Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2006)
M.-P. Cani, J. O'Brien (Editors)

Re-using Traditional Animation: Methods for Semi-Automatic Segmentation and Inbetweening

Christina N. de Juan[†] and Bobby Bodenheimer[‡]

Vanderbilt University

Abstract

A large body of traditional animation exists that contains characters with poses, expressions, and appeal not easily achievable with modern 3D techniques. To create new uses for this body of animation, this paper presents components of a system that can help incorporate the animation into re-usable libraries. In particular, we discuss two semi-automatic techniques that allow the re-use of traditional animation. First, support vector machines are used to segment cartoon images from their backgrounds for incorporation into an image library, for such applications as re-sequencing. Second, a radial basis function implicit surface modeling technique and a fast non-rigid elastic registration algorithm provide inbetween contours and textures given two key images of traditional animation. Our system is fast, model-free, and requires minimal animator intervention.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Animation

1. Introduction

Many tools developed in computer animation are designed to allow animators to obtain the expressiveness of traditional animation more easily, e.g., [FBC*95, Rad99, KMN*99], often by following the techniques discussed in Lasseter [Las87]. However, these tools primarily focus on the problem of creating new animations from models. In contrast, there has not been much study of the problem of re-using traditional animation to create new animation [BLCD02, dJB04]. Given the great aesthetics and expressiveness of traditionally animated characters, we seek to capture those features by providing components of a system to re-use the images. Part of the difficulty in studying this problem is that the forms in which traditional animation are available make it difficult to devise methods to manipulate it easily. This paper takes steps to allow the incorporation of traditional animation into a library such that the animation can be re-used.

A library that successfully captures the aesthetics of traditional animation could be used in several ways. Using re-sequencing technology described in de Juan and Bodenheimer [dJB04], one could generate new animation using classic characters. Interactive educational applications could be created with characters that respond to learners rather than through pre-scripted video sequences. For example, young learners might benefit from examples of *Wile E. Coyote* and what happens when a spring-loaded boulder launcher fails and flattens the coyote. Finally, large libraries of cartoon character data might also be useful in restoring damaged sequences of those characters.

A primary challenge in building large libraries of cartoon character data is to put the characters into a form in which the character is nicely separated from the background. Segmentation is necessary if the character is to be placed into a new environment or with a new background. Much older cartoon data suffers from noise due to changes in lighting as the cel animations were transferred to film, contamination of the cel from one use to another as it was filmed, and degradation of the animation before being transferred to an archival format. These factors make the segmentation problem quite challenging, as we discuss in Section 3.

Once the segmentation problem is solved, we address the challenge of re-using the animation. While both Bregler et al. [BLCD02] and de Juan and Bodenheimer [dJB04] dis-

[†] email:c.dejuan@alumni.vanderbilt.edu
[‡] email:bobbyb@vuse.vanderbilt.edu

cussed this problem, they did so in different ways. A true re-usable library of animation is closer in spirit to the system of [dJB04]. However, one of the limitations of the system described in that paper is the inability to generate new images when a visual discontinuity (abrupt transition) is detected in a re-sequenced animation. Addressing this issue returns one to the 2D inbetweening problem discussed by Catmull [Cat78], although the problem addressed here is more limited: it suffices to generate an inbetween of two "key images" that are somewhat similar, not two keyframes. Our work on this problem is presented in Section 4.

A fully automatic method for inbetweening would alleviate some of the tedium associated with creating a traditionally animated film. However, semi-automatic methods for inbetweening provide a more interactive environment for the artist, allowing for modifications during the creation of the inbetweens, while still improving and speeding the process. The most desirable qualities of traditional animation are the nuances an artist adds to each character, giving that character personality and style. Ensuring that the artist remains involved in the inbetweening process, albeit minimally, should provide a higher level of quality in the resulting animations.

Allowing the traditional animator to be in the process represents both a constraint and an advantage. The advantage is, as mentioned above, that we can leverage the abilities of the artist to produce superior animations. The constraint is that we must maintain our data in a form with which the traditional animator can work, i.e., line art. As a result, the methods we discuss in this paper are strictly image-based, and do not rely on underlying models for the characters such as subdivision curves, patches, or other geometry.

We present examples and results of our system using three cartoon sequences with different characters: *Bugs Bunny*, *Wile E. Coyote*, and *Daffy Duck*. The original image size of all data sets is 720 x 480. The *Coyote* data set is composed of frames from three different cartoons, with a total of 527 images. The others are composed of frames from only one cartoon, but have breaks where the scenes change. The *Bugs* data set has 553 images and the *Daffy* data set has 560 images. Figure 1 shows examples of the frames from the original data along with the corresponding segmented images generated with the methods discussed in Section 3. These characters and examples have very different color and animation properties, which demonstrate the generality and robustness of our methods.

Figure 1: *Examples of original and segmented frames from* Bugs Bunny, Wile E. Coyote, *and* Daffy Duck. Looney Tunes *characters are* ™& ©Warner Bros. Entertainment Inc.

2. Background

As mentioned above, there are few systems for re-using traditional cartoon animation, and our system dovetails with our prior work [dJB04]. In that work, pre-segmented cartoon images are re-sequenced using a manifold learning technique to create novel animations. Alternatively, Bregler et al. [BLCD02] proposed a method for re-using cartoon motion data by capturing the motion of one character and retargeting it onto a new character.

