CARTOON TEXTURES: RE-USING TRADITIONAL ANIMATION VIA METHODS FOR SEGMENTATION, RE-SEQUENCING, AND INBETWEENING

By

Christina Nereida de Juan

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

May, 2006

Nashville, Tennessee

Approved:

Robert E. Bodenheimer, Jr

Benoit M. Dawant

Richard Alan Peters II

Douglas H. Fisher

Odest Chadwicke Jenkins

COPYRIGHT © 2006 by Christina Nereida de Juan

All Rights Reserved

To my mother and father

ACKNOWLEDGEMENTS

In finding my own path toward completion of this dissertation I have learned many lessons, made lifelong friends, and have found a deep appreciation for the dedication and perseverance required to follow this path. Yet it is not possible to complete a doctorate without the personal and practical support of numerous people.

My deepest gratitude goes to my mother and father, for their everlasting love and support, and for all of their sacrifices that made it possible for me to pursue my dreams. Thank you to all my friends for their encouragement and faith in me, especially to Jessica Ellis for always listening, and to Michael Gallucci for teaching me how to think about problem solving in a new way.

I thank Bobby Bodenheimer, my advisor and friend, for giving me guidance and counsel, and for having faith and confidence in me. Bobby always allowed me to choose my own direction with research, and helped navigate through any challenges new to both of us. I am grateful to my committee members, Benoit Dawant, Alan Peters, Doug Fisher, and Chad Jenkins, for their comments and suggestions. In particular Chad, with whom many of the early ideas for this work originated. I have greatly benefited from their advice. It has been a pleasure working with my colleagues, Jing Wang and Betsy Williams, who were always willing to brainstorm ideas. Thank you to Robert Pless for helpful discussions in the early stages of the project, to Zhujiang Cao for many helpful discussions and for providing me with his level set code, to Xia Li for help with the elastic registration code, and to Tao Ju for assistance with generating mesh slicing results.

Finally, to all the wonderful people who gave me my start at Georgia Tech. Jessica Hodgins and the members of the former Animation Lab – Dave Brogan, Ron Metoyer, James O'Brien, and Victor Zordan – who's early influence provided the foundation for my growth as a researcher. I am also grateful for the many conversations with Gabriel Brostow and other members of the CPL – Arno Schödl and Tony Haro – for their early shaping of many of my research ideas.

TABLE OF CONTENTS

Page

LIST OF TABLES . . . . **vii**

LIST OF FIGURES . . . . **viii**

Chapter

I INTRODUCTION **1**

I.1 Research Goals and Contribution 1
I.2 Overview 4

II ANIMATION BACKGROUND **5**

II.1 Animation Pipeline 7
II.1.1 Principles of Animation 8
II.1.2 2D Inbetweening Problem 10
II.2 Interpolation 12
II.2.1 Contour-Based 12
II.2.2 Image-Based 16
II.2.3 Model-based 19
II.2.4 Model-based vs Model-free Representation 21

III RELATED WORK **22**

III.1 Inbetweening for 2D Animation 22
III.1.1 Spline-Based 22
III.1.2 Template-Based 24
III.1.3 Vision-Based 24
III.1.4 Shape Interpolation 25
III.1.5 Other Techniques for 2D Animation 31
III.2 Re-sequencing Animation Data 33
III.2.1 Video 33
III.2.2 Motion 35
III.3 Dimension reduction 38

IV PREPARING TRADITIONAL ANIMATION FOR RE-SEQUENCING **41**

IV.1 Pre-Processing Cartoon Data 41
IV.2 Segmentation of Color Images 42
IV.3 Ad Hoc Method 44
IV.4 Level Sets for Segmentation 45
IV.4.1 Level Sets: Method and Results 47
IV.5 Support Vector Machines 50
IV.5.1 Training SVM Model on Cartoon Images 53

IV.5.2 SVM Segmentation Results 53
IV.6 Summary 59

V DIMENSION REDUCTION FOR RE-SEQUENCING ANIMATION 60

V.1 Dimension Reduction 60
V.2 Distance Metrics 61
V.2.1 L_2 Distance 61
V.2.2 Cross-Correlation Distance 62
V.2.3 Hausdorff Distance 62
V.3 Embedding 63
V.4 Re-sequencing New Animations 66
V.4.1 Post-Processing 73
V.5 Threshold Detection 75
V.6 Re-sequencing Results 76
V.7 Summary 78

VI INBETWEENING A MOTION LIBRARY OF RE-SEQUENCED ANIMATIONS . 80

VI.1 Character Partitioning and Re-assembly 80
VI.2 Shape and Color Interpolation 81
VI.3 Semi-Automatic Inbetweening 81
VI.4 Shape Interpolation using RBFs 84
VI.4.1 Overview of Radial Basis Functions 84
VI.4.2 Interpolating Contours using RBFs 86
VI.5 Texturing or Re-coloring the Intermediate Contours 89
VI.5.1 Summary of Elastic Registration 91
VI.5.2 Results of Re-coloring Contours 92
VI.6 Summary 96

VIISUMMARY AND FUTURE DIRECTIONS 97

VII.1 Summary of Contributions 97
VII.2 Future Directions 99
VII.2.1 Threshold Detection Revisited 99
VII.2.2 Segmenting Black and White Cartoons 99
VII.2.3 Incorporating Principles of Animation 99
VII.2.4 Inbetweening Revisited 101
VII.2.5 Stop-Motion Animation 105

REFERENCES 107

LIST OF TABLES

Table Page

IV.1 Details of traditional animation data sets. * Indicates that the data set was reduced in size by removal of duplicate frames. 42

V.1 Examples of the distance values between pairs of frames using the Hausdorff distance metric on the *Daffy* data set. Adjacent frames in the original data set may not always have a low distance value, as shown in the table. The transition from frame 98 to 99 is an abrupt transition according to the distance metric. 76

VI.1 Evaluating the similarity of the elastic deformation result to the intermediate contour from the RBF contour interpolation step. 93

LIST OF FIGURES

Figure Page

I.1 On the left is the original image, the center shows the desired image mask, on the right is the segmented image. The final segmented image has the character on a constant blue background, a neutral color that does not appear in the character itself, and is easily identifiable for processing only the character. *Wile E. Coyote* is *TM* & ©*Warner Bros. Entertainment Inc.* 2

II.1 Example model sheets and character sheets showing *Dino* in various poses and expressions. *Dino* is *TM* & ©*Warner Bros. Entertainment Inc.* 6

II.2 Here is an example cartoon character at various stages of the traditional animation process. On the left is the line art, in the center is the character after coloring (ink and paint), and on the right is the final scene with the character composited with the background image. *Woody Woodpecker* is *TM* & ©*Universal Studios.* 7

II.3 The pair of images on the left shows an example of the self-occlusion problem for two-dimensional inbetweening. The character's arm covers part of the face in the first frame, but uncovers the face in the second frame. The pair of images on the right shows an example of a silhouette change. In the first frame, the character's face is towards the camera and the nose is in the center of the face, while in the second frame, the character's face has turned to be in profile and the nose is now part of the silhouette. Images from [Blair, 1994]. 11

II.4 Creating inbetweens for a pendulum swinging causes it to appear to shrink as it approaches the middle, then grow as it continues to the last keyframe (left). The correct motion is illustrated on the right. Images modified from Tony White's internet animation tutorial. 13

II.5 On the left is an example of a Bézier curve with the endpoints indicated as P_1 and P_4, and the control points for the tangents at the endpoints indicated as P_2 and P_3. On the right are the *Bernstein* polynomials, which are the blending functions for Bézier curves. Notice that at $t = 0.0$ only the B_1 polynomial is nonzero, meaning that the curve interpolates only one point, P_1. The same is true for the B_4 polynomial at $t = 1.0$, which interpolates point P_4. 15

II.6 The B-Spline blending functions. Notice that at $t = 0.0$ and $t = 1.0$, three of the polynomial functions are nonzero, unlike the Bézier blending functions, therefore no control points will be interpolated. 16

II.7 Example of a cross-dissolve between two images. The left and right show the source and destination images. The middle is the blend at exactly half way between the source and destination. 17

II.8 Example of a warp between two images. The left and right show the source and destination images. The middle is the warped image produced by either warping the source to the destination or vice versa. 18

II.9 Example of a morph between two images. The top left and bottom right show the source and destination images. The middle is the morph, which is the warped images that are cross-dissolved for the final result. 18

II.10 An example of optical flow computed from a pair of cartoon images. *Daffy Duck* is *TM & ©Warner Bros. Entertainment Inc.* 19

II.11 On the left is the input plane with a number of points. On the right is the resulting Voronoi diagram. 20

II.12 An example of Delaunay triangulation. The thick lines are the Delaunay triangles, the dashed lines represent the Voronoi diagram. 20

II.13 An example of defining an implicit function to represent a cartoon character from an image. The blue points (densely sampled) are the zero set, the green points are interior points, the red are exterior points. To facilitate viewing these points, the cartoon character's silhouette is shown instead of the full color frame. 21

III.1 A hand drawn image with the reference skeleton and relative coordinate system. Image from [Burtnyk and Wein, 1976]. 24

III.2 The left and right images are the source and destination images with sets of features indicated as directed line segments. The center image the resulting morph, with warped grid lines from the source and destination. Images are from [Beier and Neely, 1992]. 26

III.3 A visualization of two-dimensional shape transformation between an X shape and an O shape. The translucent surface indicates the isosurface of the three-dimensional variational implicit function created from the two-dimensional shapes represented by the top and bottom planes. Image from [Turk and O'Brien, 1999]. 27

III.4 An example of the boundary and normal constraints indicated by circles and plus marks on the left, an intensity image showing the implicit function on the right. Image from [Turk and O'Brien, 1999]. 28

III.5 A visualization of two-dimensional shape transformation between a pair of cartoon contours. On the left and middle are the two cartoon input shapes shown in perspective and overlayed on the mesh. On the right, the two input cartoon shapes are overlayed together, and the translucent surface indicates the isosurface of the three-dimensional variational implicit function created from the two-dimensional shapes. Notice on the isosurface that details around the head and feet are smoothed and lost because the isosurface balloons out too far from the input shapes. *Daffy* is *TM & ©Warner Bros. Entertainment Inc.* 28

III.6 We tested the method of [Alexa et al., 2000] on a pair of cartoon keyframes. The character was segmented into layers, then the shape transformation was applied to each layer separately. The top row shows the result of the transformation. The bottom row is the same transformation with the triangulation visible at each step. . 30

III.7 A close up view of three triangles from the right arm of the character in Figure III.6. Clearly, the vertex paths indicate that the triangles flipped during shape transformation. 30

III.8 An example of the shape space with the additional inbetweened key-shapes that would fall along the green line, thereby biasing the extended shape space to move away from invalid shapes shown in red. Image from [Bregler et al., 2002]. 33

III.9 A general overview of the capture and retargeting process. The source and target key-shapes are selected by the user, the key weights are determined by finding the warping function required to move from one key-shape to another in the source shape space. These are applied to the target key-shapes in the retargeting step. Image from [Bregler et al., 2002]. 34

III.10 The distance and transition probability matrices before (left) and after (right) filtering. These matrices are generated from analyzing video of a clock pendulum. Notice the probability matrix after filtering matches only forward swings of the pendulum. Image from [Schödl et al., 2000]. 35

IV.1 The top row shows a frame from the synthetic *gremlin* data set before and after processing (the images are cropped). The second row shows an original and cleaned up frame from the *Bugs Bunny* data. An example frame from the *Wile E. Coyote* data in the third row, *Daffy Duck* in the fourth row, *Michigan J. Frog* in the fifth row, and finally *Grinch* in the last row. The segmentation results for *Bugs* and *Coyote* are generated using methods described in this chapter. *Bugs*, *Coyote*, *Daffy*, and *M.J. Frog* ™& ©*Warner Bros. Entertainment Inc.*, *Grinch* ©*Turner Entertainment Co.* 43

IV.2 The deviation from the mean color value for each color channel from a single background pixel is shown as data points around a mean value line. The color of the data corresponds to the color channel. 44

IV.3 On the left is the original image, the center shows the mask generated using the ad hoc method, on the right is the cleaned up mask. *Wile E. Coyote* is ™& ©*Warner Bros. Entertainment Inc.* 46

IV.4 On the left is the original image, on the right is the segmented image that shows when the ad hoc method fails. *Bugs Bunny* is ™& ©*Warner Bros. Entertainment Inc.* 46

IV.5 On the left is the red curve representing Γ, on the right is the level set function ϕ shown as a cone in blue-green. The blue-green surface is called the level set function, because it accepts as input any point in the plane and hands back its height as output. The red front is called the zero level set, because it is the collection of all points that are at height zero. 47

IV.6 Starting on the left and moving right: the synthetic test image, the segmentation mask with $color = RED$, the segmentation mask with $color = GRN$, the segmentation mask with $color = BLU$, and finally the segmentation mask with the color stopping criteria equal to $[RED, GRN, BLU]$. Images obtained courtesy of Dr. Zhujiang Cao. 48

IV.7 On the left is the original image, the center shows the mask generated using level sets, on the right is the segmented image. Images obtained courtesy of Dr. Zhujiang Cao. This *Daffy* image is from the 1951 short "Rabbit Fire" directed by Chuck Jones. *Daffy Duck* is ™& ©*Warner Bros. Entertainment Inc.* 49

IV.8 On the left is the original image, the right shows the mask generated using only the luminance. Images obtained courtesy of Dr. Zhujiang Cao. *Wile E. Coyote* is ™& ©*Warner Bros. Entertainment Inc.* 49

IV.9 Original *Coyote* images from two different cartoons with compression artifacts, seen as blocks in the background sky. *Wile E. Coyote* is ™& ©*Warner Bros. Entertainment Inc.* 50

IV.10 An illustration of the SVM algorithm. Two classes of data are represented as red and blue points, the input space is shown before and after processing, the hyperplane is the line separating the red and blue points in the lower right. In the feature space, the margin is shown in green, the vectors (data points) that constrain the width of the margin are the support vectors. 51

IV.11 The top row shows the input image and the resulting segmentation mask, this was generated using 81 RGB samples and the corresponding optical flow magnitudes at those pixel locations. The samples came from three images in the Bugs Bunny sequence. The bottom row shows the optical flow vectors and one of the images used for the RGB samples. The red circles indicate the pixels selected for the samples. *Bugs Bunny* is ™& ©*Warner Bros. Entertainment Inc.* 54

IV.12 Here is the same image from the Bugs Bunny sequence. In the top row, the image was first blurred using a gaussian filter and the model was trained using one horizontal scanline from one image-mask pair. In the bottom row, the model was trained using only 68 RGB samples from three images. Of all of the segmentation results using SVMs, this last example shows the best classification of the cartoon character. *Bugs Bunny* is ™& ©*Warner Bros. Entertainment Inc.* 55

IV.13 The top row shows the input image and the resulting segmentation mask, this was generated using 108 RGB samples and the corresponding optical flow magnitudes at those pixel locations. The samples came from three images in the Coyote-1 sequence. The bottom row shows one of the images used for the samples and the optical flow vectors for that image. The blue circles indicate the pixels selected for the samples. *Wile E. Coyote* is ™& ©*Warner Bros. Entertainment Inc.* 56

IV.14 The top row shows the input image and the resulting segmentation mask, this was generated using 179 RGB samples and the corresponding optical flow magnitudes at those pixel locations. The samples came from three images in the Coyote-2 sequence. The bottom row shows one of the images used for the samples and the optical flow vectors for that image. The blue circles indicate the pixels selected for the samples. *Wile E. Coyote* is ™ & ©*Warner Bros. Entertainment Inc.* 57

IV.15 The top row shows the input image and the resulting segmentation mask, this was generated using 140 RGB samples and the corresponding optical flow magnitudes at those pixel locations. The samples came from three images in the Coyote-3 sequence. The bottom row shows one of the images used for the samples and the optical flow vectors for that image. The blue circles indicate the pixels selected for the samples. *Wile E. Coyote* is ™ & ©*Warner Bros. Entertainment Inc.* 58

IV.16 On the left is the segmentation result on *Bugs Bunny* using the 68 RGB sample SVM model. On the right is the result of applying morphological operations to clean up the segmentation mask. 59

V.1 An edge map in the center, and distance map on the right, for one image from the *Daffy Duck* data set. The edges on the edge map have been enhanced for easier viewing. These images represent only a single image in the data set. The edge map and distance map are computed for every image in the sequence. ™ & ©*Warner Bros. Entertainment Inc.* 62

V.2 L_2, Correlation-based and Hausdorff distance matrices for the *Daffy* data set. Each distance matrix has been normalized to the range of zero to 255 only for visual comparison. The colormap is shown on the right. 64

V.3 Hausdorff distance matrices for all cartoon data sets. Each distance matrix has been normalized to the range of zero to 255 only for visual comparison. The colormap is shown on the right. 65

V.4 Results showing the residual variance and three-dimensional projection of the neighborhood graph generated with ST-Isomap using the Hausdorff distance matrix on the *Bugs Bunny* data set. 67

V.5 Results showing the residual variance and three-dimensional projection of the neighborhood graph generated with ST-Isomap using the Hausdorff distance matrix on the *Wile E. Coyote* data set. 68

V.6 Results showing the residual variance and three-dimensional projection of the neighborhood graph generated with ST-Isomap using the Hausdorff distance matrix on the *Daffy Duck* data set. 69

V.7 Results showing the residual variance and three-dimensional projection of the neighborhood graph generated with ST-Isomap using the Hausdorff distance matrix on the *Michigan J. Frog* data set. 70

V.8 Results showing the residual variance and three-dimensional projection of the neighborhood graph generated with ST-Isomap using the Hausdorff distance matrix on the *Grinch* data set. 71

V.9 Results showing the residual variance and two-dimensional projection of the neighborhood graph generated ST-Isomap using the Hausdorff distance matrix on the *gremlin* data set. 72

V.10 Illustration of applying a pre-computed velocity vector from an original sequence to a re-sequenced frame. Top left shows a frame in a re-sequenced animation, bottom left shows the next frame in the re-sequenced animation. Top right illustrates the velocity vector computed for the frame in the top left. Bottom right shows the realignment of the character from the second re-sequenced frame. *Daffy* is ™& ©*Warner Bros. Entertainment Inc.* 74

V.11 Illustration of applying a scale factor *s* to a pair of re-sequenced frames. The top row shows the original frames that appear in a re-sequenced animation. The difference in their scale is apparent. The bottom row shows the same frames after the scale factor is applied. *Daffy* is ™& ©*Warner Bros. Entertainment Inc.* 75

V.12 An example of good, bad, and acceptable transitions for the *Daffy* data set from a path generated using ST-Isomap with two temporal neighbors. The pairs of frames shown correspond with the values shown in Table V.6. *Daffy* is ™& ©*Warner Bros. Entertainment Inc.* 77

V.13 A filmstrip of two paths without any post-processing. The bottom row shows the path generated from the *Daffy* data set with three frames removed. The top row shows the same path with inbetweens inserted at the point of highest transition cost, in this case between frames 326 and 77. ™& ©*Warner Bros. Entertainment Inc.* . 78

V.14 An example of good and bad transitions for the *Frog* data set. The first pair of images demonstrates a good transition from frame 22 to 27 with a cost of 0.198132. The second pair of images demonstrates a bad transition from frame 12 to 109 with a cost of 0.609729. ™& ©*Warner Bros. Entertainment Inc.* 79

VI.1 Moving from left to right: the first image is an example key image that will be used for inbetweening, the second image is the body layer, the third image is the head layer, the fourth image is the left arm layer, and finally the last image is the right arm layer. The character was partitioned into layers manually. *Bugs Bunny* is ™& ©*Warner Bros. Entertainment Inc.* 81

VI.2 Example of nine feature lines selected for image morphing between a pair of key image head layers. *Bugs Bunny* is ™& ©*Warner Bros. Entertainment Inc.* 82

VI.3 Example of image morphing of the head layer between key image 1 on the left and key image 2 on the right. *Bugs Bunny* is ™& ©*Warner Bros. Entertainment Inc.* . . 82

VI.4 Example of image morphing of the head layer between key image 1 on the left and key image 2 on the right, with further refinement of the feature lines. The top row shows results using 24 feature lines, the bottom row shows results using 30 feature lines. *Bugs Bunny* is ^{TM}& ©*Warner Bros. Entertainment Inc.* 83

VI.5 An example of the automatically generated contour and normal points which serve as the input to the implicit surface generation step. 86

VI.6 The top row shows an original key image on the left, the partitioned layers on the right. Notice that the head has been registered to the key image in Figure VI.8(d). The bottom row shows an example of the contours used for creating the RBF solution. *Daffy* is ^{TM}& ©*Warner Bros. Entertainment Inc.* 87

VI.7 The top image is the RBF solution for the *Daffy Duck* head layer. The head images were registered and two additional constraints were used. The bottom image is the RBF solution for the *Daffy Duck* body layer. 88

VI.8 The final inbetween frame generated using RBF interpolation method with layering and constraints. The key images are shown on the left and right. *Daffy* is ^{TM}& ©*Warner Bros. Entertainment Inc.* 89

VI.9 The final inbetween frame generated using RBF contour interpolation method with layering. The key images are shown on the left and right. *Bugs Bunny* is ^{TM}& ©*Warner Bros. Entertainment Inc.* 90

VI.10 The final inbetween frame generated using RBF contour interpolation method with layering and aligning the head and arm layers. The key images are shown on the left and right. *Wile E. Coyote* is ^{TM}& ©*Warner Bros. Entertainment Inc.* 90

VI.11 Moving from left to right: the first image is a close up of the first key image head layer, the second image is this key image after using an affine transformation to warp the image to match the inbetween contour, the third image is the automatically generated inbetween contour, the last image is an overlay showing the transformed key image atop the inbetween contour. *Daffy Duck* is ^{TM}& ©*Warner Bros. Entertainment Inc.* 91

VI.12 A comparison of results using elastic deformation for generating inbetweens on the *Bugs Bunny* head layer. On the left and right are the original key images. The image labelled “Contour” is the RBF contour interpolation result. “Deformation 1” is the result going from *key image 1* to *key image 2*, while “Deformation 2” is the result going from *key image 2* to *key image 1*. *Bugs Bunny* is ^{TM}& ©*Warner Bros. Entertainment Inc.* 93

VI.13 Moving from left to right: the first image is a close up of the first key image head layer, the second image is a close up of the second key image head layer, the third image is the automatically generated inbetween texture, the fourth image is an overlay showing the intermediate texture overlayed on the inbetween contour, and the fifth image is the final inbetween for the head layer. *Daffy Duck* is ™& ©*Warner Bros. Entertainment Inc.* 94

VI.14 The final inbetween frame generated using RBF contour interpolation method with elastic registration providing the color and texture information. The key images are shown on the left and right. *Daffy Duck* is ™& ©*Warner Bros. Entertainment Inc.* 94

VI.15 The final inbetween frame generated using RBF contour interpolation method with elastic registration providing the color and texture information. The key images are shown on the left and right. *Bugs Bunny* is ™& ©*Warner Bros. Entertainment Inc.* 95

VI.16 The final inbetween frame generated using RBF contour interpolation method with elastic registration providing the color and texture information. The key images are shown on the left and right. *Wile E. Coyote* is ™& ©*Warner Bros. Entertainment Inc.* 95

VI.17 A comparison of results using image morphing and elastic deformation for generating inbetweens on the *Bugs Bunny* head layer. On the left and right are the original key images. The image labeled “image morph” is the result of using 30 feature lines and is half way between the two key images. The images labeled “deformation 1” and “deformation 2” show the results of using the elastic registration. “Deformation 1” is the raw result, while “Deformation 2” is the manually touched up result. A small amount of correction was required on one eye and ear. *Bugs Bunny* is ™& ©*Warner Bros. Entertainment Inc.* 96

VII.1 Two pairs of images at scene cuts from the *Daffy* data set, illustrating that these images would result in a high transition cost because of the dissimilarity of the character in the images. The top row is the first scene cut, the bottom row is the second scene cut. *Daffy Duck* is ™& ©*Warner Bros. Entertainment Inc.* 100

VII.2 Two sequential images from the 1936 animated short “Porky’s Road Race,” produced by Leon Schlesinger. Notice the lighting changes on *Porky’s* face and the large white spot that appears in the second frame behind the car. Also, one of *Porky’s* dimples is missing in the first frame. *Porky Pig* is ™& ©*Warner Bros. Entertainment Inc.* 100

VII.3 An example of how to set up a mesh for use with the MVC slicing method. On the top row the two input images and the normals for the data points, red points indicate one image and green points indicate the second image. On the bottom row, the resulting mesh shown from four viewpoints. *Bugs Bunny* is ™& ©*Warner Bros. Entertainment Inc.* 102

VII.4 Slices of the mesh generated using RBF interpolation of pixel location and colors. In the top row, the slices at 0.1 distance to 0.5 distance from the top and bottom planes, and in the bottom row, the slices at 0.6 distance to 0.9 distance from the top and bottom planes. Images obtained courtesy of Dr. Tao Ju. *Bugs Bunny* is ™ & ©*Warner Bros. Entertainment Inc.* . . . 103

VII.5 Several viewpoints of the manually created mesh. The top and bottom planes are the two key images. The bottom row shows the same mesh with the underlying wireframe highlighted. *Bugs Bunny* is ™ & ©*Warner Bros. Entertainment Inc.* . . . 104

VII.6 Examples of a stop-motion animation character *Gromit*, from the film "The Wrong Trousers," exhibiting changing lighting conditions. ©*Aardman / Wallace and Gromit Ltd. 1993*. . . . 106

CHAPTER I

INTRODUCTION

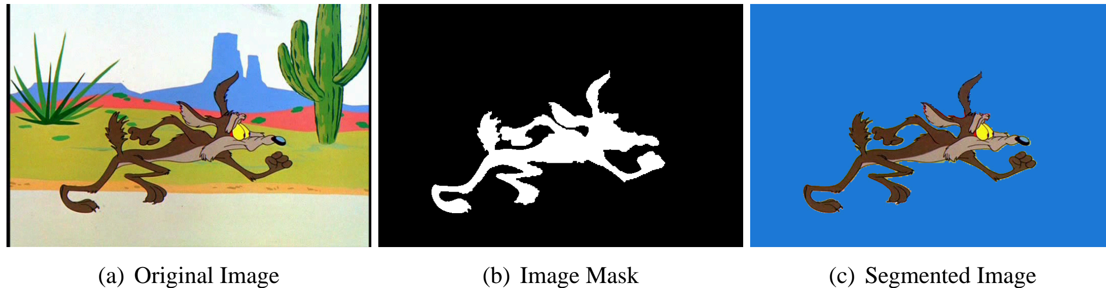
Traditional animation can be described as creating the illusion of a moving scene using a sequence of hand drawn images. A typical animation is composed of twenty-four frames per one second of animation. Other forms of animation have become increasingly prevalent, such as computer generated animation, dynamic simulation, and motion capture. This dissertation is concerned with traditional animation, specifically how to re-use existing cartoons.

Even with many other forms of animation available, traditional animation is still a popular art form. However, hand drawn animation remains a very tedious and time-consuming task despite many advances in technology that have improved the speed of producing a traditional animation. For a typical animated television series, artists bring life to familiar cartoon characters for every episode, yet no method exists that would allow them to re-use their drawings in novel situations. Clearly a character running away could be used again for future episodes. Software packages such as Toon Boom Technologies, [Fekete et al. 1995], can create simple inbetweens based on vector animation. Although an animator could re-use the original models of the characters, the basic animation still has to be created, and these animations tend to lack the expressiveness of familiar styles, such as the distinctive style of animations by Chuck Jones (creator of *Wile E. Coyote*).

The same issues arise when creating three-dimensional models for cartoon characters and "'toon-rendering" them. "'Toon-rendering" is a technique that can render three-dimensional scenes in styles that approximate the look of a traditionally animated film; it is often called "'toon shading." Aside from some of the issues already mentioned, 'toon-rendering presents many of its own challenges in creating a three-dimensional character that looks hand drawn. When the three-dimensional character moves, issues such as where to draw edge lines, how thick the lines should be, how they appear and disappear, etc., if not handled properly, are tell-tale signs that the character does not have the same style as if it was hand drawn.

I.1 Research Goals and Contribution

The goal of this research is to create novel animations from a library or database of existing cartoon data. Many of the tools and techniques developed in computer animation are designed to allow animators to obtain the expressiveness of traditional animation more easily, e.g., [Lasseter 1987; Fekete et al. 1995; Rademacher 1999; Kowalski et al. 1999]. Separately, a body of work exists to allow animators to re-use motion capture data to create new animations, e.g., [Gleicher 1998; Rose et al. 1998; Kovar et al. 2002; Lee et al. 2002; Arikan and Forsyth 2002]. In contrast, there has not been much study of the problem of re-using traditional animation to create new animation [Bregler et al. 2002; de Juan and Bodenheimer 2004]. 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 capture and manipulate it. This dissertation presents several methods that allow the incorporation of traditional



(a) Original Image (b) Image Mask (c) Segmented Image

Figure I.1: On the left is the original image, the center shows the desired image mask, on the right is the segmented image. The final segmented image has the character on a constant blue background, a neutral color that does not appear in the character itself, and is easily identifiable for processing only the character. *Wile E. Coyote* is *TM* & ©*Warner Bros. Entertainment Inc.*

animation into a motion library for re-use.

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 Chapters II, III, and IV.

One goal is to develop a method for semi-automatically segmenting the characters by creating an image mask. The masks are then applied to the original images to place the cartoon character on a neutral, or known, background. Figure I.1 shows an example of an original image, an ideal image mask, and the final segmented cartoon character on a neutral background. Three approaches will be presented, along with the benefits and drawbacks of each method: an ad hoc method using the probability of a pixel color being the character, level sets defining a speed function based on intensity and color information, and applying machine learning using Support Vector Machines [Chang and Lin 2001] to train and classify color pixels as being part of a character or part of the background.

Once the segmentation problem is solved, we can begin to address the challenge of re-using the animation. A true re-usable library of animation is closest in spirit to the system discussed in Chapter V. Drawing inspiration from the idea of video textures [Schödl et al. 2000], sequences of similar-looking cartoon data are combined into a user directed sequence. The primary goal of re-using the cartoons is to re-sequence cartoon data to create new motion from the original data that retains the same characteristics and exposes similar or new behaviors. The number of new behaviors that can be re-sequenced is restricted by the amount of data in our library for each character. Starting with a small amount of segmented cartoon data, we use an unsupervised learning method for manifold learning to discover a lower-dimensional structure of the data. The user selects a de-

sired start and end frame and the system traverses this lower-dimensional manifold to re-sequence the data into a new animation. Our method is model-free, i.e., no a priori knowledge of the drawing or character is required. The user does not need the ability to animate, or know what an acceptable *inbetween* is (defined below), since the data is already provided. The system can detect when a transition is abrupt, allowing the user to inspect the new animation and determine if any additional source material is needed. Minimal user input is required to generate new animations, and the system requires much less data than the video textures method for re-sequencing. Also, because the new animation is created from re-sequencing existing hand drawn animation, the new sequence will retain some of the *principles of animation* (discussed in Chapter II) since the images were already drawn with those principles in mind.

However, one of the limitations of the system described in Chapter V is the inability to generate new images when a visual discontinuity is detected in a re-sequenced animation. Theoretically, a visual discontinuity occurs because the manifold is not sampled densely enough by the data for its structure to be parameterized. Hand-drawn cartoon art is always going to be sparse, thus representing a fundamental obstacle for manifold learning techniques. Therefore we need to explore other techniques to overcome this limitation.

When a visual discontinuity is encountered in re-sequencing and it is determined that more source material is needed, the problem becomes one of *inbetweening*. Inbetweening is the process of drawing intermediate images that fill in the space between a pair of keyframes. An introduction to the principles of animation and definitions of terminology used in traditional animation is given in Chapter II. Addressing this issue of discontinuity returns one to the two-dimensional inbetweening problem presented in [Catmull 1978] and discussed in Chapters II.1.2 and III, although the problem we address in this dissertation is more limited in that it suffices to generate inbetweens between two images that are somewhat similar, not two keyframes. Our work on this problem is presented in Chapter VI.

The inbetweening research goal is a model-free, image-based method for generating an inbetween frame semi-automatically. By using radial basis functions (RBFs), contour information can be interpolated between two keyframes. To deal with occlusion, the character will be partitioned into layers (such as head, arm, body, etc.) manually, and each layer interpolated separately. The layers are reassembled automatically. Using RBF interpolation of keyframe contours, the goal of inbetweening is to employ a method for filling in the inbetween contour with the appropriate texture and color information taken from the two keyframes. Using a parameter-free, non-rigid elastic registration algorithm [Wirtz et al. 2004; Li et al. 2006] on the two keyframes, the resulting registration provides the texture information for an intermediate image. Image registration provides an improvement for texture filling over current methods that require the contours to be parameterized as polygons with their interiors then triangulated compatibly to preserve the texture information (Chapter III.1.4).

The most desirable qualities of traditional animation are the nuances an artist adds to each character, giving that character personality and style. The high-level goal of this work is to enable these

abilities in an artist, so our techniques use an animator as a guide in both building a library and using it. 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 up the process. Ensuring that the artist remains involved in the inbetweening process, albeit minimally, should provide a higher level of quality in the resulting animations.

I.2 Overview

The remainder of this dissertation is organized as follows. Chapter II introduces the main concepts in traditional animation and applications to animation. Chapter III discusses related research in two-dimensional animation, re-sequencing motion, and dimension reduction and its application to animation. Chapter IV describes how existing cartoon data is segmented to be prepared for use in re-sequencing. The contribution of this chapter is a robust and semi-automatic method for segmenting cartoon images. Chapter V presents the methods used in re-sequencing cartoon animation. Two main contributions of this chapter are the successful use of manifold learning applied to cartoon data for re-use and in identifying an appropriate distance metric for computing the similarity of cartoon images. Chapter VI describes generating new images, or inbetweens, for creating more visually compelling re-sequenced animations. Some of the challenges of inbetweening are overcome, and the contribution of this chapter is providing a semi-automatic method for inbetweening a pair of cartoon images. Chapter VII summarizes this work and the research contributions to the field of computer animation, and discusses future directions.

CHAPTER II

ANIMATION BACKGROUND

In addition to traditional hand drawn animation, several other forms of animation exist, and have become more prevalent in film and television. Unlike traditional hand drawn animation, computer generated animation can be either two-dimensional or three-dimensional, with the latter being most common. A three-dimensional model of a character is created using computer software and an animation is created by playing a sequence of still images the computer renders from the three-dimensional models. Procedural animation methods such as dynamic simulation create animation from carefully specified physics-based control systems that specify how objects or articulated characters should move in and interact with a virtual environment. Dynamic simulation is useful for modeling mechanical events or complex fluid flow, but also requires a level of expertise in the domain to accurately model the internal physics, and typically lacks the creative and expressive nature of hand drawn animation. Motion capture involves sensing, digitizing, and recording a subject's motion from markers placed on the subject. The recorded motion is processed to be applied to a three-dimensional model of an articulated character, thus driving its movements from the actor's motion. Motion capture is used extensively in film and video games, and examples of dynamic simulation can be seen in film special effects like smoke, fluids and fire, particularly in animated films. Stop-motion animation, like traditional animation, is extremely time consuming. To produce a stop-motion animation, the characters are static objects, typically modelled out of clay, and made to have the illusion of motion. A camera films one frame, stops to move the characters by a small amount, then the camera proceeds to film the next frame. This continues at a rate of 24 frames per second. When the film is run, the static objects appear to have fluid motion. Stop-motion is almost as old as film-making itself, dating back to 1898 with a film called *The Humpty Dumpty Circus* by Albert E. Smith and James Stuart Blackton. Stop-motion animation has a quality unlike any other form of animation, and spans a variety of films like the 1933 *King Kong* and *Jason and the Argonauts* in 1964, to more the more recent *Wallace and Gromit* films and *Tim Burton's The Nightmare Before Christmas*. Traditional animation dates back to the early 1900's. Historically, the first short animated film was *Humorous Phases of Funny Faces* released in April of 1906, once again by newspaper cartoonist J. Stuart Blackton, who pioneered "stop frame" or stop-motion animation. In *Funny Faces*, Blackton uses both stop-motion and hand drawn faces on a chalkboard. Winsor McCay is considered by many as the first animator to produce popular drawn animations such as *Little Nemo* in 1911 and *Gertie the Dinosaur* in 1914. For *Little Nemo*, Mr. McCay drew and colored all 4,000 frames himself. In 1928, Walt Disney was the first to incorporate synchronized sound with the animation in *Steamboat Willie*. These are only three of many contributors to the early developments and technological advancements made in traditional animation.

