface aims to provide a familiar look and feel to existing tools while adding our advanced functionality behind the scenes. Our Bézier curve based tool consists of a *Design View* and a collection of *Design Tools* and *Roto Tools* brought together with a *Timeline*:

- *Design Tools:* a common subset of curve-drawing operations of leading commercial software.
- *Roto Tools:* a range of tools for setting keyframes, getting instant feedback from the solver, and for some under-the-hood

roto-related features.

- *Timeline:* a thumbnail-sized view of the shot. Color coding on a thumbnail's border shows the status of that frame. We also use this view to indicate to the artist our recommendation for the next frame to edit to improve the result.

For a typical use case, when a new project is created, all the thumbnails of *Timeline* are labeled as ***Black***. If the user edits any point or curve, the related thumbnail is then labeled as ***Yellow***. Once the user confirms a curve, the related frame is considered as a new keyframe (***Green***). If more than one keyframes are annotated, the system automatically generates curves for all the frames in between (***Blue***). And the system also recommends frames (***Red***) for attention from the user.

As well as the icons on the toolbars, all of our tools were accessible via shortcut keys which were configurable and set to emulate industry standard tools. For speed of operation, roto-artists make use of pen tablets, rather than a mouse, for input, so the tool was configured to operate in a similar fashion to standard tools. A user guide for the features of the Roto++ tool is provided in the supplemental material.

Comparison Modes To provide a comparison to the baseline methods discussed in Section 2.4, we built in three different modes of operation into the tool.

- *Mode 1:* Our approach; all Roto++ features.
- *Mode 2:* Planar tracking forwards and backwards.
- *Mode 3:* Linear interpolation.

Instrumentation To provide a detailed evaluation of our method, the tool is highly instrumented to maintain a detailed log of all the user operations performed down to the level of individual mouse operations. Accurate timestamps were recorded allowing us to determine the time spent performing different operations. We also logged the current state of the roto-curves periodically to allow us to determine the accuracy of the roto output as a function of artist time expended.

3.3 Interaction

The features presented in Section 3.1 aim to reduce the number of keyframes that an artist needs to supply to produce an accurate output. While this is clearly advantageous, it is only half the story. To further improve efficiency, we would also like to reduce the amount of time spent labeling each keyframe. Beating the baseline is a challenging task since artists are highly trained and have, in many cases, years of experience using these techniques. Furthermore, they have exacting requirements on interaction, as discussed in Section 2.3; the most relevant are the need for intuition and instantaneous feedback combined with predictable results that do not corrupt previously edited results.

The limiting factor on the editing time of a keyframe for baseline methods occurs when deformable shape changes occur that cannot be accurately tracked. In most cases, the artist is then forces to edit the roto-curve control points or tangent vectors in small groups or individually. This can require a very large number of mouse operations to select the points in turn and move them to their correct locations. This is illustrated by the baseline operation in Figure 1.

Intelligent Drag Mode To assist with this editing, we can once again exploit our shape manifold. Once the manifold has been trained, new shape proposals can be generated very efficiently from locations on the manifold. This means that we can generate new sets of plausible shapes to match any input from the user. Figure 5 provides a toy example of this operation using a variety of different shapes as example keyframes. As the user selects a point on the curve and drags it, our solver uses the manifold to suggest the new best fitting shape in realtime.

When used in Roto++, we provide further control by letting the artist first select which control points on the curve they would like to update. The other points will remain fixed ensuring the requirement that if a user is happy with part of the outline, another editing operation will not corrupt it. Once the points to move are selected, they can be dragged to a new location. In realtime we run a cut-down version of our tracking solver which replaces the tracker result with the new curve location under the drag operation and then solves for the new shape recommended by the manifold. Figure 1 shows the same result achieved by a large number of control point moves being performed in a single operation with our intelligent drag tool. The operation of the tool is perhaps more clearly demonstrated in the video included in the supplemental material.

4 Technical Approach

In this section we describe how we implement the new features in our Roto++ tool described in Section 3. We first describe the notation used throughout this section. Subsequently we detail how we obtain our shape manifold from a set of keyframes edited by the artist. We then provide details on how to combine our shape manifold with an existing tracker to create the Roto++ solver that estimates the roto-shapes for the unlabeled frames. Finally we show how this solver may be used to provide the intelligent drag tool.

4.1 Notation

Table 3 in Appendix A details the notation used in this section. Throughout, we assume that we are operating on a single, closed roto-curve in a single shot. Our results can be applied in parallel to multiple roto-curves in a straight forward fashion. We also assume that there is no distinction between an interpolation control point and an tangent control point on a Bézier curve. Our closed roto-curves are made up of a closed sequence of cubic Bézier curves, each of which contains 4 control points with the last control point of the curve forming the first control point of the subsequent curve.

The k^{th} keyframe spline is denoted U_k where

$$U_k = \begin{bmatrix} U^{(k)}_{x,1} & U^{(k)}_{x,2} & U^{(k)}_{x,3} & \dots & U^{(k)}_{x,M} \\ U^{(k)}_{y,1} & U^{(k)}_{y,2} & U^{(k)}_{y,3} & \dots & U^{(k)}_{y,M} \end{bmatrix} \in \mathbb{R}^{2\times M} . \qquad (1)$$

This is illustrated in Figure 13 in Appendix A. The output splines are given as $\{Y_n\}$. Note that for consistency, we enforce that $Y_k = U_k$ for all keyframes k; we would like to produce good estimates for the remaining output splines.

All the keyframe splines must have the same number of control points (M). This is straight forward to maintain; if the artist would to add a new control point on any keyframe, the appropriate Bézier curve is subdivided at the same parametric location in all other keyframes. The manifold may then be recomputed with the increased number of control points.

It is important to note that we need a minimum of two keyframes (ideally three) before we can begin to construct a shape manifold. For this reason, the artists are asked to produce keyframes for the first and last frame before progressing. This allows the shape model to initialize; before these keyframes are present the system operates without the shape model.

4.2 Overview

Our solver may be broken down into three stages. We first estimate a rotation, translation and scale for each keyframe spline. We then remove this rigid body transformation to a produce a set of normalized keyframe splines which are aligned with one another. The differences between the normalized splines are now only the changes in deforming shape which is what we want to capture with our shape manifold.

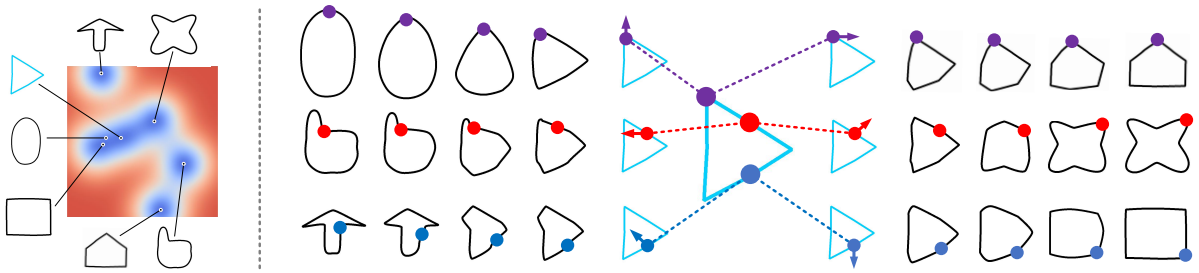

Figure 5: *A toy example of using our Intelligent Drag Tool for interactive curve editing. The manifold can be used to propagate edits to a single curve using shape information from the curves of all keyframes. To change the shape efficiently, the user can simply grab a location on the curve (examples are illustrated by red,blue, and purple dots) and drag it different directions. As the user drags the control point to a new location, our solver immediately infers the change in location on the manifold then gives best fitting shape to the current curve. Note that all the curves generated are representative of keyframe curves.*

We then take the normalized keyframe splines and fit a generative manifold model to their shape. This model embeds the high dimension spline data (the collection of all the Bézier control points) into a very low dimensional space such that every location in the low dimensional space maps to a different shape that interpolates and extrapolates from the keyframes. Furthermore, this low dimensional space is smooth such that smooth changes in the manifold space represent smooth changes in the high dimensional splines.

Finally, once we have learned the manifold model we are able to run our solver. Here we can take tracking data from any source and combine it with the shape manifold to produce a robust output where the roto-shapes are smoothly varying representative shapes, even when the tracking fails. This allows the user to insert keyframes and produce their desired shape quickly, even when parts of the spline are unable to track edges or other image features.