Some work has been done to provide a means of producing cartoon animation more easily, but does not allow for the re-use of that data. Litwinowicz [Lit91], and later Fekete et al. [FBC*95], present a complete 2D animation system, allowing the animator to draw characters directly into those systems. Corrêa et al. [CJTF98] developed a method for applying complex textures to hand-drawn animation. Finally, Petrovic et al. [PFWF00] inflate a 3D figure based on hand-drawn art to produce shadows for cel animation. Their method is also semi-automatic.

Our work was motivated by recent work in motion editing re-use systems for 3D motion capture and video data, e.g., [Gle98, SSSE00, KGP02, LCR*02, AF02, WXSC04, PW99, CB04]. This body of research has had considerable success in developing systems to present novel animation from an existing library of motion capture data. Our work emphasizes re-using the images of traditional animation from a library of images rather than a library of motion data.

Discussion of the related work for the two components of our system, segmentation and inbetweening, are divided into the following subsections respectively.

2.1. Segmentation

Image segmentation is fundamental to image processing [GW01, SS01]. Instead of approaching the problem as

a conventional segmentation problem, we take advantage of the simple colors, flat shading, and no lighting effects characteristic of these cartoons to segment them via classification using support vector machines (SVMs) [Vap98]. Since we are interested in obtaining the character from the background, SVMs are used as binary classifiers. Gómez-Moreno et al. [GMJA*03] use a similar approach for segmenting color medical images.

We experimented with level set techniques [Osh03] for segmentation. The SVM was superior both in the results it gave and in its computational speed. Because of the amount of noise in traditional cel animation, segmentation methods such as level sets proved unsatisfactory. Additionally, level set segmentation on quarter resolution color images took upwards of five minutes per image. In contrast, the SVMs required only three minutes to train the classifier and five seconds to segment each full resolution color image.

2.1.1. Support Vector Machines

The SVM algorithm operates by mapping a given training set into a high-dimensional *feature space* and finding a hyper-plane that separates the data into classes. To construct an optimal hyperplane, the SVM minimizes a particular error function, and in this work, we use the C-SVM classification [Vap98]. Given a training set of attribute-label pairs (x_i, y_i), where $i = 1 \ldots l$, training vectors $x_i \in \mathbf{R}^N$ and $y_i \in \{+1, -1\}^l$, C-SVM minimizes the following error function:

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}\xi_i$$

subject to $y_i(\mathbf{w}^T\phi(x_i) + b) \geq 1 - \xi_i$. The training vectors x_i are mapped to a higher dimension by the kernel function ϕ. An appropriate nonlinear ϕ can always map the x_i to a sufficiently high dimension that a separating hyper-plane exists. C is the penalty parameter of the error function, which controls the trade-off between allowing training errors and forcing rigid margins, $\mathbf{w}$ is a vector of coefficients, b is a constant, and ξ_i are variables for handling non-separable input data. We chose to use a radial basis function (RBF) kernel having the form $\phi = e^{-\gamma\|x_i - x_j\|^2}$, where $\gamma > 0$.

2.2. Inbetweening

Inbetweening is a studied but unsolved problem in 2D animation, introduced to the computer animation community by Burtnyk and Wein [BW76] with their template-based approach. Catmull [Cat78] described the main issues concerning the inbetweening problem. In particular, to deal with self-occlusion, Catmull [Cat78] suggested breaking the character into separate layers, a procedure we will follow (see section 4.1). Reeves [Ree81] presented a method for creating inbetweens using moving-point constraints, curves that constrain the path and speed of points on the character. Di Fiore et al. [DSEV01] present a multi-level method for inbetweening 2D animation by including 3D information as a high-level deformation tool, and 2.5D information as modeling structures. In their follow-up work, Van Haevre et al. [VDV05] unite their previous work with that of [dJB04] to produce smooth, perpetual animations from a small number of keyframes. Like their previous work, the keyframes must be drawn directly into their 2.5D system, requiring an underlying representation of each image as subdivision curves or surfaces. Kort [Kor02] introduced a method for integrating vector-based inbetweening into an animation system, which requires the user to draw the keyframes and identify the layers of each key image. Seah et al. [SL01] presented a modified hierarchical feature-based matching method for motion estimation to generate inbetween line drawings from a pair of input line drawings. None of these techniques are completely suitable for data-driven inbetweening since they require the construction of contours or other representations of the data requiring significant intervention.

The approach used here is closer to the shape interpolation approaches of [BN92, SG92, ACOL00]. However, it employs radial basis functions (RBFs) to interpolate segmented contours of images and generate implicit models. This technique was first presented by Turk and O'Brien [TO99] and refined by Carr et al. [CBC*01]. We use the machinery of [CBC*01] to generate an implicit model using RBFs, as we have found it is faster than the related methods in [TO99], which provided the inspiration for our technique. In our experience, using implicit surfaces presents a superior technique to vertex-based approaches [SG92, ACOL00], since vertex interpolation often leads to unacceptable deformations in the contours, such as arm shortening.

3. Segmentation of Cartoon Images to Establish Character Data

Segmenting traditional animation proved surprisingly difficult. Cartoon characters are usually easily identifiable, and often made up of a few solid colors. For example, *Daffy Duck* is mostly black with some orange. However, because the animation originally existed in cel format, and was often photographed as a means of transfer to film, there is significant noise in the images that makes segmenting over many frames problematic. As an example, Figure 2 shows deviations from the mean of a pixel in the background from a *Bugs Bunny* animation, a frame of which is shown in Figure 1. This pixel is not atypical, and any segmentation technique will have to deal with noisy pixels in both the foreground and background of the target images. Segmentation is the most time consuming aspect of preparing existing cartoon data.