Creating a traditional animation involves a great deal of time while skilled artists draw every frame. The traditional animation pipeline begins with a lead animator, or artist, creating each char-

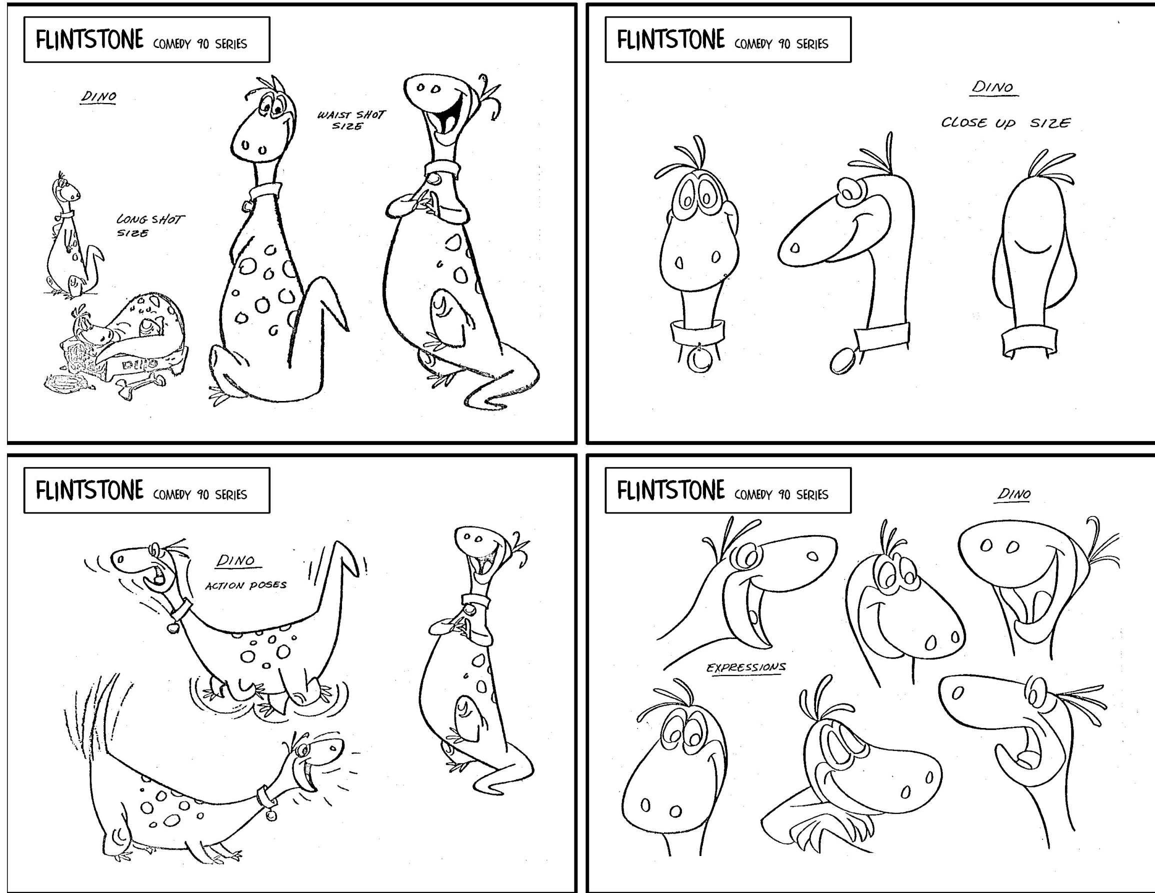


Figure II.1: Example model sheets and character sheets showing *Dino* in various poses and expressions. *Dino* is ^{TM}& ©*Warner Bros. Entertainment Inc.*

acter that will appear in the animation. The lead animator will draw *model sheets* for each character that shows what the character should look like in a neutral pose from various viewpoints. *Character sheets* are also drawn for each character, which show the character in extreme poses. The model and character sheets are used by the *clean-up artists* and *inbetween animators*, acting as a recipe for how that character should be drawn. Figure II.1 shows examples of model and character sheets. After the director approves the model sheets, the lead animator will begin drawing the keyframes for a particular shot or scene. Depending on the level of detail needed for a particular scene, the lead animator may draw all of the frames or only certain keyframes. These keyframes then go on to the clean-up artists to clean the lines of the keyframes. After, the frames go to the inbetween artists who draw the missing frames. At this stage in the pipeline, the animation consists of line drawings for each scene. Then, the line art must be colored frame by frame. Finally, the colored art is composited with a background plate and any additional effects are added. This entire process is repeated for each film or cartoon television series episode. Figure II.2 shows an example of a character as line art, colored art, and finally composited into the background.

The idea of re-using existing cartoon animation to speed up the process of creating a new two-

(a) Line Art

(b) Colored Art

(c) Final Composite

Figure II.2: Here is an example cartoon character at various stages of the traditional animation process. On the left is the line art, in the center is the character after coloring (ink and paint), and on the right is the final scene with the character composited with the background image. *Woody Woodpecker* is ™ & ©*Universal Studios.*

dimensional animation is novel. A number of issues must be addressed to re-use cartoon animations, such as how to compare the images to determine similarity, which frames can be used in succession, and how to preserve the characteristics defining the character that the animator has drawn into each image. Thus far, a general overview of the process of creating a traditional animation has been described. In Chapter II.1, the animation pipeline is broken down further and comparisons are made between traditional and computer-generated animation, highlighting the difficulties that arise with two-dimensional animation. The importance of *principles of animation* are discussed, and the two-dimensional inbetweening problem is defined. In Chapter II.2, several methods for interpolation are introduced.

II.1 Animation Pipeline

Examining the traditional animation pipeline in more detail, we describe the aspects of the pipeline that have changed over the years, and the differences between two-dimensional and three-dimensional animation. The classic two-dimensional animation process is a sequential pipeline with painting of the background images going on in parallel. The steps are:

1. Story is written
2. Visual development and Character development – the look of the film is decided; artists design the characters producing model sheets and character sheets.
3. Layout a storyboard – the pace of the film is set, the script is divided into scenes with dialog and music, timing of all the scenes are set up, and emotional ups and downs are tracked.
4. Record a sound track
5. Animate the keyframes
6. Assistant animator draws some inbetweens
7. Inbetweener draws remaining frames
8. Film the drawings from paper for a video pencil test

9. Xerox copy or ink trace the drawings onto acetate cels
10. Ink and paint – fill in the color and final ink lines for the characters on acetate cels
11. Check for errors
12. Final film from acetate cels and backgrounds
13. Edit film

A modern animation pipeline changes only after step seven. Most studios either draw the keyframes and inbetweens digitally, or the pencil drawings (line art) are scanned to make a digital pencil test. Real acetate cels are no longer used, and the "ink and paint" is also done digitally, with many studios using software to fill the line art semi-automatically. The film is composited and edited digitally. The background images are sometimes painted and scanned to make digital copies, although it is also possible to generate the background images using imaging software.

For a three-dimensional computer animated film, the pipeline differs by step four, in that the keyframes are not drawn by an animator. Rather, the animator positions a three-dimensional model of the character at key times using three-dimensional animation software. The movement of the character is generated by interpolating in three-dimensions, usually using splines, and inbetweens are generated by sampling and rendering at appropriate intervals. The equivalent of a "pencil test" is a quick rendering of the animation without any effects such as texture and lighting. The equivalent of "ink and paint" can be viewed as the final rendering with all textures, lighting, and special effects in place.

The introduction of computers in the traditional animation pipeline has improved or automated several aspects of the pipeline. These improvements have focused on such tasks as texture mapping the cels (or frames) [Corrêa et al. 1998], creating shadows [Petrovic et al. 2000], or retargeting the motion of one character onto another character [Bregler et al. 2002]. Yet, many other aspects of the pipeline remain challenging problems. The most difficult problem is still how to inbetween a two-dimensional hand drawn character.

II.1.1 Principles of Animation

Despite the differences between traditional hand drawn animation and three-dimensional computer generated animation, the art of animation consists of principles that should be incorporated into both art forms that help create the illusion of life. Frank Thomas and Ollie Johnston [Thomas and Johnston 1981], two of the original Disney animators from the 1940's, described these fundamental principles of traditional animation as:

1. SQUASH AND STRETCH – The most important principle was the discovery that objects composed of flesh or soft tissue distort their shape during an action, while maintaining the same volume whether crouched or elongated.
2. ANTICIPATION – Preparing the audience for the next action before it occurs by preceding a major action with a specific motion that anticipates what is about to happen, such as winding up before running.

3. STAGING – Dating back to classical theater, staging is presenting an idea so that it is completely and unmistakably clear. For animation, examples of staging are recognizable personalities, clearly visible expressions, and presenting a mood that affects the viewer, thereby communicating completely with the audience.
4. STRAIGHT AHEAD ACTION AND POSE-TO-POSE – Two different approaches to creating the animation: in the first, the animator works from the first frame straight through to the last in the scene; in the second, the animator creates each keyframe, planning the action throughout the scene and refining each key pose. Both have advantages, with the former spontaneity, the latter clarity and strength.
5. FOLLOW THROUGH AND OVERLAPPING ACTION – The completion of one action and establishing its relationship to the next action by extending the end of the first action with some parts of the character coming to rest at different times.
6. SLOW IN AND OUT – Spacing the inbetween frames close to each extreme keyframe with only one or two half way between the keys, to achieve subtlety of timing and staging.
7. ARCS – The path of motion to create natural looking movement. This principle is one of the most difficult to apply accurately, since drawing an inbetween halfway between the two keyframes linearly is easier than on an arc.
8. SECONDARY ACTION – Any action that results from the main action (e.g., clothing moving as a result of the character moving), and that should support the main action.
9. TIMING – Spacing actions to distinguish the personality of characters, and the speed, weight, and size of objects.
10. EXAGGERATION – Accentuating the essence of an idea, emotion, or movement by distorting the drawing to the point of extreme realism (like a caricature of reality).
11. SOLID DRAWING – Important for two-dimensional animation, describes the shape of the object being animated such that it has volume and flexibility, strength without rigidity, and is pliable.
12. APPEAL – Anything that makes the viewer enjoy looking at the drawing, a cute animal with large eyes, a dramatic villain, etc.

Lasseter [Lasseter 1987] discusses in detail eleven of these principles and their importance for producing high quality three-dimensional animation. Regardless of the medium of animation used, applying these principles has the same meaning for the motion or action. Creating motion for traditional hand drawn animation is achieved by drawing a sequence of two-dimensional images. Generating motion for a three-dimensional computer animation is achieved by using a computer model in three-dimensional space and positioning the model in key poses (or keyframes), allowing the computer to generate the inbetween frames. Lasseter describes how the principles of timing, anticipation, staging, follow-through, overlap, exaggeration, and secondary action can be applied in the same way for both two-dimensional and three-dimensional animation. However, applying the remaining principles changes because of the differences in the animation medium used. For

example, for three-dimensional animation, squash and stretch must be applied by deforming the model of the character. If the model is articulated with rigid limbs, distorting its shape is not trivial and can cause self intersection or pinching problems in the mesh. However, squash and stretch in hand drawn animation is applied by drawing the character accordingly. Two recent examples of applying extreme deformation to a three-dimensional model can be seen on *Scrat* in Blue Sky Studio's *Ice Age* and on *Elastigirl* from Pixar's *The Incredibles*. Both of these films showcase some of the most challenging aspects of animation and the time and complexity involved in creating these expressive characters.

The principles of animation became the foundation of how to create lifelike and compelling animated characters, which was important for training new animators in the techniques that distinguished the animations produced at the Disney studios in the early 1940's [Thomas and Johnston 1981]. While an artist can understand and learn these principles and how to apply them to their drawings, expressing these principles in a form that a computer can recognize is not trivial. By automating many of the processes of creating an animation using computer software, an interesting research question becomes how can these principles that act as constraints for how the artist can draw the character be formulated as mathematical constraints to be used by the computer. One challenge in our work of re-using existing cartoons is to preserve these principles that were drawn by the animator. Being able to represent these concepts as mathematical constraints is an important step in preserving the style of the existing cartoons when they are re-used to produce new animations.

II.1.2 2D Inbetweening Problem

The most challenging aspect of automating the traditional animation pipeline is inbetweening. This problem arises because each two-dimensional drawing is really an artist's representation of a three-dimensional character. Thus, trying to create an inbetween from a pair of two-dimensional drawings automatically is difficult. The motion depicted by the drawings can be classified into two categories: (1) transformations in the image plane or drawing canvas (the x-y plane), and (2) transformations outside the image or drawing plane. In the first category, typical transformations are rotations around the z-axis and translations within a plane parallel to the x-y plane. These transformations are usually easy to deal with, and the success of inbetweening them depends on the representation of the image and the interpolation method used. In the second category, the transformations are typically rotations around the x- or y-axis. These transformations are the key difficulty for automating the two-dimensional inbetweening process because these are the transformations that indicate that the drawing is really a three-dimensional object that is represented in two-dimensions. Two aspects of these transformations that cause the most problems in two-dimensional inbetweening are self-occlusion, for example if the character's arm crosses over his body; and silhouette changes or correspondence information being lost, for example when the character's head rotates from facing the camera to a profile view. The silhouette changes such that the internal features of the face are now part of the exterior contour. In a profile view of a character, the artist may draw the face anatomically incorrectly, for example with both eyes visible, emphasizing some expression or action that

Figure II.3: The pair of images on the left shows an example of the self-occlusion problem for two-dimensional inbetweening. The character's arm covers part of the face in the first frame, but uncovers the face in the second frame. The pair of images on the right shows an example of a silhouette change. In the first frame, the character's face is towards the camera and the nose is in the center of the face, while in the second frame, the character's face has turned to be in profile and the nose is now part of the silhouette. Images from [Blair, 1994].

is to follow. Figure II.3 shows typical examples of self-occlusion and silhouette changes.

Inbetweening for three-dimensional animation is not a problem because these issues of self-occlusion and silhouette changes are "defined" away. It is inherent in the definition of a three-dimensional model that all frame to frame correspondence is known. As aforementioned, creating the motion for a three-dimensional animation begins with an artist positioning a computer model at key poses (keyframes). The model is represented in three-dimensional space, so there is no loss of information by losing one dimension. A typical model for a three-dimensional character has a skeletal structure made up of limbs and joints, and usually has inverse kinematics chains that define and restrict the motion of the model. The limbs are usually a fixed length, though allowing for the incorporation of the principles of animation such as squash and stretch would indicate that some of the limbs do not have to remain of a fixed length. Depending on the complexity of the model, the limbs and joints may have a muscle layer followed by several skin layers. The external representation of the model is typically a polygonal mesh. The movement of the limbs and joints will drive the muscles and cause deformations to appear on the skin layer, which deform the external mesh. All of the three-dimensional information, from the position and orientation of the joints to the position and orientation of each vertex on the external mesh, are known for every keyframe the artist positions. Computer modeling and animation software allows the artist to simply key the position of the model at specific time intervals, and the software will interpolate between the key poses to generate the inbetweens automatically. The method of interpolation, which parts of the model get interpolated, and the timing of the interpolation are all tunable parameters of the software generating the inbetween frames. For example, creating the slow-in/slow-out timing is done by adjusting timing curves for each key pose. There is no literature on three-dimensional inbetweening because it is just interpolation between identical shapes with all information known about the shapes.

With so much control over the three-dimensional model, and the fact that inbetweening is done for free, one may ask why not create the animation with the three-dimensional model then 'toon-render it as a two-dimensional scene. Adding a great deal of deformation, like squash and stretch

or incorporating other principles of animation to a three-dimensional model of a character is often challenging, requiring the skills of a talented artist. Even when 'toon-rendering a three-dimensional character, one cannot expect it to look like the traditionally hand drawn *Wile E. Coyote* getting flattened or stretched in a visually extreme manner. Another problem with 'toon-rendering is the aforementioned silhouette changes, since the renderer can only produce accurate images from the geometry of the model. Take the head turning example again, as previously mentioned, an artist may draw a character in profile anatomically incorrectly, and in the case of a three-dimensional model it would be geometrically incorrect. The 'toon-rendered version of the model will be geometrically correct, therefore lacking the expressiveness of a hand drawn character.

II.2 Interpolation

In this section, we give an introduction to the many uses of interpolation for computer animation and the terminology of the field. Many of the methods discussed here are basic concepts taught in computer graphics and animation courses, and more details can be found in texts such as [Foley et al. 1990]. This section provides the reader with a short primer on interpolation methods typically used in computer animation.

A variety of algorithms have been applied to the inbetweening problem, specifically methods of interpolation. Interpolation is a key issue for computer generated animations. For example, in three-dimensional keyframe animation, where an animator positions a character in three-dimensional space at key times, the remaining frames are automatically generated by interpolating the positions and orientations of the character's joints. Interpolation determines the path that the character will take based on a curve that passes through a given set of control points, in this case the character's joints as positioned by the animator. The control points do not always have to be joint positions and orientations, they can be any list of values associated with a given parameter at the specific keyframe. The method of interpolation used depends on the properties that the desired path should have, such as an arc (one of the principles of animation), whether the points should be interpolated exactly or approximated, and global or local control of the method. While there are many methods for interpolation, this section discusses those that are most applicable to the inbetweening problem. That is, the interpolation methods described here have been applied to images or line drawings, as opposed to a set of joint positions and orientations of a three-dimensional character.

II.2.1 Contour-Based

Interpolation of contours can be accomplished using linear interpolation or spline interpolation. Linear interpolation is the simplest, most popular and widely used. Linear interpolation in one dimension is simply connecting a pair of points with a straight line. Specifically, let t be a number (or time interval) between 0 and 1, then the linearly interpolated value for the inbetween point $p(t)$ is:

$$p(t) = (1-t) \cdot p_1 + t \cdot p_2 \tag{II.1}$$

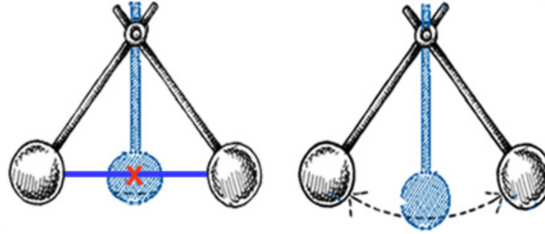


Figure II.4: Creating inbetweens for a pendulum swinging causes it to appear to shrink as it approaches the middle, then grow as it continues to the last keyframe (left). The correct motion is illustrated on the right. Images modified from Tony White's internet animation tutorial.

where p_1 and p_2 are the two keyframe points. To apply this to a pair of contours, the contours can be sampled as sets of data points, and each corresponding point from the first contour will be linearly interpolated to the second contour. The inbetween contour is then a set of data points at any intermediate value between the two input contours. However, calculating inbetweens using a linear interpolation of Cartesian coordinates does not preserve shapes or proportions; for example, if a pendulum rotates in the plane by 90°, the path that the end of the pendulum would take if it were linearly interpolated would be a straight line. The pendulum appears to shrink, then grow as it reaches the second keyframe. The desired path of the end is an arc, which requires other methods for interpolation other than linear. Figure II.4 illustrates the problem with linear interpolation of inbetweens.

Splines are a mathematical means of representing a curve, by specifying a series of points at intervals along the curve and defining a function that allows additional points within an interval to be calculated [Foley et al. 1990]. The function to approximate a curve is accomplished by means of a series of polynomials over the adjacent intervals along the curve and have high order continuity. Continuity is a mathematical measure of smoothness, i.e., the number of continuous derivatives of the curve equation. Zero-order continuity (C^0) ensures a positional continuity at a point along the curve. First-order continuity (C^1) ensures positional and tangential continuity at some point along the curve. Second-order continuity (C^2) ensures positional, tangential, and curvature continuity (the instantaneous rate of change of the tangent vector) at some point on the curve. Continuity is of concern for interpolation when a series of piecewise curves are joined together to define the path of animation. There are three major types of curves: *Hermite*, which are defined by a pair of endpoints and the tangent vectors for those endpoints; *Bézier*, which are defined by a pair of endpoints and two other points that control the tangent vectors at the endpoints; and several other *splines* defined by four control points, where the splines have either C^1 or C^2 continuity, but do not interpolate the control points. We describe three commonly used spline functions for approximating a curve: Catmull-Rom splines, Bézier curves, and B-splines. For all three, we use a parametric representation to mathematically define each spline such that the curve segment Q is given by three cubic polynomial functions x, y, and z, over the parameter t, with the curve segment equation being

$Q(t) = \begin{bmatrix} x(t) & y(t) & z(t) \end{bmatrix}$. Each of the polynomial functions have the form:

$$\begin{aligned} x(t) &= a_x t^3 + b_x t^2 + c_x t + d_x, \\ y(t) &= a_y t^3 + b_y t^2 + c_y t + d_y, \\ z(t) &= a_z t^3 + b_z t^2 + c_z t + d_z, \end{aligned} \tag{II.2}$$

and the parameter t is restricted to an interval from 0 to 1 for each curve segment. The coefficients of $Q(t)$ in equation II.2 can depend on four constraints, allowing for a compact way to express the polynomial functions. First, write the parameter t as $T = \begin{bmatrix} t^3 & t^2 & t^1 & 1 \end{bmatrix}$ and the coefficients of the three polynomial functions in a 4 x 4 matrix. We will define the matrix of coefficients for each type of spline below. We can then re-write the parametric curve $Q(t)$ as the product of:

$$Q(t) = \begin{bmatrix} x(t) & y(t) & z(t) & 1 \end{bmatrix} = T \cdot M \cdot G \tag{II.3}$$

where M is the 4 x 4 basis matrix of polynomial coefficients, defined for each specific type of spline, and $G^T = \begin{bmatrix} G_1 & G_2 & G_3 & G_4 \end{bmatrix}$, is traditionally called the *geometry matrix*. G specifies the geometric constraints that define the curve. It is a matrix of point vectors, with each row in G being either an end point, tangent vector, or other control point, depending on the type of curve being modelled. G is also a 4 x 4 matrix.

Catmull-Rom splines are cubic polynomials defined by four control points, all of which are interpolated by the curve. Tangents at interior control points are automatically generated, but the tangents at the endpoints must be specified. Some features of the Catmull-Rom spline are that the spline is C^1 continuous, so there are no discontinuities in the tangents, and it has local control (if a control point is moved, it only affects the curve locally). Each point interpolated by the spline will have a tangent direction parallel to the line between the two adjacent points. A Catmull-Rom spline has the basis matrix:

$$M_{CR} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \tag{II.4}$$

The geometric constraints in G are the endpoints and tangent vectors that define the curve.

Bézier curves are defined by four control points, two endpoints that are interpolated and two points that determine the tangent vectors at the endpoints but are not on the curve. Those four control points are the constraints represented in the geometry matrix $G_B^T = \begin{bmatrix} P_1 & P_2 & P_3 & P_4 \end{bmatrix}$. A Bézier curve has the basis matrix:

$$M_B = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{II.5}$$

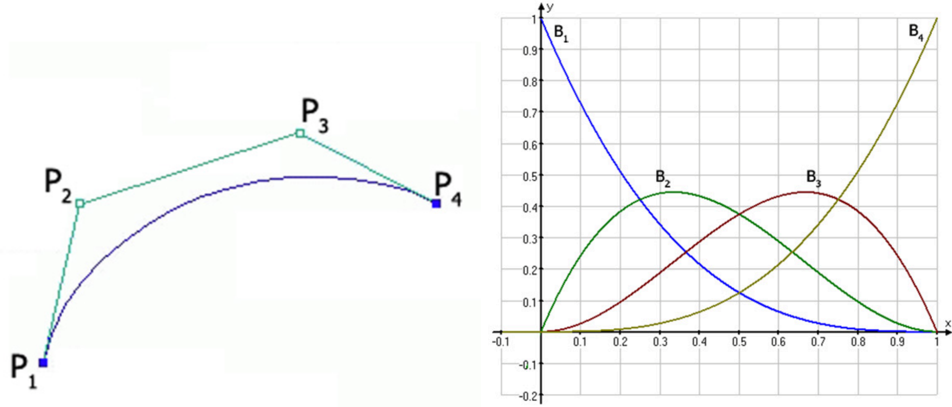


Figure II.5: On the left is an example of a Bézier curve with the endpoints indicated as P_1 and P_4, and the control points for the tangents at the endpoints indicated as P_2 and P_3. On the right are the *Bernstein* polynomials, which are the blending functions for Bézier curves. Notice that at $t = 0.0$ only the B_1 polynomial is nonzero, meaning that the curve interpolates only one point, P_1. The same is true for the B_4 polynomial at $t = 1.0$, which interpolates point P_4.

The product $Q(t) = T \cdot M_B \cdot G_B$ has the following form:

$$Q(t) = (1-t)^3 P_1 + 3t(1-t)^2 P_2 + 3t^2(1-t)P_3 + t^3 P_4, \tag{II.6}$$

which are known as the *Bernstein polynomials*. Figure II.5 shows an example of a Bézier curve and the *Bernstein* polynomials defining the blending functions. Some of the features of the Bézier curve are the convex hull property (all of the control points define a convex hull containing the resulting curve), that it is very easy to subdivide, is C^0 continuous, can be made C^1 continuous. Subdividing a curve into smaller segments may be required for some applications, in essence adding to the number of control points on the curve to provide more local control and deformability in smaller areas. Bézier curves are very easy to subdivide, where one Bézier segment with four control points becomes two Bézier segments with seven control points (sharing one control point at the seam), and the resulting curve will be identical in shape to the original segment until any of the control points are moved. The de Casteljau subdivision method is typically used [Foley et al. 1990].

B-splines are defined by four control points, but the curve in general does not interpolate any of the points. These splines are C^2 continuous everywhere. Cubic B-Splines approximate series of $m+1$ control points, but the curve consists of only $m-2$ segments. A B-spline has the basis matrix:

$$M_{B_S} = \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \tag{II.7}$$

The blending functions for this basis are shown in Figure II.6. There are several versions of B-

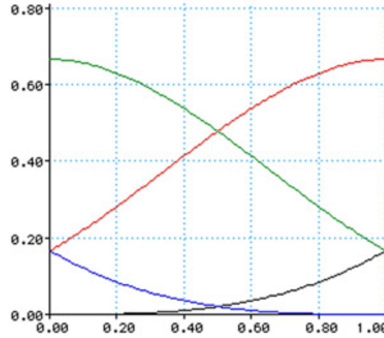


Figure II.6: The B-Spline blending functions. Notice that at $t = 0.0$ and $t = 1.0$, three of the polynomial functions are nonzero, unlike the Bézier blending functions, therefore no control points will be interpolated.

splines: uniform (equal spacing between knots) versus nonuniform; and rational (meaning each of the polynomial functions are defined as a ratio of two cubic polynomials) versus nonrational. The "B" refers to curve being represented by weighted sums of polynomial basis functions. Some of the features of the B-spline representation are that they satisfy the convex hull property and they are C^2 continuous. However, subdividing becomes more challenging.

One of the advantages of using splines to interpolate is that a spline can approximate the desirable arc for the path of the animation. However, compared to linear interpolation, splines are more difficult and expensive to implement. Splines can be used to represent the contours of a line drawing of a character, and used for interpolation given corresponding spline representations of a pair of keyframes. However, converting the artwork into a spline representation that has corresponding features is not a trivial problem. Defining a spline that captures subtle details and sharp edges or tufts of hair or fur can be challenging. Assuming one does choose a spline representation, once the line drawing is converted into a spline (or several splines to include interior features), the path of interpolation should also be defined to achieve desirable inbetweens. We chose to avoid having to define corresponding splines for line drawings, and instead work with image based representations of our images. One advantage is that we can analyze colored drawings and use this information for coloring any automatically generated inbetweens, which would not be possible with a strict spline representation of cartoon characters.

II.2.2 Image-Based

Interpolation can also be performed on images, and is referred to as blending, warping, or morphing. The simplest form of blending is called a cross-dissolve. Two images are blended by linearly interpolating between pixel colors in first image to pixel colors in second image over time. Figure II.7 illustrates a cross-dissolve between a pair of images. This method is simple and flexible but often appears unrealistic. For example, because the two ovals in Figure II.7 are identical but perpendicular to each other, the intermediate shape is no longer oval. Also, cross-dissolve is most

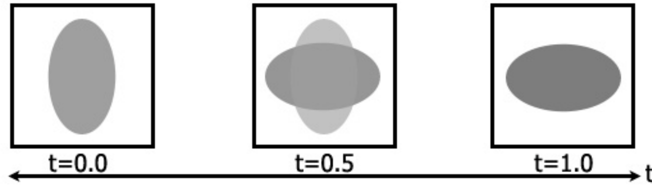


Figure II.7: Example of a cross-dissolve between two images. The left and right show the source and destination images. The middle is the blend at exactly half way between the source and destination.

often used for linking two shots in a film, so we are used to seeing the effect, but not for creating intermediate frames that resemble what we expect an inbetween to look like.

Image warping is a method for mapping one image onto a second image in an interesting way. Once a mapping is known, the source image can be transformed to match the destination image. Figure II.8 illustrates a warp between two simple ovals, with the intermediate shape being a circle. The mapping can be created as a forward map or an inverse map, which defines a geometric transformation that determines the relationship between pixels in the two images. With a forward map, each pixel in the source image is mapped to an appropriate location in the destination image. The forward transformation uses the centers of the pixels in the source image and maps them to locations in the destination image. However, the mapping may not fall at pixel centers in the destination image. To correctly distribute the value of a source pixel to one or several destination pixels, some method of filtering must be used, which essentially computes a weighted average to compute the destination pixel's intensity. For example, if the pixels are regarded as squares, and the fraction of the area of the source pixel that covers the destination pixel will be used as a weighting factor for filling the destination pixel. In this way, no holes will exist in the destination image. With an inverse map, every pixel in the destination is mapped to an appropriate location in the source image, leaving no holes in the destination image since the image is scanned pixel by pixel. Like the forward mapping, the inverse transformation uses the centers of the pixels in the destination image mapped to locations in the source image that may not be pixel centers. Again, some method of interpolation is used, but in this case pixels in the source image are interpolated. In either case, how the pixels should be mapped must be specified, which determines how the pixels move between the two images. For example, specifying a set of important pixels in the two images (as control points, lines, or curves), then extrapolating information about the control pixels to determine the motion of the rest of the pixels. Or, we can map the images onto a regular shape such as a plane or sphere. Then we know how to warp the images from source to destination by using the mapping from the source to the regular shape, followed by the inverse mapping from the regular shape to the destination.

Image morphing is a technique that combines warping with cross-dissolve [Wolberg 1990]. The images are first warped to each other or some regular shape using forward or inverse mapping. During the morph, the intermediate shapes created from the warp are cross-dissolved with each other produce the final intermediate shapes. Figure II.9 illustrates the concept of a morph.

Analyzing a series of images to determine the difference between the images caused by a char-

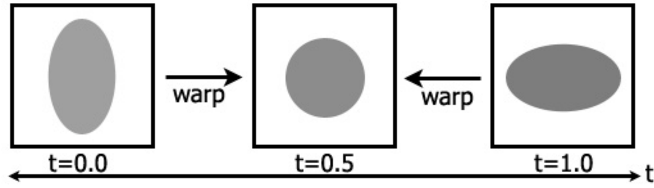


Figure II.8: Example of a warp between two images. The left and right show the source and destination images. The middle is the warped image produced by either warping the source to the destination or vice versa.

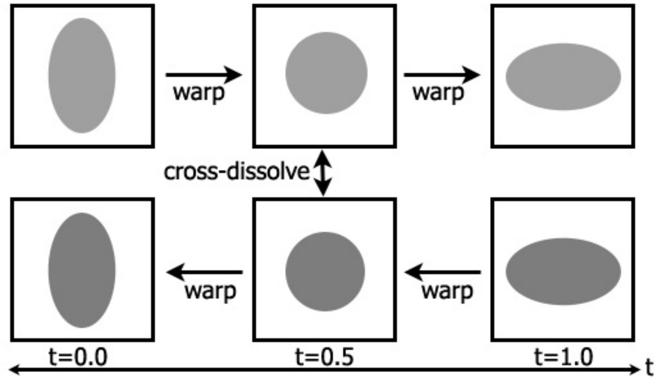


Figure II.9: Example of a morph between two images. The top left and bottom right show the source and destination images. The middle is the morph, which is the warped images that are cross-dissolved for the final result.

acter's or object's motion can be useful for producing an intermediate frame. Optical flow is a method of computing the velocity field from the motion of an object in an image sequence. The velocity of every pixel in the image is calculated, which indicates how quickly something crossed that pixel and which direction it moved. The velocity field can be used to warp one image onto another, assuming the two images are very similar. However, estimating optical flow is typically not a robust procedure, will fail with fast moving objects, and is sensitive to noise. Many assumptions are made when calculating optical flow, for example, for most pixels in an image, the neighboring pixels will have approximately the same brightness. Another problem is the assumption that only a single motion is present in the image sequence. There is extensive literature about optical flow and many extensions for improving robustness, which is beyond the scope of this dissertation. As an example of applying the optical flow to a pair of cartoon images, we use the algorithm of [Lucas and Kanade 1981] to compute the velocity field. The result is shown in Figure II.10. Even though the images are fairly similar, the flow vectors do not capture all of the motion exhibited by the character. In particular, the character's left hand is occluded in the first image, then extends out and is visible in the second image. Also, small details of the mouth opening are lost among the general flow vectors.

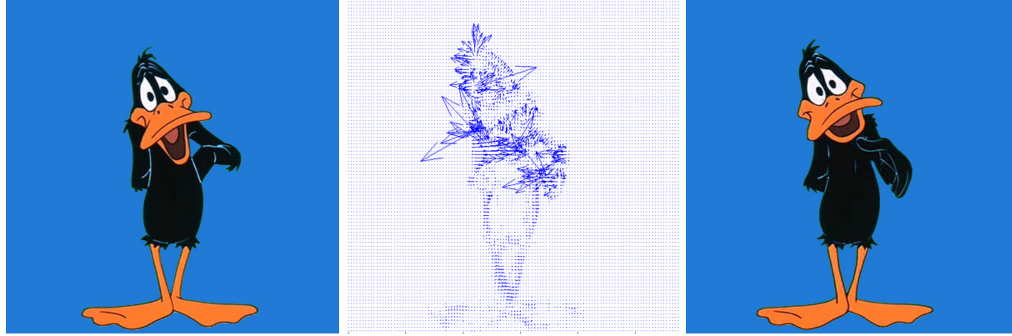


Figure II.10: An example of optical flow computed from a pair of cartoon images. *Daffy Duck* is ^{TM}& ©*Warner Bros. Entertainment Inc.*

II.2.3 Model-based

In addition to the methods already described for blending between shapes using contours (splines) and images, there are several methods for transforming shapes defined by a model representation. The simplest way to transform one object into another is when both objects have the same vertex-edge topology. The correspondence between the objects is established and shape transformation proceeds by interpolating the vertex positions using any interpolation method desired. Suppose we wish to represent a pair of cartoon images (either line art or full color) as polygonal models, then generate the inbetweens by transforming one polygonal representation into the second. Assuming two-dimensional models, the geometry could be defined as the triangulation of the contours (polygons) of each cartoon character. Triangulation is the division of a surface or plane polygon into a set of triangles, usually with the restriction that each triangle side is entirely shared by two adjacent triangles. One of the most common methods of triangulating a polygon is called Delaunay triangulation [O'Rourke 1994]. Delaunay triangulation is the dual of the Voronoi diagram. The Voronoi diagram is a subdivision of the plane containing a number of points into convex polygons such that each polygon contains exactly one point, and every point of a polygon is closer to its generating point than to any other on the Voronoi diagram. Figure II.11 illustrates the subdivision of a plane of points into the Voronoi diagram. If one draws a line between any two points whose Voronoi domains touch, a set of triangles is obtained, these triangles are the Delaunay triangulation. Generally, this triangulation is unique. Figure II.12 shows the Delaunay triangulation and its dual Voronoi diagram.

Once we have the two-dimensional polygonal models of a pair of cartoon characters created with a Delaunay triangulation, we would like to continue with the shape transformation. However, in general the models will not have the same topology. Ensuring the same topology for both polygonal models is the problem of generating compatible triangulations from a pair of polygons. One would begin with a pair of polygons that already have corresponding vertices, then a compatible triangulation can be created. Compatible triangulation for both two- and three-dimensional models is an active area of research in computational geometry [Etzion and Rappoport 1997], and there is extensive literature describing the many techniques, which is beyond the scope of this dissertation.

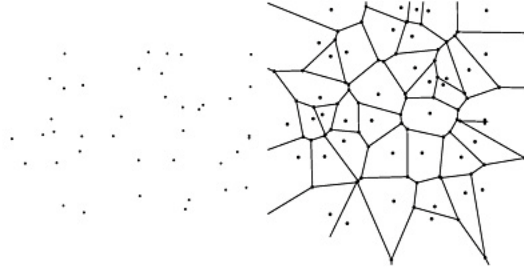


Figure II.11: On the left is the input plane with a number of points. On the right is the resulting Voronoi diagram.

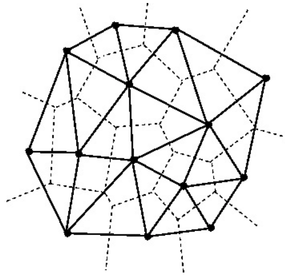


Figure II.12: An example of Delaunay triangulation. The thick lines are the Delaunay triangles, the dashed lines represent the Voronoi diagram.