4.3 Keyframe Alignment

We align the keyframe splines by estimating a rotation θ_k, translation $\mathbf{t}_k$, and scale s_k for every keyframe. We do this to a high degree of accuracy by using an energy model to estimate the transformation and a reference shape at the same time as a generalized Procrustes analysis problem. We denote the mean reference spline as R. For each keyframe k, our alignment energy is

$$E^{(k)}_{\text{align}}(U_k,\theta_k,\mathbf{t}_k,s_k,R) = \sum_{m=1}^{M}\left[\begin{bmatrix}U^{(k)}_{x,m}\\ U^{(k)}_{y,m}\end{bmatrix} - \left(s_k Q_k \begin{bmatrix}R_{x,m}\\ R_{y,m}\end{bmatrix} + \mathbf{t}_k\right)\right]^2 \quad (2)$$

where Q_k is the 2D rotation matrix

$$Q_k = \begin{bmatrix}\cos(\theta_k) & -\sin(\theta_k)\\ \sin(\theta_k) & \cos(\theta_k)\end{bmatrix}. \quad (3)$$

We solve for the energy

$$E_{\text{align}}(\{U_k,\theta_k,\mathbf{t}_k,s_k\},R) = \sum_{k}^{K} E^{(k)}_{\text{align}}(U_k,\theta_k,\mathbf{t}_k,s_k,R) \quad (4)$$

over all keyframes to find the optimal $\{\theta_k,\mathbf{t}_k,s_k\}$. We initialize with the mean shape and linear estimates for the transformation variables before applying the non-linear least squares Ceres solver [Agarwal et al. 2015] directly to Equation 4 using the Gauss-Newton L-BFGS method.

Normalizing the keyframes Once we have obtained the rotation, translation, and scale estimates we can produce normalized keyframe shapes $\{Y_k\}$ by removing the rigid body transformation as

$$\begin{bmatrix}Y^{(k)}_{x,m}\\ Y^{(k)}_{y,m}\end{bmatrix} = Q_k^T\left(\frac{1}{s_k}\begin{bmatrix}U^{(k)}_{x,m}\\ U^{(k)}_{y,m}\end{bmatrix} - \mathbf{t}_k\right) \quad \forall\, m,k\,. \quad (5)$$

4.4 Generating the Shape Manifold

Once we have the normalized keyframe splines, we are able to learn the shape manifold. We use the Gaussian Process Latent Variable Model (GP-LVM) [Lawrence 2005] to produce our shape manifold. While this is a standard model in machine learning, we include a brief introduction in Appendix B.

There are two outputs from the manifold learning process. The first is a set of low dimensional locations $\{\mathbf{x}_k\}$ that correspond to the normalized keyframe splines $\{Y_k\}$. The second is a non-linear mapping function $\mathbf{F}(\cdot)$ that maps from the low-dimensional manifold locations to produce output splines. The two outputs are related in the sense that the function will reproduce the keyframe splines from their manifold locations such that

$$Y_k = \mathbf{F}(\mathbf{x}_k) \quad \forall\, k \quad . \quad (6)$$

Once the manifold is learned, the mapping function is very efficient to evaluate as a matrix multiplication (for further details see Appendix B, Equation 20). We should also note that the training complexity of the GP-LVM is governed by the number of keyframes K, not the number of control points of the splines themselves. Therefore, training of the model is very efficient and may be performed in realtime for shots with typical frame counts.

Manifold Dimensionality The dimensionality can actually be learned automatically using auto-relevance determination kernels [Rasmussen and Williams 2006] as in the Bayesian GPLVM of Titsias and Lawrence [2010]. We used this in early experiments on the roto dataset and found that the most common dimensionality was 2 which we then used in the tool. While we could adaptively estimate the dimensionality dynamically, in practice there was little advantage and it is a slower optimization to perform; a lower dimensionality also speeds up our interactive response times.

4.5 The Roto Solver

As we have seen in Figure 3; once we have obtained a shape manifold from the keyframes, we expect the splines of intervening frames to be generated from locations in the manifold (since they generate appropriate shapes). Furthermore, we expect the path taken in the manifold space between frames to be smooth since nearby manifold locations will generate similar, smoothly varying shapes. To take advantage of this we use the manifold to constrain an energy model. We may take data terms from any type of tracker (*e.g.* based on points, planes, edges, color, etc…) to try and push and pull the roto-splines between the keyframes but we constrain the estimated shapes to be generated from a location on the manifold with neighboring frames to have nearby manifold locations.

Tracker Selection We consider several popular rotoscoping trackers: *KLT*, the *Nuke Planar Tracker*, and the *Blender Planar Tracker*. These implementations are comprehensively compared on our ground truth dataset (see Figure 7). We used the Blender Planar Tracker in our final approach for the *Expert Study* (see Section 5.3) since it was the best tracker with a publicly available implementation, and has been widely applied by the opensource community. It also provides realtime performance; this is vital in order to update seamlessly when new keyframes are added. It is important to note that our model is capable of fusing information from any other tracker. However, we cannot integrate proprietary trackers directly into our tool since we do not know how they are implemented. We also observe that many more advanced or proprietary trackers will not run in realtime.

Tracking Residuals For each output frame n (between the keyframes) we estimate a manifold location $\mathbf{x}_n$ that will generate a normalized spline Y_n. We then combine these with estimates for a rotation θ_n, translation $\mathbf{t}_n$ and scale s_n. This gives us final spline which we may compare with L different trackers which output points $\mathbf{p}_{m,n}^{(l)}$ for the m^{th} key point for tracker $l \in [1..L]$. We note that we do not need dense tracks for every key point or for every frame. The residual for the l^{th} tracker of the m^{th} point in the n^{th} frame is given by

$$C_{\text{track}}(l,m,n) = \left[s_n Q_n [\mathbf{F}(\mathbf{x}_n)]_m + \mathbf{t}_n \right] - \mathbf{p}_{m,n}^{(l)} \,. \tag{7}$$

Again, Q_n is the 2D rotation matrix of angle θ_n.

Smoothness Terms As well as the tracker terms, we include costs that penalize large discontinuities between consecutive frames in the the manifold space, and in the transformations (rotation, translation and scale). These residuals take the form

$$C_{\text{manifold}}(n,n') = \mathbf{x}_n - \mathbf{x}'_n \tag{8}$$

$$C_{\text{rot}}(n,n') = \begin{bmatrix} \cos(\theta_n) \\ \sin(\theta_n) \end{bmatrix} - \begin{bmatrix} \cos(\theta'_n) \\ \sin(\theta'_n) \end{bmatrix} \tag{9}$$

$$C_{\text{trans}}(n,n') = \mathbf{t}_n - \mathbf{t}'_n \tag{10}$$

$$C_{\text{scale}}(n,n') = s_n - s'_n \tag{11}$$

where n and n' are neighbouring frames.

Final Energy The final energy may be expressed as a sum-of-squares with appropriate weights being allocated to the smoothness terms

$$\begin{aligned} E_{\text{solver}} = & \sum_l \sum_m \sum_n \|\rho(C_{\text{track}}(l,m,n))\|^2 + \\ & \lambda_{\text{manifold}} \sum_{n,n'} \|C_{\text{manifold}}(n,n')\|^2 + \lambda_{\text{rot}} \sum_{n,n'} \|C_{\text{rot}}(n,n')\|^2 + \\ & \lambda_{\text{trans}} \sum_{n,n'} \|C_{\text{trans}}(n,n')\|^2 + \lambda_{\text{scale}} \sum_{n,n'} \|C_{\text{scale}}(n,n')\|^2 \,. \end{aligned} \tag{12}$$

The function $\rho(\cdot)$ is a robust loss function to protect against errors from the tracking data (for example a tracker that has failed and drifted away). We use an aggressive loss function

$$\rho(c) = \arctan(c/\gamma) \tag{13}$$

with a scale parameter of $\gamma = 10$ pixels to discount outliers. The weight parameters in the energy were set to $\lambda_{\text{manifold}} = 0.001$, $\lambda_{\text{rot}} = 0.1$, $\lambda_{\text{trans}} = 1.0$, and $\lambda_{\text{scale}} = 0.1$. These parameters were determined by running the solver over a random subset of the ground truth rotoscoping dataset (Section 5.1) with a range of different settings. The L2 (Euclidean) norm between the estimated and ground truth spline control points was used to determine which settings were the most successful. None of the sequences used in the experiments (Section 5) were used to train the parameters. We found our method was robust to parameter choice since all the terms have natural normalizations. The manifold locations have a prior to have a unit Gaussian distribution, the translations may be normalized by the frame size, the scales by the keyframes and the rotations have an inherent scale.