The first step for using SVMs to segment cartoon images is to classify the training data by selecting the appropriate attribute-label samples. Several features can be identified in the characters that can be used as samples for training and classifying. The most natural choice of a feature is the color

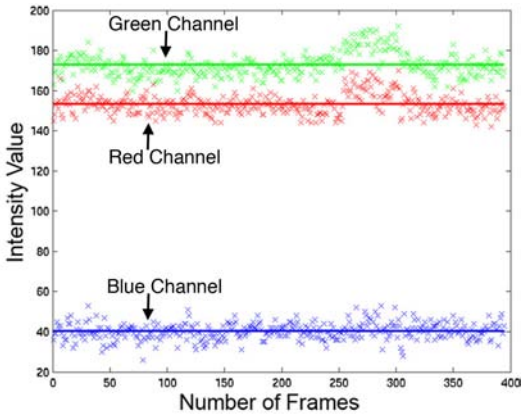


Figure 2: *The deviations from the mean color value for each color channel from a single background pixel for the* Bugs Bunny *sequence are shown as data points around a mean value line.*

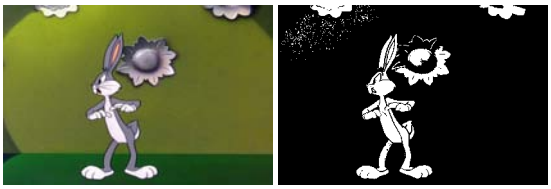


Figure 3: *An input image and resulting segmentation mask using 81 RGB samples and corresponding optical flow vector magnitudes.* Bugs Bunny *is* ™& ©Warner Bros. Entertainment Inc.

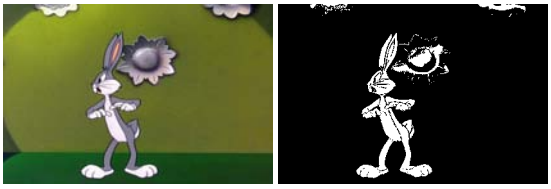


Figure 4: *Segmentation with the SVM classifier model trained using 68 RGB samples from three images. This example shows the best classification of the* Bugs *character.* Bugs Bunny *is* ™& ©Warner Bros. Entertainment Inc.

of the character. Another choice of feature is the optical flow. Many times the character will be on a moving background, and one unique characteristic of hand-drawn cartoons is that there are no shading or lighting changes in the images, so the optical flow may be useful in locating the character. We use two feature sets for classifying the data: color alone or color with optical flow vector magnitudes. For both variations, the user selects pixels from one or several reference images, labels each selection as part of the character (1) or part of the background (-1), and the RGB values are represented in the range of $[0,1]$ and stored in x_i. When using color with optical flow vector magnitudes, the user again selects several pixels from a reference image as before, and the optical flow vector magnitudes are looked-up and included with the RGB values. The optical flow vector magnitudes are pre-computed using a standard algorithm [LK81] from the temporally adjacent frames in the cartoon image sequences. In our experience, user annotation of one reference image per animation scene with a static background and three images per animation scene with a moving background usually suffices, although nothing prevents a user from annotating multiple images per scene. For all of our examples, the user selected samples from three reference images, typically the first, last, and middle frames of the sequence.

Once the training data is classified, the SVM is trained to create a classifier model for each character. As mentioned above, we use an RBF kernel, and we use the LIBSVM library [CL01] to train the SVM model. A grid search and cross-validation on the training data is computed to find the best C and γ parameters for the error function and RBF kernel. For each cartoon character, the best C and γ parameters are found and the whole training set is trained again to generate the final SVM classifier model. The SVM model generated for each character is then applied to every image in that character's data set. The output of classification are the predicted labels for each pixel, which become the resulting binary segmentation mask. Figure 3 shows the result of using 81 RGB samples with optical flow magnitudes for the SVM model. Figure 4 shows an improved result using only 68 RGB samples. This last result is the best achieved on the *Bugs Bunny* data set.

The top row of Figure 5 shows the results of an SVM classifier trained on 108 RGB samples with optical flow vector magnitudes, applied to the first of three *Coyote* sequences. The center row of Figure 5 is the second *Coyote* sequence, using an SVM classifier trained on 179 RGB samples with optical flow vector magnitudes. The results for the third *Coyote* sequence are shown in the bottom row of Figure 5, using an SVM classifier trained on 140 RGB samples with optical flow vector magnitudes. One of the difficulties with these particular cartoon examples is that the character is walking across a moving background in the first two sequences. Thus, there are new color samples revealed throughout the sequences that may not be accounted for in the SVM models.

In all of the examples, there are some pixels that the SVM model erroneously classifies as part of the character. For example, the flower in the background of the *Bugs Bunny* sequence. The color of the flower and the color of the character have exactly the same RGB values. To further improve the segmentation masks, simple morphological operations are performed. The segmentation mask is a binary image. First each 8-connected region in the mask is labeled with a number. For each region found, the area of the region is cal-