Therefore, representing cartoon images as two-dimensional polygonal models for shape transformation requires a great deal of manual effort that we wish to avoid. In Chapter III.1.4, we show an application of the interpolation method described by [Alexa et al. 2000] using two-dimensional polygonal models of a pair of cartoon characters and the limitations of such a method.

Shape transformation can also be accomplished using implicit functions. Representing a model as an implicit surface eliminates the need for a compatible triangulation. Correspondence does not need to be specified to generate a transformation between a pair of implicit surfaces. Given two implicit surfaces specified by $f(x,y) = 0$ and $g(x,y) = 0$ that are defined with a common sign convention (i.e., positive on the inside, negative on the outside), then the shape transformation to define a blend of the two shapes is simply $f + g = 0$, which represents the intersection of the two surfaces. A good reference book for implicit surfaces is [Bloomenthal 1997]. As with representing a cartoon image with a polygonal model, defining the cartoon image as an implicit model still requires some effort. We can create an implicit function that represents a cartoon image by defining the function to be equal to 0 at the cartoon character's contour, then any point inside the character has a positive value while any point outside the character has a negative value. These definitions are illustrated in Figure II.13. Given a pair of cartoon keyframes represented as implicit models, shape transformation is then accomplished by interpolating between the two implicit functions. We will discuss further details regarding shape transformation using implicit functions further in Chapter III.1.4.

Figure II.13: An example of defining an implicit function to represent a cartoon character from an image. The blue points (densely sampled) are the zero set, the green points are interior points, the red are exterior points. To facilitate viewing these points, the cartoon character's silhouette is shown instead of the full color frame.

II.2.4 Model-based vs Model-free Representation

Using an underlying model to represent the characters in the images can provide a great deal of control over the interpolation between keyframes. One can create a three-dimensional computer model of a character based on the model sheet and character sheet, and deform the computer model to match a set of keyframes. Then generating any number of inbetweens from the computer models of the keyframes is straightforward. However, the task of creating these models requires a great deal of manual effort, whether they are two-dimensional polygonal representations or three-dimensional computer representations matching the images. Therefore, simply hand drawing the inbetweens would be faster. Also, there are issues in how to transform two-dimensional polygonal models in that they must have compatible triangulations and the paths that the vertices take during transformation must be constrained such that no triangles flip, causing a degenerate polygon.

The methods presented in this dissertation for re-using existing cartoon animation are model-free. As such, semi-automatic and automatic methods are used to analyze the images without a priori knowledge of the character. Once a data set for a particular character has been processed, the system can create new animations very quickly. A user who wishes to create a new animation by re-sequencing existing data needs only to select sets of keyframes (start and end poses) for an initial animation to be created by the system. However, by not including an underlying model, some limitations exist. For example, a model of the character may be useful in determining image similarity for designing a distance metric. We believe that the advantages of a model-free method, in particular only requiring a small amount of user input, outweigh the more time consuming methods of developing model-based representations. By requiring a small amount of user input, refinements can be made where the model-free representation reaches its limits.

CHAPTER III

RELATED WORK

In this chapter, we break the literature related to work in this dissertation into several topics and sub-topics: work on 2D animation, including spline-based, template-based, vision-based, and shape transformation methods for inbetweening; work on re-sequencing animation data represented as video and motion data; and work on dimension reduction.

III.1 Inbetweening for 2D Animation

Inbetweening is a studied but unsolved problem in two-dimensional animation. An inbetween is a figure drawn by a person or computer program based on two extreme poses of a character. Catmull [Catmull 1978] describes the main issues in dealing with the inbetweening problem, and discusses the principal difficulty being that the drawings are really two-dimensional projections of three-dimensional characters as visualized by a skilled artist. Because of this, a great deal of information is lost, and problems of self-occlusion and correspondence arise. To deal with self-occlusion, [Catmull 1978] suggests breaking the character into separate layers before processing with a computer program. For the correspondence problem, he suggests the program operator specify the correspondence of the lines and hidden lines. Even with human intervention, the problems of occlusion and correspondence are still difficult to overcome.

III.1.1 Spline-Based

Reeves [Reeves 1981] presents a method for creating inbetweens by using moving-point constraints. A moving-point is a curve in space and time that provides a constraint on the path and speed of a specific point on the keyframe for a character. The moving points paradigm is at the center of this method for inbetweening. Other points on keyframes that are not directly constrained by a moving point are constrained by a "smooth blending" of their neighboring moving points. Correspondence is established "automatically," in that the intersection of two curves in different keyframes with the same moving point determines the correspondence. Multiple moving points can control the interpolation, therefore it is not linear interpolation on a path. The user must define a pair of keyframes for the start and end poses, manually select the moving points, and those moving points must be defined to constrain the motion of the ends of all the curves in the keyframes. A patch network is defined as a data structure of sets of keyframes and moving points. Each completed patch network is subdivided. To generate an inbetween, the patches are evaluated at an intermediate time step using one of three interpolation methods described by the author. Miura interpolation (defined by the author) is similar to linear interpolation, but causes a lot of contortions and discontinuities along patch boundaries. Coons patch interpolation (defined by the author) controls the normal derivatives at patch boundaries and uses two blending functions. Contortions interior to the patch rarely occur, and can be remedied with adding constraints. Finally, the cubic metric space interpolation method,

as described by the author, is a method of interpolation defined by cubic equations. Contortions occur when there are regions of high curvature in its cubic basis curves. While this method provides control in creating a new animated sequence by generating the inbetweens "automatically," a great deal of manual effort is involved.

Di Fiore et al. [Di Fiore et al. 2001] present a multi-level method for inbetweening computer-assisted 2D animation by including 3D information as a high-level deformation tool, and 2.5D information as modeling structures. They address the inbetweening issues of self-occlusion and silhouette changes. The multi-level method is "2.5D" for modeling and animating, with four levels described: (1) the basic building primitives such as sets of 2D subdivision curves in level 0, (2) all 2.5D modeling information is in level 1, (3) 3D skeletons are used to convey 3D information in level 2, and (4) high-level deformation tools are in level 3. The key to this method is in the 2.5D modeling level, where the animator defines sets of depth ordered primitives (subdivision curves) with respect to both x-axis and y-axis rotations of the character at extreme poses or camera angles, while drawing order for each of the primitives is also stored. Correspondence information stored in level 0 is defined manually, and those curves are interpolated linearly. The notion of 3D skeletons incorporated in level 2 simply define a local coordinate system that provides a region of the character (represented as sets of control points on 2D curves) that will be influenced by a particular bone and any transformation it may cause. The high-level deformation tools of level 3 are applied to the control points of the 2D curves or the position of the 3D skeleton. While this method addresses the difficulties of 2D inbetweening, the user or animator still has to manipulate non-intuitive control points for the underlying spline representation of the curves, and specify the correspondence information manually. The skeleton information included in level 2 also requires specification of the region of influence over the underlying curves, and is only a marginal improvement over the 2.5D information for the extreme poses from level 1, which the animator must specify as well.

[Kort 2002] introduced a method for integrating vector-based inbetweening into an animation system that requires the user to draw the keyframes and identify the layers of each key image. The layers are included to overcome the occlusion problem in 2D animation. Each drawing is analyzed and classified into components called strokes, chains of strokes and relations between them. A set of rules is used to match parts of different drawings and specify allowable changes between the relations associated with each drawing. Their method assumes: (1) a vectorized stroke is defined as the path specified by the movement of the input pen, (2) stroke chains are structures consisting of one or more strokes, (3) a stroke chain may or may not have a corresponding stroke chain in another key drawing, (4) animation paths that specify the correspondences between stroke chains and the interpolation between them. Each vectorized stroke is represented as a Bézier curve. The inbetweens are generated by interpolating the strokes using a Coons patch transformation. Although the results are promising, the method still requires the animator to draw the full keyframes directly into their system. The system is also restricted in the type of animation that can be inbetweened in this fashion, such as those animations in which the layering order of the cels is invariant.

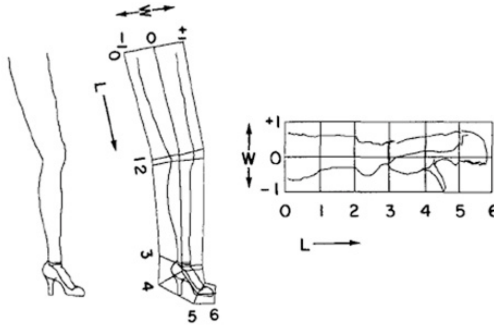


Figure III.1: A hand drawn image with the reference skeleton and relative coordinate system. Image from [Burtnyk and Wein, 1976].

III.1.2 Template-Based

Burtnyk and Wein [Burtnyk and Wein 1976] describe an inbetweening method for keyframe animation in which the animator traces key drawings into a computer program and uses an underlying skeleton to drive the motion for the interpolation phase. By drawing the keyframes into the computer, the system keeps track of the order of the strokes for stroke to stroke matching. The order of the strokes in each image determines how the interpolation will generate an inbetween image, also keeping track of separate layers for later determination of hidden lines. The skeleton used to represent the image data defines a coordinate space within each image. Each skeleton defines an axis and a boundary of influence over a region of the image, which is essentially a mesh. Figure III.1 shows an example of a drawn image, the reference skeleton for the image, and the coordinate system derived. Each region of influence from the skeleton will cause a distortion in the image that is used for interpolation, and any image region that is not covered by a skeleton will remain unaffected by the distortion. The interpolation method used is linear interpolation of the skeletons. To smooth any discontinuities that result from linear interpolation, a parametric method of curve fitting is used, producing a smooth path for each interpolated point. One limitation of this system is that defining a skeleton to control the interpolation of every part of an animation sequence would be extremely time consuming. The authors note that they envision their system being used selectively over previously created sequences of animation to improve the motion dynamics.

III.1.3 Vision-Based

Seah et al. [Seah and Lu 2001] discuss using a modified hierarchical feature-based matching method for motion estimation to generate inbetween line drawings from a pair of input line drawings. The authors modify the hierarchical feature-based matching algorithm from [Weng et al. 1993]. Some of the features are intensity, edgeness (magnitude of gradient), cornerness (instantaneous rate of change in the gradient direction along an edge), displacement orientation, and magnitude smoothness. These features are used to compute motion vectors between a pair of images, and is structured as an optimization problem solved by using the least squared error. The method is hierarchical to overcome the initial value problem, so the feature-based matching is applied to an image pyramid

for each image pair from a low resolution to a high resolution, and passes each set of matched features to the next resolution level. The authors modified the algorithm of [Weng et al. 1993] to normalize all of the features and apply different weights to each feature. Their modified feature-based matching is applied to the pair of keyframes; however, the inbetween image generated is just a linear interpolation of the matched features, and some artifacts are introduced during interpolation. Also, this method will fail in the case of silhouette changes when new features appear and have no corresponding features to match to, while other matched features will disappear.

Wang et al. [Wang et al. 2004] developed a technique for applying non-photorealistic effects to video, giving the video a cartoon-like look. Their method applies a modified mean-shift segmentation to the video to construct volumes of contiguous pixels with similar color. The mean-shift segmentation uses an anisotropic kernel to account for spatio-temporal information in the video. Once the segmentation volume is generated, the user outlines semantic regions in certain keyframes in the video, indicating which low-level segments should be merged, which is propagated to all frames thereby maintaining interframe correspondence. The video is then represented as three-dimensional polyhedral regions. Two-dimensional style effects are applied along the surface of regions and within regions by taking slices of the volume along the time axis. The slices yield solid areas and curves along edges that may be rendered according to the desired style effects. Their method of building a three-dimensional volume from video may be applied to the inbetweening problem, and might work for contour data. However, there is a fair amount of user input required to define the semantic regions and the method is fairly complex to be used on a single pair of keyframes that require an inbetween.

III.1.4 Shape Interpolation

Beier and Neely [Beier and Neely 1992] discuss how to create a morph of a pair of images that is a combination of a cross dissolve and an image warp. A warp is calculated from the source to the destination image, and likewise for the destination to the source. The warping method used is a reverse mapping, where the destination image is scanned pixel by pixel and the correct pixel from the source image is sampled. Their morphing method is based on fields of influence surrounding two-dimensional control primitives. To deform the images, sets of features (directed line segments) are defined for both source and destination images. A coordinate system is built based on the line segments and a distortion field. Figure III.2 shows a source and destination image with sets of features and the resulting morph. Once the warping function is defined for each image, the morph proceeds as a cross dissolve with the start image being the unwarped source image, the image half way between is a combination of the source image half way warped and the destination image half way warped, and the final image is the destination image unwarped. Any transformation based on a single pair of lines is affine. Any transformation with multiple pairs of lines has weights assigned to each line, and is usually non-affine. While this method produces nice intermediate images during the morph, it suffers from two disadvantages, speed and control. Because the features are globally defined the line segments need to be referenced for every pixel. Also, some artifacts can appear as

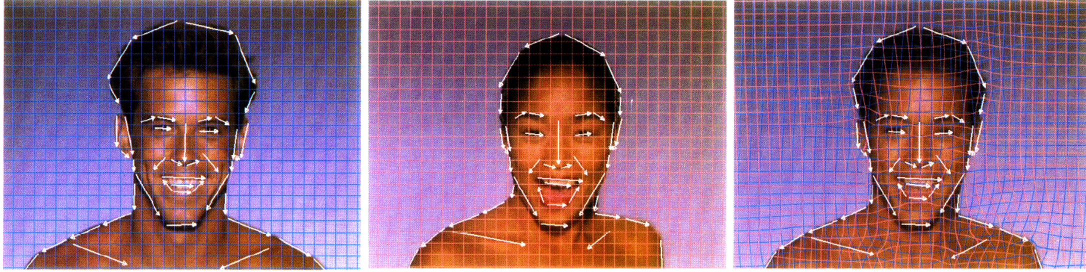


Figure III.2: The left and right images are the source and destination images with sets of features indicated as directed line segments. The center image the resulting morph, with warped grid lines from the source and destination. Images are from [Beier and Neely, 1992].

unexpected interpolations of pixels that are far away from the line segments.

Sederberg and Greenwood [Sederberg and Greenwood 1992] studied how to smoothly blend between a pair of two-dimensional polygonal shapes. By modeling the input shapes as being composed of thin wires, the shape transformation is achieved by minimizing equations of work for deforming the wire from one shape to another. The user can specify physical attributes of the wire, thereby controlling the ease or difficulty of bending or stretching it to conform from one shape to another. To prevent the shape from self-intersecting, a penalty is added for any deformation that crosses a zero degree angle, which implies the shape has intersected itself. The deformation work equations are separated into stretching work and bending work, the first is applied to adjacent vertex pairs, the second is applied to sets of three adjacent vertices. They address the problem of vertex correspondences by specifying a small number of initial corresponding point pairs on the input shapes, and can add vertices based on some heuristics. The polygonalized shapes do not have to have the same number of vertices, but every vertex in each shape must have a correspondence to the other shape. While their results show nice shape blending, the shapes must be polygonal, therefore using existing animations would require polygonalizing every image. Their results also depend on the initial manual placement of the corresponding vertex pairs. All of the transformations, while constrained by the work equations for bending wires, still suffer undesirable effects, such as an arm shortening, because the corresponding vertices are linearly interpolated.

Turk and O'Brien [Turk and O'Brien 1999] present a method for shape transformation using implicit functions for both representing and interpolating the shapes. Their method relies on scattered data interpolation, produces smooth intermediate shapes, and will work for shape transformation in any number of dimensions. Each shape is defined implicitly in a higher dimension, thereby combining the representation and interpolation of the shapes into one step. Figure III.3 shows a visualization of a transformation between a pair of two-dimensional shapes. Using implicit functions to interpolate between a pair of shapes is similar to the technique we discuss using radial basis functions (RBFs) for interpolation. However, their method requires specifying interior and exterior points for the shapes being transformed, and assume a specific gray-scale image from which to de-

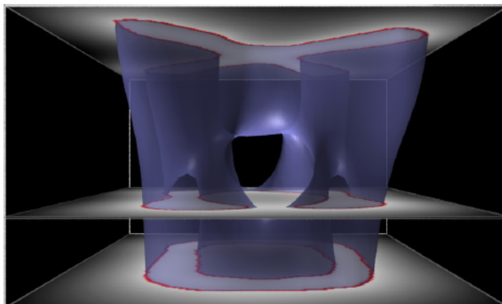


Figure III.3: A visualization of two-dimensional shape transformation between an X shape and an O shape. The translucent surface indicates the isosurface of the three-dimensional variational implicit function created from the two-dimensional shapes represented by the top and bottom planes. Image from [Turk and O'Brien, 1999].

fine the interior, exterior, and boundary points. White regions represent the interior of the shape, black regions represent the exterior of the shape, and intermediate gray values define the boundary of the shape. This gray-scale image representation allows for creating a smooth shape from the image, while defining the boundary and normal constraints by looking at the pixel neighbors and the gradient, respectively. Figure III.4 shows an example of the implicit function created from a gray-scale image with boundary and normal constraints. While the variational implicit shape transformation method would work with cartoon images, the user would have to prepare each image to work with this representation by either manually specifying the interior, exterior, and boundary points with normal constraints, or by modifying each image to be in the same gray-scale image as described. We experimented with the variational implicit shape transformation on a pair of two-dimensional cartoon contours. Figure III.5 shows the result. Interior and exterior points were specified manually for each contour (48 interior and exterior for the first contour, 44 interior and 53 exterior for the second contour), and all points along the contour were used. No correspondence information was specified. While the result is able to generate a surface between the two cartoon contours, the surface between the contours loses a great deal of small detail, smoothes out some of the larger details around the feet, and extrapolates out beyond both contours. As we will see in Chapter VI.4.2, our method of using radial basis functions (RBFs) for generating an implicit surface between a pair of contours performs better, smoothing still occurs on fine details, but larger details around the head and feet are preserved.

Alexa et al. [Alexa et al. 2000] describe a method for generating nonlinear shape transformations from a pair of two- or three-dimensional input shapes, while maintaining internal textures and features during the morph. Unlike [Beier and Neely 1992], this transformation is performed in object-space, that is, the representation of the two- or three-dimensional shapes are polygons or polyhedra respectively. In dealing with objects rather than images, the morphing process requires generating a correspondence between the geometric features (vertex correspondence) and interpolating the boundaries of the shapes (vertex path). The authors present a morphing technique that blends the interior of the shapes rather than their boundaries to achieve intermediate shapes that are

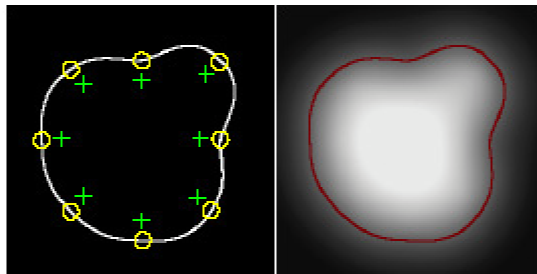


Figure III.4: An example of the boundary and normal constraints indicated by circles and plus marks on the left, an intensity image showing the implicit function on the right. Image from [Turk and O'Brien, 1999].

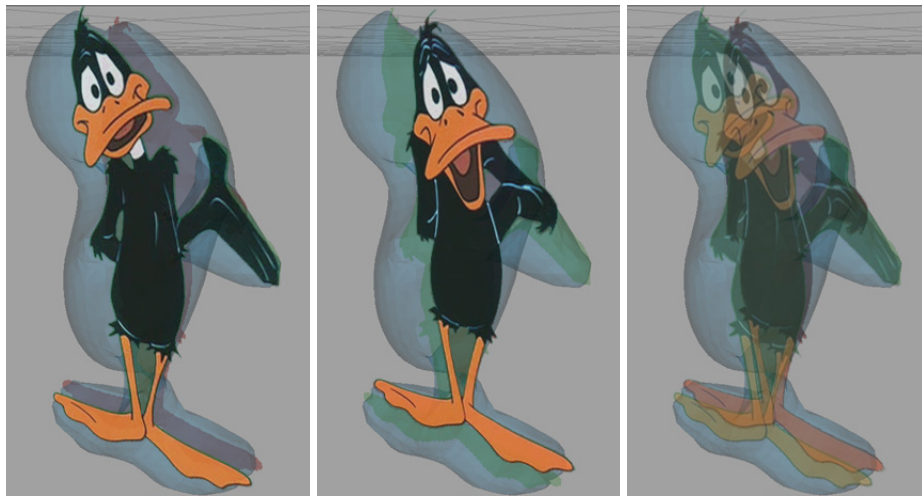
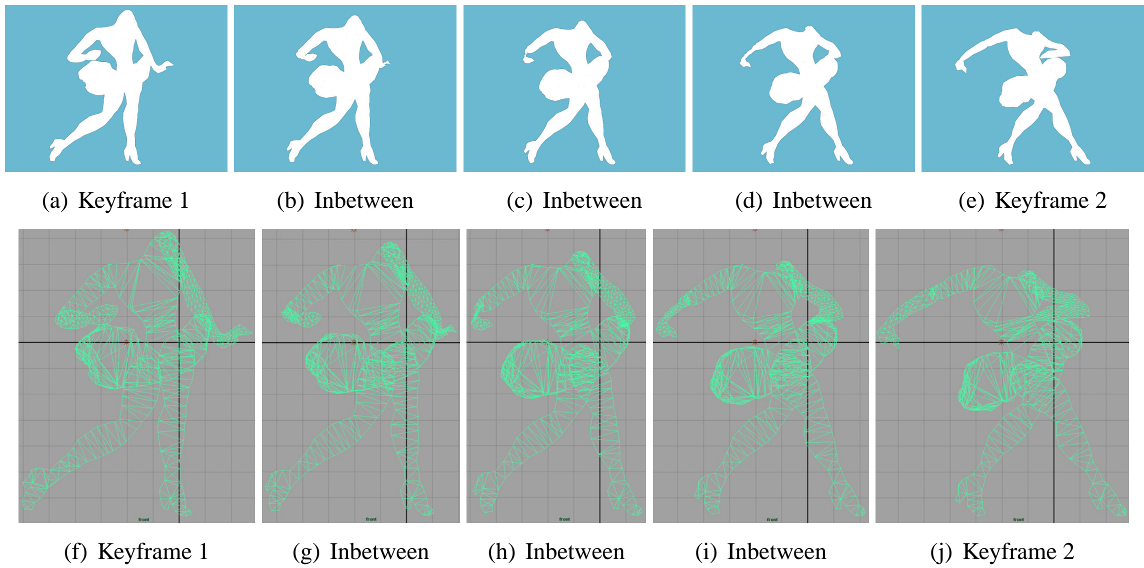


Figure III.5: A visualization of two-dimensional shape transformation between a pair of cartoon contours. On the left and middle are the two cartoon input shapes shown in perspective and overlayed on the mesh. On the right, the two input cartoon shapes are overlayed together, and the translucent surface indicates the isosurface of the three-dimensional variational implicit function created from the two-dimensional shapes. Notice on the isosurface that details around the head and feet are smoothed and lost because the isosurface balloons out too far from the input shapes. *Daffy* is ^{TM}& ©*Warner Bros. Entertainment Inc.*

locally least-distorting. Their method assumes that correspondence has already been determined for the boundary vertices before applying their algorithm for generating a compatible triangulation into triangles or tetrahedra (in two- and three-dimensions respectively). For the two-dimensional case, the compatible triangulations are generated with a Delaunay triangulation, then vertex positions are optimized based on maximizing the minimum interior angle to help eliminate long skinny triangles. This mesh smoothing step must be done on both triangulations to ensure that they remain compatible. The general overview of their morphing algorithm is to first determine an optimal least-distorting transformation between source and target, which is locally as similar as possible between each pair of corresponding triangulations. An affine transformation is determined for every pair of source-target triangles in the triangulation. The transformation is decomposed into a rotational component (rigid) and a scale-skew component (non-rigid), both of which are interpolated separately. Each transformation is decomposed using singular value decomposition (SVD) to separate into a rotation matrix and a scale-shear matrix. The rotation is represented as a quaternion and interpolated using spherical-linear interpolation. The scale-shear component is interpolated linearly. Now an optimal transformation is known for each pair of source-target triangles, however, these cannot simply be applied as the shape will come apart because shared vertices will have different optimal paths to follow. The paths the vertices take are expressed as a vertex configuration that minimizes the quadratic error between the actual transformation matrices and the desired transformation matrices. Their method produces nice shape interpolation for a single pair of input shapes, and preserves interior details very well for all of their examples. However, this method has several limitations. Corresponding vertices on the pair of input polygons must be manually specified. We implemented this method and found that it cannot be easily applied to the two-dimensional inbetweening problem. It cannot handle occlusions or out-of-plane rotations, which are the two main problems for two-dimensional inbetweening. Generating good compatible triangulations is difficult, and while Delaunay triangulation tends to create long, skinny triangles, the optimization step does not always remove all defective triangles, resulting in numerical problems. Most importantly, there is no constraint in the method that prevents triangles from flipping during shape transformation. The authors state "In all our examples no simplex changed orientation (i.e. flipped), however, we have not been able to prove this to be a property of our approach." Figure III.6 shows the result of applying [Alexa et al. 2000] to a pair of cartoon keyframes. The character was manually segmented into layers to overcome the self-occlusion problem, and each pair of shapes had manually specified boundary vertex correspondence. Compatible triangulations were generated, though as can be seen in the figure, some could not be optimized to remove the defective triangles. Even with a good triangulation of the right arm, the shape transformation produced undesirable results for the inbetween shapes. Examining the arm closer reveals that the triangles changed orientation during the transformation. Figure III.7 shows a close up view of only three triangles extracted from the arm, and the paths that the vertices followed during the transformation. Yet, other parts of the character produced very reasonable inbetween shapes. Due to the unpredictable nature of triangles flipping during shape transformation, we abandoned this method for generating inbetweens.



(a) Keyframe 1 (b) Inbetween (c) Inbetween (d) Inbetween (e) Keyframe 2

(f) Keyframe 1 (g) Inbetween (h) Inbetween (i) Inbetween (j) Keyframe 2

Figure III.6: We tested the method of [Alexa et al., 2000] on a pair of cartoon keyframes. The character was segmented into layers, then the shape transformation was applied to each layer separately. The top row shows the result of the transformation. The bottom row is the same transformation with the triangulation visible at each step.

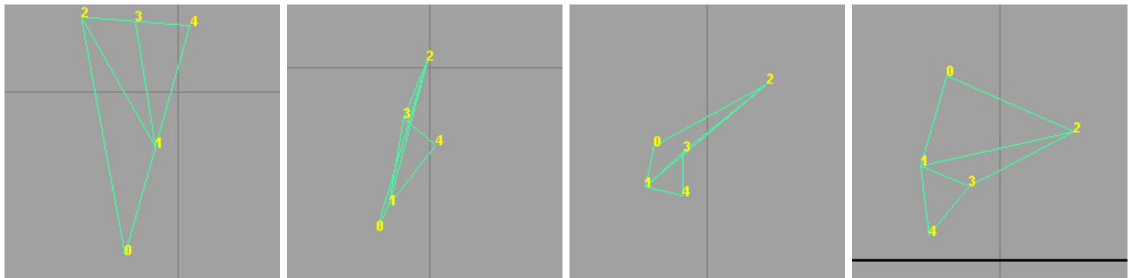


Figure III.7: A close up view of three triangles from the right arm of the character in Figure III.6. Clearly, the vertex paths indicate that the triangles flipped during shape transformation.

III.1.5 Other Techniques for 2D Animation

Fekete et al. [Fekete et al. 1995] present a complete 2D animation system for vector-based sketching and painting. Their software is designed to support a paperless 2D animation pipeline. In doing so, several technical issues are addressed. To draw the keyframes directly into the system, strokes are captured and vectorized from a pen-tablet input device. The vector-based strokes are internally represented as Bézier curves, which are created by a quick curve fitting algorithm to the drawn brush stroke. Painting the keyframes requires gap filling (making sure the strokes are closed) and a planar map applied to the planar topology that is defined by a set of Bézier paths. Rendering breaks down the strokes into shaded polygons and scan-converts them. While their method presents an approach to the entire animation process as vector-based sketching, they also discuss the advantages and disadvantages of automatic inbetweening. The advantages include a reduction of the number of hand-drawn inbetweens and the possibility for procedural rendering or texture mapping. One disadvantage mentioned is that automation changes the nature of inbetweening and limits its complexity. Both template based systems [Burtnyk and Wein 1976] and explicit correspondence systems [Sederberg and Greenwood 1992] would be restricted to fairly standard drawings. One of the features described as future work when this paper was published was incorporating automatic inbetweening of simple objects such as a bouncing ball or falling snow. Their current software system (by Toon Boom Technologies) can create simple inbetweens from the vector animation. However, this technique requires the basic animation from which to produce the inbetweens, and an artist is still required to draw the keyframes.

Corrêa et al. [Corrêa et al. 1998] developed a method for applying complex textures to hand-drawn animation. For every shot in the animation, the camera parameters are known and fixed per shot, their system uses a silhouette detection scheme and a warping algorithm to modify a three-dimensional model of the character to match to the hand-drawn line art. From the warped model, the texture is rendered and composited with the line art for the final result. The user must create a three-dimensional model that approximates the shape of the hand-drawn character, and help guide the correspondence of edges and curves in the line art that correspond to those features in the three-dimensional model. Each curve is represented as a uniform cubic B-spline. To calculate the control points of the spline, an overdetermined linear system is solved using least squares data fitting to minimize the root mean squared error. The system is overdetermined because the curves to be fitted are hundreds of pixels in the curve of the line art. The warping algorithm uses a forward mapping, in contrast to the inverse mapping of [Beier and Neely 1992]. Markers are specified on both the model and drawing, and used to define a coordinate system for both model space and drawing space. Each marker has a slightly different coordinate system, so a weighted average is used for each marker pair. Two parameters control the smoothness and precision of the warp, and how much influence a certain marker contributes to its neighbors. The authors discuss several ways of tweaking the results by adding more markers, adjusting the ordinate direction of the coordinate system for the warp, and reparameterizing the texture, allowing even more low level control of the process. While their results achieve the goal of applying a complex texture to a hand-drawn

character, the entire process is still quite labor intensive. Also, there are several types of line art for which their system will not work. The authors describe four classes of line art that their system would fail to texture: (1) a character with tufts of fur that are suggested by sharp wiggles, (2) characters that would be difficult to approximate with a three-dimensional model, such as clothing, (3) characters that have no reasonable three-dimensional representation, and (4)complex shapes that could not be represented with a single B-spline patch. Because of the labor intensive process of this system and its limitations, this method would not be well suited for transferring texture information from a keyframe to an inbetween generated by our proposed system.

Petrovic et al. [Petrovic et al. 2000] inflate a 3D figure based on hand-drawn art to produce shadows for cel animation. Their method is not fully automatic, the user must specify camera position, ground plane, background objects, depth for character, and lights. To create shadows, they use three types of shadow mattes: (1) tone mattes, these are self shadowing, cause other shadows on the character; (2) contact shadow mattes, these are shadows cast by the character onto the ground, and in contact with the character at all times; (3) cast shadow mattes, shadows that the character casts onto the background. Constructing the background requires the assumption of a fixed field of view and aspect ratio for the camera and an upright camera roll and ground plane pitch. Next, the user defines several parameters: (1) the pitch of the camera, obtained by the user drawing two parallel lines in the ground plane; (2) a coordinate system, defined such that the origin is the center of the image plane; (3) a ground intersection line, specified by the user so that walls must be perpendicular to the ground. More complex objects like stairs require more user input, such as polylines on the object starting with the contact points with the ground plane. The inflation method is a three step process. First, the line art is converted into character mattes. These are divided into multiple layers to give depth information. The inflation method is based on “teddy” by finding the chordal axis of a closed curve, lifting it out of the plane, and lofting a surface between the curve and axis. Next, the shape is adjusted for a perspective camera. Although this method produces nice results for casting shadows in cel animation, this technique can be considered a post-process or special effect in the traditional animation pipeline, and does not help in the actual creation of the animation itself.

Bregler et al. [Bregler et al. 2002] proposed a method for re-using cartoon motion data by capturing the motion of one character and retargeting it onto a new cartoon character. Their system can be broken down into two steps: capturing and retargeting. In the capture phase, a set of key-shapes are identified in the source cartoon. Each key-shape is parameterized to capture the motion. An affine transformation describes the coarse motion (the general translation, rotation, and scale) of the character. To capture the nonlinear motion, such as the extreme distortions cartoon characters typically undergo, key-shape deformations are defined by selecting the set of key-shapes that include all possible extreme deformations. To reduce the number of key-shapes the user must select, the cartoon shape space defined by the key-shapes is extended by linearly interpolating the key-shapes. Since linear interpolation cannot accurately produce good inbetween shapes, the linear key-shape set is first preprocessed and extended using [Alexa et al. 2000], then PCA is applied to the extended shape space that has been biased by seeding with the inbetweened key-shapes. Biasing the extended

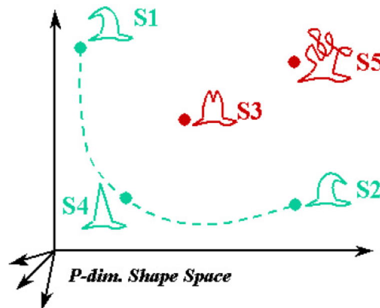


Figure III.8: An example of the shape space with the additional inbetweened key-shapes that would fall along the green line, thereby biasing the extended shape space to move away from invalid shapes shown in red. Image from [Bregler et al., 2002].

shape space before applying PCA helps to constrain the number of invalid shapes that occur with just extending the shape space by linear interpolation. Figure III.8 shows an example of the shape space with the additional inbetweened key-shapes that would fall along the green line, thereby biasing the extended shape space to move away from invalid shapes shown in red. With the key-shapes selected and the cartoon shape space defined, a warp from one key-shape to another is calculated. The retargeting step requires the user to define a set of corresponding key-shapes on the target cartoon character. The warping functions calculated from the source key-shapes are applied to the target key-shapes to create the animation for a new character from the motion from the source character. Figure III.9 shows an overview of the capture and retargeting steps described. While this approach produces interesting results for applying the motion of one character onto another character, this approach does not generate a new cartoon motion. Their system requires a great deal of expert user intervention to train the system and a talented artist to draw all the key-shapes. Each of the key-shapes must be manually specified for the source and target character, and parameterized by hand to find the affine deformations that the source key-shapes undergo before applying them to the target key-shapes. Their work provides a method for re-using the overall motion of the cartoon data, but it does not look at the structure of the data itself and therefore cannot re-sequence the data to expose meaningful new behaviors.

III.2 Re-sequencing Animation Data

III.2.1 Video

We are motivated by the work of Schödl et al. [Schödl et al. 2000] on video textures to retain the original images in motion sequences but play them back in non-repetitive streams of arbitrary length. A video texture is derived from video by changing the order in which the original frames of the video are played. The video frames are played out of the original order only at specific places that are most unnoticeable to the viewer. This re-sequencing then produces a smoothly playing infinite video from the finite duration input clip. Video textures is most similar to our goal of re-sequencing cartoon images. To determine when to transition from one frame to another, the L_2 distance is

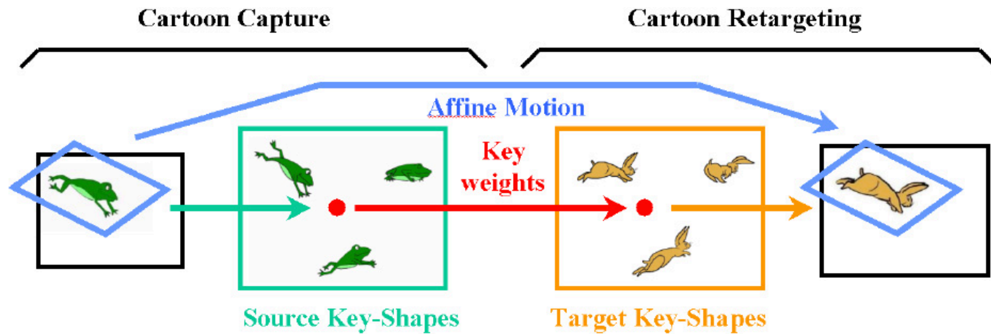


Figure III.9: A general overview of the capture and retargeting process. The source and target key-shapes are selected by the user, the key weights are determined by finding the warping function required to move from one key-shape to another in the source shape space. These are applied to the target key-shapes in the retargeting step. Image from [Bregler et al., 2002].

used to compute the differences between frames for building the video structure. These distances are used to produce a matrix of probabilities, where each entry in the matrix is the probability of transitioning from one frame to the next. If the L_2 distance between a pair of images is small, then the probability of transitioning between those images is high. The probability matrix is further refined to take into account the dynamics of the motion. A filter is used over the distance matrix by using a weighted diagonal matrix to take into account the similarity of temporally adjacent frames. In other words, the similarity computation can be said to match subsequences of frames instead of individual frames. Figure III.10 shows examples of the distance and probability matrices before and after filtering. To further improve the transition points to make them more unnoticeable, morphing and blending techniques are used. We want to compare the differences between frames in a similar fashion to analyze the traditional animation data for re-sequencing. [Schödl et al. 2000] assume a large data set with incremental changes between frames. Their methods do not extend well to cartoon data, which is inherently sparse and contains exaggerated deformations between temporally adjacent frames.