Optimization Details For each of the optimizations we made use of the highly efficient C++ Ceres solver [Agarwal et al. 2015] for both non-linear least squares (using the Levenberg-Marquart algorithm) for Equation 12 and non-linear unconstrained optimization (for the GP-LVM learning). In all cases we either obtained explicitly coded analytic Jacobians and derivatives (for example to exploit sparsity in the GP-LVM kernel gradients) or made use of the effective auto-differentiation code built into the solver. In particular we can obtain efficient analytic derivatives of the GP-LVM prediction function in Equation 20 to calculate the Jacobian of Equation 7.

When performing the GP-LVM learning we initialized the low dimensional space with linear PCA. The high dimensional training data (the keyframe curves) were normalized to have zero mean and unit variance and then the hyperparameters and noise precision were all initialized with unity. The hyperparameters were given an uninformative log-Normal prior with unit variance. Everytime the GP-LVM was relearned the parameters were reinitialized; this does not create problems with the keyframes moving around since they are all guaranteed to return the same shape since the keyframes are fixed shapes used to train the GP-LVM.

The roto solver optimization of Equation 12 was initialized by linearly interpolating the scale, translation and rotation parameters between the keyframes. The manifold locations were initialized by linear interpolation between the corresponding keyframe manifold locations. We will release a reference implementation of the model along with the Roto++ GUI tool.

4.6 Interaction

Timing The multithreaded, C++ implementation of our solver is very efficient with all three optimizations normally performed in a fraction of a second, increasing to a second for the longest shot (150 frames) we tried. This is clearly advantageous for an interactive workflow, where the addition of a new keyframe can be propagated in realtime. Another advantage is that we solve for the whole sequence at once so there is no waiting whilst information propagates frame by frame. The speed of the trackers can vary by type and implementation. The fastest planar trackers also solved in under a second for average length sequences (30 frames).We note, as well, that the initial solve is always longer and as keyframes are added, any trackers only need to update locally.

Next Keyframe Recommendation The suggestions for the next keyframe to edit are obtained by comparing the output of the tracker and the output of the solver. The L2 (Euclidean) error between the control points from the solver and from the tracker is calculated. If this error is lower than an artist determined threshold (we used 0.5 pixels) the frame is ignored. The remaining frames with larger errors are ranked in descending order of errors and the top 10% of the frames are labeled as suggestions for the artist to edit.

Control Point Insertion If the user wishes to insert a new control point into an existing curve we subdivide all the roto-curves in all other frames at the same parameter value so that the curves themselves do not change shape (simply contain an additional control point). This does not corrupt the manifold since the changes will not be incorporated until the keyframe is marked as finished by the user. At this point the manifold will be update but all existing

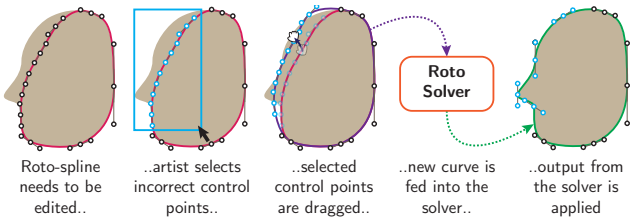

Figure 6: *An illustrative example of the Intelligent Drag Tool.*

keyframes (that are marked as finished) will not be modified even if their locations on the manifold are changed since they form the training data for the GP-LVM.

Intelligent Drag Tool Figure 6 shows an illustration of how the intelligent drag tool is implemented. First the artists selects the control points that they wish to move. The selected control points are then dragged to a new location; this defines a new curve, shown in purple, made from the unmoved control points and the new positions of the selected points. As the dragging occurs, this new curve is fed into the roto solver as if it were a tracker output. The solver will then estimate a new position for the selected control points that obeys the shape manifold in realtime. Once the artist is happy with the shape they can release the drag and the new positions will be applied. In order to prevent unwanted movement, only the selected points will be moved and the unselected points will retain their original positions, even if the manifold would have moved them to new locations. This allows the tool to be used to create new keyframes that do not currently exist on the manifold.

5 Experiments

5.1 Real-World Rotoscoping Dataset

To evaluate our method we produced an extensive rotoscoping dataset specifically designed to be truly representative of real-world commercial rotoscoping in the post-production industry.

The dataset consists of a five minute short movie , that has been professionally rotoscoped by a services company that works on preparation and compositing for high-end VFX and stereoscopic conversion across film, television and commercials. This dataset will be made available under the Creative Commons license agreement (by-nc-sa)[1] for non-commercial use.

The short movie depicts a story unfolding around a night club and contains shots typical to live-action movies. This represents the range of complexity for rotoscoping that would be expected in live-action film post-production. The footage covers the space of roto-shapes that would be required, from simple rigid objects, to isolated articulation motion, more complex articulations with occlusions and intersections, and thin structures for close-ups of hair. The scene content and camera effects cover locked-off and hand-held camera moves, shifts in focus and motion blur, close-up and wide-angle shots, and bright and dark environments. They also include isolated and interacting characters, plus complex environmental effects with water.

Rotoscoping Effort The dataset is broken down into 158 shots, with over 10,000 frames of footage. The shots range from 13 frames to over 300 frames long, with an average shot length of 67 frames; the majority of shots in live-action movies are less than 100 frames long. The shots have been rotoscoped to separate the main elements and the rotoshapes are provided as a script for each shot for compositing software typically used in post-production for high-end VFX.

[1] https://creativecommons.org/licenses/by-nc-sa/3.0/

Complexity	Typical Shot Description	Rating
Easy	• Single isolated characters • Trackable objects • Simple manual keyframing	1 2
Medium	• Limited motion blur • Limited articulation • Several characters	 3
Hard	• Lengthy camera shots • High-speed shots with motion blur • Many characters with detailed articulation • Detailed shapes for hair / fur	4 5

Table 1: *Complexity examples for different rotoscoping shots. Ratings and descriptions provided by professional artists.*

Complexity Rating	Number in Dataset	Rotoscoping effort
2	38 shots	28 frames / day
3	94 shots	14 frames / day
4	14 shots	11 frames / day
5	12 shots	6 frames / day

Table 2: *Breakdown of the complexity and rotoscoping effort required in the rotoscoping dataset. The total effort for the entire dataset was 734 person days. Ratings and effort provided by professional artists.*

Shot Complexity The dataset has been categorized by professional artists on a scale (1) - (5) to define the amount of effort required in rotoscoping. This is a sliding scale from (1) which represents an easy shot that can be tracked to (5) which is the most complex with highly detailed interacting shapes. Table 1 provides further information with typical examples of the types of shots at each rating. Table 2 details a breakdown of the shots present in the dataset. The simplest shots consist of a static camera or slow pan with limited detail, and the most complex contain detailed hair strands, motion blur, water bubbles, dust, and debris. The effort to rotoscope the data-set was a total of 734 person days with an average of 14 frames per day. Table 2 also shows the relative efforts for different shot ratings.

Errors All of the shots are recorded in HD and a typical error of a few pixels is within the motion blur of the majority of scenes at this resolution. For professional artists to achieve sub-pixel accuracy for this footage would require feathering to be used which we do not currently have data for. We discuss this as future work in Section 5.4.

5.2 Roto Solver Evaluation

In our first evaluation we evaluate our solver against some benchmarks (Section 2.4) in terms of reducing the number of keyframes to achieve a given accuracy. We can perform a quantitative evaluation of this without including users by taking advantage of the ground truth of our dataset. We conduct two experiments. In the first, we uniformly sample keyframes from some test curves and then determine how accurate the different methods estimate the unknown interim curves. In the second test we make use of our frame suggestion feedback to provide the keyframes to the algorithm in the order it suggests.