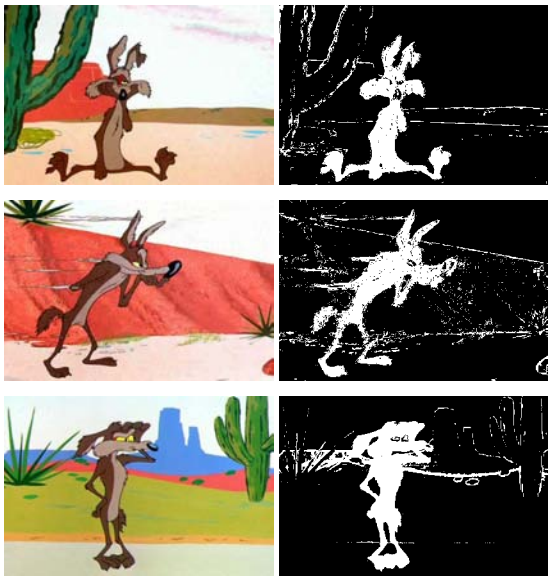


Figure 5: *An input image and resulting segmentation mask for the first* Coyote *sequence (top), the second* Coyote *sequence (center), and the third* Coyote *sequence (bottom).* Wile E. Coyote *is* TM& ©Warner Bros. Entertainment Inc.

culated and stored. The area of a region is the actual number of pixels in that region. The region with the largest area, or larger than a preset value (e.g., 10,000 pixels), is likely to be the character, so the region that meets that criteria is kept in the mask while all others are removed. Finally, any remaining 4-connected foreground pixels identified as holes are automatically flood-filled. To avoid filling regions of the character that are supposed to be small holes (e.g., a character making a circle shape with their hand), the flood-fill can be done interactively instead of automatically. However, in all of our examples we use the automatic flood-fill to reduce the amount of user intervention required. Figure 6 shows the results of applying this method to one of the SVM segmentation masks. If any remaining stray pixels are present, those are easily cleaned up manually. For the *Bugs Bunny* data set, only 20% of the masks needed minimal manual touching up, and slightly more for the *Coyote* data set. Manual touch up of these masks takes less than one minute per frame.

4. Inbetweening an Image Library for Re-Sequenced Animations

Once a library of character image data has been assembled using the techniques of the previous section, a method such as that described in [dJB04] can be used to generate novel sequences. However, as noted by the authors, that system cannot generate new images when a visual discontinuity (abrupt transition from re-sequencing) occurs in a novel sequence. To supplement such a method, we need to generate an inbetween shape between two frames of data. Our procedure

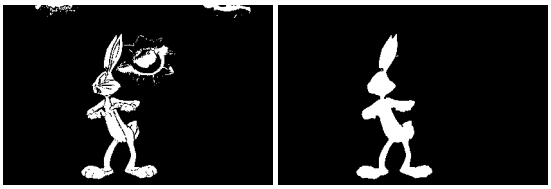


Figure 6: *The result of applying morphological operations to clean up the SVM segmentation result on* Bugs *using the 68 RGB sample SVM model. On the left is the SVM result, on the right is the automatically cleaned mask.*

involves three steps. In the first step, the character is partitioned into several layers such as head and torso. In the second step inbetween shape contours are generated for each layer using an RBF-based technique, and in the final step the cartoon color or texture is fit to the inbetween shape using an elastic registration technique.

4.1. Character Partitioning and Re-assembly

Given a pair of character images to be inbetweened, the pair is first partitioned into character layers. This partitioning is done to alleviate the problem of self occlusion [Cat78]. It is carried out manually and takes only a few moments per image to split the character into layers. For our examples, we typically partition the characters into head, body, arms, and legs layers.

After the inbetween is generated for each layer in the process described below, the layers are automatically reassembled. The location and scale of each layer are lost in the inbetween generation, but are computed using the original silhouettes and partitioned layers as references. To determine the location, a translation, we use the average of the centroid positions of each character layer from the original key images. The scale factor is computed using the average pixel area of the key images defined as $s = \sqrt{\frac{A_{ave}}{A_{tween}}}$, where A_{ave} is the average area of pixels from the key images and A_{tween} is the area of pixels of the inbetweened contour (filled in to be a silhouette) determined by the slicing operation described next. By area, we mean the total number of pixels belonging to the character or character layer.

4.2. Shape Contour Generation

Given input data from the previous segmentation and layering operations, we next generate an inbetween shape contour. The idea is to generate a 3D mesh from two key images and slice that mesh in the middle to extract the inbetween shape contour, as outlined in the following steps. First, silhouettes are created automatically from the key images. The silhouettes are then used to create contours defining the shapes to be inbetweened. The contours are generated by starting at a pixel on the edge of the silhouette and

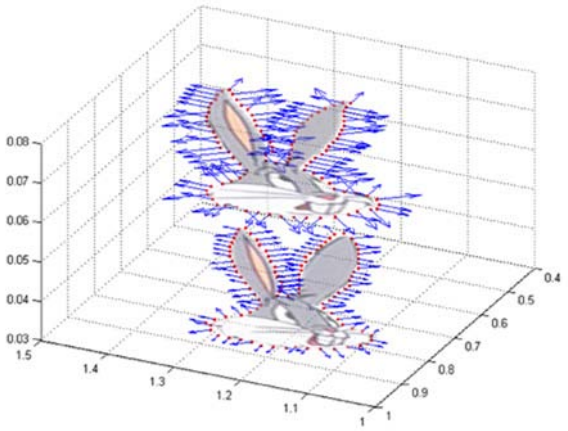
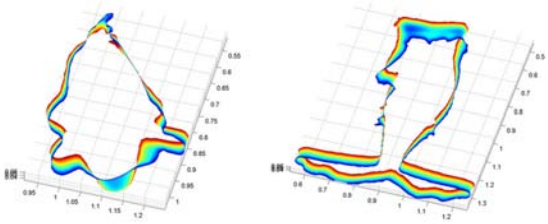


Figure 7: *Visualization of the automatically generated contour and normal points that serve as the input to the implicit surface generation step.*