In their follow-up work [Schödl and Essa 2002], they create new character animations with user-directed video sprites. The authors describe video sprites as "animations created by rearranging recorded video frames of a moving object." As has been described, there are a number of ways of generating motion for animation. Video textures rearranges recorded video frames, but to direct the video sprites, the authors developed a cost function to define the desired motion of the sprites. The sprites are captured from video footage of the desired character on a constant background. The video is processed to extract the sprite from the background, and correct for perspective effects of the character moving closer and farther from the camera. Unlike the L_2 distance that was used to compute the similarity of images in [Schödl et al. 2000], transition costs are defined by training a linear classifier, learning from 1000 pairs of good or bad transitions that are manually classified. The cost function used sums the costs of all constraints and time steps, examples of some of the constraints are location, path, and anti-collision. The cost function is optimized using a hill-climbing

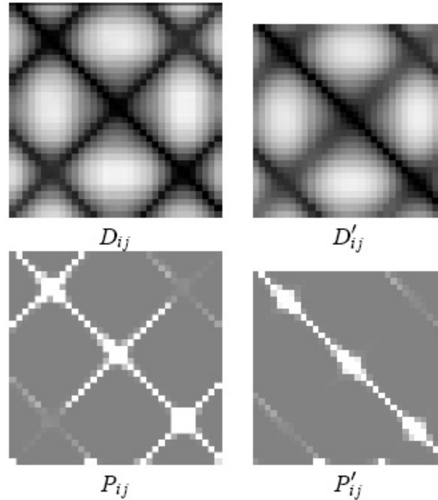


Figure III.10: The distance and transition probability matrices before (left) and after (right) filtering. These matrices are generated from analyzing video of a clock pendulum. Notice the probability matrix after filtering matches only forward swings of the pendulum. Image from [Schödl et al., 2000].

method over the entire sequence as a whole, and determines transitions in no particular order. The results are generated by specifying constraints for the desired animation, for example, to make a character walk around in a circle, a path constraint is used. However, not all motions can be animated, and there is no guarantee that there will be useful motions and good transitions captured. The examples shown require a vast amount of video data: 30 minutes of video footage for a hamster yielding 15,000 sprite frames (30,000 after mirroring). In our work, the largest cartoon data set we use has 2,000 frames, yet we still achieve good results with sparser data of 560 frames.

III.2.2 Motion

Recently, other researchers have found inspiration from video textures and have applied it to motion capture data. Sidenbladh et al. [Sidenbladh et al. 2002] employ a probabilistic search method to find the next pose in a motion stream and obtain it from a motion database. Arikan and Forsyth [Arikan and Forsyth 2002] construct a hierarchy of graphs connecting a motion database and use randomized search to extract motion satisfying specified constraints. Lee et al. [Lee et al. 2002] model motion as a first-order Markov process and also construct a graph of motion. They demonstrate three interfaces for controlling the traversal of their graph. Kovar et al. [Kovar et al. 2002] use a similar idea to construct a directed graph of motion that can be traversed to generate different styles of motion. In our work, once the structure of the data is learned, the manifold that represents the data can be traversed to re-sequence the data.

With the recent research efforts in re-using video data [Schödl et al. 2000; Schödl and Essa 2002] and motion data [Sidenbladh et al. 2002; Arikan and Forsyth 2002; Lee et al. 2002; Kovar et al. 2002; Kovar and Gleicher 2004], research questions arise about how to re-use the data in novel

ways. Given a corpus of video or motion data, how can the data be re-used to create new motions not previously seen in the database? The goal of our work is to re-use existing traditional animation to create new motions, representing the data in a lower-dimensional graph structure (manifold) and re-sequencing the images by traversing the graph from a given start and end pose. Using a graph structure to represent the motion data is not a new idea. [Schödl et al. 2000] also use image-based motion for creating infinitely long smoothly playing video from short video clips. [Kovar et al. 2002] use motion capture data for synthesizing new motion paths.

Looking specifically at Motion Graphs, [Kovar et al. 2002] present a method for creating directable motion given a database of motion capture data. Motion Graphs is a framework for generating different styles of locomotion along arbitrary paths through the motion capture database. The system automatically constructs a directed graph (the motion graph) of connections in the database of motion capture data. The idea is to automatically add the transitions between clips of motion within the motion database. The transitions are segments of two motion clips that are blended (interpolated). The completed motion graph then has original motion capture data as well as the synthesized transitions. Each edge in the graph represents the motion clip, and consists of the position of the root joint and quaternions representing the orientation of each of the joints in the character. Nodes in the graph represent the transition points. A "walk" of the graph is the new synthesized motion. By representing clips of motion capture data as a motion graph a user can direct a new motion by defining several functions specifying the goal of the motion, allowing the system to search through the motion graph to build a "walk" (i.e. traverse the graph). The user can also specify a path that the character should follow, and using an error function and halting criteria, the system can find the pieces of motion that best represents the path supplied by the user.

To build a motion graph, each clip of motion capture data is annotated with constraint information (e.g. foot plant constraints) and a descriptive label (e.g. sneaking). Then the goal is to find the best transition points for each motion clip. Transition points are found for every pair of motions in the database. To find the transitions, a distance metric is defined; the one used here is the sum of squared distances of point clouds. The distance between two frames from motion A and motion B is calculated by using a window of 10 frames around each frame in motions A and B. The window of frames for each motion is converted into point clouds around the joints. The point clouds are aligned using a linear transformation applied to one of the motion window s, and finally the sum of squared distances is calculated using the point clouds. This calculation is done for every pair of frames in motions A and B, and for every pair of motions in the database, resulting in a distance matrix for each pair of motions in the database. Once a distance matrix is computed, the local minima points are extracted using a pre-defined threshold. The threshold is defined by the user on a per-motion basis, since the motions being compared will vary and have different thresholds for different types of motions. The next step is to create the synthesized transitions between the motions; these transitions are added to the motion graph as edges. The final graph is pruned to remove any dead ends by saving only the strongly connected components of the graph.

Synthesizing new motions requires a traversal of the motion graph. The authors call this a

“walk,” which is cast as an optimization problem to search the motion graph. The user defines a cost function for adding edges to the synthesized motion, a legality function for determining which clips are legal to add to the synthesized motion based on constraints and such, and a halting function. An incremental search then builds the “walk.” Path synthesis requires supplying a desired path, an error function a nd halting function. The system measures the actual path travelled by the character during a graph walk and measures the difference to the user-supplied path.

One of the difficulties with the motion graphs approach is that the user must specify a cost function that does not over specify the goal of the new motion. That is, if the user simply restricts the beginning and ending constraints of the motion, the out come will most likely not be a desirable motion. The cost function should guide the entire motion. Another issue is that using just the transition points from the distance matrices alone is not sufficient for synthesizing new motion. The transition points are used for determining which sections of motions need to be blended to then be able to use the transition from one motion to another.

Our work for re-sequencing cartoon images to create new motions is similar in that a suitable distance metric must be identified for the results of the re-sequencing to yield appropriate new motions. A question that can be asked is, once a suitable distance metric is identified, isn’t the distance matrix that is calculated enough to generate new motions, like with the motion graphs paradigm? A distance matrix alone is not sufficient to produce new motions; the distance matrix will help in determining the transition points only, but will not solve our problem of re-sequencing to create new animations. Comparing images is not a trivial problem, and the quality of the re-sequenced motions depends on the quality of the distance metric. Using nonlinear dimension reduction, we create a manifold structure of the data then use this lower-dimensional graph structure for re-sequencing. The graph does not need to be pruned, rather, the entire graph is a strongly connected component, the nodes represent the images and the edges represent the transitions that can be taken. The user only needs to specify a start and end pose to create the new motion; the new motion is then created by traversing the graph and playing back the frames in the sequence determined by the traversal. By representing the image-based motion data in a lower-dimensional structure, we can also use the structure of the manifold to generate new images. A distance metric alone could not provide enough information to generate inbetweens. In this respect, the paradigm of motion graphs is not appropriate for re-using cartoon motion. If image-based cartoon data were used to create a motion graph, the distance metric would provide the transition points, but there is no way to synthesize blended transitions that are intelligible from blending the images. Also, we are re-sequencing the order of the motion clips so to speak (re-sequencing the order of the frames from the cartoon data) for creating new motions, not just playing back segments of motion pieced together by blended segments. Another issue is temporal coherence. In creating a lower-dimensional structure, temporal information from the original motion can be easily preserved.

III.3 Dimension reduction

An alternate way of thinking about the problem of re-using and inbetweening animation data and how to represent the data mathematically is by using dimension reduction. Cartoon animation data can be viewed as requiring a high-dimensional space to represent all possible variations or degrees of freedom. Methods for learning low-dimensional models from this high-dimensional data have been recently used in animation systems [Kovar et al. 2002; Jenkins and Matarić 2003]. Using a low-dimensional representation of the data may provide insights into the structure of the data and how best to re-use it. The goal of dimension reduction is to represent the data in a lower-dimensional space in such a way as to preserve certain properties of the data as faithfully as possible. The measure of how well the data are approximated by the lower-dimensional space is referred to as the residual error. There are linear and nonlinear methods of dimension reduction, and we discuss the most common and relevant methods here.

Principle Component Analysis (PCA) [Jolliffe 1986] is a linear projection on a subspace of the original data that best preserves the variance in the data. PCA uses eigenvalue decomposition on the covariance matrix of the data to produce a subset of eigenvectors that represent the principle variations in the data. These eigenvectors form a linear subspace for representing the data, generating a mean image and eigenvectors that span the principle shape variations in the image space. However, PCA assumes that the structure of the data are linear, and that the input data are independent. We know that animation data are temporally correlated, with a specific sequential order. Thus, PCA is probably not be the best choice for cartoon data.

Independent Component Analysis (ICA) is a statistical method that separates the independent components in a multivariate signal by maximizing the statistical independence of the estimated components. For example, ICA of a random vector x consists of finding a linear transformation $s = Wx$ such that the components s_i are as independent as possible, in the sense of maximizing some function F that measures independence. There are both linear and nonlinear forms of ICA. However, we do not expect statistically independent components to give us any advantage over the nonlinear method we chose (Isomap).

Multidimensional Scaling (MDS) [Kruskal and Wish 1978] is a nonlinear approach to dimension reduction that preserves pairwise distances to uncover the structure of the data. MDS allows for the visualization of how near data points are to each other for many kinds of distance or dissimilarity measures and produces a representation of the data in a lower dimension. MDS is a generic term that includes many different types, which can be classified according to whether the data are quantitative (metric MDS) or qualitative (nonmetric MDS). The variants of metric versus nonmetric MDS differ in their cost functions and optimization algorithms. With metric MDS, the cost function measures the distance between pairwise data. With nonmetric MDS, the cost function ranks the dissimilarity of the data.

There are several methods of unsupervised learning that use eigendecomposition to obtain a lower-dimensional embedding of data lying on a nonlinear manifold. The two we are most interested in are Local Linear Embedding (LLE) [Roweis and Saul 2000] and Isomap [Tenenbaum et al. 2000].

We briefly discuss the two methods, and defer a more detailed look at Isomap until Chapter V.

LLE looks for a lower-dimensional embedding that preserves the local geometry of a small neighborhood for each data point. It is assumed that the data lies on a manifold and that each data point and its local neighborhood are approximately a linear subspace. Briefly, the algorithm approximates the data with many linear patches. The patches are assembled together on a low-dimensional subspace such that the relationships between patches are preserved. The assembly of the patches is achieved by global optimization.

Isomap generalizes MDS to nonlinear manifolds. Instead of using Euclidean distances between pairwise data, Isomap approximates the geodesic distances on the manifold. Isomap works well for nonlinear data, it preserves the global data structure, and it optimizes globally. The basic algorithm begins by constructing a neighborhood graph for each data point, pairwise distances are computed, then MDS is used to reduce the dimensionality.

However, neither LLE nor Isomap account for temporal structure in cartoon data. A modified version of Isomap, called Spatio-Temporal Isomap (ST-Isomap) [Jenkins and Matarić 2003; Jenkins and Matarić 2004], can account for the temporal dependencies between sequentially adjacent frames. We borrow the idea of extending Isomap using temporal neighborhoods, and use ST-Isomap for dimension reduction of cartoon data to maintain the temporal structure in the embedding. [Jenkins and Matarić 2003] focuses on synthesizing humanoid motions from a motion database by automatically learning motion vocabularies. Starting with manually segmented motion capture data, ST-Isomap is applied to the motion segments in two passes, along with clustering techniques for each of the resulting sets of embeddings. Motion primitives and behaviors are then extracted and used for motion synthesis. This type of analysis and synthesis also requires more data than is typically available for cartoon synthesis. Thus, we adapt the methods of [Jenkins and Matarić 2003] to use images as input, and use only one pass of ST-Isomap for creating the embedding used for re-sequencing.

Pless [Pless 2003] has investigated using Isomap for exploring and analyzing video sequences as a trajectory through large image spaces. The idea is to have automatic tools for analyzing video by representing the video as a trajectory through the image space in a lower dimension. The author defines a video trajectory as a representation of changes in a video sequence based upon the nonlinear dimension reduction method of Isomap. The analysis of the video trajectories through the lower-dimensional spaces reveals five categories describing the shapes of the trajectories: (1) cyclic, which are repetitive video sequences, (2) helical, a periodic action viewed by a moving camera, (3) knotted, non-periodic or dynamic motions such as fountains, smoke, and flames, (4) linear, smoothly changing but not repetitive motions such as a slow pan across a scene, and (5) combinations of the four categories with distinct transitions between each type. One application of these video trajectories is to segment video based on which category the trajectory falls into, such as a transition from a bird in flight to a bird gliding. While this method is similar to our work in creating a lower-dimensional image space for video using Isomap, their work is focused on using the embedding space to analyze the video for such purposes as segmentation or classification of the motion

portrayed in the video. Our work is concerned with re-using the video by re-sequencing based on traversing the embedding space.

Recent work in dimension reduction/manifold learning include Manifold Charting [Brand 2003], Maximum Variance Unfolding (MVU) [Weinberger and Saul 2004], and hessian locally linear embedding (hLLE) [Donoho and Grimes 2003]. [Brand 2003] recovers a Cartesian coordinate system for a manifold of sampled data by constructing a nonlinear mapping from a high-dimensional sample space to a low-dimensional vector space. [Weinberger and Saul 2004] use semidefinite programming to do unsupervised learning of image manifolds and formulate the manifold learning problem by defining a set of constraints that are optimized to "unfold" a manifold. hLLE is a modification of the LLE method designed to produce linear embedding functions that exactly recover a hidden parametrization for data lying on a manifold that is locally isometric to an open connected subset of Euclidean space. All are more computationally expensive and account for structure in the data set that we do not expect, for example, non-convexity.

CHAPTER IV

PREPARING TRADITIONAL ANIMATION FOR RE-SEQUENCING

As mentioned in Chapter I, 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. Segmentation is necessary to facilitate the image comparison method, discussed in the next chapter, and to create new animations that can be taken from various episodes with differing backgrounds. This chapter examines three methods for semi-automatic segmentation of cartoon images in preparation for use in the re-sequencing method.

IV.1 Pre-Processing Cartoon Data

All data used in this dissertation comes from two-dimensional animated video or 'toon-rendered motion capture. As such, the video must be pre-processed to remove the background and align the character relative to a fixed location throughout the sequence, allowing for easy calculation of image similarity later. There are a number of video-based tracking techniques that can be used for background subtraction or segmentation. Three segmentation methods have been applied to the cartoon data and are described in this chapter. When the methods do not completely segment the character, a small amount of manual clean up of the images is required. Since the representation of the data is model-free, identification of specific regions of the character, i.e., limbs or joints, is not necessary, so it does not matter that the characters may undergo deformation. The alignment of the character in image space is done using the centroid of the character in each frame and repositioning it to the center of the image, facilitating the computation of a distance matrix, which is described in Chapter V.2.

Six cartoon sequences with different characters are used throughout this work, a *gremlin*, *Bugs Bunny*, *Wile E. Coyote*, *Daffy Duck*, *Michigan J. Frog*, and the *Grinch*. Table IV.1 shows the number of images used in each data set, the size of the original images and the cropped or scaled image size used for re-sequencing. The only synthetic data set is the *gremlin*, which is created using three clips of motion capture of free-style dancing performed by the same subject, and is played through a gremlin model that is 'toon-rendered on a constant white background. 'Toon-rendering motion capture data onto the *gremlin* allows for creating a large data set of images that are already segmented and aligned, providing the proof of concept for re-sequencing cartoon-like images to create new animations. Of the hand drawn cartoon examples, the *Coyote* data set is composed of frames from three different cartoons, called Coyote-1, Coyote-2, and Coyote-3. The others are composed of frames from only one cartoon, but have breaks where the scenes change. The *Bugs Bunny* images

Table IV.1: Details of traditional animation data sets. * Indicates that the data set was reduced in size by removal of duplicate frames.

Data Set	Number of Images	Original Image Size	Modified Image Size
Bugs Bunny	553*	720 x 480	360 x 240
Wile E. Coyote	527*	720 x 480	360 x 240
Daffy Duck	560	720 x 480	310 x 238
Michigan J. Frog	146	640 x 480	320 x 240
Grinch	295	640 x 480	320 x 240
Gremlin	2000	320 x 240	150 x 180

are from the 1946 short "Slick Hare" directed by Isadore "Friz" Freeleng. The Coyote-1 images are from the 1953 short "Stop! Look! Hasten!," the Coyote-2 images are from the 1954 short "Ready.. Set.. Zoom!," and the Coyote-3 images are from the 1955 short "Guided Muscle." The *Daffy Duck* images are from the 1953 short "Duck Amuck," the *Michigan J. Frog* images are from the 1955 short "One Froggy Evening," and the *Grinch* images are from the 1966 film "How the Grinch Stole Christmas." All of these cartoons were directed by Chuck Jones. Figure IV.1 shows examples of the frames from the original data along with the corresponding segmented images. This chapter focuses on the *Bugs Bunny* and *Wile E. Coyote* data sets for segmentation, as the other data sets were processed manually at an earlier time.

IV.2 Segmentation of Color Images

In analyzing objects in images, it is essential that we distinguish between the objects of interest and "the rest" of the image. The objects of interest are referred to as the foreground, while "the rest" is referred to as the background. Techniques used to find objects of interest are referred to as segmentation techniques, i.e., segmenting the foreground from background. Segmentation involves understanding that objects of interest, in this case cartoon characters, follow some specific form or have some known properties. For instance, we know cartoon characters are usually easily identifiable, and often made up of a few solid colors (*Daffy Duck* is mostly black with some orange). Creating a library of animation data that includes a variety of cartoon characters requires extensive pre-processing time, particularly if it is done manually. To generate good re-sequenced animations, there must be a way to compute the similarity of the character in the images. As such, segmenting and aligning the character is a vital step in our process, particularly with the distance metrics we describe in Chapter V.2. Segmentation is the most time consuming aspect of preparing existing cartoon data for discovering the structure of the data via manifold learning methods.

Image segmentation is fundamental to image processing [Gonzalez and Woods 2001; Shapiro and Stockman 2001]. We examined three techniques for potential use in our system: ad hoc method using simple probability of color distribution, level sets [Osher 2003], and support vector machines [Vapnik 1998] (SVMs). Some of the more common techniques such as thresholding or

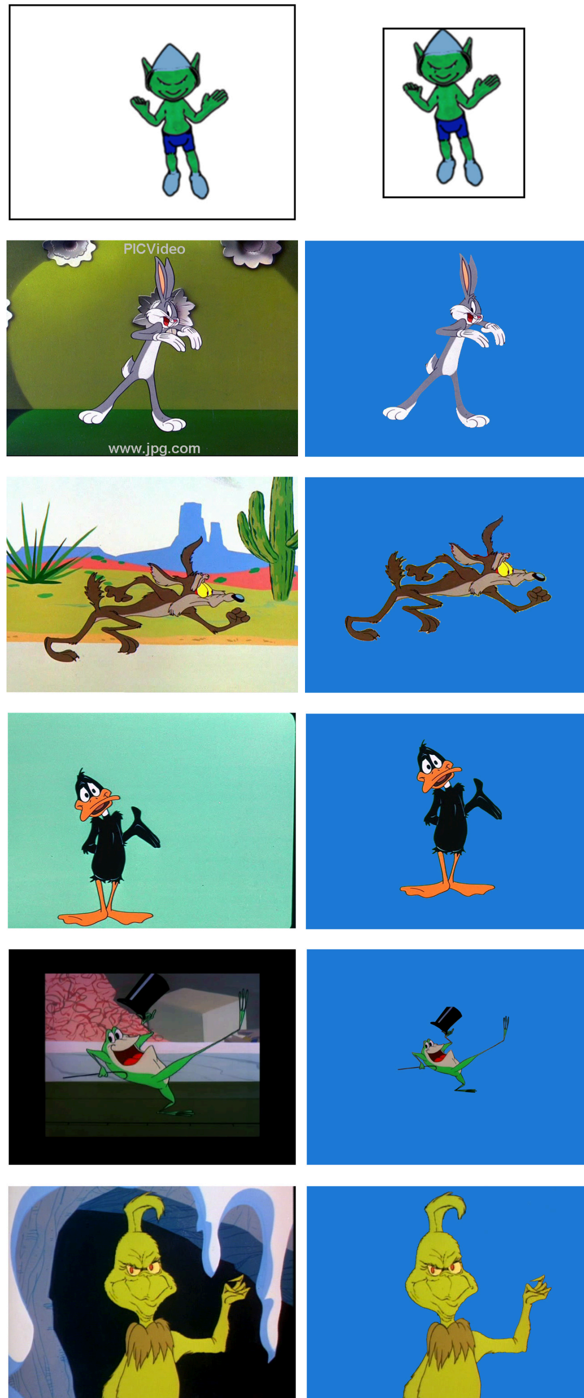


Figure IV.1: The top row shows a frame from the synthetic *gremlin* data set before and after processing (the images are cropped). The second row shows an original and cleaned up frame from the *Bugs Bunny* data. An example frame from the *Wile E. Coyote* data in the third row, *Daffy Duck* in the fourth row, *Michigan J. Frog* in the fifth row, and finally *Grinch* in the last row. The segmentation results for *Bugs* and *Coyote* are generated using methods described in this chapter. *Bugs*, *Coyote*, *Daffy*, and *M.J. Frog* ™ & ©*Warner Bros. Entertainment Inc.*, *Grinch* ©*Turner Entertainment Co.*

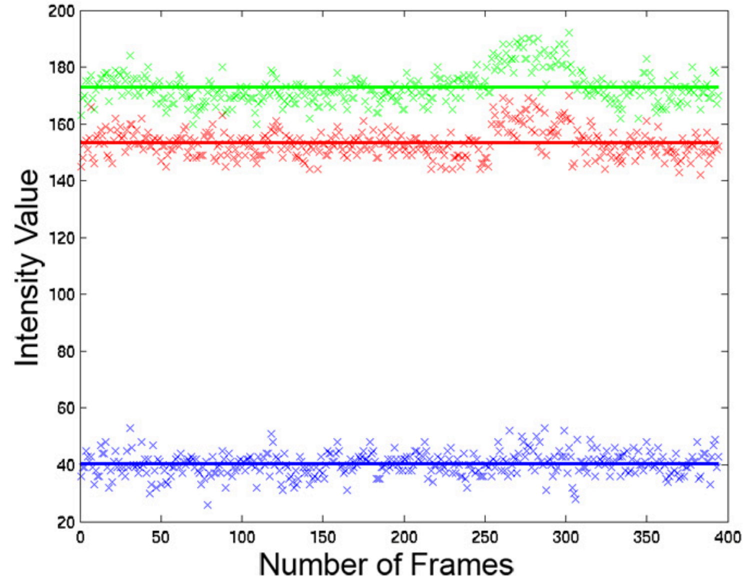


Figure IV.2: The deviation from the mean color value for each color channel from a single background pixel is shown as data points around a mean value line. The color of the data corresponds to the color channel.

edge detection work well for certain types of images. With thresholding, one can segment based on intensity of a grayscale image, or use a histogram to find the region to segment. However, in color cartoon images, using only the intensity of the images loses a great deal of information, and often more than one threshold level is required to define the object of interest. With edge detection, gradient methods can find edges of objects, but are sensitive to noise. As mentioned previously, the problem of segmenting traditional animation is difficult. Because the source of the cartoon data used throughout this dissertation comes from DVDs, there is noise in the animations from both image compression artifacts and color variations in the 50 year-old cartoons. Even without MPEG-2 compression artifacts on a DVD, the original cels also have noise and artifacts due to changes in lighting as the cels were transferred to film and other degradation issues before being archived. Simple thresholding is not robust enough to deal with the noisy images. As an example, Figure IV.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 IV.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.

IV.3 Ad Hoc Method

The first method we describe for segmenting color cartoon images is an ad hoc method that uses the probability of color values and some thresholding for cleaning up the image masks. For each character data set, the user selects several example pixels (colors) that represent the character to be segmented stored in $R^n_{r,g,b}$, along with one pixel (for every frame) containing the (x,y) coordinates

in image space where a part of the character is located, stored in $P^m_{x,y}$. $R^n_{r,g,b}$ is an n x 3 vector of RGB values representing n reference colors of a character, and $P^m_{x,y}$ is an m x 2 vector of (x,y) pixel coordinates for m frames in the sequence. Selecting the $R^n_{r,g,b}$ and $P^m_{x,y}$ points is a small task for the user, taking less than 10 minutes for both the *Bugs* and *Coyote* data sets. Basically, this ad hoc method finds regions of connected pixels whose colors are within a defined tolerance, tol, of reference colors $R^n_{r,g,b}$. The tolerance is a scalar value set by the user. A mask for each image in the sequence, $mask_m$, is initialized to be the same size as the input image to be segmented with all pixels set to a value of $mask_m = false$. The value at each element of $mask_m$ is compared to the sum of squared color differences, i.e., if the differences are within the tol^2, then $mask_m(i,j) \vee 1$ otherwise $mask_m(i,j) \vee 0$. The logical $\vee$ operator is equal to 1 unless both elements are 0. Because $mask_m$ is initialized to $false$, or all zeros, only the sum of squared differences between the current pixel color and the reference pixel color $R^n_{r,g,b}$ that fall within the tol^2 are included in the mask. The first step in the image mask calculation can be summarized in the following equation:

$$mask_m = mask_m \vee ((C_r - R_r)^2 + (C_g - R_g)^2 + (C_b - R_b)^2) \leq tol^2 \qquad \text{(IV.1)}$$

where C_r, C_g, C_b are the current RGB colors for image m, and R_r, R_g, R_b are the RGB colors from $R^n_{r,g,b}$ for each color sample n.

Next, the number of regions in $mask_m$ with a value of 1 that have 8-neighbor connected components are labelled as potential character objects. If one of those objects overlaps with the reference pixel $P^m_{x,y}$ for image m, that object is included in the output mask, otherwise it is removed. Finally, morphological close is used to clean up the resulting image mask. Figure IV.3 shows an example of a successful segmentation of *Coyote*. Notice that in the middle image there is a region in the center of *Coyote's* head that has a hole. Any similar areas in the resulting masks are either filled semi-automatically by the user selecting one point in the region to be flood filled, or by using morphological operators. However, there are some problems with this ad hoc method of segmentation. Figure IV.4 shows that part of the background was included with *Bugs*. Because the flower in the background is a similar color to *Bugs*, it is included as part of the foreground. Adjusting the only parameter, the tol, to exclude the background flower results in losing parts of the character. When dealing with similarly colored foreground and background elements, the ad hoc method reaches its limit and fails to segment the character from the background.

IV.4 Level Sets for Segmentation

The second method used for cartoon segmentation is a level set algorithm developed by [Cao and Dawant 2005], adapted to both grayscale and color images. There is much ongoing research using level sets from applications in medical image segmentation to fluid model animation of smoke and fire. First, a brief introduction to level sets: this is a numerical method that models propagating surfaces with time-varying, curvature-dependent speeds, introduced by Osher and Sethian [Osher and Sethian 1988; Osher 2003]. The surfaces are viewed as a specific *level set* of a higher-dimensional function. In two dimensions, the level set method represents a closed curve Γ (the interface bound-

Figure IV.3: On the left is the original image, the center shows the mask generated using the ad hoc method, on the right is the cleaned up mask. *Wile E. Coyote* is ™ & ©*Warner Bros. Entertainment Inc.*

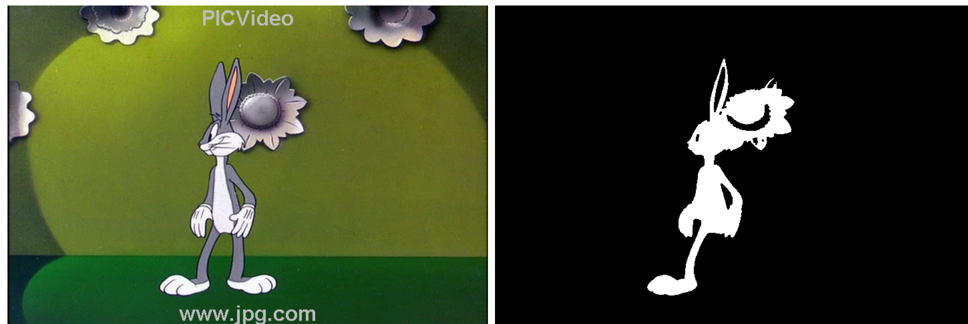


Figure IV.4: On the left is the original image, on the right is the segmented image that shows when the ad hoc method fails. *Bugs Bunny* is ™ & ©*Warner Bros. Entertainment Inc.*

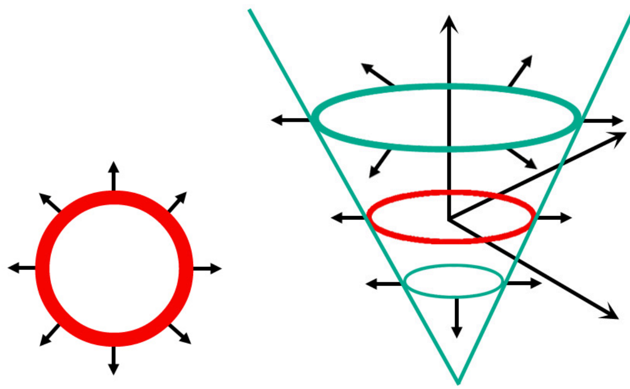


Figure IV.5: On the left is the red curve representing Γ, on the right is the level set function ϕ shown as a cone in blue-green. The blue-green surface is called the level set function, because it accepts as input any point in the plane and hands back its height as output. The red front is called the zero level set, because it is the collection of all points that are at height zero.

ary) as being embedded into a 3D function ϕ. In other words, the curve Γ is defined as a function of time as the zero level set, i.e., by $\Gamma(t) = \{(x,y)|\phi(x,y,t) = 0\}$, where ϕ (the embedding function) is usually chosen as a signed distance function. For this type of function, ϕ is positive on the exterior of the region bounded by Γ, and negative on the interior of Γ. A useful property of this representation is that the level set function ϕ remains a valid function while the embedded curve Γ can change topology.

Figure IV.5 illustrates the relationship between Γ and ϕ. The level set approach takes the initial position of the curve Γ (the red curve on the left in figure IV.5), and embeds it into a higher-dimensional surface (the blue-green cone on the right in figure IV.5). That cone-shaped surface has a great property in that it intersects the *xy*-plane exactly where the curve sits. Once Γ is embedded into a higher-dimension, the evolution of the embedding function ϕ can be linked to the propagation of the curve through a time-dependent initial value problem. The evolution of the level set function is determined by a user-defined speed function F. Then the evolution equation, an initial-value partial differential equation (PDE), for the level set function ϕ has the form: $\phi_t + F|\nabla\phi| = 0$, given $\phi(x,t = 0)$. Here, ϕ is the embedding function and F is the speed function, i.e., the speed of a point on the curve along its normal direction. With these equations, the method can be summarized as follows: the curve Γ is in a plane propagating in a direction normal to itself with a certain speed F so that at time t we can solve for the position of the curve $\Gamma(t)$ using the initial-value PDE.

IV.4.1 Level Sets: Method and Results

Knowing the structures of interest can be used for evolving an initial curve towards the boundaries of the structure, thereby helping understand what needs to be segmented. The interface between the object of interest and other parts of the image is the zero level set. Tracking the interface using level

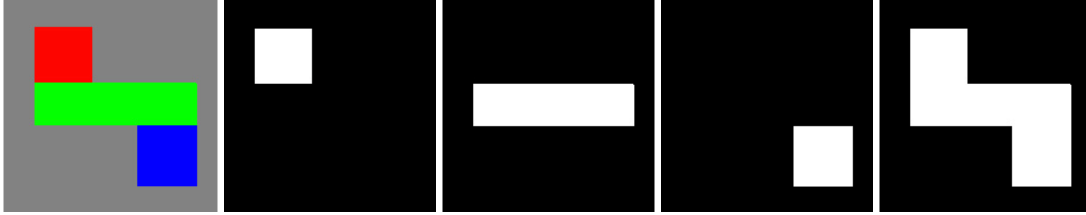


Figure IV.6: Starting on the left and moving right: the synthetic test image, the segmentation mask with $color = RED$, the segmentation mask with $color = GRN$, the segmentation mask with $color = BLU$, and finally the segmentation mask with the color stopping criteria equal to $[RED, GRN, BLU]$. Images obtained courtesy of Dr. Zhujiang Cao.

sets allows the front to move in either a positive or negative direction from its initial placement. Typical level set methods rely on careful placement of the initial curve for successful segmentation, and use gradient-based speed functions. The level set method we use overcomes many of these problems as it is a region-based algorithm that combines regional statistics with the curve evolution. The speed function F is defined by the curve evolution derived to minimize the total energy. The image is modelled as either a two-class or three-class case, and statistics (average intensities) for each region are used to derive the energy function. In the two-class model, a single curve is evolved for any number of features (grayscale or color images). In the three-class model, two curves are used, and are coupled to each other to maintain a global segmentation.

As such, the level set method we use works with both grayscale and color images [Cao and Dawant 2005]. An initial proof of concept test using the three-class model of the level set algorithm was applied to a simple synthetic image of a gray background with three rectangles of pure red, pure green, and pure blue. The simple image is size 200 x 200, and was used at this resolution for processing. When given the stopping criteria of $color = RED$, where $RED = [255, 0, 0]$, $GRN = [0, 255, 0]$, and $BLU = [0, 0, 255]$, only the red square was segmented. Likewise for the green rectangle and blue square. When all three colors were given as stopping criteria ($color = [RED, GRN, BLU]$), all three colors were correctly segmented. The time to segment the test image was four minutes. Figure IV.6 shows the test image used along with the segmentation results.

We applied the two-class model to an image of *Daffy Duck* to evaluate the segmentation, with the assumption that *Daffy* can be easily identified using features in only one color channel. Figure IV.7 shows the segmentation results on the red channel of the *Daffy* image. While this method works fairly well on a simple character such as *Daffy*, who is mostly black with a little orange, and proved easily identifiable in the red color channel, there are more challenges to segmenting the *Coyote* and *Bugs* characters. Figure IV.8 shows the original coyote image and the segmented image mask generated from applying the two-class model to the luminance of the image. The remainder of this section describes the results of the two-class and three-class models for segmentation applied to two cartoon characters.

While the two-class model level set method worked fairly well on *Daffy Duck* by using either

Figure IV.7: On the left is the original image, the center shows the mask generated using level sets, on the right is the segmented image. Images obtained courtesy of Dr. Zhujiang Cao. This *Daffy* image is from the 1951 short “Rabbit Fire” directed by Chuck Jones. *Daffy Duck* is ™ & ©*Warner Bros. Entertainment Inc.*

Figure IV.8: On the left is the original image, the right shows the mask generated using only the luminance. Images obtained courtesy of Dr. Zhujiang Cao. *Wile E. Coyote* is ™ & ©*Warner Bros. Entertainment Inc.*

a single color channel or the luminance of the image, the *Coyote* and *Bugs Bunny* characters are more complicated. In particular, some of the colors in both characters are also the same as some background elements. Also, there is some color variation due to noise from the age of the cartoons as well as DVD compression artifacts. Two of the three *Coyote* sequences suffer from a fair amount of compression artifacts, as shown in figure IV.9. When segmenting *Daffy* using all three color channels and the three-model method, the character was either over-segmented or the segmentation failed. The three-model method also failed on the *Coyote* color images. The resulting three-model color segmentation for *Bugs* only picked up small parts of the character, with a few pixels in the background on the flower that is the same color as the character. The over-segmented result of one *Bugs Bunny* image at $\frac{1}{4}$ resolution took sixteen minutes. Even the segmentation results using one color channel of *Daffy* with the two-class model, using an image also at $\frac{1}{4}$ resolution, averaged about five minutes per frame.