Curve Selection In consultation with roto-artists, we picked four 30 frame long shots; each is representative of one or more real-world effects and/or typical objects. The shots *Arm* and *Head* provide large illumination changes on deformable objects with complex outlines. The shot *Wheel* provides large perspective changes of a rigid object with occlusions. Finally, the shot *Face* shows a moving

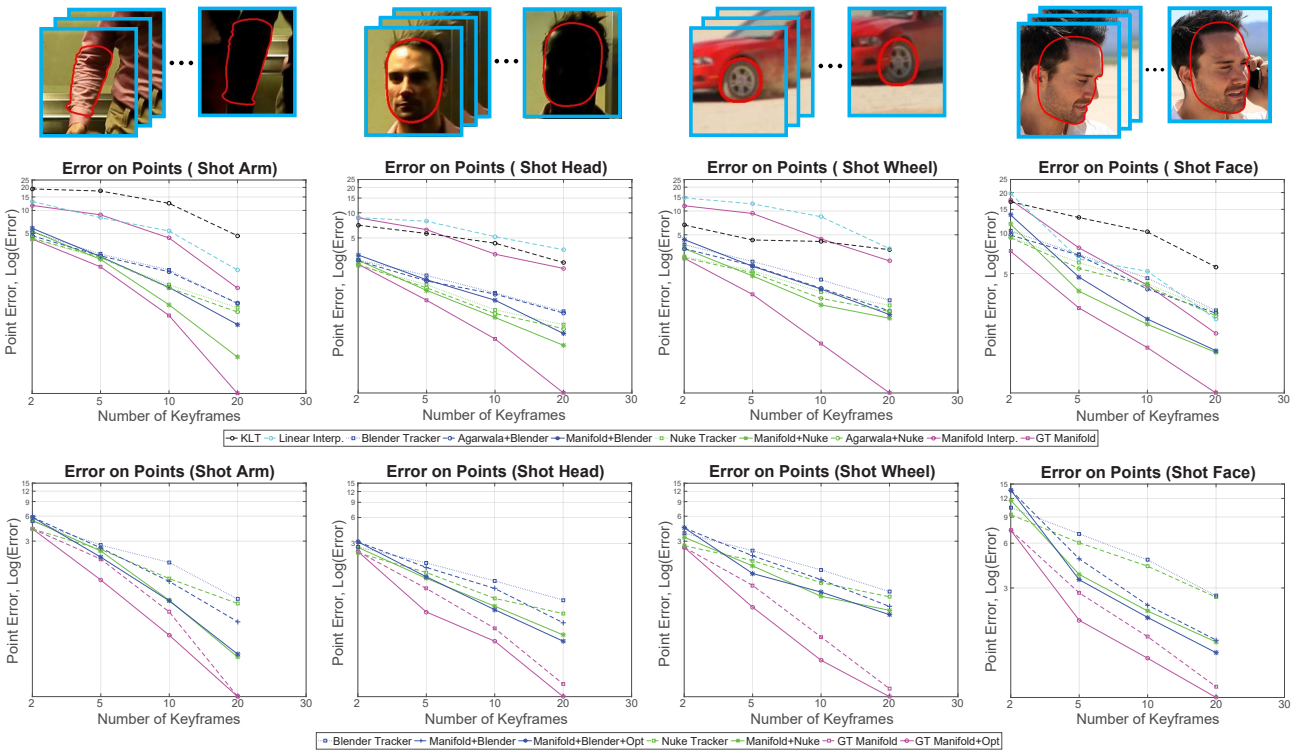

Figure 7: *Quantitative evaluations on selected shots of our Real-World Rotoscoping Dataset. The **First Row** (from left to right) shows the shots i.e. **Arm**, **Head**, **Wheel** and **Face** that represent several typical objects and motions. The **Second Row** shows the evaluation on the selected shots by using various interpolation methods, i.e. our Manifold with either Ground Truth tracking ("GT"), Nuke Planar Tracker or Blender Planar Tracker; the regularization of Agarwala et al. [2004]; the KLT Tracker; and Linear Interpolation. The **Third Row** illustrates the performance improvement on our manifold methods using optimal keyframe selection (for methods marked with "+OPT"). Errors (shown on log plots) are calculated using the RMS pixel error between the control points of estimated curves and the ground truth (both have the same parameterization).*

and turning human face which is among the most typical, but also challenging, targets for professional rotoscoping. According to the artists, the human face is the most frequently repeated object for a movie and it is also a high deformable object that is readily affected by illumination, blur and occlusion. The top row of Figure 7 shows some thumbnails from the test shots.

Error Metric Since we use the same input keyframes for each method (from the ground truth of the dataset) we know that the roto-splines will be parameterized identically. This allows us to evaluate the error by considering the RMS pixel error between the control points of the estimated curves and the ground truth splines.

Experiment 1: Uniform Keyframes The second row of Figure 7 provides the results of various automatic estimation methods on each of the four shots. The first result to note is that the straight-forward interpolation methods (either linear interpolation or interpolation in the manifold) space perform considerably worse than the other approaches, alongside the basic KLT tracker. This is indicative of the difficulty of the shots; even after keyframing 20 of the available 30 frames the linear interpolation method has an error above two pixels.

At the other end of the scale, the *GT Manifold* result is always the best performer. This is result obtained if the ground truth is passed as a "tracker" to our roto solver. It acts as a lower bound on what can be achieved with the shape manifold; *i.e.* can the manifold express all the required shapes after seeing a certain number of keyframes. Although there is nothing theoretical that prevents other methods from improving below this line, the fact that is is the best performer means that we are not losing and representational power by using the manifold.

Now comparing the two planar trackers, we see that the Nuke tracker (with its propriety additions) does in general outperform the Blender tracker by they display similar characteristics. If we combine the raw trackers with the regularization of Agarwala *et al.* [2004] we notice improved performance suggesting that the regularization is helping to compensate for tracking errors such as drift. However, the heuristic curve based regularizer is effectively acting as a smoothing interpolator for the tracking results which limits its efficacy.

Combining the trackers with our shape manifold offers further improvement which supports the hypothesis that the shape manifold should help reduce the number of keyframes that are needed to achieve a given degree of accuracy. The generative shape model is learned from the specific keyframes present in each shot (rather than a generic solution) and is more expressive with its ability to interpolate and extrapolate shape in a non-linear manner.

The *Manifold Interp* result provides the output obtained by the direct application of the method of Campbell and Kautz [2014] and we can see the comparative improvement of our method that uses a more complex model containing the manifold regularization and the tracker guidance.

Experiment 2: Keyframe Suggestion The third row of Figure 7 shows some of the same results (for reference) combined with results obtained by selecting the next keyframe according to the frame suggestion feedback (methods labeled with *+OPT*). We

can see that by using the suggested frames the *GT Manifold* result improves which indicates that the suggested keyframes are more indicative of the range of shapes in the shot and therefore providing them as keyframes produces a more accurate shape manifold.

The second result of interest is that making use of the Blender tracker with the optimal suggestions also results in improved accuracy which supports our hypothesis that the suggested frames are useful for both improving the shape model and the tracker result.

Conclusion We can conclude from these automatic experiments that the combination of our shape manifold helps to improve tracking results beyond all the baselines across a range of shots. Furthermore, following the frame suggestion feedback also improves accuracy such that a meaningful reduction in the number of keyframes required can be achieved.

While these methods do offer improvement, we should also note the overall difficulty of the shots. Even with the best methods the artist will still have to label a significant proportion of the frames in the shot to achieve professional quality. This motivates our next set of experiments where we will investigate our interaction contribution to see if we can reduce the amount of time required to edit these keyframes.

5.3 Expert Study

To evaluate the Intelligent Drag Tool and our Roto++ tool as a whole we conducted an Expert Study. The aim was to investigate performance with respect to two baseline workflows in a real-world situation with experienced rotoscoping artists and shots from a commercial movie. Figure 8 provides an overview of the expert study that we will now discuss in more detail. First we will describe the protocol used and then we will analyse the quantitative results.

Evaluation Protocol We invited seven professional roto-artists, from movie post-production houses, to take part in the study; the artists all had between two and nine years of rotoscoping experience. We divided the artists into three teams as shown in Figure 8. Each team had a similar distribution of experience to allow for fair comparisons between the teams. Our Roto++ tool was run in three different modes, *Mode 1* presents our method; *Mode 2* denotes the Blender planar tracker; and *Mode 3* is the linear interpolation. The artists were unaware of the technical details of any of the differences between any of the modes. In addition to our solver, Mode 1 also made the Intelligent Drag Tool available. The roto artists were instructed on how to use it but they were free to use it or not during the study. Similarly for the next keyframe suggestions.

Figure 8 also shows the four shots given to the artists; these were chosen from our dataset. Each shot contained 30 frames with one shot for training and the others for the tests. Each artist was asked to roto a different shot for each mode to avoid any prior learning for the shot; every artist used each shot once and each mode once. The table in Figure 8 shows the randomized ordering used to avoid bias. The shots comprised three different complexity ratings from the dataset (Table 1). Test A was level 2, Test B level 3, and Test C level 4; the training shot was level 2. These shots were chosen to represent typical photometric effects, *i.e.* motion blur, deformable objects, illumination changes and large occlusion.

The artists spent the first 30 minutes working on the training shot to learn to use the software. They were then required to roto one curve within 15 minutes for each of the three tests. As described in Section 3.2, the logging facility of the Roto++ tool was used to recorded all the interaction. We also took snapshots of the current roto-curves every 20 seconds for numerical analysis. All the workstations involved in this expert study had a similar configuration: Intel i7 3.00GHz, 16GB RAM and nVidia Quadro K2200 4GB GPU.