(a) Head Layer RBF (b) Body Layer RBF

Figure 8: *Left: RBF solution for the* Daffy *head layer. The head images were registered and two additional constraints were used. Right: RBF solution for the* Daffy *body layer.*

tracing around the silhouette in clockwise order. The contour image is just a clockwise ordered list of (x,y) pixel coordinates. Next, using the ordered list of contour points, a set of normals are computed. These normals (shown in Figure 7) define the interior and the exterior of the contour for an RBF interpolation algorithm. The set of contour points and normals are then given a z coordinate placing them in 3D space. The value of the z coordinate is a small number relative to the scale of the contour points such that the contour points are close enough not to cause any inward bowing when fitting the RBFs. These point sets are used as input to the RBF methods, which generate an implicit surface interpolating the contour points. We are using the RBF interpolation and fast evaluation methods developed by [CBC*01]. A marching cubes algorithm then creates a mesh describing the implicit surface, and the mesh is sliced in the middle to create the inbetween contour. The process is quite fast and completes in approximately one minute on a 1.4GHz Pentium.

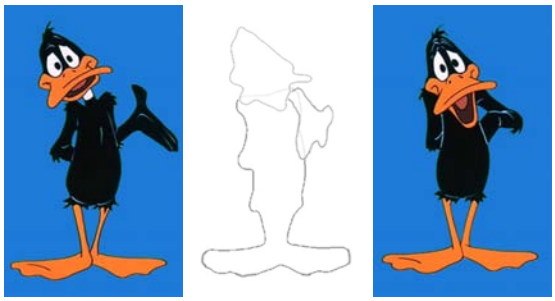
Sometimes the inbetween contour needs further refinement, and then the following steps can be taken: (1) the individual layers can be registered, (2) constraint points can be added for the RBF contour interpolation, or (3) both methods can be used in conjunction. By default, the layers are aligned using the centroid of the character layer (e.g., the centroids of the heads for the head layer). To improve the alignment, a more sophisticated transformation can be applied using an iterative closest point (ICP) registration algorithm [BM92]. However, using ICP requires the user to select feature points on both layers for registration. Constraint points can be used to improve the inbetween contours. The user can select desired constraint points on the previous inbetween contour image to serve as extra data points that must be interpolated by the RBFs. Alternatively or in combination with the above, the user can select desired constraint points on each of the original contour images. Normals are calculated for the selected constraint points, and are passed on with the original contour points and normals into the RBF contour interpolation routine.

Now we describe the RBF interpolation results, and note when any of the additional refinement methods are used. The first example uses two key images of *Daffy Duck* and both techniques for contour refinement. The character is partitioned into three layers for both key images: a head layer, an arm layer, and a body layer. The head layers are registered using the ICP method with 12 control points for each image. Once the head images are registered, the contours are generated and the RBF interpolation method is employed. Two additional constraint points are manually specified along with the contour points on the head layers, which are inserted at a z value half-way between the two key images. These constraint points are used to restrict the fitting of the RBFs around the lower part of *Daffy's* beak. The arm layer and body layers did not require any contour refinement. Figure 8 shows the RBF interpolation results for the *Daffy* head and body layers. Figure 9 shows the inbetween contour for the *Daffy* example described above. The inbetween contours for each layer are reassembled automatically after using the RBF contour interpolation method, as described previously. The entire refinement process took two minutes.

Figure 10 shows the contour inbetweening results for *Bugs Bunny*. For this example, *Bugs* was partitioned into four layers: head, body, left and right arms. Figure 11 is an example contour inbetween generated for *Wile E. Coyote*. The *Coyote* was partitioned into four layers: head, left arm, right arm, and body. The RBF solutions for these two examples did not require any additional contour refinement.

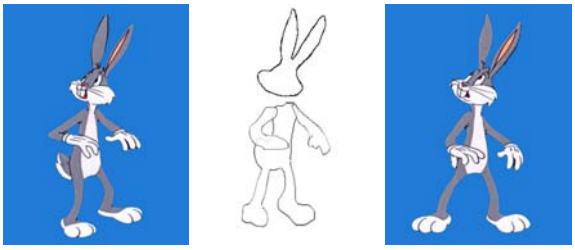
4.3. Texturing or Re-coloring the Inbetween Contours

The final step in creating an inbetween is filling in the color and texture information. We have the color and texture information for the original key images. The issue is how to color and texture the inbetween contour based on this information.



Key image 1 Inbetween Contours Key image 2

Figure 9: *Inbetween contours generated using RBF contour interpolation method. Although the feet in the contours became larger due to the auto-reassembly scaling described in Section 4.1, the position relative to the other layers is correct. The texture filling described in Section 4.3 will not be scaled, so these scaling artifacts will be corrected.* Daffy Duck *is* TM& ©Warner Bros. Entertainment Inc.



Key image 1 Inbetween Contours Key image 2

Figure 10: *Inbetween contours for* Bugs *generated using RBF contour interpolation.* Bugs Bunny *is* TM& ©Warner Bros. Entertainment Inc.

In a production studio, a similar issue occurs when the line art is scanned and goes to the next step of ink and paint. Traditionally, the ink and paint process was done manually. Some studios use a simple flood fill for each region of closed contours in the line art. But an artist is still required to ensure that all contours are closed, else the flood fill would fail. While some of the color information can be passed along from one frame to the next, an artist is still required to touch up many frames before they are finalized.

We use the two key images to fill the inbetween contour by registering the key images and generating an intermediate image based on the registration. There is a large amount of literature on image registration in the medical imaging community, where we looked for inspiration. The method we employ is non-rigid elastic image registration, described by Wirtz et al. [WFMS04], to register the two key images.