We believe that there is sufficient variation in the pixel values for the cartoon characters due to compression artifacts as well as film degradation from digitization of 50+ year old cartoons, that even with a small amount of tolerance for variation in the colors selected as the stopping criteria, the character will either be over-segmented or the algorithm will produce empty regions and run into numerical instability. The other issue with this method is the amount of computation time required to segment one image at $\frac{1}{4}$ resolution. The time to segment one cartoon image is between five to

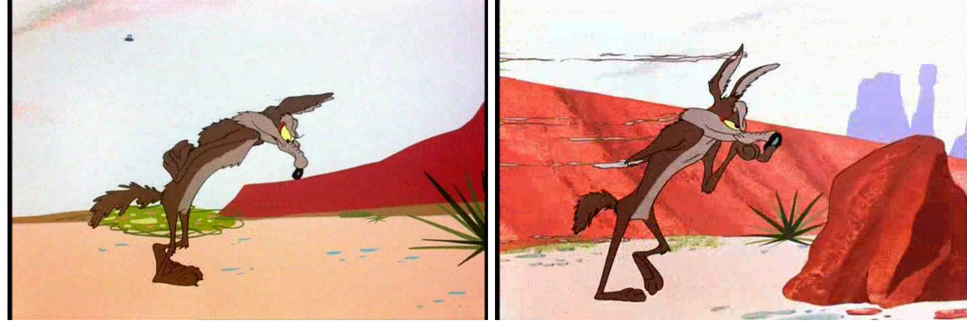


Figure IV.9: Original *Coyote* images from two different cartoons with compression artifacts, seen as blocks in the background sky. *Wile E. Coyote* is ^{TM}& ©*Warner Bros. Entertainment Inc.*

sixteen minutes. Assuming the worst case scenario of sixteen minutes per image, with a total of 1,080 images to be segmented (527 *Coyote* and 553 *Bugs*), the time to segment all images would take about 288 hours, or twelve days. In comparison with the simple ad hoc method, segmentation on a full size image requires only twelve seconds per image.

IV.5 Support Vector Machines

The third method applied to segmenting color cartoon images is the use of Support Vector Machines (SVMs). A support vector machine is a supervised learning algorithm based on the concept of a hyperplane that defines a boundary separating sets of objects that have different class memberships [Vapnik 1998]. It is a supervised algorithm because the examples in the training data are input/output pairs, where each pair is an example input object with its associated class label. The SVM algorithm operates by mapping the given training set into a high-dimensional *feature space* and finding in that space a plane that separates the data into the appropriate classes (two classes in the case of a binary SVM). Any consistently labelled training data set can be made separable. To avoid over-fitting the data by finding trivial solutions, SVMs choose the maximum margin separating hyperplane from among the many hyperplanes that can separate the examples in the feature space. For example, given training examples labelled either "yes" or "no," a maximum-margin hyperplane is found that splits "yes" examples from the "no" examples in such a way that the distance between the hyperplane and the closest example (called the margin) is maximized.

Figure IV.10 presents an overview of the SVM algorithm, mapping the input space to a feature space, finding the hyperplane separating clusters of data to fall on either side of the plane, and the margin (distance between separating line and closest data points to the line). The points that constrain the width of the margin are the support vectors. Using a soft margin allows some training examples to fall on the wrong side of the hyperplane, therefore making the algorithm robust to mislabelled examples. The feature space can be defined by vectors in the input space and dot products in the feature space, so the SVM can find the separating hyperplane without the need for representing the feature space explicitly by defining a kernel function (the so-called "kernel-trick").

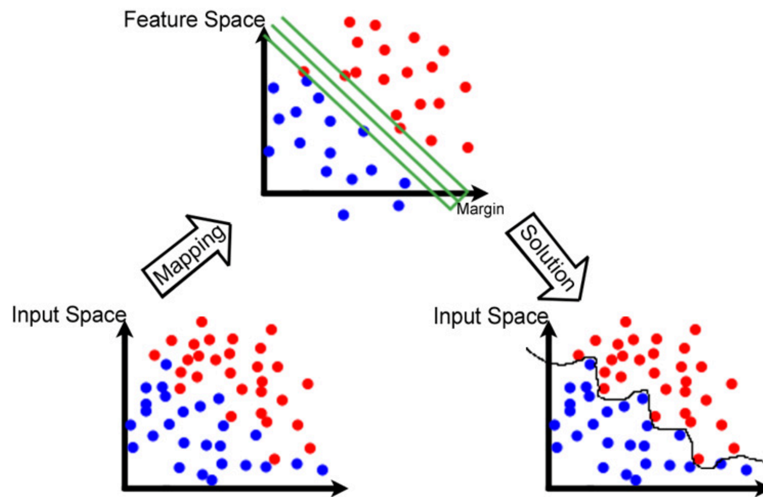


Figure IV.10: An illustration of the SVM algorithm. Two classes of data are represented as red and blue points, the input space is shown before and after processing, the hyperplane is the line separating the red and blue points in the lower right. In the feature space, the margin is shown in green, the vectors (data points) that constrain the width of the margin are the support vectors.

There are a number of reasonable kernel functions that can be used in SVM models, such as linear, polynomial, radial basis function, and sigmoid. In fact, if given a way of computing the inner product between a test point and training point in the feature space directly as a function of the original input points, it can be used as a kernel function. The goal of the SVM model is to predict labels of data instances in a testing set given only attributes. To construct an optimal hyperplane, the SVM is an iterative algorithm used to minimize an error function. Depending on the form of the error function, SVM models are classified into four groups: C-SVM classification (C-SVC), ν-SVM classification (ν-SVC), ε-SVM regression (ε-SVR), and ν-SVM regression (ν-SVR) [Chang and Lin 2001]. For classification, the difference between C-SVC and ν-SVC is in the penalty term of the error function. With C-SVC, the penalty parameter of the error function (equation IV.2 below), is the constant $C > 0$, which determines the trade-off of allowing training errors. C provides an upper bound on outliers and limits the influence of those potential outliers. With ν-SVC, the C parameter in the error function is replaced by a parameter $\nu \in [0,1]$ that determines the lower bound on the number of examples that are support vectors, and the upper bound on the number of examples that are allowed to lie on the wrong side of the hyperplane (outliers). For regression, ε-SVR is analogous to the C-SVC for classification and has a similar error function. The model produced by ε-SVR depends on a subset of the training examples and ignores any examples within a threshold of ε to the model prediction, similar to the upper bound provided by C. Finally, ν-SVR is similar to ν-SVC in that it uses the parameter ν to control the number of support vectors, and replaces the parameter ε of ε-SVR.

In this work, we are training a binary SVM with the C-SVM classification. Regression models are not appropriate for our problem of segmentation. Given a training set of attribute-label pairs (x_i, y_i), where $i = 1 \dots l$, training vectors $x_i \in \mathbf{R}^N$ and labels $y_i \in \{+1, -1\}^l$, C-SVM minimizes the following error function:

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

subject to the constraint

$$y_i(\mathbf{w}^T\phi(x_i) + b) \geq 1 - \xi_i. \tag{IV.3}$$

The training vectors x_i are mapped to a higher dimension by the kernel function ϕ. C is the penalty parameter of the error function, which controls the tradeoff between allowing training errors and forcing rigid margins. Increasing C increases the cost of misclassifying points and forces the creation of a more accurate model that may not generalize well. The vector $\mathbf{w}$ is a vector of coefficients, b is a constant, and ξ_i are variables for handling non-separable input data. ξ_i are called the *slack variables*, and allow for the possibility of data examples violating the constraint given in IV.3, should the inequality not have the ξ_i on the right hand side. Essentially, the ξ_i take into account any noise in the data that would otherwise prevent the hyperplane in the feature space from separating the differently labelled examples. We chose to use an RBF kernel that has the form $\phi = e^{-\gamma\|x_i - x_j\|^2}$, where $\gamma > 0$, and the indices i and j run over the training set, i.e., $i, j = 1 \dots l$. We are trying to find a separating hyperplane that will classify the data as having a label of 1 or -1. As such, the RBF kernel can be viewed as a similarity measure that will divide the input data (x_i, x_j) into one of those two classes. Using the RBF kernel with C-SVM is the most popular mainly because it requires the user to set only two parameters, C and γ. The other variables $(\mathbf{w}, b, \xi_i)$ are solved for when minimizing the error function given in equation IV.2. In our work, we use the LIBSVM library [Chang and Lin 2001] with the RBF kernel to train the C-SVM model.

Because the accuracy of the SVM model largely depends on the selection of the model parameters, a grid search is used to find the optimal C and γ parameters for the RBF kernel. Using cross-validation, pairs of C and γ are tried over a specified grid and the pair with the best cross-validation accuracy is picked. For a specified value of C and γ, the cross-validation accuracy is computed as follows: train an SVM model using a subset of examples $(l-1)$ of attribute-label pairs, one example is left out, and the model predicts the label for the unseen example. The accuracy of that prediction is calculated for all examples in turn leaving one out, and the average of accuracy on predicting the sets is the cross-validation accuracy. We define a grid (m, n) using exponentially growing sequences of C and γ, such that $C = 2^{2m-1}$ and $\gamma = 2^{2n-1}$, $m \in [-2, \dots, 8]$, and $n \in [-7, \dots 3]$. We chose these values for our grid search based on examples provided by [Chang and Lin 2001]. Although other methods exist to find the best accuracy for a pair of inputs, a grid search is very simple and can be easily sped up, if necessary, by parallel processing since each (C, γ) pair is independent. The C and γ with highest cross-validation accuracy, in our experience typically above 96%, is selected.

IV.5.1 Training SVM Model on Cartoon Images

To segment cartoon images, we first train an SVM model by selecting the appropriate attribute-label examples. 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 of the character, since for example, *Coyote* will always be two shades of brown with yellow eyes. Another choice of feature is using the optical flow. Many times the character will be on a stationary background, and one unique characteristic of hand-drawn cartoons is that there are not shading or lighting changes in the images, so the optical flow may be useful in locating the character. We used four variations on the color and optical flow features for setting up the data. One way is to have the user select several pixels from a reference image, label each point as part of the character (1) or part of the background (-1). Here, only RGB values are used as the samples. A second way to set up the training data is to have the user select one scanline from a reference image, have labels for each color value applied automatically by looking up the appropriate label from a corresponding pre-segmented (manually) mask, and once again, use only RGB values as the samples. A third variation is to have all RGB values from one reference image as samples, and have labels applied automatically by looking up the appropriate label from a corresponding pre-segmented (manually) mask. Finally, the user can select several pixels from a reference image, as in the first approach, but RGB values and optical flow vector magnitudes are used as the samples. The optical flow vector magnitudes are pre-computed using [Lucas and Kanade 1981] from the temporally adjacent frames in the cartoon image sequences. The main reason for using optical flow is to identify moving background pixels in the Coyote-1 and Coyote-2 data sets. Using the location of the reference pixels selected by the user, the corresponding optical flow vector magnitudes are included as samples and given the same labels as the reference pixels. The method used to select the features for training an SVM model for each particular character is noted in the results section. Also, with any of these variations for setting up the data, multiple reference images can be used. One important note is that all RGB values are scaled to be in $[0,1]$ range.

IV.5.2 SVM Segmentation Results

Segmentation of cartoon images uses the classified SVM model (one for each character) on each image in the data set. The SVM output of the predicted labels is the resulting segmented mask. Figure IV.11 shows the result of using 81 RGB samples and optical flow magnitudes for training the SVM model. The top row of figure IV.12 shows the result of using one scanline of a blurred image for training the SVM model, and the bottom row uses 68 RGB samples for training the SVM model. The best segmentation result achieved on the *Bugs Bunny* data set is this last example where only 68 color samples from three images in the data set were used to train the SVM model. Any more samples or additional information resulted in over-classification, seen as more background pixels and noise in the segmentation masks (compare results in figures IV.11 and IV.12). For instance, including the optical flow vector magnitudes in the *Bugs Bunny* data set essentially picks up more of the background pixels, in particular the gray ones. The reason is because the gray pixels on *Bugs*

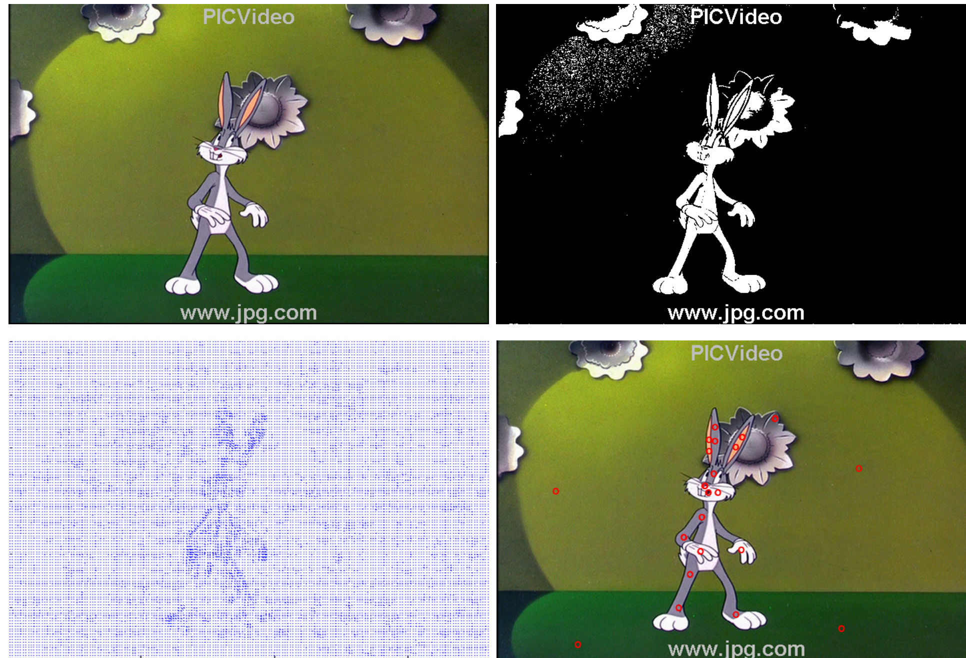


Figure IV.11: The top row shows the input image and the resulting segmentation mask, this was generated using 81 RGB samples and the corresponding optical flow magnitudes at those pixel locations. The samples came from three images in the Bugs Bunny sequence. The bottom row shows the optical flow vectors and one of the images used for the RGB samples. The red circles indicate the pixels selected for the samples. *Bugs Bunny* is ^{TM} & ©*Warner Bros. Entertainment Inc.*

Bunny typically have small optical flow vectors, which is also true of the gray background pixels.

Figure IV.13 shows the results of an SVM trained on 108 RGB samples with optical flow vector magnitudes, applied to the Coyote-1 data set. Figure IV.14 is the Coyote-2 data set, using an SVM trained on 179 RGB samples with optical flow vector magnitudes. The results for the Coyote-3 data set can be seen in figure IV.15, using an SVM trained on 140 RGB samples with optical flow vector magnitudes. Compare these results to the ad hoc segmentation method discussed in Chapter IV.3. One of the difficulties with these particular cartoon examples is that the character is walking across a moving background in the Coyote-1 and Coyote-2 data sets. Because of this, there are new color samples revealed throughout the sequences that may not be accounted for in the SVM model. It is also easy to see some of the compression artifacts in the segmented images, those show up as small regions of blocks in and around the background elements.

In all of the examples, there are some pixels that the SVM model erroneously classifies as part of the character. To further improve the segmentation masks, simple morphological operations are performed. The segmentation mask is a binary image with regions either equal to 0 or 1. First, each 8-connected region in the mask is labelled with a number. For each region found, the area (total number of pixels) of that region is calculated and stored. The region with the largest area, or larger than a preset value (i.e., 10,000 pixels), is likely to be the character and is kept in the mask, all others

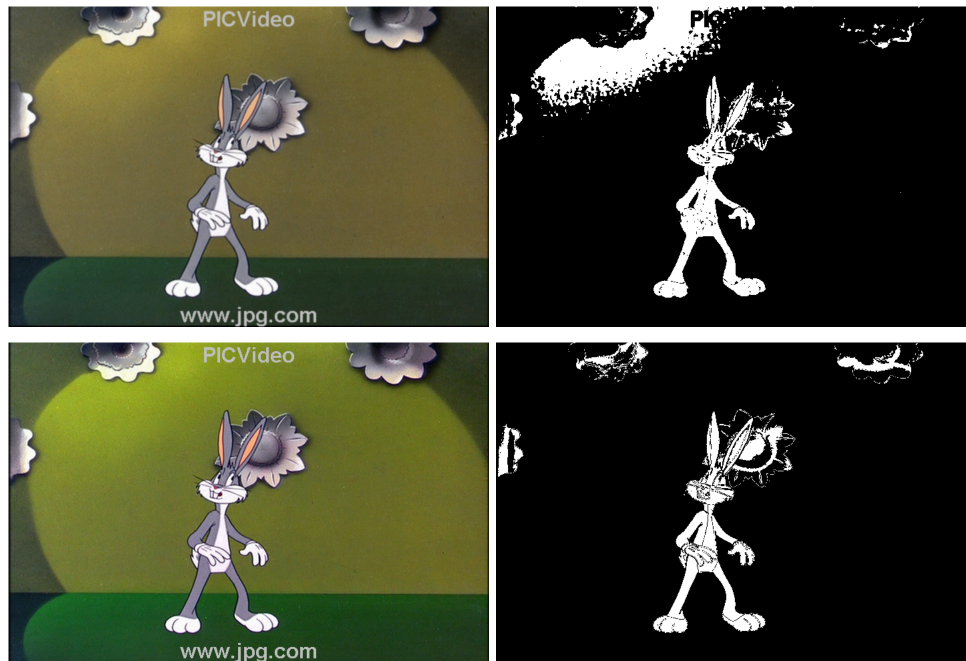


Figure IV.12: Here is the same image from the Bugs Bunny sequence. In the top row, the image was first blurred using a gaussian filter and the model was trained using one horizontal scanline from one image-mask pair. In the bottom row, the model was trained using only 68 RGB samples from three images. Of all of the segmentation results using SVMs, this last example shows the best classification of the cartoon character. *Bugs Bunny* is ^{TM}& ©*Warner Bros. Entertainment Inc.*

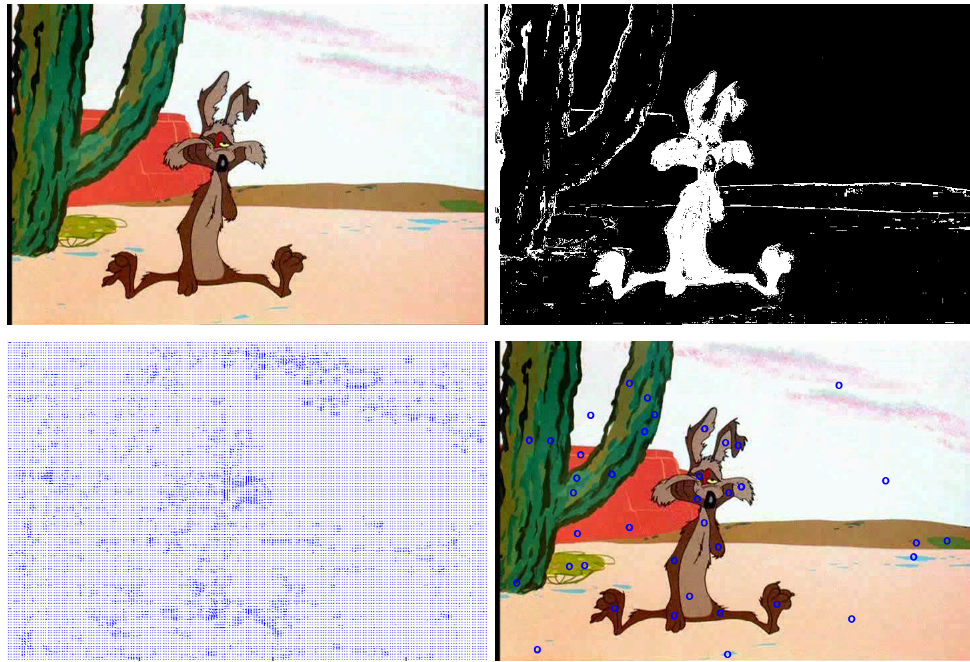


Figure IV.13: The top row shows the input image and the resulting segmentation mask, this was generated using 108 RGB samples and the corresponding optical flow magnitudes at those pixel locations. The samples came from three images in the Coyote-1 sequence. The bottom row shows one of the images used for the samples and the optical flow vectors for that image. The blue circles indicate the pixels selected for the samples. *Wile E. Coyote* is ^{TM}& ©*Warner Bros. Entertainment Inc.*

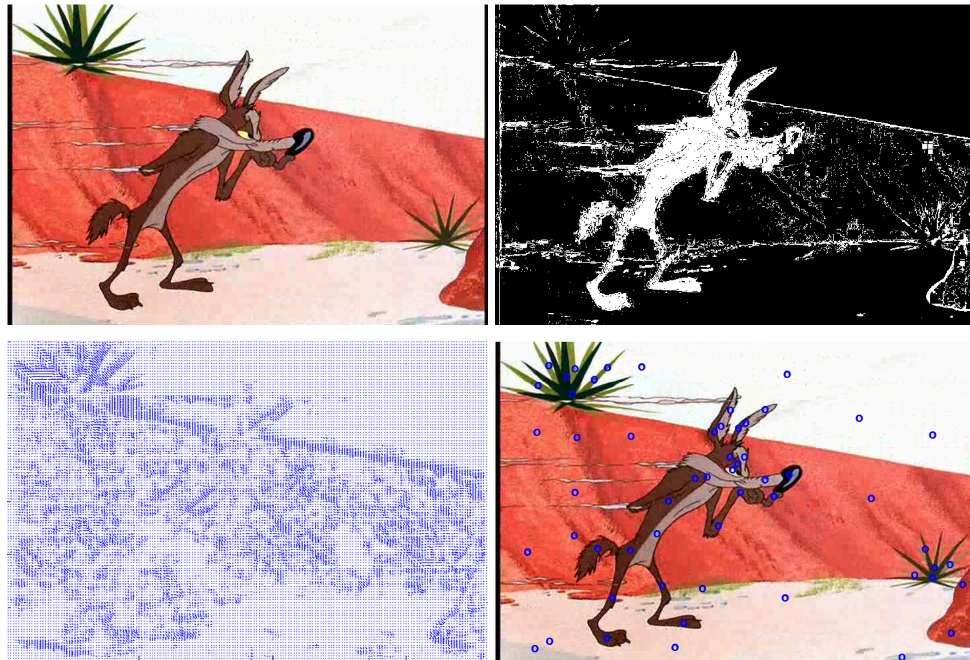


Figure IV.14: The top row shows the input image and the resulting segmentation mask, this was generated using 179 RGB samples and the corresponding optical flow magnitudes at those pixel locations. The samples came from three images in the Coyote-2 sequence. The bottom row shows one of the images used for the samples and the optical flow vectors for that image. The blue circles indicate the pixels selected for the samples. *Wile E. Coyote* is ™& ©*Warner Bros. Entertainment Inc.*

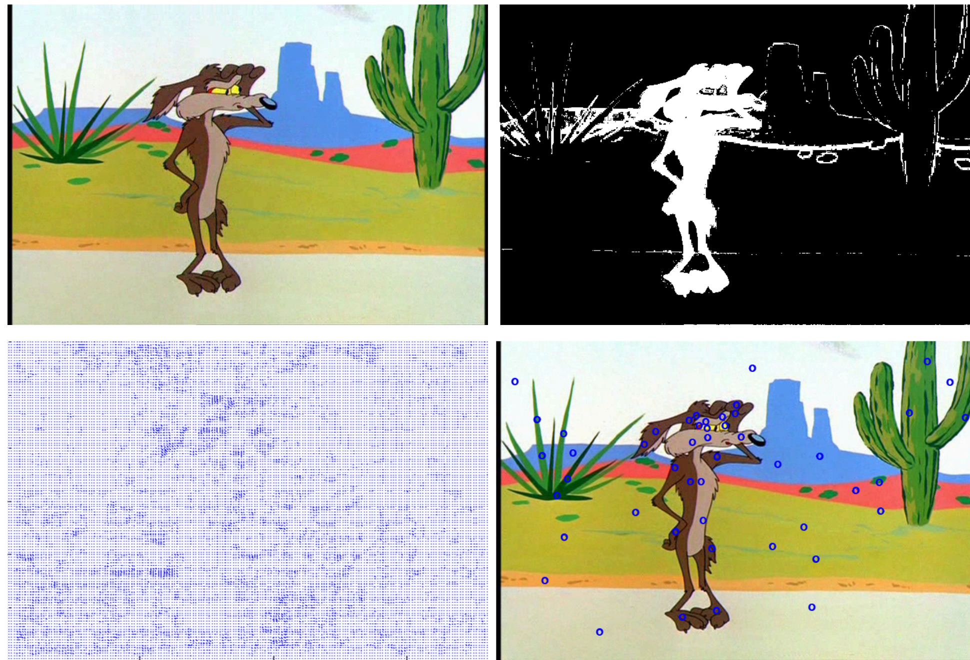


Figure IV.15: The top row shows the input image and the resulting segmentation mask, this was generated using 140 RGB samples and the corresponding optical flow magnitudes at those pixel locations. The samples came from three images in the Coyote-3 sequence. The bottom row shows one of the images used for the samples and the optical flow vectors for that image. The blue circles indicate the pixels selected for the samples. *Wile E. Coyote* is ^{TM}& ©*Warner Bros. Entertainment Inc.*

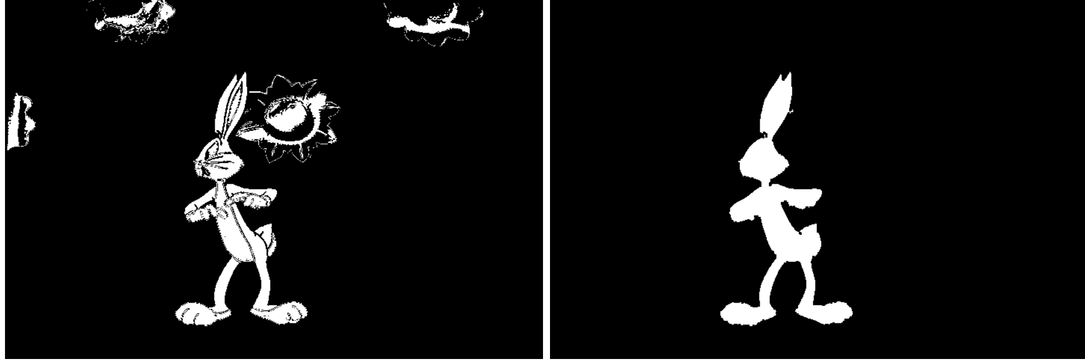


Figure IV.16: On the left is the segmentation result on *Bugs Bunny* using the 68 RGB sample SVM model. On the right is the result of applying morphological operations to clean up the segmentation mask.

are discarded. Finally, any remaining small holes are automatically filled. Figure IV.16 shows the results of applying this method to one of the SVM segmentation masks. Any remaining stray pixels are easily cleaned up manually.

IV.6 Summary

The segmentation methods presented in this chapter, in particular the support vector machine method, are robust and work well on all examples we have tried them on. This chapter presents the first necessary component of a system for building re-usable motion libraries of traditional animation: a method for semi-automatic segmentation of the images. Minimal user intervention is required, and is essentially a guide for the animator in building the motion library. The SVM technique works well for classifying cartoon characters, exploiting the strong color information that make up each character, despite any noise or artifacts in the images. We rejected using level sets because the time for segmentation of a $\frac{1}{4}$ resolution image was too slow, averaging ten minutes per image on a 1 GHz pentium. Both the ad hoc and SVM methods were very fast, with the ad hoc method segmenting full size images at a rate of ten seconds per image, and the SVM method segmenting full size images at a rate of about three seconds per image. We believe the robustness of classifying cartoon images using support vector machines provides a reliable and intuitive segmentation method.

CHAPTER V

DIMENSION REDUCTION FOR RE-SEQUENCING ANIMATION

Once a library of character data has been assembled using the techniques of the previous chapter, a method such as that described in this chapter can be used to generate novel sequences. We describe the re-sequencing process without generating new frames. The types of new motions that can be re-sequenced are restricted by the amount of data in the library for each character. This method is model-free, requiring no a priori knowledge of the cartoon character. First, the cartoon data is pre-processed, as described in the previous chapter, creating the library of data for each character. Next, nonlinear dimension reduction is used to learn the structure of the data, which requires a metric for comparing the similarity of the images in the data sets. Sometimes these techniques and others [Jenkins and Matarić 2004; Roweis and Saul 2000] are collectively called manifold learning techniques. However, in this dissertation we apply the older terminology. Finally, by selecting a start and end frame from an original data set, the data is re-sequenced to create a new motion.

V.1 Dimension Reduction

Nonlinear dimension reduction finds an embedding of the data into a lower-dimensional space. We use a simplified version of ST-Isomap [Jenkins and Matarić 2004] to perform the manifold-based nonlinear dimension reduction. Like standard Isomap, ST-Isomap preserves the intrinsic geometry of the data as captured in the geodesic manifold distances between all pairs of data points. It also retains the notion of temporal coherence, which is critical to the resulting output for cartoon data. We are using segmented cartoon images to learn the lower-dimensional manifold that parameterizes all of the images of a particular cartoon character. Each segmented image of a cartoon character represents one data point, and a manifold is learned for each character. ST-Isomap uses an algorithm similar to Isomap; here we summarize a simplified version of ST-Isomap and refer the reader to [Jenkins and Matarić 2004] for more details:

1. Compute the local neighborhoods based on the distances $D_X(i,j)$ between all-pairs of points i,j in the input space X based on a chosen distance metric (described below).
2. Adjust $D_X(i,j)$ to account for temporal neighbors.
3. Estimate the geodesic distances into a full distance matrix $D_{Iso}(i,j)$ by computing all-pairs shortest paths from D_X, which contains the pairwise distances.
4. Apply Multi-dimensional Scaling (MDS) [Kruskal and Wish 1978] to construct a d-dimensional embedding of the data.

The difference between Isomap and ST-Isomap is in step 2, where the temporal dependencies are accounted for. Here, we simply force the temporal neighbors to remain as part of the matrix D_X

whether or not those neighbors fall within the k spatial neighborhood that is specified. D_{Iso} are the approximate geodesic distances based on calculating the shortest paths between all pairs of data points, while D_X is simply the distances between all pairs of data points using some distance metric.

One issue with Isomap is determining the size of the spatial neighborhoods. If the data is sufficiently dense, Isomap can form a single connected component, which is important in representing the data as a single manifold structure. The connected components of a graph represent the distinct pieces of the graph. Two data points (nodes in the graph) are in the same connected component if and only if there exists some path between them.

Our experimental results found that varying the size of the neighborhood (step 1) will ensure that a single connected component is formed (one manifold) regardless of the sparseness of the data. However, depending on the distance metric used and the sparseness of the data, the spatial neighborhoods would need to be increased to a point such that no meaningful structure will be found. This issue arises with Isomap since its main objective is in preserving the global structure and preserving the geodesic distances of the manifold. ST-Isomap, by including adjacent temporal neighbors, remedies this deficiency, allowing a smaller spatial neighborhood size while forming a single connected component. Having all of the data points in the same embedding is desirable for re-sequencing. Using from one to three temporal neighbors and a small spatial neighborhood results in a meaningful structure that is usable for re-sequencing.

V.2 Distance Metrics

The key to creating a good lower-dimensional embedding of our data is the distance metric used to create the input to Isomap. When computing the local neighborhoods for $D_X(i,j)$, we examined three different distance metrics: the L_2 distance, the cross-correlation between pairs of images, and an approximation to the Hausdorff distance [Huttenlocher et al. 1993]. As mentioned previously, video textures uses the L_2 distance for computing the similarity between video frames. Although this works well for densely sampled video, it is insufficient for dealing with sparse cartoon data.

V.2.1 L_2 Distance

The first distance metric is the L_2 distance between all-pairs of images. Given two input images I_i and I_j:

$$d_{L2}(I_i, I_j) = \sqrt{\|I_i\|^2 + \|I_j\|^2 - 2 * (I_i \cdot I_j)} \tag{V.1}$$

Only the luminance of the images is used for the L_2 distance. The distance matrix $D_{L2}(i,j)$ is created such that

$$D_{L2}(i,j) = d_{L2}(I_i, I_j) \tag{V.2}$$

This metric is simple and works well for large data sets with incremental changes between frames, but is unable to handle cartoon data, which is inherently sparse and contains exaggerated deformations between frames.

V.2.2 Cross-Correlation Distance

The second distance metric is based on the cross-correlation between a pair of images. This metric also uses only the luminance of the images. Given two input images I_i and I_j:

$$c_{i,j} = \frac{\sum_m \sum_n (I_{i_{mn}} - \bar{I}_i)(I_{j_{mn}} - \bar{I}_j)}{\sqrt{(\sum_m \sum_n (I_{i_{mn}} - \bar{I}_i)^2)(\sum_m \sum_n (I_{j_{mn}} - \bar{I}_j)^2)}} \tag{V.3}$$

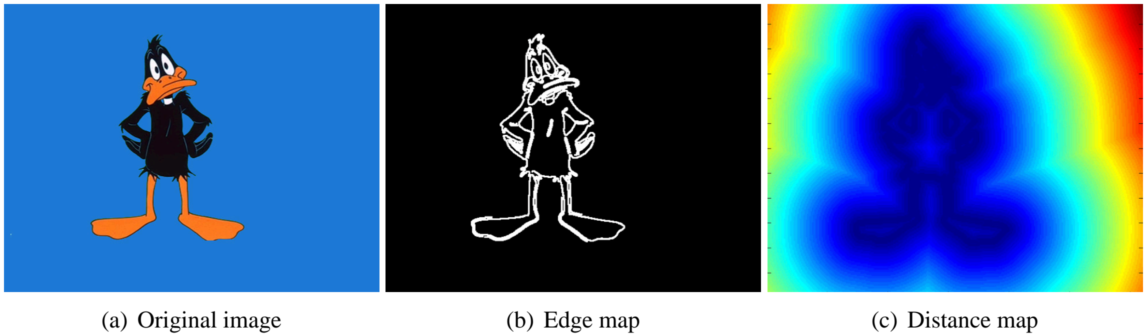
where $\bar{I}_i$ and $\bar{I}_j$ are the mean values of I_i and I_j respectively. This equation gives us a scalar value $c_{i,j}$ for the correlation coefficient between image I_i and image I_j in the range $[-1.0, 1.0]$. However, we want the correlation-based distance metric to be 0.0 for highly correlated images and 1.0 for anti-correlated images. Therefore the correlation-based distance matrix between images I_i and I_j is $D_{corr}(i,j) = (1.0 - c_{i,j})/2.0$.

V.2.3 Hausdorff Distance

The third distance metric is an approximation to the Hausdorff distance. This metric uses an edge map and a distance map of each image. The edge map E is computed using a standard Canny edge detector [Canny 1986]. The distance map X is the distance transform calculated from E, and represents the pixel distance to the nearest edge in E for each pixel in X. Then, the Hausdorff distance between a pair of images I_i and I_j is:

$$D_{Haus}(i,j) = \frac{\sum_{(x,y) \in E_i \equiv 1} X_j(x,y)}{\sum_{(x,y) \in E_i \equiv 1} E_i(x,y)} \tag{V.4}$$

where E_i is the edge map of image I_i, X_j is the distance map of image I_j, and (x,y) denote the corresponding pixel coordinates for each image. Figure V.1 shows an example of the edge map and distance map for a single image, note that the distance computation is between a pair of images and is done for all pairs of images in the data set.