Error Metrics Each individual was given an instruction sheet for the test showing the desired roto shape in the first and the last frame. Unlike in the previous experiments, because the artists create their own curves they will not be parameterized in the same manner as the ground truth. We, therefore, computed the error by rasterizing both curves to a dense polylines, aligning them and taking the RMS pixel error. For each test we selected the best experimental subject from each mode for comparison and we compare our approach against the two baselines of the planar tracker and linear interpolation. The variance in the results are from the samples over different experts in each group. We distributed the experience of the artists evenly across the groups.

Rotoscoping Time Figure 9 depicts the point error over time for all baselines on the three different shots. Across all three tests (differing complexities) our solver improves over the baselines. We also note that an earlier version of our solver was used for the expert study that occasionally produced a lag in response (this was commented on by the experts when they were debriefed). We have since improved the solver speed by an order of magnitude and no have no lag in subsequent tests. The dashed line on the plots represents a replay of the log with the solver times set to the new speed; this leads to an additional improvement in performance, really demonstrating the value of our tool.

Shape Interaction Using our instrumentation we were able to measure various mouse (pen and tablet) operations; the results are presented in Figure 10. We observe that our solver and intelligent drag tool require fewer mouse operations and achieve a greater accuracy. This is due to the value of the intelligent drag tool moving the control points to the correct location in far fewer moves than editing individual control points.

Intelligent Drag Mode Figure 11 demonstrates the efficiency of editing individual keyframes. Every keyframe edited by the artist is a line on the graph (considered in isolation). Each line is a plot of the error in the roto-curve of that keyframe against the time spent in editing the keyframe. The lines corresponding to our method are located in the bottom left indicating the at the intelligent drag mode is much more efficient for individual keyframe editing since the error reduces rapidly in a small amount of editing time.

We also conducted an additional study with a longer sequence (110 frames rather than 30) of the Test A shot; the result is shown in Figure 12. As well as validating the ground truth results of Section 5.2, the plot also shows a significant reduction in time taken to edit a keyframe as more frames are labeled, indicating that the manifold is becoming increasingly useful as its shape model improves when presented with more keyframes. We also observe higher initial errors than previously (when only a few keyframes are provided) since the manifold takes longer to learn the various shapes present in the much longer shot (requires more training data).

The other plots in Figure 12 also highlight areas where tracking fails. While we are also susceptible to failures in tracking, we can also produce significantly more reliable results than the planar tracker alone. These are challenging sequences, containing nonrigid deformation (A), illumination changes (B), and large motion blur (C) which are very challenging to the tracker. manifold constraint to produce a plausible shape is particularly valuable in these settings where it ensures that the artist is presented with a much better starting point for keyframe editing as well as having the advantages of the intelligent drag tool.

Further Comparisons We also compared our method to the brush-stroke interaction method of Video SnapCut [Bai et al. 2009]. During our pilot study we asked artists if they ever used non-curve based tools with the specific example of the Adobe RotoBrush.

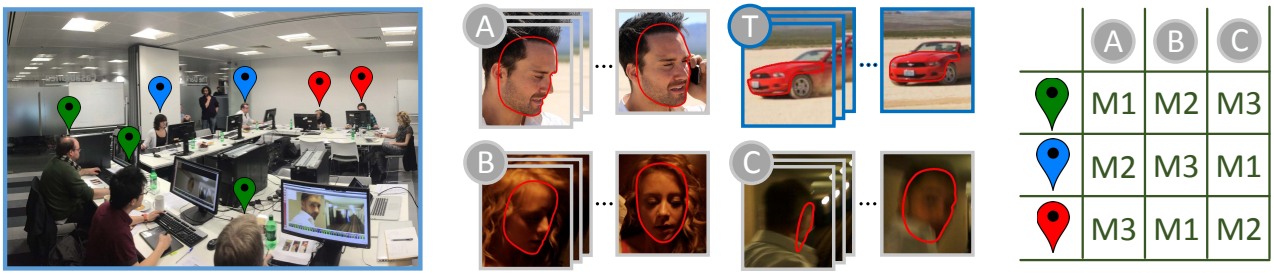

Figure 8: *Our expert study. We invited seven experienced rotoscoping artists for the study; all the artists were asked to roto three different shots for test after an initial training shot. The shots used represent several main difficult cases for the real-world shots:* ***(T)*** *Training Shot, a rigid object with camera shake;* ***(A)*** *Test A, nonrigid deformation;* ***(B)*** *Test B, large illumination change;* ***(C)*** *Test C, motion blur and large shape change. Each team of artists was also required to use different modes (workflow) for different shots in order to avoid prior learning on specific shot. This was a blind study with the artists unaware of any of the technical details of the different modes. The three modes used were:* ***M1*** *Our Roto Solver;* ***M2*** *Planar Tracker; and* ***M3*** *Linear Interpolation.*

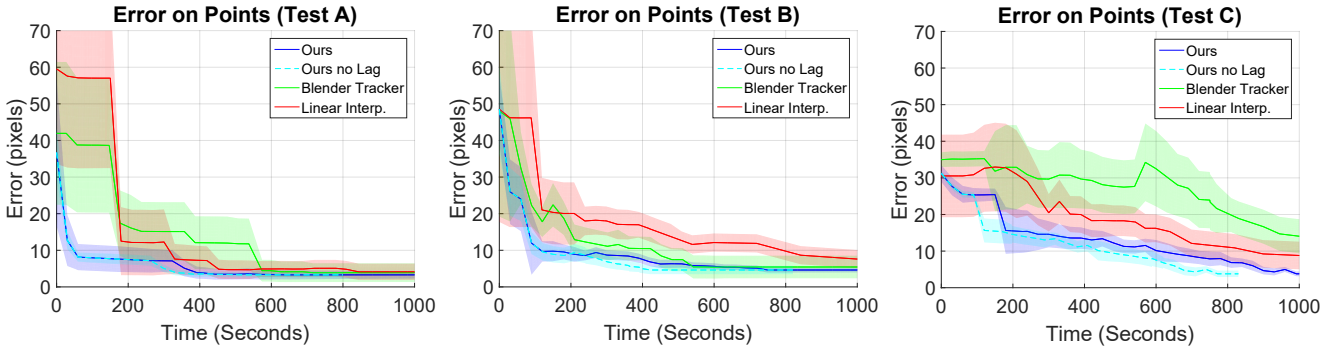

Figure 9: *Quantitative measure of roto accuracy over time for the expert study.. The shaded regions around each error curve represents one standard deviation. Our approach is more effective than the other baselines with the artists taking less time to achieve a specific accuracy. During the user study we used a slower implementation of our solver that led to a laggy response. We have since increased the speed of our solver by an order of magnitude. The dashed line ("Ours no lag") represents a replay of the recorded log if the solver had been running at the new speed. We can see that with the solver running at full speed, the improvement is even greater.*

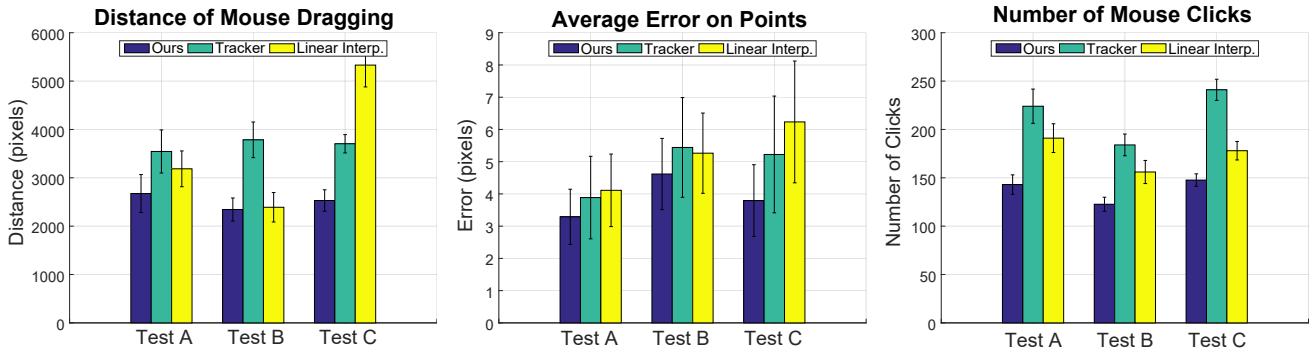

Figure 10: *Quantitative measures of mouse move distance, average point error, and number of mouse clicks. The error bars represent one standard deviation. A mouse click or drag is counted only if it changes the location of a point. Our method achieves lower scores across all categories while still outperforming the other methods in terms of accuracy.*

There was a universal response that this interaction mode was not widely used for movie postproduction since it was often difficult to control. We provide illustration of the interaction comparison in the supplemental video.