In this method, a preprocessing step is performed that compensates for any artifacts due to rotation or translation before the elastic registration proceeds. The process of elastic registration is now described briefly; refer to [WFMS04]

Key image 1 Inbetween Contours Key image 2

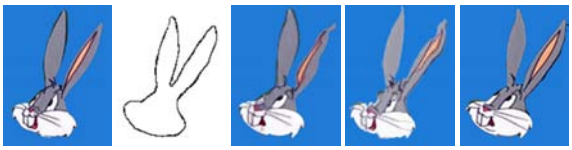
Figure 11: *Inbetween contours for* Coyote *generated using RBF contour interpolation.* Wile E. Coyote *is* TM& ©Warner Bros. Entertainment Inc.

for details. Each image, represented as slices in 3D space, requires finding a transformation based on displacement fields for each slice. A minimization of a functional consisting of a distance metric and smoother (the elastic potential energy) becomes the main objective. The distance metric is the sum of squared intensity differences of each image after undergoing a transformation given by the displacement field. Two parameters, λ and μ, are Lamé's material constants. μ governs how far the material will stretch and is defined as the stress divided by the area. λ governs how fast the material will stretch, and is dependent on μ. Minimizing the series (or pair) of images and displacement fields results in a system of nonlinear partial differential equations, or the Navier-Lamé (NLE) equation, given by

$$\mu\nabla^2\vec{u} + (\lambda+\mu)\nabla(\nabla\cdot\vec{u}) + f(\vec{u}) = 0$$

where $\vec{u}$ is the displacement field that tries to minimize the sum of squared intensity differences of the images, $f(\vec{u})$ is the derivative of the distance metric. The second term imposes a restriction that the entire image (or surface material) is as "stretchable" everywhere on the surface, while the first term enforces a constraint on how far the material will stretch. Simply put, the NLE equation describes the elastic deformation of an object subject to a force, and here it is simply the derivative of the distance metric. The object is deformed until an equilibrium is reached between the forces. Setting the material constants λ and μ of the object are important for ensuring a good registration. Large material constants make the object more rigid, while small material constants are more susceptible to noise effects but allow for larger deformation.

Once a deformation is known for registering the key images, the transformation can be applied to generate an intermediate image and used as a preliminary texture for the inbetween contour. To extend the algorithm for elastic registration from grayscale images to color images, the deformation is computed on the luminance of the two key images and stored. This deformation is then applied to each color channel separately, resulting in the final inbetween color image to use for filling the inbetween contour. In our experience, the material parameters μ and λ need only be set once, as the amount of deformation allowed for the different cartoon



KI_1 C ER_1 ER_2 KI_2

Figure 12: *Comparing elastic registration results for generating inbetweens on the* Bugs Bunny *head layer.* Bugs Bunny *is* ™& ©Warner Bros. Entertainment Inc.

characters was the same. We found that the values $\mu = 2.0$ and $\lambda = 4.0$ worked well.

The final inbetween typically requires only a small amount of touch up, similar to the final touch up done in a production studio pipeline. Our method requires only one step where the artist touches up the result after generating the inbetween texture. In contrast, three touch up steps are typically used in a production studio, one when closing the contours, one when filling, and one as a final pass.

The inbetween contours that were previously computed are used with the elastic registration results in two ways. First, the contour is used to automatically re-assemble the character layers quickly, as described in Section 4.1. Second, and more importantly, the contour is used to determine the correct direction that the elastic registration is applied to the key images. For example, the amount of force required to deform image A into image B will be different than the amount of force required to deform image B into image A. We define ER_1 as the elastic registration result using *key image 1* as the source and *key image 2* as the destination. ER_2 is the elastic registration result using *key image 2* as the source and *key image 1* as the destination. Figure 12 shows the two key images, the inbetween contour C, and the results of ER_1 and ER_2. We can see that ER_1 is visually better than ER_2. However, to automatically determine which registration result more closely matches the contour C, we use the Hausdorff distance as applied in [dJB04] to compute the similarity of C to ER_1 and C to ER_2. A smaller similarity value indicates a better match. ER_1 has a similarity value of 2.05, while ER_2 has a similarity value of 2.33. As we expected, the registration ER_1 is a better match to C, which is used in the final inbetween.

We show the results of elastic registration for filling the inbetween contours from Section 4.2. The same characters and pairs of key images are used. Figure 13 shows a close up of the *Daffy* head layer with the two key images, the inbetween texture generated using the elastic registration, an overlay of the inbetween texture on the inbetween contour, and the final result after a small amount of manual touch up. Cleaning up the final result takes about one minute using image editing software. Figures 14, 15, and 16 show the final results on the three characters. Using a purely image-based interpolation method may introduce new artifacts, but we believe any errors on a single frame of a sequence are insignificant enough to not be easily seen.

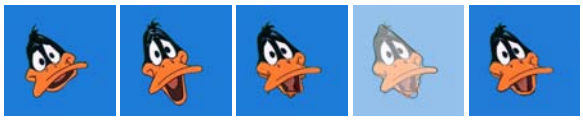
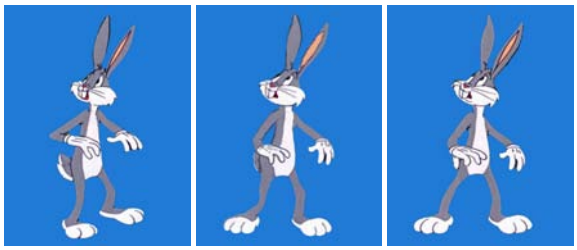


Figure 13: *Moving from left to right: a close up of the first key image head layer, a close up of the second key image head layer, the automatically generated inbetween color, the intermediate color overlaid on the inbetween contour, and the final inbetween for the head layer.* Daffy Duck *is* ™& ©Warner Bros. Entertainment Inc.