(a) Original image (b) Edge map (c) Distance map

Figure V.1: An edge map in the center, and distance map on the right, for one image from the *Daffy Duck* data set. The edges on the edge map have been enhanced for easier viewing. These images represent only a single image in the data set. The edge map and distance map are computed for every image in the sequence. ^{TM}& ©*Warner Bros. Entertainment Inc.*

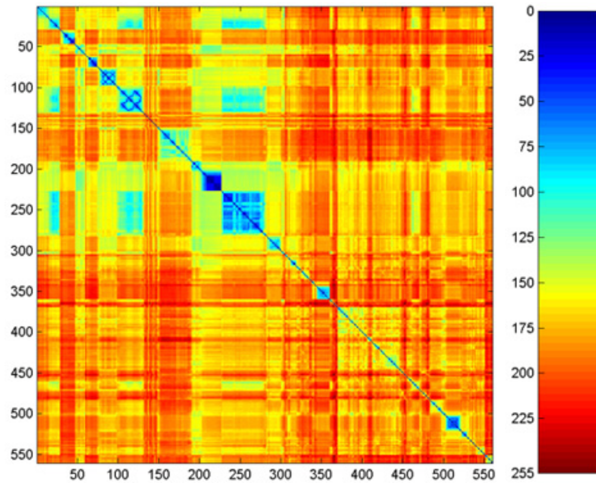
Figure V.2 shows an example of the L_2, correlation-based distance and the Hausdorff distance matrices for the *Daffy* data set. Each distance matrix has been normalized to have values in the range of zero to 255 for visual comparison, while the color map assigns blue to zero and red to 255. A value of zero corresponds to similar images, while a value of 255 corresponds to dissimilar images. Note that the diagonal is zero as expected, and the banding indicates structure in the data. It is that structure that will become important when generating the lower-dimensional manifold for each data set. Since the geodesic distances are estimated by computing the shortest cost paths between all pairs of points, identifying the distance metric that best characterizes the similarity of the images is important. By using the each of the different distance metrics, generating lower-dimensional manifolds of each data set, and creating a few example re-sequenced animations, we found that the Hausdorff distance metric works best for all data sets. Figure V.3 shows the resulting Hausdorff distance matrices for all data sets. The important thing to note from these figures is that while there is much similarity in the structure of the motion when considered as a function of the distance metric, there are important differences too.

V.3 Embedding

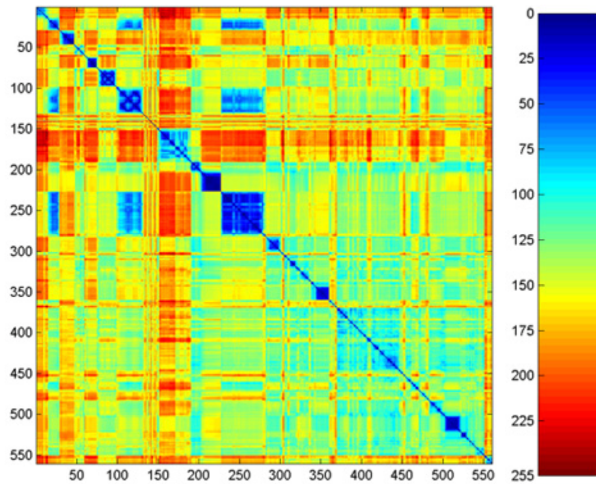
Once the distance matrix for a data set is computed, we apply ST-Isomap to obtain the lower-dimensional embedding of a manifold that parameterizes the cartoon data. The dimensionality of the manifold must be determined. Choosing a dimensionality too low or too high results in incoherent re-sequencing.

Estimating the true dimension of the data using ST-Isomap is different than with PCA. In PCA, picking the dimension of a reduced data set can be done automatically such that the proportion of variance (shape variations) retained by mapping down to n-dimensions can be found as the normalized sum of the n-largest eigenvalues. This residual variance is typically chosen to be greater than 80% (usually 90%), while the remaining variance is assumed to be noise. PCA seeks to maximize the principal shape variations in the data, while minimizing the error associated with reconstructing the data from the lower-dimensional representation. The intrinsic dimensionality of the data estimates the lower-dimensional subspace where the high-dimensional data actually "lives."

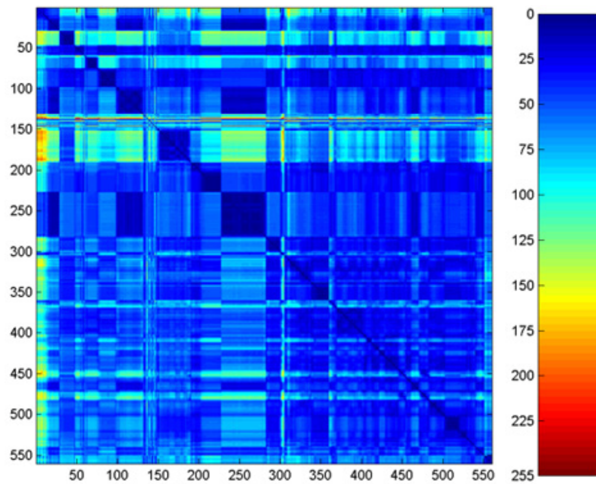
In ST-Isomap, the residual variance is computed using the intrinsic manifold differences, which take into account possible nonlinear folding or twisting. Restated, the residual variance is the correlation coefficient between the embedded distances (the distances between data points on the manifold) and the estimated geodesic distances. We pre-select the number of dimensions in which to embed the data, from one to ten dimensions, and the residual variance is calculated for each dimension. The true dimensionality of the data can be estimated from the decrease in residual variance error as the dimension of the manifold is increased. We select the "knee" of the curve by simply eye-balling the curve. There are statistical methods for selecting the knee of the curve similar to ones used for PCA [Jolliffe 1986], but we believe that simply eye-balling the curve gives us an appropriate lower-dimensional manifold with which to traverse for re-sequencing. Figures V.4 to V.9 show the residual variances and the two-dimensional or three-dimensional projections of the neigh-



L_2



Correlation-Based



Hausdorff

Figure V.2: L_2, Correlation-based and Hausdorff distance matrices for the *Daffy* data set. Each distance matrix has been normalized to the range of zero to 255 only for visual comparison. The colormap is shown on the right.

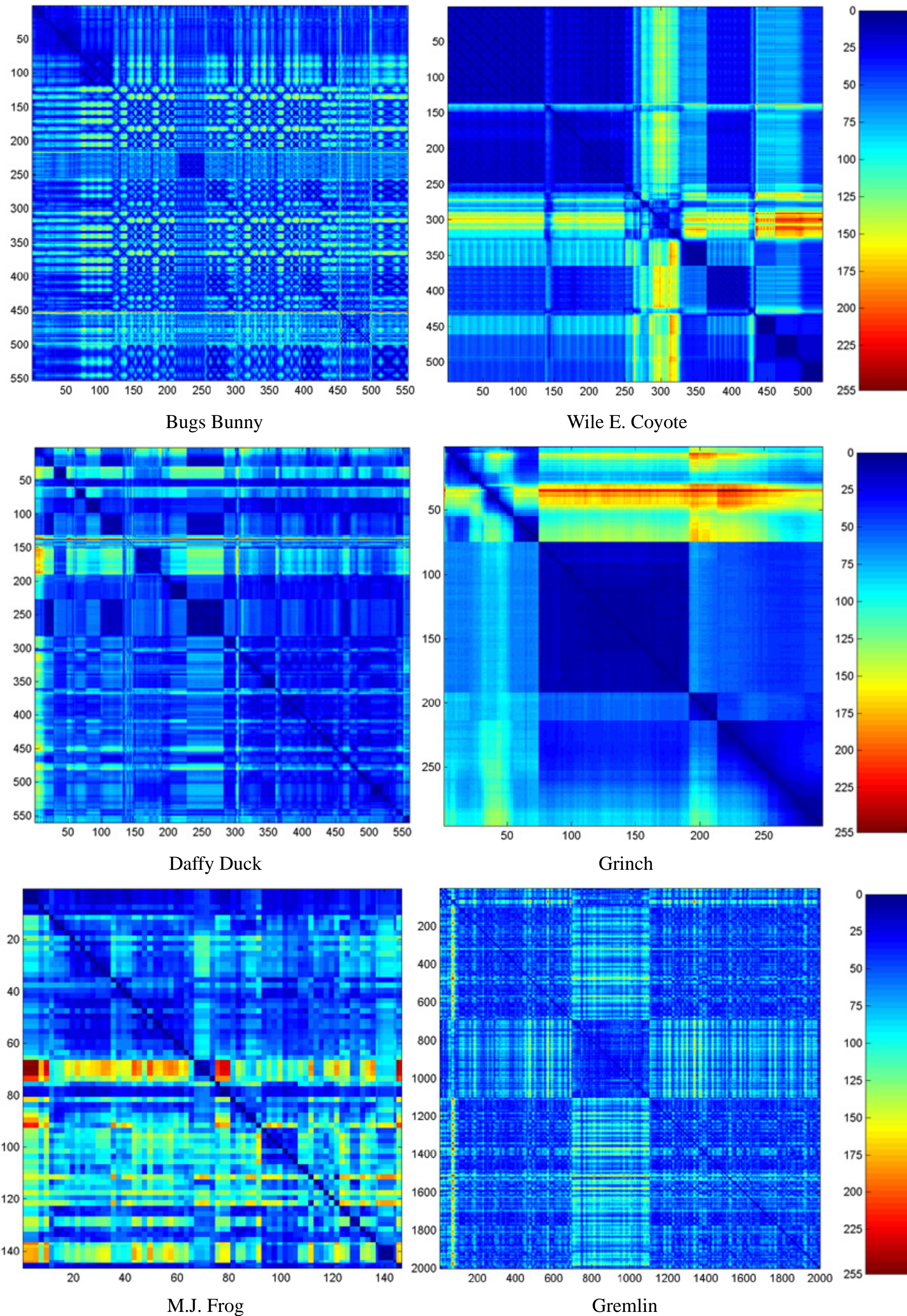


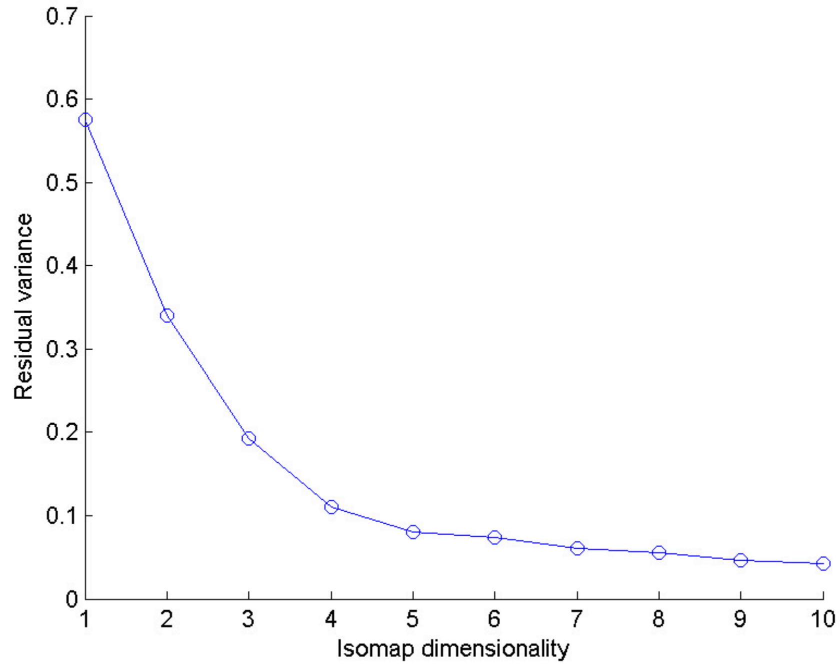
Figure V.3: Hausdorff distance matrices for all cartoon data sets. Each distance matrix has been normalized to the range of zero to 255 only for visual comparison. The colormap is shown on the right.

borhood graphs for all the data sets. The neighborhood graphs represent the manifold structure of the data. Notice that the *gremlin*, *Bugs*, and *Daffy* data sets are reduced to about four or five dimensions (figure V.9, figure V.4, figure V.5, and figure V.6 respectively), as indicated by the variance plots. The *Coyote* and *Grinch* data sets can be reduced down to a three-dimensional manifold (figure V.8). The *Frog* data set is very sparse, and can at best be reduced to a five-dimensional manifold. It is difficult to see structure in the four- and five-dimensional manifolds that have been projected down to a three-dimensional graph. However, the three-dimensional projection of the *Grinch* data (figure V.8) gives a good representation of what the true three-dimensional manifold looks like. Examining the data more closely, there are three scene cuts in the *Grinch* data. The large loop structure on the left in figure V.8(b) depicts the first scene where *Grinch* rotates his head around 360°, the small loop on the top and the branch to the right of the loops represent the next two scenes where the *Grinch* is speaking and making exaggerated facial gestures. This kind of grouping based on the scenes the images come from is not uncommon. In the *Coyote* manifold, represented by a three-dimensional projection of the three-dimensional manifold, the clustering of images from the same scenes remains. The cluster on the lower right side of the manifold in figure V.5(b) are images predominantly from the Coyote-1 segment where the *Coyote* is crawling then sitting and eating a fly. The lower cluster in the center is the *Coyote* climbing on a rock, the loop on the right represents the cyclical motion of the *Coyote* walking, and has images from both Coyote-1 and Coyote-2 data sets. The upper cluster in the center represents images in the Coyote-3 data set where the character is standing scratching his head and making facial expressions. We expect to see these loops when the motion is a cyclical motion such as walking or running. In figure V.4(b), although the four-dimensional manifold is represented as a three-dimensional projection, we can still see similar loop structures of cyclical motion. The large loop on the left represents the repetitive and cyclic motion of *Bugs Bunny* performing a samba dance.

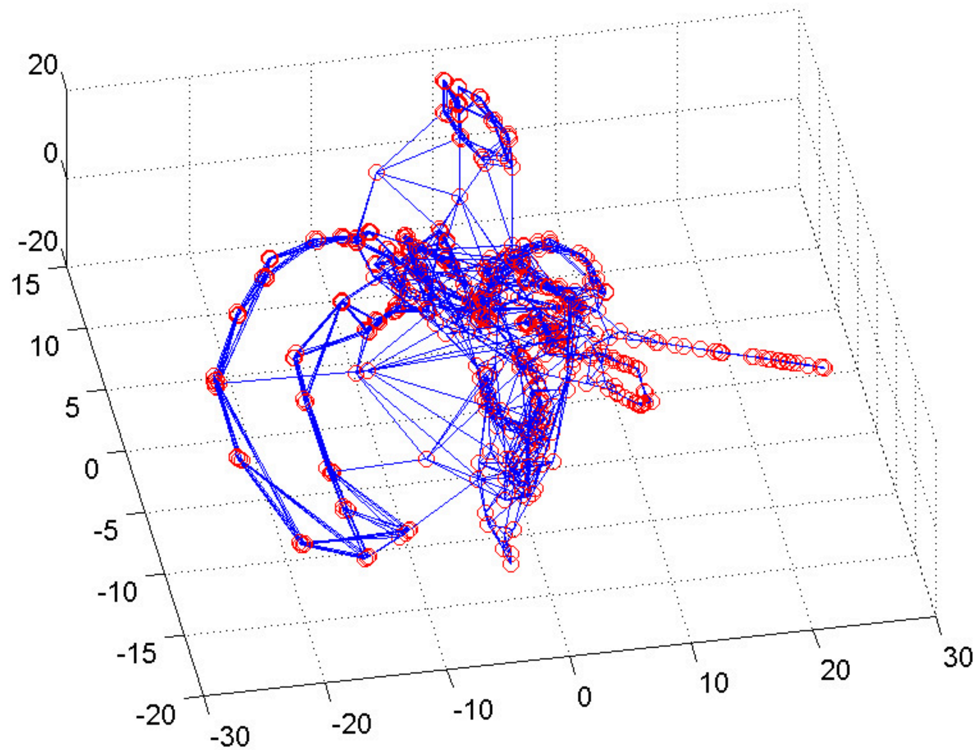
For all of the manifolds shown in figures V.4 to V.9, the number of temporal neighbors is noted in the figures. Most of the data sets use a spatial neighborhood of seven, with the exception of the *Coyote* data set, which uses only three spatial neighbors. Varying the number of spatial neighbors changes how many similar looking images are clustered together, but can also affect the manifold negatively if the number is too high. In that case, images that are not very similar could be grouped into the same spatial neighborhood because of the fixed neighborhood size. Setting the spatial neighborhood too small may cause more than one connected component to be embedded. This means that the data is split among two or more manifolds, and re-sequencing between the different manifolds becomes another problem. To simplify the re-sequencing, we require a single embedded manifold from which to traverse and generate new animations.

V.4 Re-sequencing New Animations

To generate a new animation, the user selects a start frame and an end frame, and the system traverses the manifold to find the shortest cost path. In traversing the manifold, some temporal information has been preserved, but cyclical data such as walking, groups similar images together.

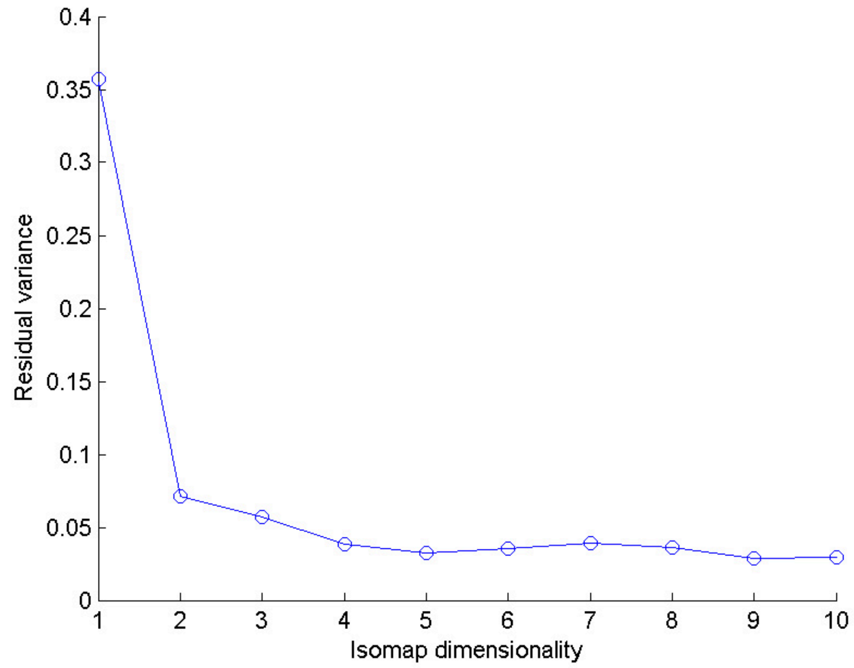


(a) ST-Isomap with two temporal neighbors, variance plot

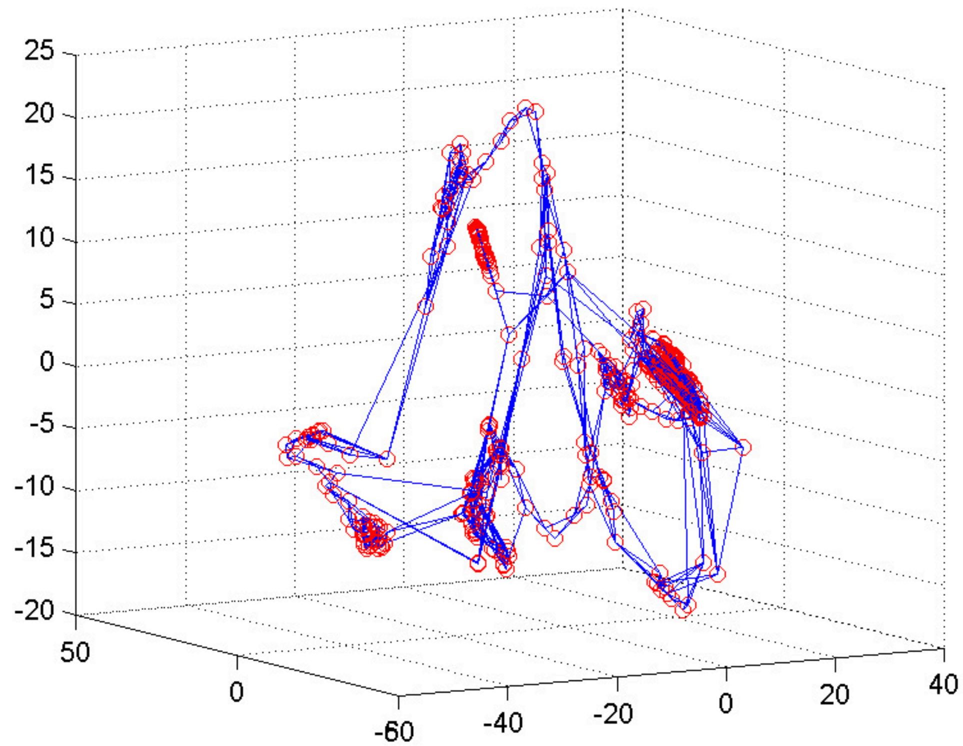


(b) ST-Isomap with two temporal neighbors, 3D graph

Figure V.4: Results showing the residual variance and three-dimensional projection of the neighborhood graph generated with ST-Isomap using the Hausdorff distance matrix on the *Bugs Bunny* data set.

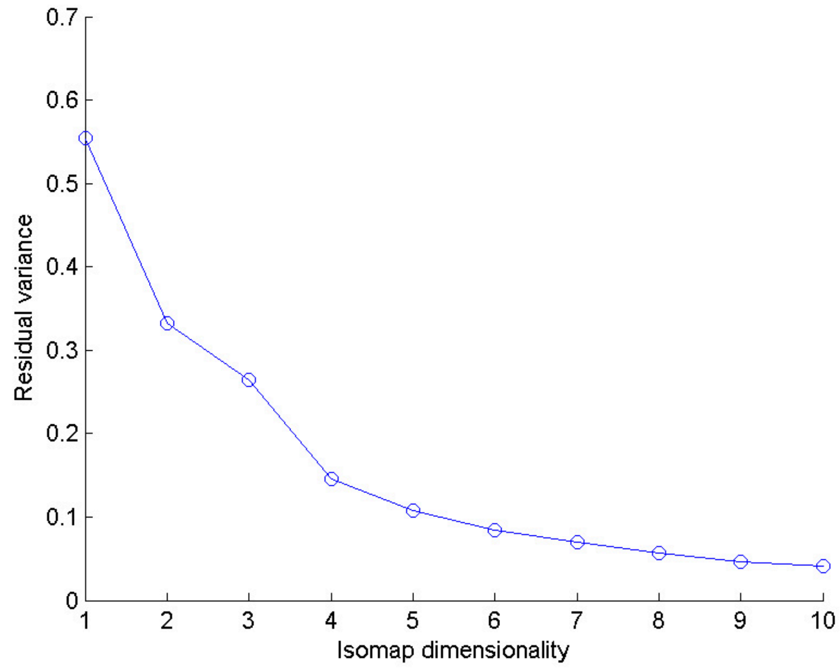


(a) ST-Isomap with two temporal neighbors, variance plot

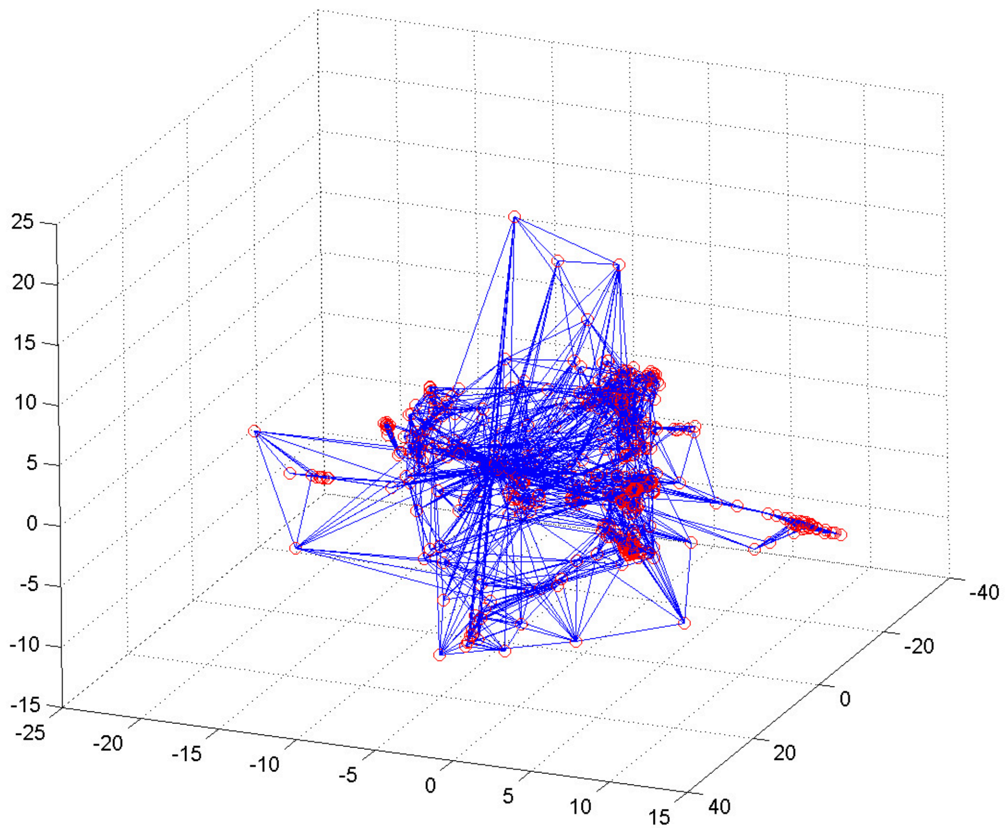


(b) ST-Isomap with two temporal neighbors, 3D graph

Figure V.5: Results showing the residual variance and three-dimensional projection of the neighborhood graph generated with ST-Isomap using the Hausdorff distance matrix on the *Wile E. Coyote* data set.

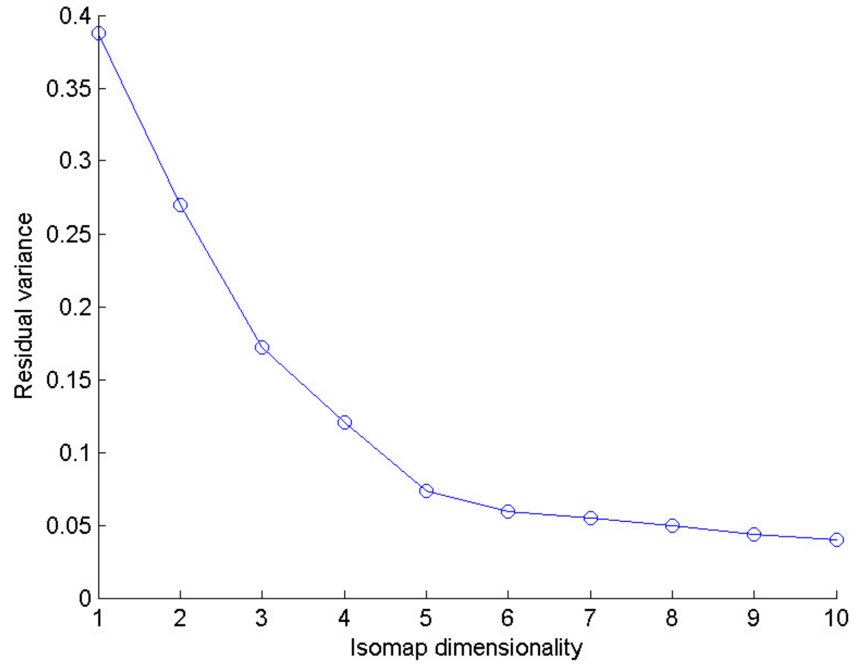


(a) ST-Isomap with two temporal neighbors, variance plot

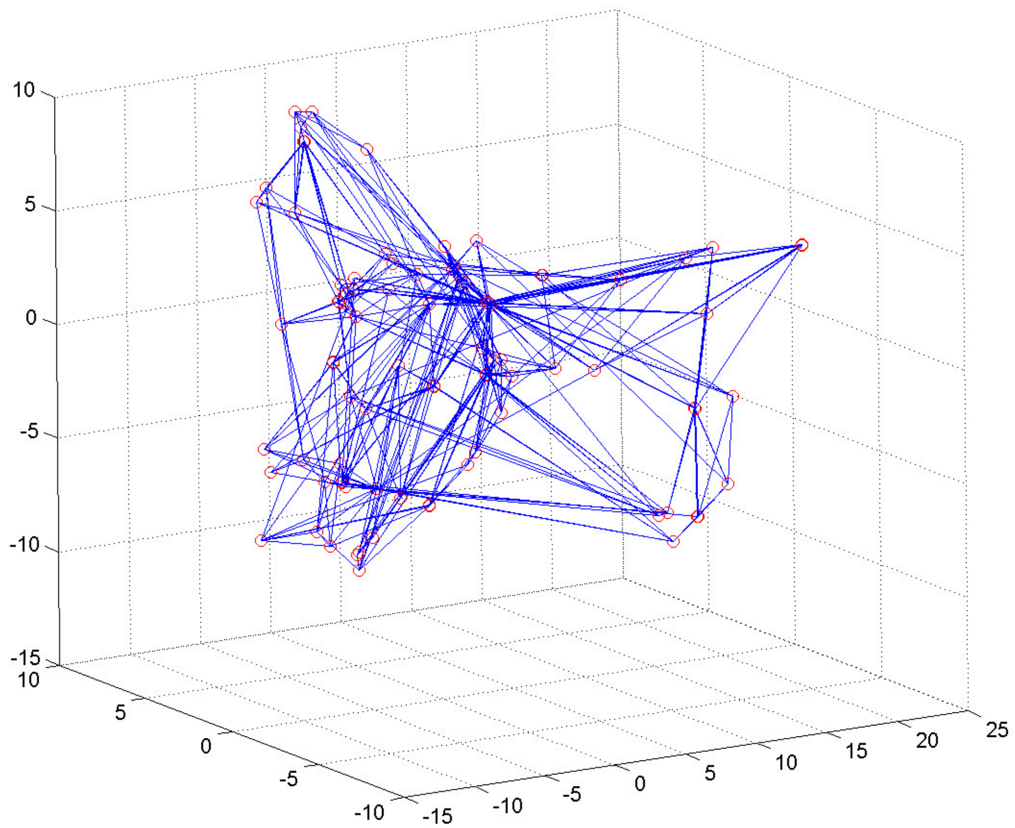


(b) ST-Isomap with two temporal neighbors, 2D graph

Figure V.6: Results showing the residual variance and three-dimensional projection of the neighborhood graph generated with ST-Isomap using the Hausdorff distance matrix on the *Daffy Duck* data set.

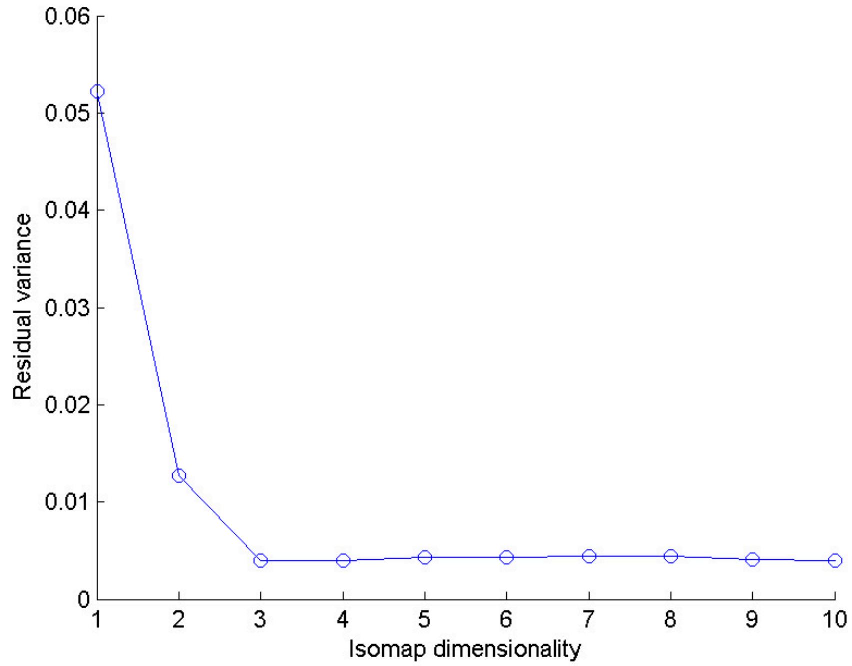


(a) ST-Isomap with three temporal neighbors, variance plot

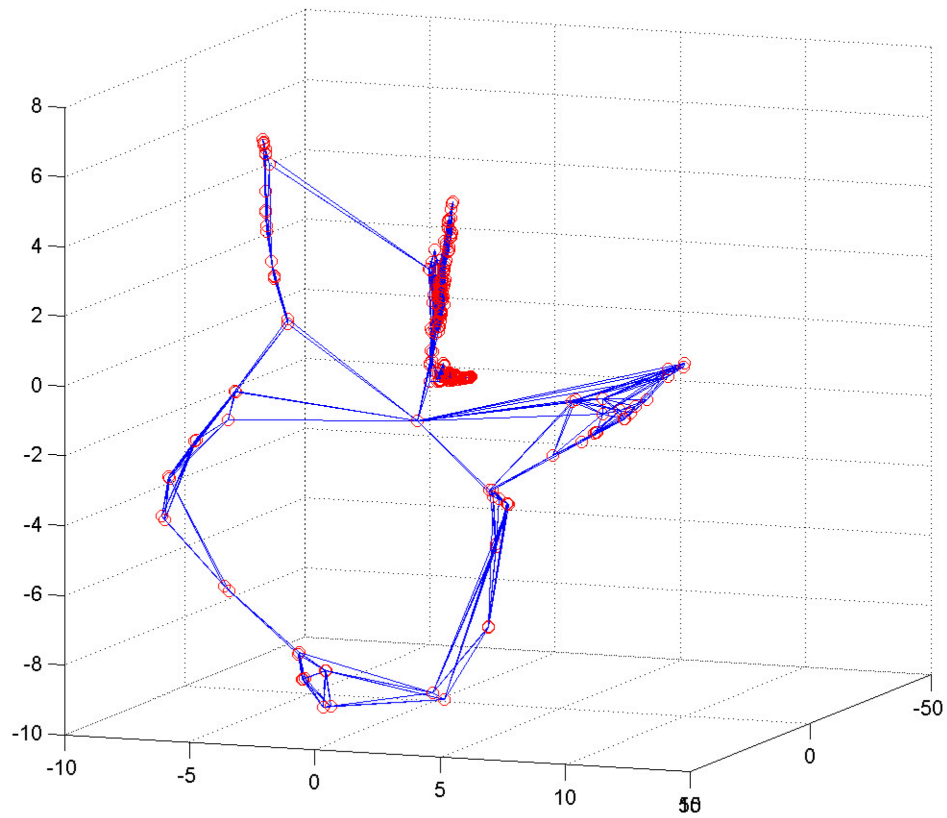


(b) ST-Isomap with three temporal neighbors, 2D graph

Figure V.7: Results showing the residual variance and three-dimensional projection of the neighborhood graph generated with ST-Isomap using the Hausdorff distance matrix on the *Michigan J. Frog* data set.

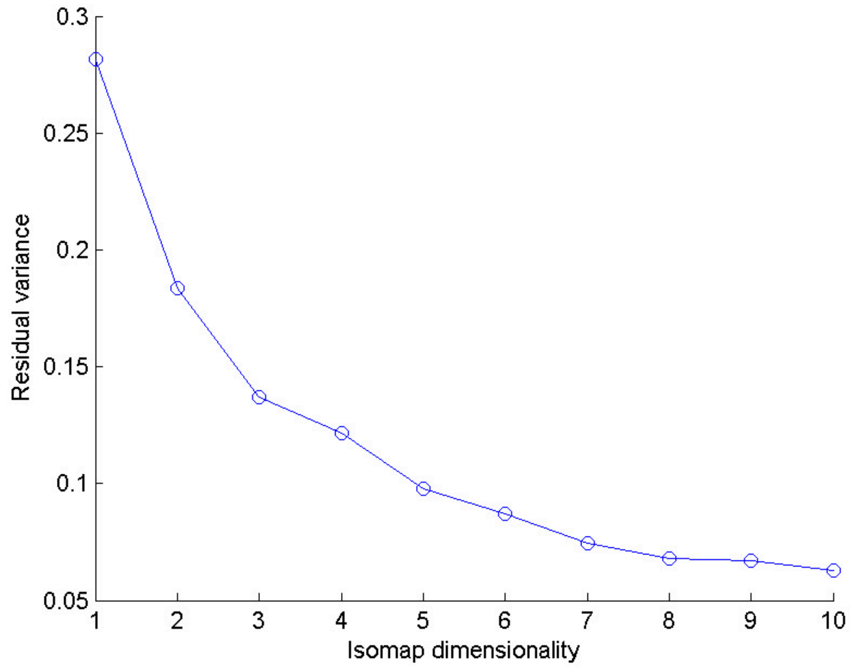


(a) ST-Isomap with two temporal neighbors, variance plot

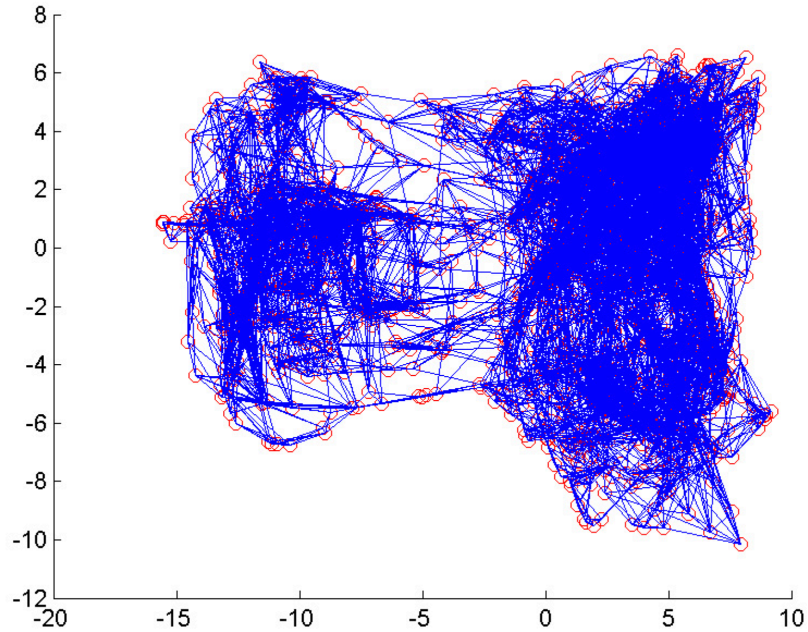


(b) ST-Isomap with two temporal neighbors, 2D graph

Figure V.8: Results showing the residual variance and three-dimensional projection of the neighborhood graph generated with ST-Isomap using the Hausdorff distance matrix on the *Grinch* data set.