Appendix C contains additional results and illustrations. In Figure 14, we show a qualitative comparison of JumpCut, RotoBrush, the Mocha Tracker, and our approach for propagation from two keyframes. In Figure 15 we show a quantitative comparison with RotoBrush and the proprietary Mocha tool. We observe that our method performs favourable compared to the commercial alternatives. The Mocha planar tracker confers good performance and we believe that our method could be further improved by incorporating this tracker however we cannot perform this test since the details of the tracker are proprietary.

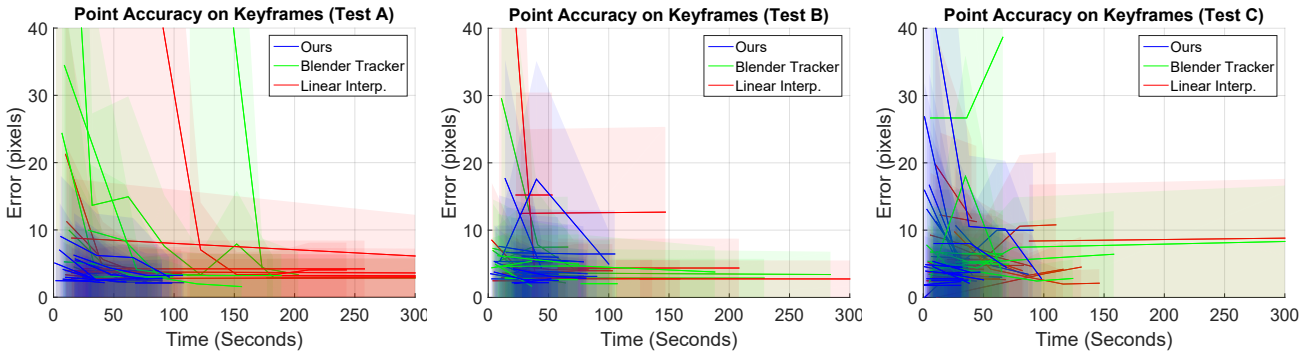

Figure 11: *Quantitative measure of roto accuracy over time editing individual keyframes. The shaded regions around each error curve represents one standard deviation. This plots describes the efficiency of each artist on a keyframe by keyframe basis. For each keyframe they edited we plot the reduction in error over the time spent editing in that keyframe (thus there is a line on the graph for every keyframe edited). We can see that our method lies in the bottom left corner indicating that the starting errors were lower (due to the improved solver) and that the time to edit each keyframe was reduced (the intelligent drag mode).*

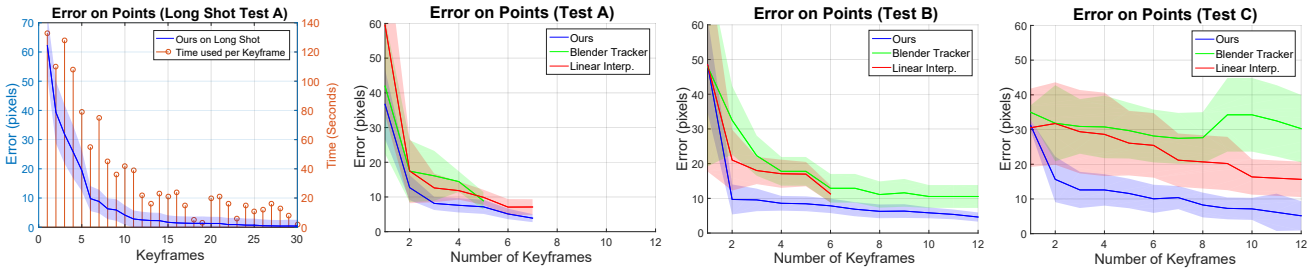

Figure 12: *Quantitative measure on points error of curves against the number of keyframes edited. The shaded regions around each curve represents one standard deviation. Our method yields best accuracy (especially for the first couple of keyframes) overall. In the first plot, we illustrate the effect of an extended shot (increasing the number of frames from 30 to 110) for our method. As well as a similar improvement in accuracy against keyframes as our ground truth evaluation, we also plot the time spent editing each keyframe and demonstrate a significant acceleration in editing rate as the manifold improves with the knowledge of more keyframes.*

Conclusion In conclusion, the user study provides a range of quantitative data that support the adoption of our new tool Roto++ as a serious contender for professional rotoscoping. This improvement is achieved via a three advantages. The improved solver is more robust to errors in tracking and provides a measurable reduction in the number of keyframes required for a given accuracy on challenging, real-world shots. The keyframe suggestion mode confers two valuable points in that it identifies frames that simultaneously help the shape model to improve and reduce the errors in the tracker; both again help to reduce the number of keyframes. Finally, our expert study demonstrated that our intelligent drag tool quantifiably improved the time to edit keyframes when used by professional artists. This is a very promising result since we should also note that the roto-artists have years of experience using the two other baseline techniques and had only been using our drag tool for 30 minutes before the user study.

During the exit interviews, after the user study, the artists provided some qualitative feedback. They were very enthusiastic about the intelligent drag tool and the frame suggestions saying that they found them to integrate well into their existing workflow and were useful for real-world tasks. They also liked the clean interface of the Roto++ tool and said it was enjoyable to work with.

5.4 Limitations and Future Work

Complexity Rating 5 In our work we have not attempted to rotoscope some of the incredibly challenging shots. Although these are not very common, for example they only account for 8% of our representative dataset, they are known to perform very poorly to all attempts at automation with the roto artists usually forced to draw in each frame by hand. This can often be due to completely missing data, where the artists are using their artistic knowledge to actually fill in the gaps (for example through thick smoke or very strong motion blur). These sequences would have taken too long to run for our expert study; in future work we would like to investigate what can be done to help artists at these extremes.

Feathering Throughout this work we have not considered "feathering", the use of soft alpha borders to the segments that vary in width under the artist's control. This would be a fairly straight forward extension to our framework since the feather offset for each control point on the roto-spline could be appended to the vector modeled by the GP-LVM that could then estimate the feather width for new frames as well as the roto shape. The main consideration for not including it in our current model is that the ground truth dataset does not currently have the feathering information so we would have no way of evaluating the performance. In the future we would like to extend the roto dataset with this information.

Appearance Snapping Our intelligent drag tool does not snap to appearance edges, instead it uses its generative model to suggest shapes that respect local artist edits. This is more reliable because it will continue to function in the presence of lighting changes, motion blur and regions with many/ambiguous edges. The Roto++ tool includes a snapping active contour tool but artists in our pilot study disliked it; they found it faster to place control points themselves. That said, it may well be of use to amateur users and for simpler videos. As further work we will investigate more advanced appearance models that may make this tool more useful.

Assistive Curve Creation At the moment we provide no assistance in creating the first keyframe. Although our expert users claim not to want it, we would like to explore an optional mode that would help the user create curves in the first place, *e.g.* incorporating the snap-to functionality of [Su et al. 2014].

Acknowledgments

The "MODE" footage is provided courtesy of James Courtenay Media and the rotoscoping work was kindly performed for The Foundry by Pixstone Images. All data-sets were prepared by Courtney Pryce with special thanks to John Devasahayam and the PixStone Images team for turning around a vast amount of work in such a short period of time.

The study was coordinated by Juan Salazar, Dan Ring and Sara Coppola, with thanks to experts from Lipsync Post, The Moving Picture Company, and Double Negative – Kathy Toth, Andy Quinn, Benjamin Bratt, Richard Baillie, Huw Whiddon, William Dao and Shahin Toosi.

This research has received funding from the EPSRC "Centre for the Analysis of Motion, Entertainment Research and Applications" CAMERA (EP/M023281/1) and "Acquiring Complete and Editable Outdoor Models from Video and Images" OAK (EP/K023578/1, EP/K02339X/1) projects, as well as the European Commission's Seventh Framework Programme under the CR-PLAY (no 611089) and DREAMSPACE (no 610005) projects.

References

Agarwal, S., Mierle, K., and Others, 2015. Ceres solver. http://ceres-solver.org.

Agarwala, A., Hertzmann, A., Salesin, D. H., and Seitz, S. M. 2004. Keyframe-based tracking for rotoscoping and animation. *ACM Trans. on Graphics 23*, 3.

Bai, X., and Sapiro, G. 2007. A geodesic framework for fast interactive image and video segmentation and matting. In *IEEE Int. Conf. on Computer Vision (ICCV)*.

Bai, X., Wang, J., Simons, D., and Sapiro, G. 2009. Video snapcut: Robust video object cutout using localized classifiers. *ACM Trans. on Graphics 28*, 3.