Key image 1 Final Inbetween Key image 2

Figure 14: *The final inbetween frame for* Daffy *using elastic registration for the color. See color plate.* Daffy Duck *is* ™& ©Warner Bros. Entertainment Inc.



Key image 1 Final Inbetween Key image 2

Figure 15: *The final inbetween frame for* Bugs *using elastic registration for the color. See color plate.* Bugs Bunny *is* ™& ©Warner Bros. Entertainment Inc.

5. Discussion

This paper presents two necessary components of a system for building image libraries of traditional animation and then re-using them. These techniques could work to extend the system earlier of de Juan and Bodenheimer [dJB04] or to further automate the system presented in Bregler et al. [BLCD02]. Such extensions could find application in film

Key image 1 Final Inbetween Key image 2

Figure 16: *The final inbetween frame for* Coyote *using elastic registration for the color. See color plate.* Wile E. Coyote *is* ™& ©Warner Bros. Entertainment Inc.

restoration or interactive educational technology. The techniques are semi-automatic, but require only minimal intervention to guide the animator in building the image library or touching up novel inbetweens.

The segmentation method seems robust and works well on all examples we have tried it on. Indeed, the support vector machine technique outperformed the level set segmentation technique that we tried, which was difficult to tune, could not effectively deal with the amount of noise in the animations, and most importantly required significant computation time. In contrast, the SVM method required only about five seconds to classify each full resolution image. Training and cross-validation to find the best parameter values took only two to three minutes per SVM classifier model. However, the SVM classifier is trained on only one character, so if two characters appear in the same scene, we require two passes, one for each SVM classifier. The same is true for the morphological operations used to clean up the segmentation masks, as only one character is assumed to be in the image.

The inbetweening procedure is also robust. Limitations of this component include the requirement that a character be manually partitioned into separate layers. Also, the method will not produce a reasonable, detailed inbetween if the deformation between the key images is too great. In this case, animator intervention is required. The processes described here can, however, aid animators in the process of generating inbetweens, as it provides a strong template for a finished product. Considering the minimal amount of user interaction involved for a strictly image-based approach, we believe that this method yields better inbetweens for two-dimensional animation than has previously been reported. Future work in this area will examine whether techniques such as Ju et al. [JSW05] can be used to reduce the limitations described above. Also, the quality of the resulting animations using our inbetweening procedure depends on more than simply re-arranging similar looking frames. Some incorporation of dynamics, such as velocity of neighboring frames, or higher level cues such as timing of the original animation, are future work.

6. Acknowledgments

The authors thank Warner Bros. Entertainment Inc., for permission to use the images of the *Looney Tunes* characters, and Julie Heath for facilitating the process. We also thank the reviewers for their helpful comments. This material is based upon work supported by the National Science Foundation under Grant IIS-0237621. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

References

[ACOL00] ALEXA M., COHEN-OR D., LEVIN D.: As-rigid-as-possible shape interpolation. In *Proceedings of SIGGRAPH 2000* (2000), ACM Press/Addison-Wesley Publishing Co., pp. 157–164.

[AF02] ARIKAN O., FORSYTH D. A.: Interactive motion generation from examples. *ACM Transactions on Graphics 21*, 3 (July 2002), 483–490.

[BLCD02] BREGLER C., LOEB L., CHUANG E., DESHPANDE H.: Turning to the masters: Motion capturing cartoons. *ACM Transactions on Graphics 21*, 3 (July 2002), 399–407.

[BM92] BESL P. J., MCKAY N. D.: A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell. 14*, 2 (1992), 239–256.

[BN92] BEIER T., NEELY S.: Feature-based image metamorphosis. In *Proceedings of SIGGRAPH 1992* (1992), ACM Press, pp. 35–42.

[BW76] BURTNYK N., WEIN M.: Interactive skeleton techniques for enhancing motion dynamics in key frame animation. *Communications of the ACM 19*, 10 (1976), 564–569.

[Cat78] CATMULL E.: The problems of computer-assisted animation. In *Proceedings of SIGGRAPH 1978* (1978), ACM Press, pp. 348–353.

[CB04] CALLENNEC B. L., BOULIC R.: Interactive motion deformation with prioritized constraints. In *2004 ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (July 2004), pp. 163–171.

[CBC*01] CARR J. C., BEATSON R. K., CHERRIE J. B., MITCHELL T. J., FRIGHT W. R., MCCALLUM B. C., EVANS T. R.: Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of SIGGRAPH 2001* (2001), ACM Press, pp. 67–76.

[CJTF98] CORRÊA W. T., JENSEN R. J., THAYER C. E., FINKELSTEIN A.: Texture mapping for cel animation. In *Proceedings of SIGGRAPH 1998* (1998), ACM Press, pp. 435–446.

[CL01] CHANG C.-C., LIN C.-J.: *LIBSVM: a library for support vector machines*, 2001. Software available at

http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[dJB04] DE JUAN C., BODENHEIMER B.: Cartoon textures. In *Eurographics/ACM SIGGRAPH Symposium on Computer Animation* (2004), Boulic R., Pai D., (Eds.).