(a) ST-Isomap with two temporal neighbors, variance plot



(b) ST-Isomap with two temporal neighbors, 2D graph

Figure V.9: Results showing the residual variance and two-dimensional projection of the neighborhood graph generated ST-Isomap using the Hausdorff distance matrix on the *gremlin* data set.

Also, similar images from different cartoon animations (like the *Coyote* data set) tend to be near each other on the manifold. The traversal path gives the indices of the images used for the new animation, which is created by re-sequencing the original, unregistered images. No new images are generated, only the order of the original images is changed, and images may come from different cartoons. Dijkstra's algorithm [Sedgewick 2002] is used to find the shortest cost path through the manifold. The dimension of the manifold used for re-sequencing varies for each data set. The *Daffy* data set and the *Frog* data set use a five-dimensional manifold, the *gremlin* data set uses a four-dimensional manifold, and the *Grinch* data set uses a three-dimensional manifold.

V.4.1 Post-Processing

To ensure the smoothest looking re-sequenced animations, we add a small amount of automatic post-processing. Only the start and end keyframes for each re-sequenced segment are specified, but currently there are no restrictions on the number of inbetweens that the path should have. As such, the shortest cost path may not visit all temporally adjacent frames in the manifold. To improve the re-sequenced animation, we process the frames specified from the path using the following automatic techniques. First, any missing sequentially adjacent frames within eight frames are inserted, helping to smooth some of the choppiness associated with skipping the missing frames. Sequentially adjacent frames are those that are adjacent in the original sequence. For example, if the re-sequenced path selected is [20 24 60 70] before inserting the sequentially adjacent frames, the resulting path becomes [20 21 22 23 24 60 70]. Using up to eight sequentially adjacent frames does not significantly change the overall re-sequenced path since the temporally adjacent frames are usually near each other in the manifold.

After adding these frames, we further improve the smoothness of the re-sequenced animations by matching the velocity of the centroid of each character from frame to frame in the new path. The new sequence was found based on the Hausdorff distance metric using character aligned images, as described in Chapter IV.1. The aligned images thus no longer possess any offset of the character within the frame. In post-processing, the original images are used. For each original image in the data set, the character's centroid is calculated and stored. Then a velocity vector is computed based on each frame's previous and next temporal neighbor in the original (not aligned) sequence. When given a path for re-sequencing, the position and velocity of the centroid for the character in every frame are known. The position of the character is adjusted from one frame to the next in the new sequence based on the projected position indicated by the first frame's velocity vector from the original sequence. Figure V.10 illustrates the repositioning of the character in image space using the pre-computed velocity vector. This adjustment is done whenever the path jumps from one single frame or subsequence in the path to another. Subsequences in the path are handled such that the first frame in the subsequence has its character repositioned based on the previous frame's projected position, while the remaining frames in that subsequence are adjusted to the first frame's new position.

Finally, if the character translates along the z-axis then the figure often changes in size within

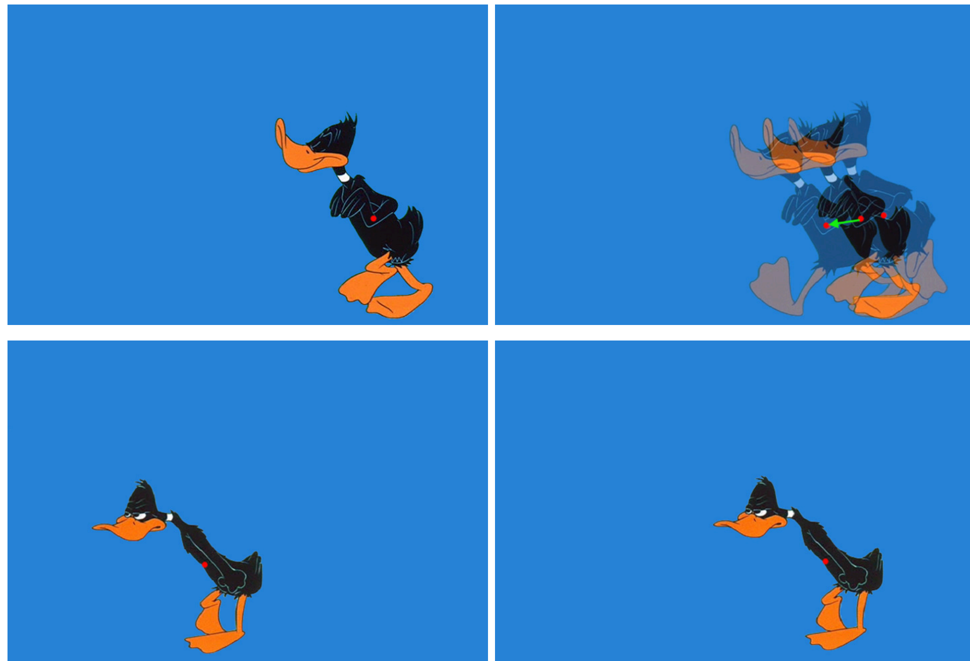


Figure V.10: Illustration of applying a pre-computed velocity vector from an original sequence to a re-sequenced frame. Top left shows a frame in a re-sequenced animation, bottom left shows the next frame in the re-sequenced animation. Top right illustrates the velocity vector computed for the frame in the top left. Bottom right shows the realignment of the character from the second re-sequenced frame. *Daffy* is ^{TM} & ©*Warner Bros. Entertainment Inc.*

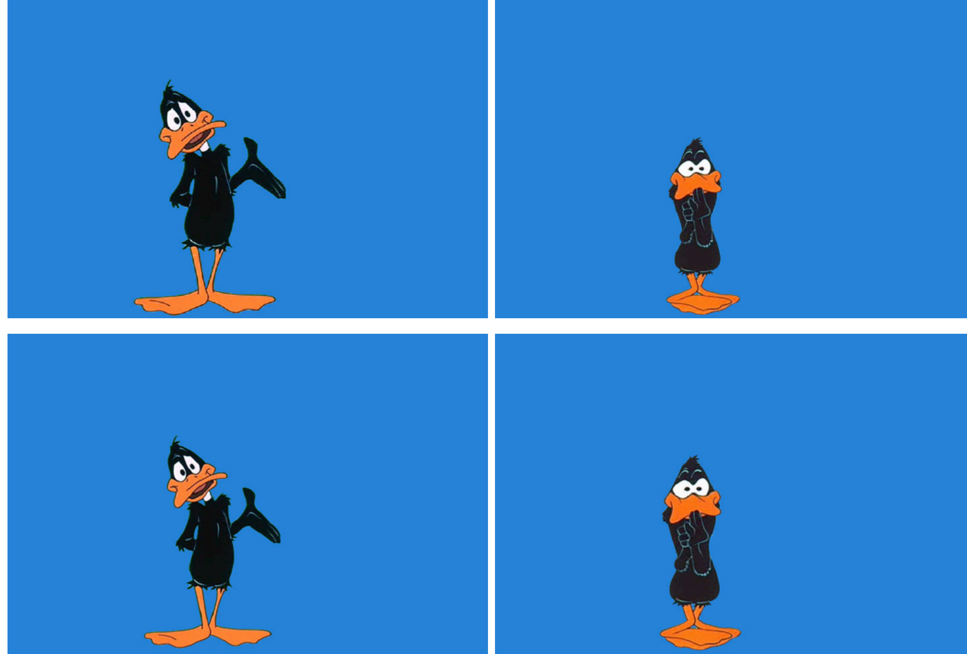


Figure V.11: Illustration of applying a scale factor s to a pair of re-sequenced frames. The top row shows the original frames that appear in a re-sequenced animation. The difference in their scale is apparent. The bottom row shows the same frames after the scale factor is applied. *Daffy* is ^{TM} & ©*Warner Bros. Entertainment Inc.*

the frame. The final re-sequenced frames are adjusted using a scale factor based on the average pixel area in the sequence. By pixel area, we mean the total number of pixels comprising the character. The scale factor s is defined as

$$s = \sqrt{\frac{A_{ave}}{A_{seq}}} \tag{V.5}$$

where A_{ave} is the average pixel area in the entire path, and A_{seq} is the average pixel area of a subsequence (or just the pixel area of a single frame). Then s is applied to each frame of the subsequence (or single frame) in the path. Figure V.11 illustrates the scale factor s applied to a re-sequenced frame.

V.5 Threshold Detection

In re-sequencing cartoon data, the transitions from the shortest cost path may result in visual discontinuities. A small cost (embedded distances on the manifold) indicates a good transition, while a large cost indicates a bad transition. The system can automatically identify when the cost of a transition is too large. A threshold is determined for each data set, and notifies the user of abrupt transitions in the re-sequenced animation. The threshold is currently determined manually for each data set by examining the manifold and all associated transition costs. Notification allows the user to decide if additional source material is needed to produce a more visually compelling sequence.

V.6 Re-sequencing Results

After testing all of the data sets by generating lower-dimensional manifolds for each and varying the number of temporal neighbors, using two temporal neighbors yielded the best re-sequencing results. The *gremlin* data set is well populated with only a few large jumps at the transitions between motion capture clips, but the Hausdorff distance metric is an improvement over the L_2 distance. For the *Bugs*, *Daffy*, and *Coyote* data sets, there are also a few large jumps in the original data resulting from scene cuts and images from different cartoons. The Hausdorff distance metric works significantly better than the L_2, and reasonable paths are found through the lower-dimensional manifold for each data set.

We are able to re-sequence the *gremlin* data into a short motion clip that retains the same characteristics of the original dance motion, but shows a new dance behavior. This result was achieved by selecting six keyframes (sets of start and end frames) and applying ST-Isomap with two temporal neighbors, and post processed as described in Chapter V.4.1. The result is a sequence with a total of 57 frames.

We also re-sequence the *Daffy* data into two short motion clips, each retaining the original characteristics of the gesturing motion, but showing a new gesturing behavior. The clips were created by selecting six and seven keyframes and applying ST-Isomap with two temporal neighbors. The first clip was minimally post-processed, only the missing temporally adjacent frames were inserted, and resulted in a sequence with a total of 59 frames. The second clip was post-processed by including any missing temporally adjacent frames and velocity-matching the centroids, resulting in a sequence with a total of 98 frames.

Table V.1: Examples of the distance values between pairs of frames using the Hausdorff distance metric on the *Daffy* data set. Adjacent frames in the original data set may not always have a low distance value, as shown in the table. The transition from frame 98 to 99 is an abrupt transition according to the distance metric.

Daffy	246 → 235	0.413511	good
Daffy	326 → 77	6.173898	bad
Daffy	99 → 243	3.010666	accept
Daffy	235 → 236	0.094055	good
Daffy	98 → 99	7.270829	bad

After generating several re-sequenced animations for a particular data set, we inspect the cost values associated with the transitions and determine a threshold value for abrupt transitions. Those cost values come from the embedded distances in each lower-dimensional manifold, represented here as D_{emb} Once the threshold is determined, the system can use threshold detection to indicate to the user when a large transition cost has occurred. As an example using the *Daffy* data set, our findings indicate that a threshold value of $D_{emb} < 2.2$ represents a good transition while $D_{emb} > 3.9$ represents an abrupt transition. Table V.6 shows some of the distance values associated with the transitions for a re-sequenced animation, while Figure V.12 shows the frames referred to in the

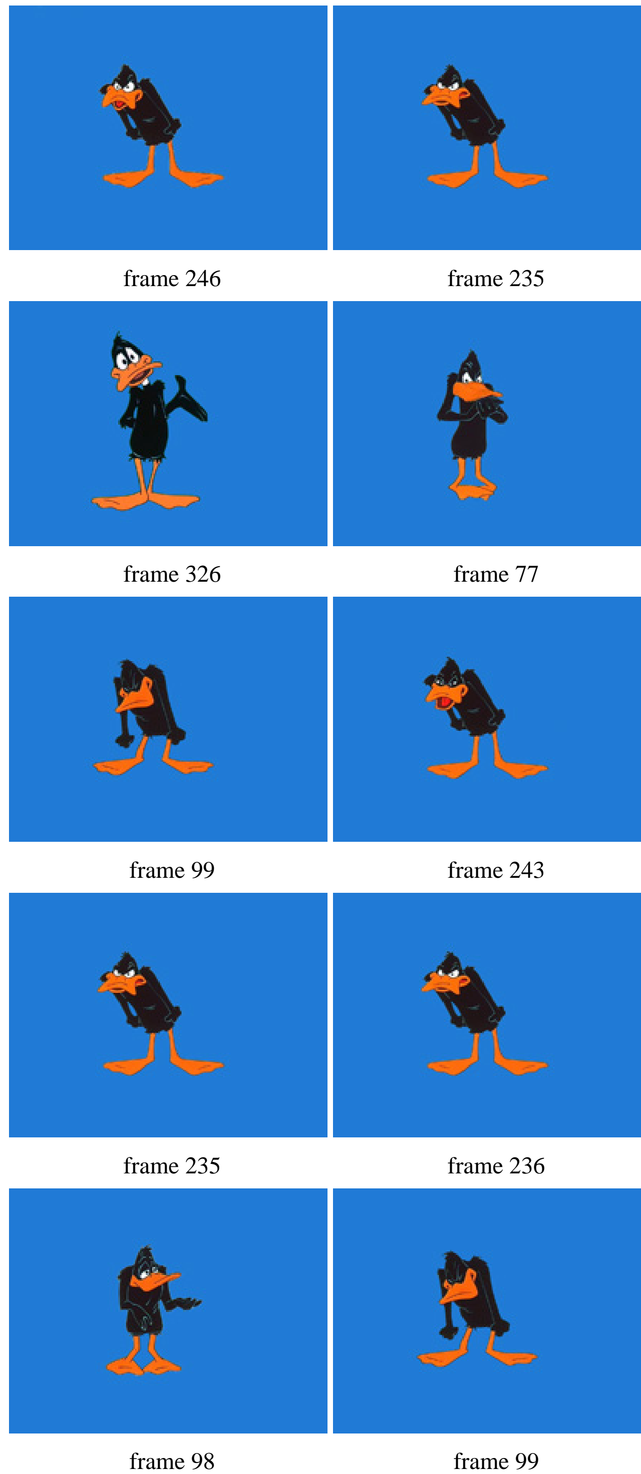


Figure V.12: An example of good, bad, and acceptable transitions for the *Daffy* data set from a path generated using ST-Isomap with two temporal neighbors. The pairs of frames shown correspond with the values shown in Table V.6. *Daffy* is ™ & ©*Warner Bros. Entertainment Inc.*

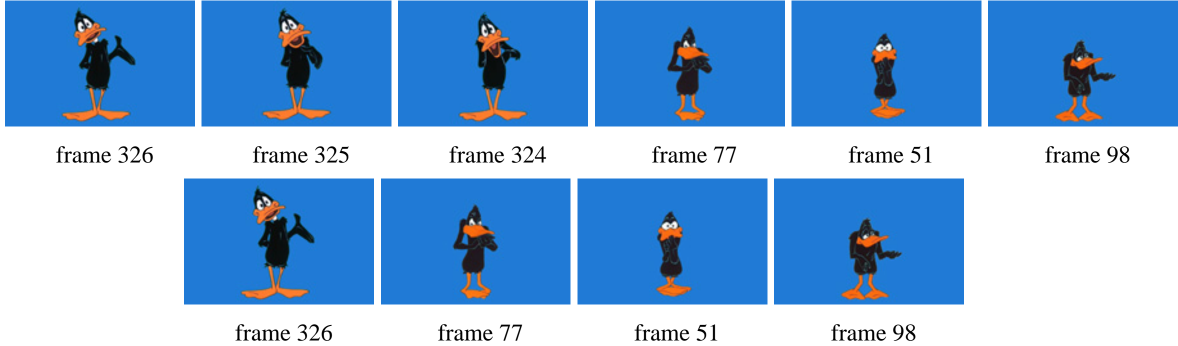


Figure V.13: A filmstrip of two paths without any post-processing. The bottom row shows the path generated from the *Daffy* data set with three frames removed. The top row shows the same path with inbetweens inserted at the point of highest transition cost, in this case between frames 326 and 77. *TM & ©Warner Bros. Entertainment Inc.*

table. The transition from frame 99 to 243 has a value $2.2 \leq D_{emb} \leq 3.9$, representing a region that should be inspected by the animator before accepting or rejecting. In this case it is accepted.

To test the system's ability to detect a large transition, an example is generated with three images from the *Daffy* data set removed. ST-Isomap is applied using two temporal neighbors and seven spatial neighbors. In the path generated from the data set with missing frames, the transition cost exceeded the pre-set threshold and resulted in a sequence with visual discontinuities. Inserting inbetween frames at the point of highest transition cost generates an improved sequence. Figure V.13 shows the two paths without any post-processing. The sequence generated from the data set with missing images differs from the other sequence only in the transition from the first frame 326 to the second frame 77, which is where the inbetweens were added.

The *Michigan J. Frog* data set illustrates the challenges in re-sequencing cartoon data. This data set has 146 frames, of which only 73 are unique. Although ST-Isomap can reduce the data to approximately five dimensions, traversing the resulting five-dimensional manifold for re-sequencing yields jumpy motion. A transition threshold can still be found even though the data set is so sparse. A threshold value $D_{emb} \leq 0.58$ represents a good transition. Figure V.14 shows examples of good and bad transitions for the *Frog*, and the corresponding transition costs, for a path generated using ST-Isomap with three temporal neighbors to create the manifold.

V.7 Summary

We foresee that this system of re-sequencing, and specifically the ability to identify visual discontinuities automatically, will be useful as an aid to artists charged with generating inbetweens in traditional animation. If a sufficient body of prior animation is available, the inbetween artist could use the system to match keyframes in a new animation and generate inbetweens from existing data. Only if the keyframes were sufficiently novel or the transition cost too high would the inbetween artist be required to generate new art. By providing a method for re-using cartoon images through

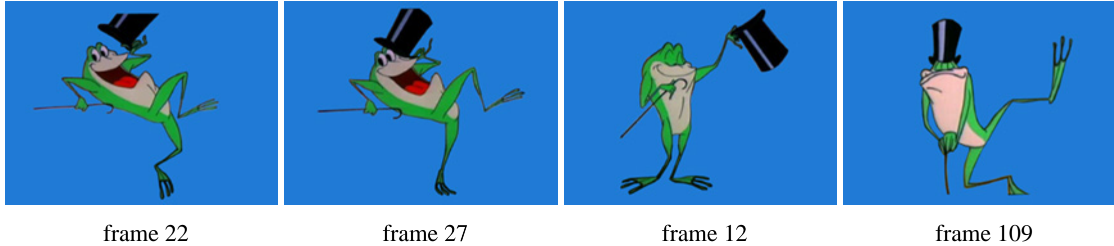


Figure V.14: An example of good and bad transitions for the *Frog* data set. The first pair of images demonstrates a good transition from frame 22 to 27 with a cost of 0.198132. The second pair of images demonstrates a bad transition from frame 12 to 109 with a cost of 0.609729. *TM & ©Warner Bros. Entertainment Inc.*

the use of manifold learning, this chapter presents the second necessary and most important component of a system for building re-usable motion libraries of traditional animation.

Thus far, we have implemented and discussed two research goals: semi-automatic segmentation of existing cartoon images (in Chapter IV), and how to re-sequence cartoon data to create new animations that retain the characteristics of the original motion. Our methods are model-free, i.e., no a priori knowledge of the drawing or cartoon character is required. The keys to the re-sequencing method are the identification of a suitable metric to characterize the differences in cartoon images and the use of a nonlinear dimension reduction and embedding technique, ST-Isomap. The system can characterize when a novel re-sequencing requires additional source material to produce a visually compelling animation. The methods and results discussed in Chapter V have been published [de Juan and Bodenheimer 2004]. In the next chapter, we address the issue of generating new images in the cases of high transition costs, extending the re-sequencing capabilities to go beyond just the existing library of cartoon data.

CHAPTER VI

INBETWEENING A MOTION LIBRARY OF RE-SEQUENCED ANIMATIONS

Several avenues exist for expanding and building upon the re-sequencing cartoon animation work described thus far. One of the largest limitations of the re-sequencing system as described in Chapter V is the inability to generate new images when a visual discontinuity is detected in a re-sequenced animation. Addressing the issue of synthesizing new data leads us back to the two-dimensional inbetweening problem introduced in Chapter II.1.2. In this case, the problem is restricted in that we must generate transitions by blending or interpolating of a pair of images from a re-sequenced path, not a series of keyframes. We refer to those pairs of images as “key images” in this chapter.

One of the most challenging aspects of automating the traditional animation pipeline is inbetweening. In this chapter, we develop methods that allow for the semi-automatic generation of inbetweens for re-sequenced traditional animations. Recall that the principal difficulty with two-dimensional inbetweening is that the drawings are actually two-dimensional projections of three-dimensional characters as visualized by skilled artists, discussed in Chapter III.1, which results in problems of self-occlusion and lack of correspondences. To deal with self-occlusion, [Catmull 1978] suggests partitioning the character into separate layers before processing with a computer program. For the correspondence problem, he suggests the program operator specify the correspondence of the lines and hidden lines of the character from frame to frame. However, even with human intervention, the problems of occlusion and correspondence are still difficult to overcome.

VI.1 Character Partitioning and Re-assembly

To overcome the self-occlusion problem, a pair of key images to be inbetweened are first partitioned into separate character layers. Partitioning the character into layers is done manually, and a semi-automatic inbetweening method (Chapter VI.4) is then applied to each layer. The location and scale of each layer is “lost” in the inbetweening process, so the layers are reassembled after the intermediate images are generated. The layer reassembly is done automatically using the original silhouettes and inbetweened results (contours) as references for calculating the translation and scale factor for each layer. To determine the 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

$$s = \sqrt{\frac{A_{ave}}{A_{tween}}} \tag{VI.1}$$

where A_{ave} is the average pixel area from the key images and A_{tween} is the pixel area of the inbetweened result. By area, we mean the total number of pixels belonging to the character layer. We are using the contour (Chapter VI.4.2) for computing A_{tween}. As such, the contour is filled in to be a silhouette. Figure VI.1 shows a character and the four layers used for inbetweening.

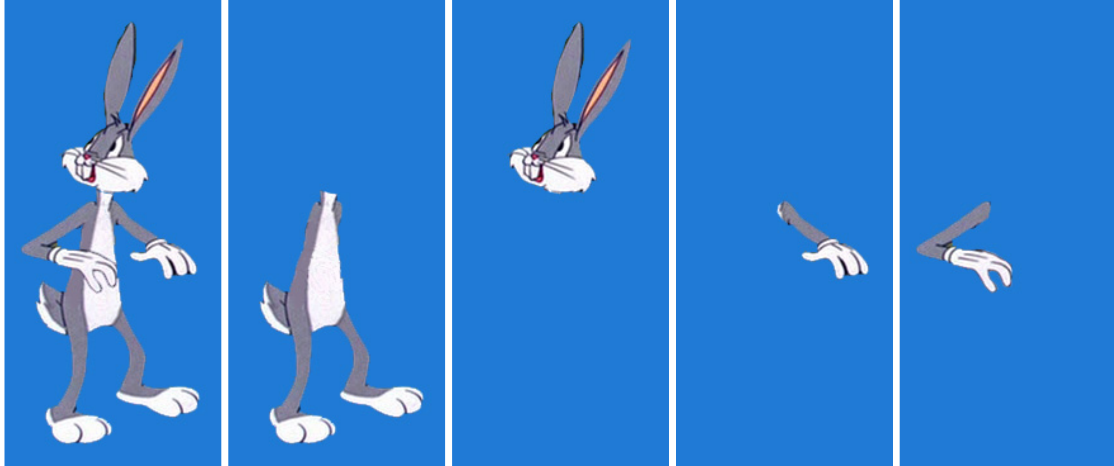


Figure VI.1: Moving from left to right: the first image is an example key image that will be used for inbetweening, the second image is the body layer, the third image is the head layer, the fourth image is the left arm layer, and finally the last image is the right arm layer. The character was partitioned into layers manually. *Bugs Bunny* is ™ & ©*Warner Bros. Entertainment Inc.*

VI.2 Shape and Color Interpolation

There is a well known algorithm that would provide the best approximation to an inbetween, and is a natural choice for shape and color deformation or interpolation. The method of image metamorphosis by [Beier and Neely 1992], discussed in Chapter III.1.4, requires a fair amount of user input and tweaking to generate a reasonable looking inbetween. Figure VI.2 shows the two key images of *Bugs Bunny's* head layer with nine user selected feature lines. Figure VI.3 shows the result of the morphing at intervals of 0.25, 0.5, and 0.75. The morphing algorithm generates these results in minutes, but the selection of appropriate feature lines can be more time consuming, specifically in determining how many are necessary and which features should be highlighted. In the examples shown, determining the feature lines and selecting them for each image was an iterative process that took approximately 25 minutes. Further tweaking results in improved intermediate images, as shown in figure VI.4, but the time spent deciding on which features to select and making small refinements exceeds the time it would take to manually draw an inbetween image. In addition to the time spent tweaking the results, the algorithm is not intuitive to use. However, it is important to note that the results presented here serve as a baseline for the more automated methods we present later in this chapter.

VI.3 Semi-Automatic Inbetweening

Our approach to the inbetweening problem is similar to the shape interpolation approaches of [Beier and Neely 1992; Sederberg and Greenwood 1992; Alexa et al. 2000]. However, it employs radial basis functions to generate implicit models as presented by Turk and O'Brien [Turk and O'Brien 1999] and refined by Carr et al. [Carr et al. 2003]. In our experience, using implicit surfaces presents

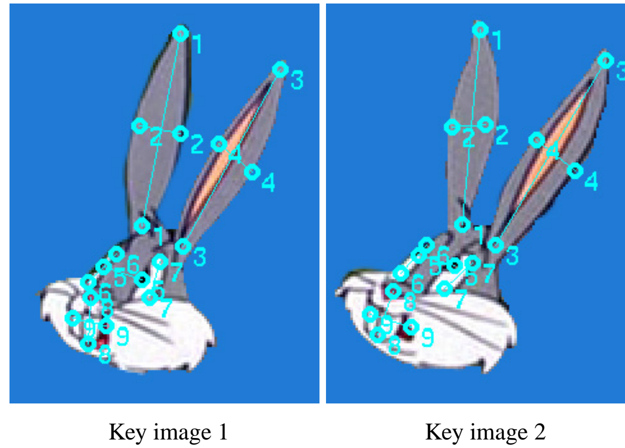


Figure VI.2: Example of nine feature lines selected for image morphing between a pair of key image head layers. *Bugs Bunny* is ™ & ©*Warner Bros. Entertainment Inc.*

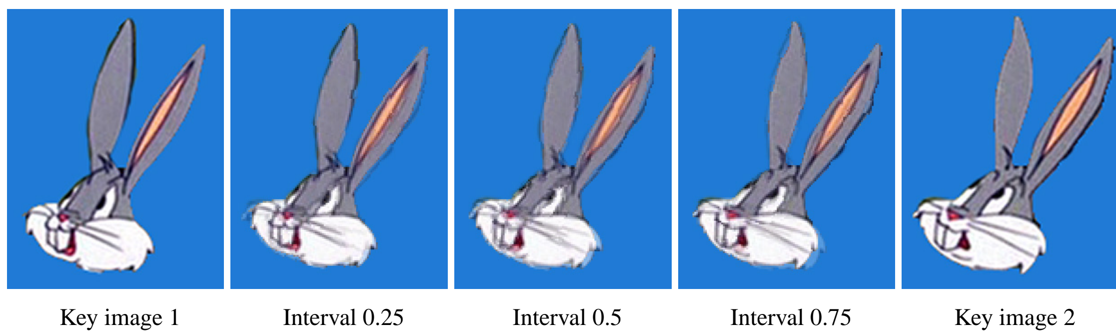


Figure VI.3: Example of image morphing of the head layer between key image 1 on the left and key image 2 on the right. *Bugs Bunny* is ™ & ©*Warner Bros. Entertainment Inc.*

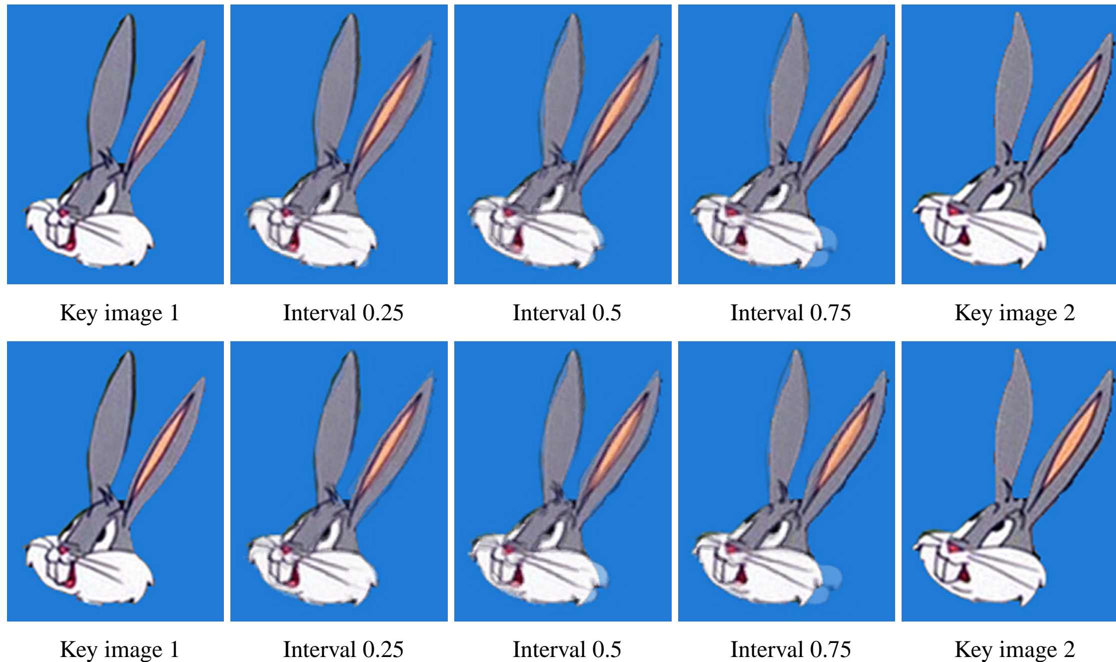


Figure VI.4: Example of image morphing of the head layer between key image 1 on the left and key image 2 on the right, with further refinement of the feature lines. The top row shows results using 24 feature lines, the bottom row shows results using 30 feature lines. *Bugs Bunny* is ™& ©*Warner Bros. Entertainment Inc.*

a superior technique than vertex-based approaches [Sederberg and Greenwood 1992; Alexa et al. 2000], since vertex interpolation often leads to unacceptable deformations in the contours, such as arm shortening. To alleviate the problem of self-occlusion, we employ the partitioning and layering strategy mentioned above. Although the techniques presented here are not fully automatic, they do not require the intervention of a skilled animator for quality results. We use the machinery of [Carr et al. 2003] to generate an implicit model using radial basis functions (RBFs), as we have found it produces better results and is faster than the related methods in [Turk and O'Brien 1999], which provided the inspiration for our technique.

We describe a method that is a combination of two techniques, RBF interpolation and non-rigid image deformation, for creating inbetween images from the existing key images in Chapter VI.4. The key images used for inbetweening are any pair of images that are identified as having a large transition cost, or visual discontinuity, in the re-sequenced animation. The goal in generating new images will be to maintain the model-free representation of the characters being inbetweened, as has been discussed throughout this dissertation. Also, the methods used should remain simple with little to no user input required. Aside from partitioning the character into layers manually, very little user input is necessary to achieve the inbetweening results presented here.

The system for re-using cartoon images described in Chapter V re-sequences animated images by parameterizing the input images with a lower-dimensional manifold and then traversing that

manifold to produce original animation, given start and end data. Examining the geodesic distance determines if additional source material needs to be provided in the form of an inbetween. In this chapter, we extend the re-sequencing method: given a geodesic distance, we compute an inbetween. Once the re-sequenced animation is created and the visual discontinuities are identified, the interpolation to correct the discontinuities are restricted to a pair of images, instead of the traditional animation problem of having a series of keyframes that require a large number of inbetweens. In practice, typically only one or two intermediate frames will be required for smoothing the visual discontinuity in the re-sequenced animation.

VI.4 Shape Interpolation using RBFs

Before describing the details of our method of shape interpolation using radial basis functions (RBFs), a brief overview of the algorithm is presented. RBFs are used to interpolate the outer contours of the character in the images. A mesh is created from the two contours, and slices of that mesh are extracted and becomes the intermediate contours used to create the inbetween. In our experience, using implicit surfaces presents a superior technique than vertex-based approaches [Sederberg and Greenwood 1992; Alexa et al. 2000], since vertex interpolation often leads to unacceptable deformations in the contours, such as arm shortening. Once we have an intermediate contour, that contour must be filled in with color information. Using the two key images as references for the intermediate contour, the images are registered and an intermediate color image is generated. That intermediate image is used to fill in the contour, completing the inbetween. The rest of this section describes the details of the inbetweening algorithm.

VI.4.1 Overview of Radial Basis Functions

We begin with a brief introduction to RBFs and the fast evaluation methods developed by [Carr et al. 2003]. Radial basis functions are techniques used for interpolation in a multidimensional space. RBFs offer a compact functional representation of a set of data points used to define a surface. The functions also have a distance criterion with respect to a center used for evaluation. That makes RBFs useful for interpolation, extrapolation, and smoothing data. RBFs can be evaluated anywhere on the surface to produce a mesh of any specified resolution. An object's surface is defined implicitly as the zero set of RBFs fitted to the given surface data points. The main contribution of [Carr et al. 2003] is in the fast evaluation methods that are able to model large data sets consisting of millions of points with a single function that is continuous and differentiable. Their method involves three steps to fit RBFs to 3D data sets without restrictions on surface topology. The steps are (1) construct a signed-distance function, (2) fit RBFs to the distance function, and (3) iso-surface the fitted RBFs. Essentially, given a set of zero-valued surface points and non-zero off-surface points, fitting the RBFs is a scattered data interpolation problem. These methods are commercially available as a toolbox for Matlab, known as FastRBF.

The general functional form of the RBFs over a set of N data points is given by $s(x) = p(x) + \sum_{i=1}^{N} \lambda_i \phi(|x - x_i|)$ where p is a polynomial of low degree, ϕ is a basis function (a radially symmetric

real valued function on $[0,\infty)$, and x_i are the centers of the RBFs. There are several functions that can be used as the basis function ϕ, such as the thin-plate spline, a Gaussian, biharmonic, and triharmonic splines. [Carr et al. 2003] chose the biharmonic spline function $\phi(r) = r$, which can be rewritten as $s^*(x) = p(x) + \sum_{i=1}^{N} \lambda_i |x - x_i|$ for the RBFs, where p is a linear polynomial, λ_i are real numbers, and $||$ is the Euclidean norm on $\mathbf{R}^3$. The biharmonic spline requires that data points be not co-planar, and there are constraints on the coefficients λ ensuring that the function is second-order differentiable. We also chose the biharmonic spline as the basis functions in our interpolation work described below. The authors claim that non-compactly supported basis functions are better for extrapolation and interpolation of non-uniformly sampled data. Because the biharmonic spline is not compactly supported, direct evaluation of the RBFs using a biharmonic basis function is unreliable and extremely time consuming. Their fast evaluation method allows for the use of the biharmonic spline on data sets consisting of tens of thousands of data points. They use the Fast Multipole Method (FMM), originally developed for the fast evaluation of polyharmonic splines in two and three dimensions. The idea behind the FMM is best described by citing [Carr et al. 2003]: "The FMM makes use of the simple fact that when computations are performed, infinite precision is neither required nor expected. Once this is realized, the use of approximations are allowed." In the evaluation of RBFs, the *approximations* are defined as two parameters, one for fitting accuracy and one for evaluation accuracy. Basically, this provides a boundary for where to evaluate the RBFs, and if the RBF centers are far from an evaluation point then an approximate evaluation is satisfactory. Typically all the input data points are used as RBF centers, and as nodes of interpolation. The fitting accuracy specifies the amount of deviation from a data point to the fitted RBF value, and can be specified for each data point. The evaluation accuracy determines the precision for evaluating the fitted RBFs, and acts as a boundary to the actual evaluated function. Another improvement to speed the evaluation is to reduce the number of RBF centers. Basically, a subset of the interpolation nodes are selected, the residual error of the approximation is computed, and if the residual error is smaller than the fitting accuracy, the RBFs are fitted. By using this method iteratively, although not essential for the fast evaluation method, the surfaces are approximated to within the specified accuracy. The more centers used in the fitting of the RBF, the more closely the zero set surface approximates the entire set of input data points. Once the RBFs are fitted to a set of data points, the RBFs define an implicit model of an object. An explicit representation, such as a triangular mesh, can be extracted using iso-surfacing algorithms like marching cubes.