Baker, S., and Matthews, I. 2004. Lucas-kanade 20 years on: A unifying framework. *Int. Journal of Computer Vision 56*, 3.

Baumberg, A. M., and Hogg, D. C. 1994. An efficient method for contour tracking using active shape models. In *IEEE Workshop on Motion of Nonrigid and Articulated Objects*, 194–199.

Benhimane, S., and Malis, E. 2004. Real-time image-based tracking of planes using efficient second-order minimization. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, vol. 1, 943–948.

Blender Online Community. 2015. *Blender - a 3D modelling and rendering package*. Blender Foundation.

Boykov, Y. Y., and Jolly, M.-P. 2001. Interactive graph cuts for optimal boundary amp; region segmentation of objects in n-d images. In *IEEE Int. Conf. on Computer Vision (ICCV)*, vol. 1, 105–112.

Bratt, B. 2011. *Rotoscoping: Techniques and Tools for the Aspiring Artist*. Taylor & Francis.

Campbell, N. D. F., and Kautz, J. 2014. Learning a manifold of fonts. *ACM Trans. on Graphics 33*, 4.

Chuang, Y.-Y., Agarwala, A., Curless, B., Salesin, D. H., and Szeliski, R. 2002. Video matting of complex scenes. *ACM Trans. on Graphics 21*, 3.

Cootes, T. F., Taylor, C. J., Cooper, D. H., and Graham, J. 1995. Active shape models and their training and application. *Computer Vision and Image Understanding 61*, 1, 38–59.

Criminisi, A., Sharp, T., Rother, C., and Pérez, P. 2010. Geodesic image and video editing. *ACM Trans. on Graphics 29*, 5.

Dame, A., Prisacariu, V. A., Ren, C. Y., and Reid, I. 2013. Dense reconstruction using 3D object shape priors. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*.

Erofeev, M., Gitman, Y., Vatolin, D., Fedorov, A., and Wang, J. 2015. Perceptually motivated benchmark for video matting. In *British Machine Vision Conference (BMVC)*.

Fan, Q., Zhong, F., Lischinski, D., Cohen-Or, D., and Chen, B. 2015. JumpCut: non-successive mask transfer and interpolation for video cutout. *ACM Trans. on Graphics 34*, 6.

Grochow, K., Martin, S. L., Hertzmann, A., and Popović, Z. 2004. Style-based inverse kinematics. *ACM Trans. on Graphics 23*, 3.

Grundmann, M., Kwatra, V., Han, M., and Essa, I. 2010. Efficient hierarchical graph based video segmentation. *IEEE Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*.

Kwatra, V., Schödl, A., Essa, I., Turk, G., and Bobick, A. 2003. Graphcut textures: image and video synthesis using graph cuts. *ACM Trans. on Graphics 22*, 3.

Lawrence, N. 2005. Probabilistic non-linear principal component analysis with gaussian process latent variable models. *Journal of Machine Learning Research 6*, 1783–1816.

Lebeda, K., Hadfield, S., Matas, J., and Bowden, R. 2016. Texture-independent long-term tracking using virtual corners. *IEEE Trans. on Image Processing 25*, 1, 359–371.

Levin, A., Lischinski, D., and Weiss, Y. 2008. A closed-form solution to natural image matting. *IEEE Trans. on Pattern Analysis and Machine Intelligence 30*, 2, 228–242.

Li, Y., Sun, J., and Shum, H.-Y. 2005. Video object cut and paste. *ACM Trans. on Graphics 24*, 3.

Li, W., Cosker, D., Brown, M., and Tang, R. 2013. Optical flow estimation using laplacian mesh energy. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'13)*, IEEE, 2435–2442.

Loper, M. M., Mahmood, N., and Black, M. J. 2014. MoSh: Motion and shape capture from sparse markers. *ACM Trans. on Graphics 33*, 6.

Lucas, B. D., and Kanade, T. 1981. An iterative image registration technique with an application to stereo vision. In *Int. Joint Conf. on AI*, vol. 2, 674–679.

Mortensen, E. N., and Barrett, W. A. 1995. Intelligent scissors for image composition. In *Proc. of SIGGRAPH*, ACM, 191–198.

Prisacariu, V. A., and Reid, I. 2011. Nonlinear shape manifolds as shape priors in level set segmentation and tracking. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*.

Rasmussen, C. E., and Williams, C. 2006. *Gaussian Processes for Machine Learning*. MIT Press.

Rhemann, C., Rother, C., Wang, J., Gelautz, M., Kohli, P., and Rott, P. 2009. A perceptually motivated online benchmark for image

matting. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*.

Rother, C., Kolmogorov, V., and Blake, A. 2004. "GrabCut": Interactive foreground extraction using iterated graph cuts. *ACM Trans. on Graphics 23*, 3.

Rzeszutek, R., El-Maraghi, T., and Androutsos, D. 2009. Interactive rotoscoping through scale-space random walks. In *IEEE Int. Conf. on Multimedia and Expo (ICME)*, 1334–1337.

Santosa, S., Chevalier, F., Balakrishnan, R., and Singh, K. 2013. Direct space-time trajectory control for visual media editing. In *ACM SIGCHI Conf. on Human Factors in Computing Systems*, 1149–1158.

SilhouetteFX. *Silhouette 5.2 User Guide 2014*.

Smith, A. R., and Blinn, J. F. 1996. Blue screen matting. In *Proc. of SIGGRAPH*, ACM, 259–268.

Su, Q., Li, W. H. A., Wang, J., and Fu, H. 2014. EZ-Sketching: Three-level optimization for error-tolerant image tracing. *ACM Trans. on Graphics 33*, 4.

Subr, K., Paris, S., Soler, C., and Kautz, J. 2013. Accurate binary image selection from inaccurate user input. *Computer Graphics Forum 32*, 2, 41–50.

Titsias, M., and Lawrence, N. 2010. Bayesian gaussian process latent variable model. In *Proc. of 13th Int. Workshop on AI and Stats.*

Torresani, L., Yang, D. B., Alexander, E. J., and Bregler, C. 2001. Tracking and modeling non-rigid objects with rank constraints. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*.

Torresani, L., Hertzmann, A., and Bregler, C. 2008. Nonrigid structure-from-motion: Estimating shape and motion with hierarchical priors. *IEEE Trans. on Pattern Analysis and Machine Intelligence 30*, 5, 878–892.

Tsai, D., Flagg, M., Nakazawa, A., and Rehg, J. M. 2012. Motion coherent tracking using multi-label mrf optimization. *Int. Journal of Computer Vision 100*, 2, 190–202.

Turmukhambetov, D., Campbell, N. D. F., Goldman, D. B., and Kautz, J. 2015. Interactive sketch-driven image synthesis. *Computer Graphics Forum 34*, 8, 130–142.

Wang, J., and Cohen, M. F. 2007. Image and video matting: A survey. *ACM Foundations and Trends in Computer Graphics and Vision 3*, 2.

Wang, T., and Collomosse, J. 2012. Probabilistic motion diffusion of labeling priors for coherent video segmentation. *IEEE Trans. on Multimedia 14*, 2, 389–400.

Wang, J., Bhat, P., Colburn, R. A., Agrawala, M., and Cohen, M. F. 2005. Interactive video cutout. *ACM Trans. on Graphics 24*, 3.

Wang, J., Agrawala, M., and Cohen, M. F. 2007. Soft scissors: An interactive tool for realtime high quality matting. *ACM Trans. on Graphics 26*, 3.

Appendix

A Notation for the Technical Approach

Table 3 contains a summary of the notation used in Section 4 with Figure 13 illustrating the layout of the control points for an input keyframe.

N	Number of frames in the shot
$n \in 1..N$	Index of a frame
K	Number of available keyframes
$k \in 1..K$	Index of a keyframe
M	Number of control points in the closed spline
$m \in 1..M$	Index of a control point
L	Number of trackers
$l \in 1..L$	Index of a tracker
$\{U_k\}$	Set of *input* keyframe splines
$\{Y_n\}$	Set of *output* splines
$\theta, \mathbf{t}, s$	The rotation, translation and scale of a spline
Q	A rotation matrix
R	A mean reference spline for alignment
$\mathbf{x}$	A location on the manifold
$V = \mathbf{F}(\mathbf{x})$	A new spline generated from the manifold
$\mathbf{p}_{m,n}^{(l)}$	l^{th} Tracker output point for the m^{th} control point in the n^{th} frame

Table 3: *Summary of notation used in Section 4.*

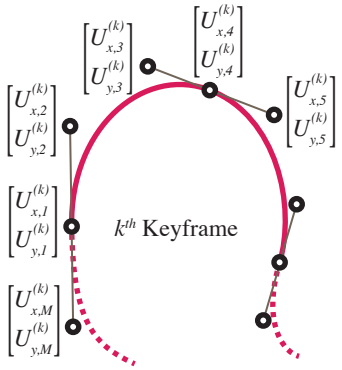

Figure 13: *Notation for the control points of the k^{th} input keyframe.*

B The GP-LVM

We begin by introducing Gaussian Processes (GPs) and the GP Latent Variable Model (GP-LVM) which may be used to perform unsupervised manifold learning. We will then provide details of how we use this to generate our shape model.