[DSEV01] DI FIORE F., SCHAEKEN P., ELENS K., VAN REETH F.: Automatic in-betweening in computer assisted animation by exploiting 2.5D modelling techniques. In *Proceedings of Computer Animation* (November 2001), pp. 192–200.

[FBC*95] FEKETE J., BIZOUARN É., COURNARIE É., GALAS T., TAILLEFER F.: TicTacToon: A paperless system for professional 2-D animation. In *Proceedings of SIGGRAPH 1995* (1995), Cook R., (Ed.), Addison Wesley, pp. 79–90.

[Gle98] GLEICHER M.: Retargeting motion to new characters. In *Proceedings of SIGGRAPH 1998* (1998), ACM SIGGRAPH, pp. 33–42.

[GMJA*03] GÓMEZ-MORENO H., JIMÉNEZ P. G., ARROYO S. L., BUENO R. V., SÁNCHEZ R.: *Recent Advances in Intelligent Systems and Signal Processing*. WSES Press, USA, 2003, ch. Color images segmentation using the Support Vector Machines, pp. 151 – 155.

[GW01] GONZALEZ R. C., WOODS R. E.: *Digital Image Processing*. Prentice Hall, Upper Saddle River, N.J., 2001.

[JSW05] JU T., SCHAEFER S., WARREN J.: Mean value coordinates for closed triangular meshes. *ACM Transactions on Graphics 24*, 3 (July 2005), 561–566.

[KGP02] KOVAR L., GLEICHER M., PIGHIN F.: Motion graphs. *ACM Transactions on Graphics 21*, 3 (July 2002), 473–482.

[KMN*99] KOWALSKI M. A., MARKOSIAN L., NORTHRUP J. D., BOURDEV L., BARZEL R., HOLDEN L. S., HUGHES J. F.: Art-based rendering of fur, grass, and trees. In *Proceedings of SIGGRAPH 1999* (1999), pp. 433–438.

[Kor02] KORT A.: Computer aided inbetweening. In *NPAR '02: Proc. of the 2nd int'l symposium on Non-photorealistic animation and rendering* (2002), ACM Press, pp. 125–132.

[Las87] LASSETER J.: Principles of traditional animation applied to 3d computer animation. In *Proceedings of SIGGRAPH 1987* (1987), pp. 35–44.

[LCR*02] LEE J., CHAI J., REITSMA P. S. A., HODGINS J. K., POLLARD N. S.: Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics 21*, 3 (July 2002), 491–500.

[Lit91] LITWINOWICZ P. C.: Inkwell: A 2-d animation system. In *Proceedings of SIGGRAPH 1991* (1991), ACM Press, pp. 113–122.

[LK81] LUCAS B., KANADE T.: An iterative image registration technique with an application to stereo vision. In *Proc. of the 7th Int'l Joint Conference on Artificial Intelligence* (1981), pp. 674–679.

[Osh03] OSHER S.: *Geometric level set methods in imaging, vision, and graphics*. Springer-Verlag, New York, 2003.

[PFWF00] PETROVIC L., FUJITO B., WILLIAMS L., FINKELSTEIN A.: Shadows for cel animation. In *Proc. of ACM SIGGRAPH 2000* (2000), ACM Press / ACM SIGGRAPH / Addison Wesley Longman, pp. 511–516.

[PW99] POPOVIĆ Z., WITKIN A.: Physically based motion transformation. In *Proceedings of SIGGRAPH 1999* (1999), pp. 11–20.

[Rad99] RADEMACHER P.: View-dependent geometry. In *Proceedings of SIGGRAPH 1999* (1999), pp. 439–446.

[Ree81] REEVES W. T.: Inbetweening for computer animation utilizing moving point constraints. In *Proceedings of SIGGRAPH 1981* (1981), pp. 263–269.

[SG92] SEDERBERG T. W., GREENWOOD E.: A physically based approach to 2-D shape blending. In *Proceedings of SIGGRAPH 1992* (1992), Catmull E. E., (Ed.), pp. 25–34.

[SL01] SEAH H. S., LU J.: Computer-assisted inbetweening of line drawings: Image matching. In *CAD/Graphics* (August 2001).

[SS01] SHAPIRO L. G., STOCKMAN G. C.: *Computer Vision*. Prentice Hall, Upper Saddle River, N.J., 2001.

[SSSE00] SCHÖDL A., SZELISKI R., SALESIN D. H., ESSA I.: Video textures. In *Proceedings of SIGGRAPH 2000* (2000), ACM Press / ACM SIGGRAPH / Addison Wesley Longman, pp. 489–498.

[TO99] TURK G., O'BRIEN J. F.: Shape transformation using variational implicit functions. In *Proceedings of SIGGRAPH 1999* (1999), ACM Press/Addison-Wesley Publishing Co., pp. 335–342.

[Vap98] VAPNIK V. N.: *Statistical Learning Theory*. Wiley, New York, 1998.

[VDV05] VAN HAEVRE W., DI FIORE F., VAN REETH F.: Uniting cartoon textures with computer assisted animation. In *Proceedings of GRAPHITE 2005* (2005), ACM Press, pp. 245–253.

[WFMS04] WIRTZ S., FISCHER G., MODERSITZKI J., SCHMITT O.: Superfast elastic registration of histologic images of a whole rat brain for 3d reconstruction. In *Medical Imaging 2004, Proceedings of the SPIE* (2004), vol. 5370, pp. 328–334.

[WXSC04] WANG J., XU Y., SHUM H.-Y., COHEN M. F.: Video tooning. *ACM Transactions on Graphics 22*, 3 (August 2004), 574–583.