Gaussian Processes Gaussian Processes [Rasmussen and Williams 2006] represent distributions over functions and may be used as a powerful tool to learn a smooth, non-linear mapping from one space $\mathbf{x} \in \mathbb{R}^Q$ to another $\mathbf{y} \in \mathbb{R}^D$. They are particularly effective when mapping between spaces with different dimensions, in our case we have $Q \ll D$, when the vectors of interest in the higher dimensional space actually lie on a manifold of far lower dimension. We instinctively believe this to be the case for roto-curves since the there are a very large number of possible splines that can be drawn with M control points but only very few of them will map to the shape of the object we are segmenting.

The GP achieves this mapping by modelling the covariance of the higher dimensional vectors as a kernel function in the lower dimensional space. If we assume that all our vectors are normalized to have zero mean then the GP models the probability of a high dimensional vector $\mathbf{y}$ as the multivariate Gaussian

$$P(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \kappa(\mathbf{x}, \mathbf{x})) \ . \tag{14}$$

Here, $\kappa(\mathbf{x}, \mathbf{x})$ is a kernel function that encodes how distances in the

low dimensional correlate to similar high dimensional vectors.

GP-LVM The GP-Latent Variable Model, [Lawrence 2005], takes an unsupervised approach to learn a generative model. Unlike, the standard GP, we only provide a set of high dimensional vectors $\{\mathbf{y}_k\}$ (the training data) and a desired kernel mapping function $\kappa(\cdot,\cdot)$. The GP-LVM then estimates the best corresponding set of low dimensional vectors $\{\mathbf{x}_k\}$ that generate the high dimensional vectors when passed through the GP. Whilst the full derivation is quite involved, the effect is that we end up with a set of low dimensional vectors and a mapping function to generate high dimensional vectors that are representative of the training data. For our purposes we set the high dimensional vectors to be the spline keyframes (in dimension $D = 2M$) and we learn a set of points $\{\mathbf{x}_k\}$ in $\mathbb{R}^Q$ (we use $Q = 2$) and a mapping such that for every point in the 2D space, we generate a different roto spline.

The nature of the mapping between the spaces is determined by the kernel function. We use the Radial Basis Function (RBF) kernel

$$\kappa(\mathbf{x}_i,\mathbf{x}_j\,|\,\alpha,\beta) = \alpha\exp\left(-\frac{1}{2}\beta\|\mathbf{x}_i-\mathbf{x}_j\|^2\right), \tag{15}$$

where α and β are the variance and length-scale hyper parameters. The RBF kernel is inherently smooth, therefore, as we move around in the low dimensional manifold space the high dimensional space will also vary smoothly. This provides the desirable property for using the manifold for rotoscoping: neighboring frames should have smoothly varying shape and therefore should have nearby embedded spaces. This is demonstrated by Figure 3 where we embed 19 consecutive roto-curves in a 2D manifold.

Learning the Manifold We learn the manifold model by optimising the hyper parameters (α,β) a noise parameter (σ) and the embedded locations $(X = \{\mathbf{x}_k\})$ of the corresponding training keyframe splines $(\{\mathbf{y}_k\})$. We maximise the likelihood of the training examples factored across each of the dimensions as

$$P(Y|X,\Omega) = \prod_k^K \mathcal{N}\Big(Y_k(:)|\mathbf{0},\kappa(X,X\,|\,\alpha,\beta)+\sigma^2 I\Big), \tag{16}$$

where Ω denotes the hyper and noise parameters and Y and X are stacked collections of training row vectors such that

$$Y = \begin{bmatrix} Y_1(:) \\ Y_2(:) \\ \vdots \\ Y_K(:) \end{bmatrix} \tag{17}$$

and similarly for X. Here we used the notation

$$Y_k(:) = [Y^{(k)}_{x,1}\ Y^{(k)}_{y,1}\ Y^{(k)}_{x,2}\ Y^{(k)}_{y,2}\ \dots\ ] \in \mathbb{R}^{2M} \tag{18}$$

to unwrap Y_k (a $2\times M$ matrix of spline keypoints) as $1\times 2M$ row vector. Also, we use the notation $\kappa(X,X\,|\,\alpha,\beta)$ to denote the covariance matrix

$$[\kappa(X,X\,|\,\alpha,\beta)]_{i,j} = \kappa(\mathbf{x}_i,\mathbf{x}_j\,|\,\alpha,\beta)\,. \tag{19}$$

Since the kernel matrix is a non-linear function we use a gradient based non-linear optimiser to maximise Equation 16 and initialize the low dimensional vectors using linear PCA to reduce the high dimensional vectors to the appropriate number of dimensions (Q). We also place a unit Gaussian prior on the manifold locations to maintain a natural scale.

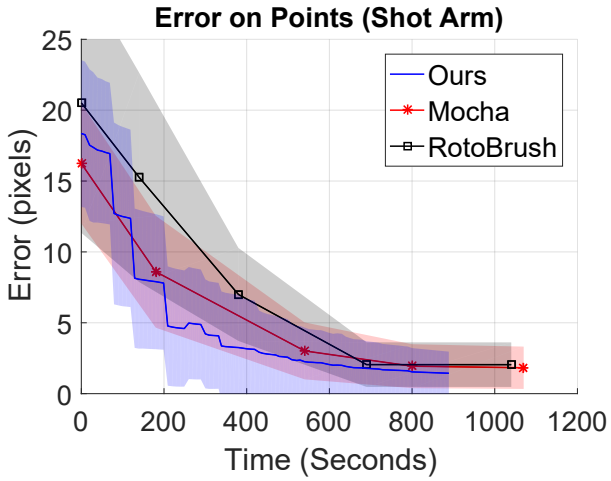

Figure 15: *Quantitative measure of error against time for shot Arm. We quantitatively compare our method to the commercial Mocha workflow and RotoBrush on our Arm shot (Containing 30 frames, the same shot used in Figure 14). The shaded regions around each curve represents one standard deviation.*

Generating New Splines Once we have trained the GP-LVM, we can generate new splines from a low dimensional location $\mathbf{x}'$ by conditioning the distribution of Equation 16 on the training data. In practical terms, this produces a simple matrix multiplication

$$\mathbf{y}' = \mathbf{F}(\mathbf{x}') = \kappa(\mathbf{x}',X|\alpha,\beta)[\kappa(X,X|\alpha,\beta)]^{-1}Y \tag{20}$$

to generate the new normalized spline $\mathbf{y}'$.

C Additional Results

Figure 14 shows our additional visual comparison to the JumpCut, RotoBrush and Mocha Tracker. The selected clip contains eight adjacent frames (1080p), and represents the difficulties of non-rigid deformation and illumination changes. Within this comparison, we keyframe the first and the last frames across all the trials. For the tests of JumpCut and RotoBrush, we show the curve propagation results in forward and backward separately because the related software does not support the propagation using two keyframes. Our method outperforms the JumpCut and RotoBrush, and yields competitive accuracy when compared to the proprietary Mocha Tracker (4.36 v.s. 6.42 average pixel RMS).

Figure 15 shows our additional quantitative comparison to the Mocha Tracker and RotoBrush. An experienced artist performs the rotoscoping on a difficult shot (the Arm, 30 frames) by using Mocha Pro 4.1.0 and AfterEffect CC'15 (RotoBrush) respectively. We shows the points error against the time elapsed. The the commercial packages, the artist is required to output the alpha masks at 5 minute intervals after the first two keyframes. Note that this comparison does not include JumpCut because the GUI is not publicly available.

Figure 14: *Qualitative comparisons of different rotoscoping methods. We propagate a curve from a reference frame to the other frames of the arm shot. From **Top to Bottom**, The **First** group shows the JumpCut results by propagating the reference curve in forward (first row) and backward (second row) directions respectively. The **Second** group illustrates the results of RotoBrush (Adobe AfterEffect CC 2015). The **Third** group shows the results by using the Mocha Tracker (ver. 4.1.0) which takes into account two reference frames. The **Fourth** group gives the results of our method which uses the manifold (trained by two reference keyframes) and the blender planar tracker. The **Bottom** row shows the ground truth for each frame. Note that the cyan circles highlight details for the results.*