Having a compact functional representation of a surface, which also has the ability to interpolate and extrapolate, make using RBFs appealing for interpolating between pairs of key images. The points used as input data to be fitted with the RBFs can be the outer contours of the cartoon characters, or all of the pixels of a character, including color values. The fast evaluation method developed by [Carr et al. 2003] provides a simple framework in the form of a toolbox in Matlab, allowing for ease of use and accessibility to test the representation of cartoon data with RBFs.

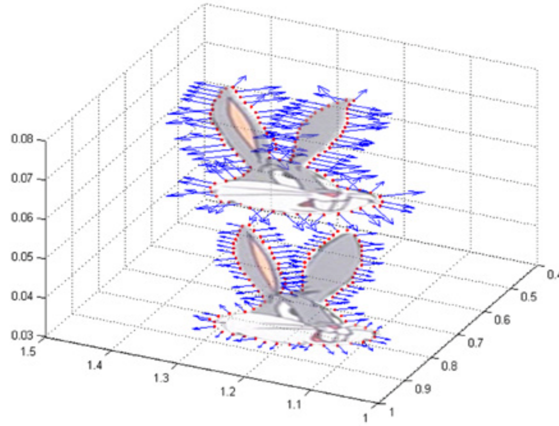
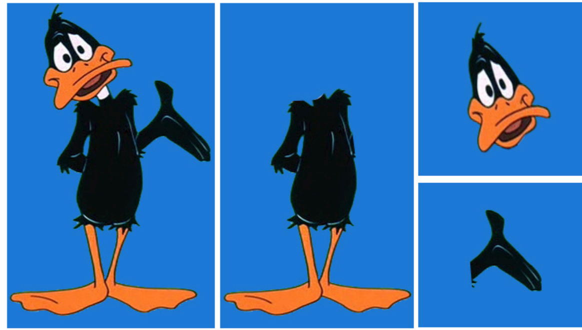


Figure VI.5: An example of the automatically generated contour and normal points which serve as the input to the implicit surface generation step.

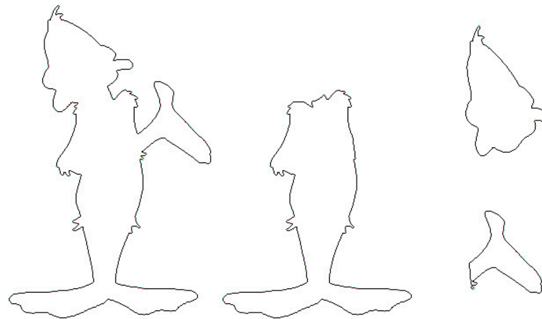
VI.4.2 Interpolating Contours using RBFs

To continue the discussion of using RBFs for inbetweening, we begin with the task of interpolating the outer contours of a pair of images of cartoon characters. Given input data from the previous segmentation and layering operations, the RBF contour inbetweening algorithm proceeds as follows. First, silhouettes are created automatically from the input data. The color of the background pixels is known, so every pixel not equal to the background color is set to black and all background pixels are set to white. The silhouettes are then used to create contours defining the shapes to be inbetweened. The contours are generated from the silhouette images by starting at a pixel on the edge of the silhouette and tracing around the silhouette in clockwise order. The contour image is just a clockwise ordered list of (x, y) pixel positions. Next, using the ordered list of contour points, a set of normals are computed. These normals (shown in Figure VI.5) define the interior and the exterior of the contour for the RBF interpolation algorithm. The set of contour points and normals are then given a z coordinate, placing them in 3D space, and these point sets are used as input to the RBF method, which generates an implicit surface interpolating the contour points. A marching cubes algorithm then creates a mesh describing the implicit surface, and the mesh is sliced at intermediate points to create the intermediate contours. This process is quite fast and completes in approximately one minute on a 1GHz Pentium.

If the intermediate contour needs further refinement, the following steps can be taken: (1) the individual layers can be aligned, (2) constraint points can be added for the RBF contour interpolation, or (3) all methods can be used in conjunction. The alignment of layers can be done simply using the centroid of the character layer (for example the centroids of the heads for the head layer), or a more complicated transformation can be applied using an iterative closest point registration algorithm [Besl and McKay 1992]. Constraint points can be used to improve the intermediate contours. The user can select desired constraint points on the previous intermediate contour image to serve as extra data points that must be interpolated by the RBFs. Alternatively or in combination with the



(a) Partitioned Layers

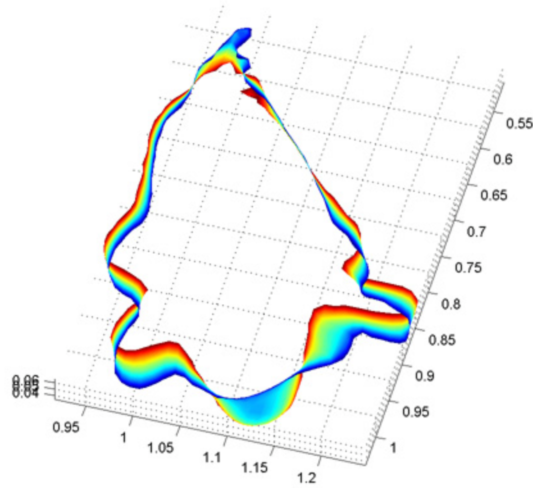


(b) Partitioned Contours

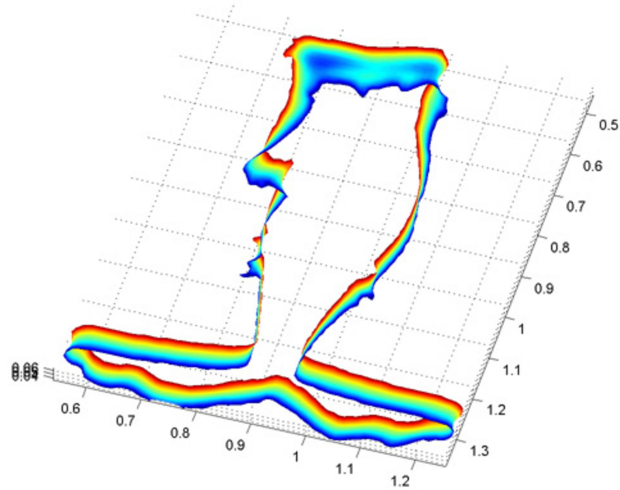
Figure VI.6: The top row shows an original key image on the left, the partitioned layers on the right. Notice that the head has been registered to the key image in Figure VI.8(d). The bottom row shows an example of the contours used for creating the RBF solution. *Daffy* is ™ & ©*Warner Bros. Entertainment Inc.*

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. Next we describe the RBF interpolation results, and note when any of the additional refinement methods are used.

The first example we describe uses two key images of *Daffy Duck* and both techniques for intermediate contour refinement. The character is partitioned into three layers for both key images: a head layer, an arm layer, and a body layer, as described in Chapter VI.1. The head layers are registered using the iterative closest point method with 12 control points for each image. Figure VI.6 shows the partitioned layers and final contours for one key image. Once the head images are registered, the contours are generated and the RBF interpolation method is employed. Two additional constraint points are included 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 layers are registered, but required no further contour refinement. The body layers did not require any contour refinement. Figures VI.7(a) and VI.7(b) show the RBF interpolation results for the *Daffy* head and body layers, respectively.



(a) Head Layer RBF



(b) Body Layer RBF

Figure VI.7: The top image is the RBF solution for the *Daffy Duck* head layer. The head images were registered and two additional constraints were used. The bottom image is the RBF solution for the *Daffy Duck* body layer.

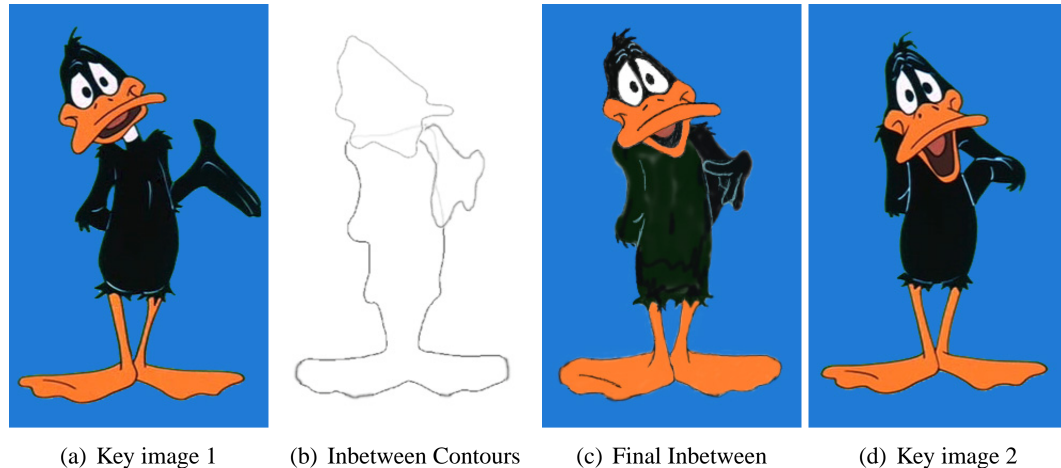


Figure VI.8: The final inbetween frame generated using RBF interpolation method with layering and constraints. The key images are shown on the left and right. *Daffy* is ^{TM}& ©*Warner Bros. Entertainment Inc.*

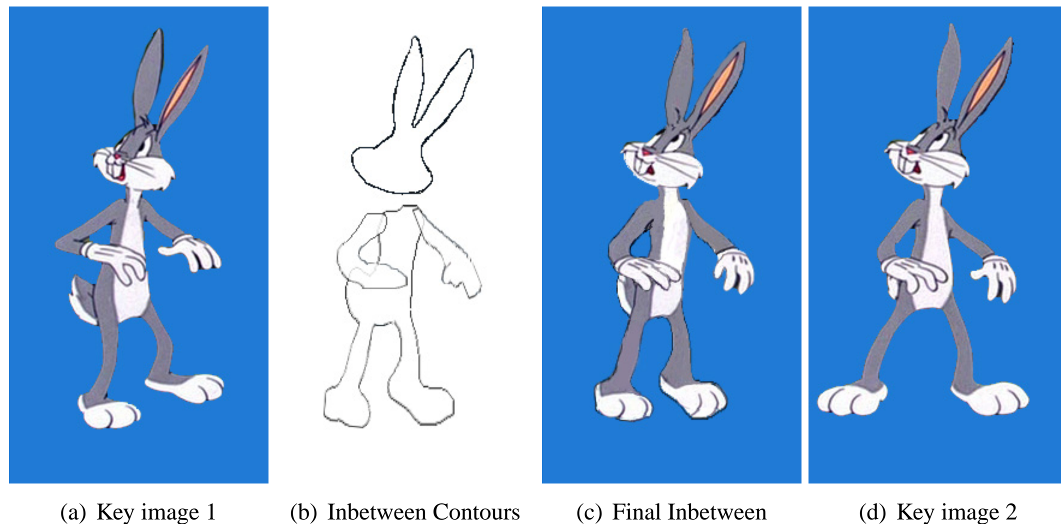
Figure VI.8 shows the final inbetween for the *Daffy* example described above. The intermediate contour for each layer is reassembled after using the RBF contour interpolation method. The color information is filled in manually for this example, and an automatic method will be discussed later.

Two more examples of the RBF contour interpolation method are described for *Bugs Bunny* and the *Coyote*. Figure VI.9 shows the inbetweening results for *Bugs Bunny*. For this example, *Bugs* was partitioned into four layers: head, body, left and right arms. The only contour refinement used aligning the heads in the head layer using the centroids. Figure VI.10 is an example inbetween generated for *Wile E. Coyote*. The *Coyote* was partitioned into four layers: head, left arm, right arm, and body. The head and arm layers were aligned using their centroids. The RBF solution for each of the *Coyote* layers did not require any additional contour refinement. All of the examples shown in this section have the color information for the intermediate contours filled in manually.

VI.5 Texturing or Re-coloring the Intermediate Contours

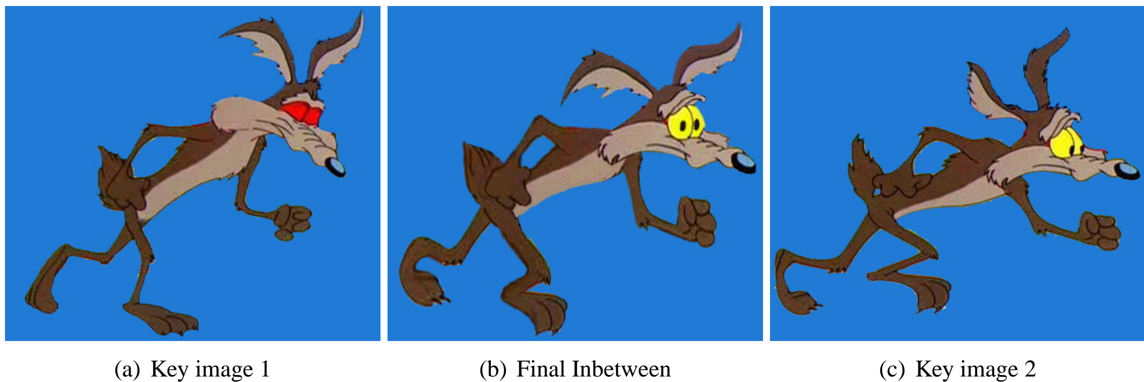
The final step in creating an inbetween is filling in the color and texture information. We have the color and texture information for the original key images available, from which we generated the intermediate contour, and the issue is how to best transfer this information and blend it into the intermediate contour. In a production studio, a similar process is done when the line art is scanned in and goes to the next step of ink and paint. Traditionally, the ink and paint process was all done manually. Some studios now 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



(a) Key image 1 (b) Inbetween Contours (c) Final Inbetween (d) Key image 2

Figure VI.9: The final inbetween frame generated using RBF contour interpolation method with layering. The key images are shown on the left and right. *Bugs Bunny* is ^{TM}& ©*Warner Bros. Entertainment Inc.*



(a) Key image 1 (b) Final Inbetween (c) Key image 2

Figure VI.10: The final inbetween frame generated using RBF contour interpolation method with layering and aligning the head and arm layers. The key images are shown on the left and right. *Wile E. Coyote* is ^{TM}& ©*Warner Bros. Entertainment Inc.*

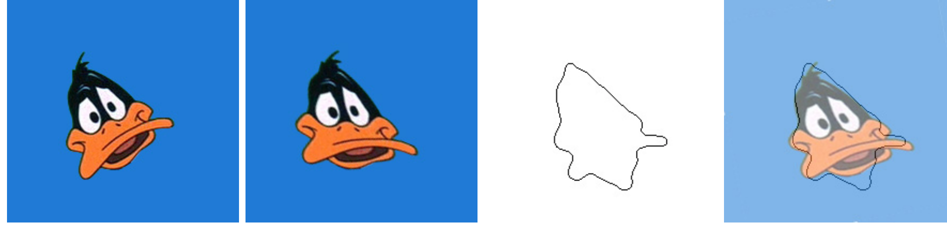


Figure VI.11: Moving from left to right: the first image is a close up of the first key image head layer, the second image is this key image after using an affine transformation to warp the image to match the inbetween contour, the third image is the automatically generated inbetween contour, the last image is an overlay showing the transformed key image atop the inbetween contour. *Daffy Duck* is *TM* & ©*Warner Bros. Entertainment Inc.*

generating an intermediate image based on the registration. As a proof of concept test, we first try a simple affine transformation to register the two key images. Figure VI.11 shows an example of applying an affine transformation (done manually in Adobe Photoshop) to one key image to align it with the intermediate contour. As is clearly visible, a better registration of the key image to the intermediate contour is necessary for improving the result. There is vast literature on image registration in the medical imaging community, where we looked for inspiration. The method we employ to register a pair of key images is non-rigid elastic image deformation, developed by [Wirtz et al. 2004], and recently adapted by [Li et al. 2006]. The next section briefly summarizes the original method, followed by the results of the image registration.

VI.5.1 Summary of Elastic Registration

Although the method of the fast non-rigid elastic registration by [Wirtz et al. 2004] was developed for the purpose of producing a high resolution three-dimensional reconstruction of a rat brain from a series of images (slices of the brain) in a short amount of time, the technique adapts very well for the purposes of registration and deformation of cartoon images. The authors chose to use elastic registration due to the distortions introduced in slicing and digitization of a rat brain, and cite several methods of elastic registration. They also use a fast implementation method for a system of nonlinear equations based on the work of [Fischer and Modersitzki 1999], and a Gaussian pyramid for evaluation of downsampled images to build up to a high-resolution final result.

A preprocessing step is performed to compensate for any artifacts due to rotation or translation before the elastic registration proceeds. The authors briefly describe the process of elastic registration, which is paraphrased here. We refer the reader to the references provided in [Wirtz et al. 2004] for more details. Each image, or digitized brain slice, 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 of images and displacement fields results in a system of nonlinear partial differential equations, or the Navier-Lamé (NLE) equation, given by

$$\mu\nabla^2\overrightarrow{u} + (\lambda+\mu)\nabla(\nabla\cdot\overrightarrow{u}) + f(\overrightarrow{u}) = 0 \quad \text{(VI.2)}$$

where $\overrightarrow{u}$ is the displacement field that tries to minimize the sum of squared intensity differences of the images, $f(\overrightarrow{u})$ 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, which in [Wirtz et al. 2004] is simply the derivative of the distance metric. The object is deformed until an equilibrium is reached between the forces. The system of equations for the three-dimensional registration differs slightly from that of a system for a pure two-dimensional registration, so the authors use the solution from [Fischer and Modersitzki 1999]. 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.

In the multi-resolution step, the images are analyzed and registered at low levels of resolution before proceeding to higher levels. The authors use a Gaussian pyramid. At the lower levels, the number of iterations are restricted. After each level of deformation, the error function is recomputed before performing registration at a higher level of resolution. Multi-resolution registration is both computationally less expensive than trying to deform the whole series of images into place all at once and more likely to result in a satisfactory convergence.

VI.5.2 Results of Re-coloring Contours

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 intermediate contour. We extended the algorithm for elastic registration of grayscale images [Li et al. 2006] to color images. Basically 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 intermediate 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 characters was the same. By using the color information in the intermediate image, most of the contour will be filled, requiring 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 for the artist to touch up after generating the intermediate texture, as opposed to the three steps for closing contours, filling, then the final touching up, thereby speeding up the process. Also, because the method is model-free, no user input is required to generate the results, just the two input images.

The intermediate contours that were previously computed (Chapter VI.4.2) are used with the elastic registration results in two ways. First, the contour is used to automatically re-assemble the

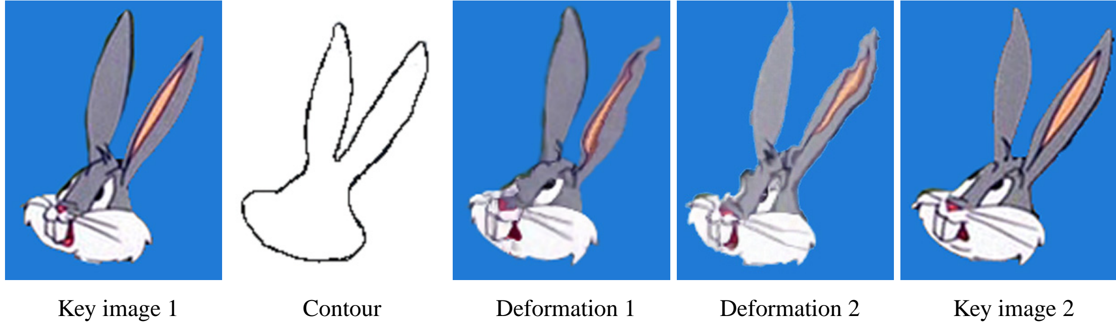


Figure VI.12: A comparison of results using elastic deformation for generating inbetweens on the *Bugs Bunny* head layer. On the left and right are the original key images. The image labelled "Contour" is the RBF contour interpolation result. "Deformation 1" is the result going from *key image 1* to *key image 2*, while "Deformation 2" is the result going from *key image 2* to *key image 1*. *Bugs Bunny* is ™& ©*Warner Bros. Entertainment Inc.*

Table VI.1: Evaluating the similarity of the elastic deformation result to the intermediate contour from the RBF contour interpolation step.

Images	Hausdorff Distance
C to ER_1	2.0545
C to ER_2	2.3324

character layers quickly, as described in Chapter VI.1. Second, and more importantly, the contour is used to determine the correct direction that the elastic deformation 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 elastic deformation result using *key image 1* as the source and *key image 2* as the destination. ER_2 is the elastic deformation result using *key image 2* as the source and *key image 1* as the destination. Figure VI.12 shows the two key images, the intermediate contour C, and the results of ER_1 and ER_2. We can see that ER_1 is visually better than ER_2. To determine which deformation result more closely matches the contour C, we use the Hausdorff distance described in Chapter V.2.3 to compute the similarity of C to ER_1 and C to ER_2. The results are shown in Table VI.5.2. As we expected, the deformation ER_1 is a better match to C, which is used in the final inbetween.

We compare the results of elastic registration to the manually filled in results of Chapter VI.4.2. The same characters and pairs of key images are used. Figure VI.13 shows a close up of the *Daffy* head layer with the two key images, the intermediate texture generated using the elastic registration, an overlay of the intermediate texture on the inbetween contour, and the final result after a small amount of manual touch up. Figures VI.14, VI.15, and VI.16 show the final results on the three characters. A comparison is also be made between the image morphing results of Chapter VI.2 and those here. Figure VI.17 shows a close up of the *Bugs Bunny* head layer with the results of the image

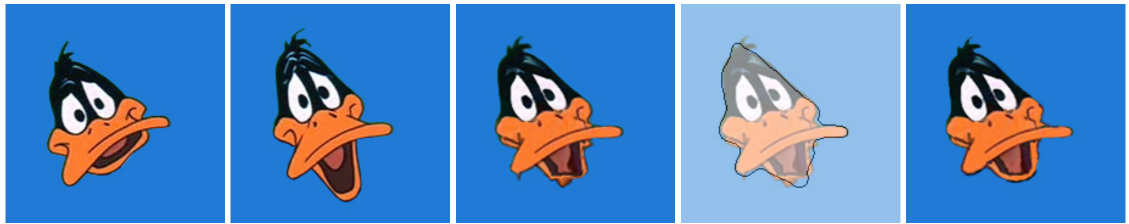
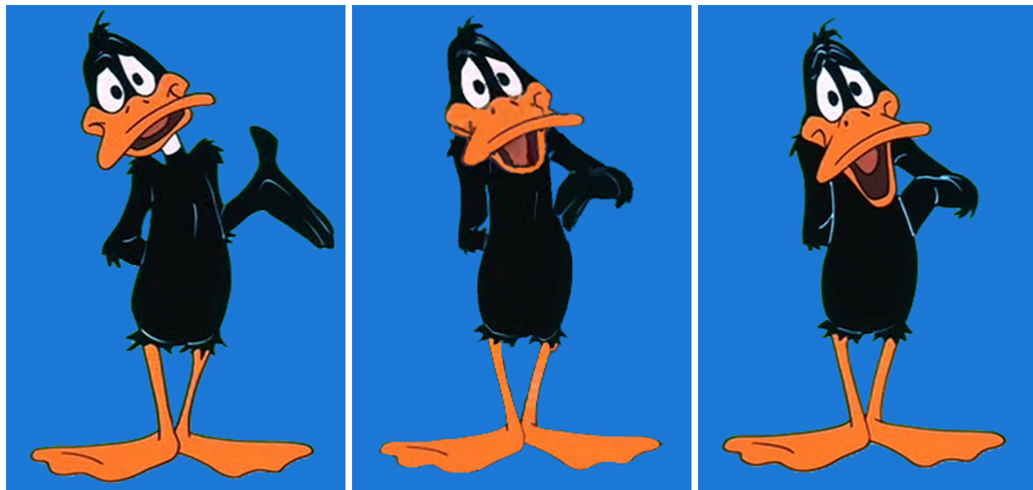


Figure VI.13: Moving from left to right: the first image is a close up of the first key image head layer, the second image is a close up of the second key image head layer, the third image is the automatically generated inbetween texture, the fourth image is an overlay showing the intermediate texture overlayed on the inbetween contour, and the fifth image is the final inbetween for the head layer. *Daffy Duck* is ™ & © *Warner Bros. Entertainment Inc.*



Key image 1 Final Inbetween Key image 2

Figure VI.14: The final inbetween frame generated using RBF contour interpolation method with elastic registration providing the color and texture information. The key images are shown on the left and right. *Daffy Duck* is ™ & © *Warner Bros. Entertainment Inc.*

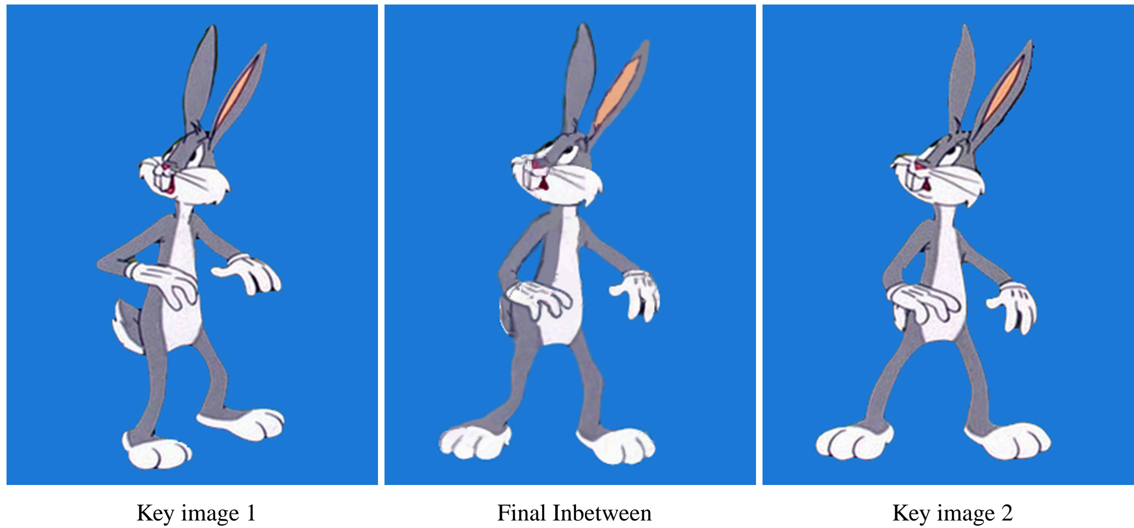


Figure VI.15: The final inbetween frame generated using RBF contour interpolation method with elastic registration providing the color and texture information. The key images are shown on the left and right. *Bugs Bunny* is ^{TM}& ©*Warner Bros. Entertainment Inc.*

Figure VI.16: The final inbetween frame generated using RBF contour interpolation method with elastic registration providing the color and texture information. The key images are shown on the left and right. *Wile E. Coyote* is ^{TM}& ©*Warner Bros. Entertainment Inc.*

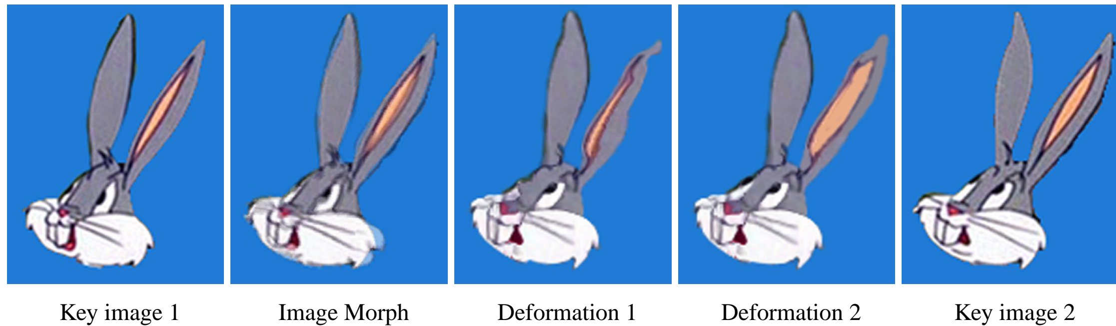


Figure VI.17: A comparison of results using image morphing and elastic deformation for generating inbetweens on the *Bugs Bunny* head layer. On the left and right are the original key images. The image labeled “image morph” is the result of using 30 feature lines and is half way between the two key images. The images labeled “deformation 1” and “deformation 2” show the results of using the elastic registration. “Deformation 1” is the raw result, while “Deformation 2” is the manually touched up result. A small amount of correction was required on one eye and ear. *Bugs Bunny* is *™& ©Warner Bros. Entertainment Inc.*

morphing side by side with the elastic deformation. Using the semi-automatic elastic deformation algorithm produces inbetweens of comparable quality, yet with no significant burden placed upon the user.

VI.6 Summary

The results in this chapter show that using implicit surfaces together with elastic deformation is a viable and robust technique for generating inbetweens for cartoon animation. It works well on three cartoon characters, each of which exhibit differences in color and shape. By providing a semi-automatic method for inbetweening, this chapter presents a third necessary and important component of a system for building re-usable motion libraries of traditional animation.

Including the artist or animator in the process of creating the inbetweens ensures the quality of the resulting images, and the methods presented here require only minimal intervention for touching up novel sequences of animation. The processes described here can 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, we believe that this method yields better inbetweens for two-dimensional animation than has previously been reported.

CHAPTER VII

SUMMARY AND FUTURE DIRECTIONS

VII.1 Summary of Contributions

In this dissertation, we have presented the components of a system to develop existing cartoon animation into motion libraries and novel animation sequences. Traditional animation is an art form everyone is familiar with, but successfully merging it with computer animation techniques has historically been one of the most challenging areas of research in computer animation. Traditional animation for film and television has become very expensive and time consuming in spite of the many technological advances in computer animation. The components developed here accomplish three necessary tasks of any re-use system: segmentation, re-sequencing, and inbetweening.

This research makes technical inroads into the problem of re-using traditional animation, and is focused on the three key tasks of building a re-usable cartoon motion library. First, we developed a fast and robust method for segmenting cartoon images requiring very little user input. Second, we determined that manifold learning techniques allow for a parameterization of large amounts of cartoon data, providing a simple way of re-using existing cartoons. Third, we showed that implicit surface techniques along with an elastic deformation technique can be successfully applied to the inbetweening problem, creating a semi-automatic and interactive method for generating inbetweens. All of these techniques can be successfully modified and applied to aid in the problem of merging traditional animation methods with computer animation. We believe the utility of these techniques will open new avenues of research in two-dimensional animation and may lead to a resurgence in interest to cartoon animation.

Isolating the characters from completed animations and creating a character library is the first necessary step in re-using cartoon data. We explored three techniques: an ad hoc method based on the probability of encountering certain pre-selected color values, a level set method with the speed function defined by either gray-level intensity or color values, and classifying the color and optical flow magnitudes using support vector machines (SVMs). We demonstrated the success of classifying cartoon data using SVMs, requiring minimal user intervention, and applied simple morphological operations to clean up any incorrectly classified regions of the images. The same morphological operations can be used to clean up results from the ad hoc method as well. The support vector machine technique outperformed the ad hoc and level set segmentation techniques that we tried, which could not effectively deal with the amount of noise in the animations. The SVM technique also proved to be the fastest at segmenting full images.

Once the characters are segmented, the second step for re-using cartoon data is how to access and re-sequence the data itself. An appropriate distance metric for comparing the similarity of the images is crucial, and we showed that an approximate Hausdorff distance metric was superior to the more common L_2 and cross-correlation metrics. We also demonstrated that the use of a manifold learning algorithm, in particular ST-Isomap, could successfully create a manifold of sparse data such

as those found in hand-drawn cartoons. The key to building a good manifold representing the data was in the identification of an appropriate distance metric. A simple traversal of the manifold based on the shortest-cost path results in a sequence of image indices used for re-sequencing the frames to create novel animations. A cost is associated with each edge taken along the re-sequenced path, and that value is a useful measure of the visual discontinuity between frames of the new animation. Theoretically, the discontinuity exists because the manifold is not sampled densely enough by the data for its structure to be parameterized. Hand-drawn cartoon data is inherently sparse, representing a fundamental obstacle for manifold learning techniques.

Because of this sparsity, we developed a semi-automatic method of generating inbetweens to make the available data more dense and provide increased smoothness to animation streams. Our procedure involves three steps. In the first step, the character is partitioned into several layers such as head and torso. In the second step, intermediate 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 intermediate shape using an elastic deformation technique. All of these steps require some user intervention; however having a semi-automatic method for generating new images is an important factor in quality control. Including the animator or artist in various aspects of building a motion library and re-using the cartoon data becomes necessary to ensure that the results maintain some nuances of cartoon animation that give it the characteristics we find most appealing.

Creating and re-using three-dimensional computer models for animation are fast and easy to manipulate, and many studios have turned to these as their preferred method of animation, particularly in film production. Animated films have gained much popularity in recent years, in particular three-dimensional computer animations, which can be seen in the dramatic increase of animated films being released today. Many of the television cartoon series use 'toon-rendering of three-dimensional computer models to help reduce production costs. Yet, several TV cartoons are still hand-drawn (like *The Simpsons*), but now use computers instead of actual cels. Since [Catmull 1978], there have only been a handful of contributions to two-dimensional animation, and it has been one of the most interesting challenges since the dawn of computer graphics. It is sad to see the traditionally animated TV cartoon or film be lost simply because of production costs. Many of us grew up watching the *Looney Tunes* and *Mickey Mouse* cartoons on Saturday mornings, and marvelled at the expressiveness and exaggerated motions the characters exhibited. Much of that charm is lost today in favor of more simplistic motion from 'toon-rendered models and three-dimensional cartoons. Providing a means of re-using existing cartoon data can help in speeding up the production time of creating new hand-drawn animations. In particular, the artists charged with generating inbetweens could use a system like the one proposed here to match keyframes as start and end poses, while the system generates the inbetween frames either by re-sequencing or creating new images. The idea of re-using cartoon data by re-sequencing is a novel one, and may revive interest in traditional animation as a viable form of art, thereby encouraging more studios to revisit creating traditional animations.

VII.2 Future Directions

This research, while answering several important questions about re-using traditional animation, gives rise to other interesting questions and potential topics of further research. To make re-using cartoons economically viable, the first step would be to build a complete system including a graphical user interface (GUI) that would encapsulate all of the components presented in this dissertation. Building such a GUI should take into account the needs of both the animator and the production studio environment. For studios still working on traditional animations, the animators could add new source material to the motion library as they draw new characters, and access existing drawings easily. Having a user friendly GUI would make re-using traditional animation much more accessible.

VII.2.1 Threshold Detection Revisited

Another aspect of the GUI would be to automate the selection of a threshold value for determining a visual discontinuity (discussed in Chapters V.5 and V.6). Statistical analysis on the cost of transitions might reveal a good threshold value. For example, with the *Daffy* data set, there are a total of 559 transitions with an average transition cost of 1.883, a minimum transition cost of 0.001, and a maximum transition cost of 20.987. There are three scene cuts with a cost of the transition at the scene borders greater than 15.1, which is expected since the character may not be facing the same direction in one scene to another. Figure VII.1 shows two original images at a scene cut. Our manually selected threshold value of good transitions being less than a cost of 2.2 represents 76.6% of all transitions, a bad transition with a cost of greater than 3.9 represents 12.9% of all transitions, with the remaining 10.5% of transitions falling between those threshold values. Determining the threshold value of good transitions can be automatically set based on selecting the cost value that represents approximately 75% or more of the total transitions.

VII.2.2 Segmenting Black and White Cartoons

In addition to building a GUI, there are other technological innovations that could improve such a system. One challenge is how to improve the segmentation methods to handle old black and white cartoons. Although we found the SVM technique superior to the others, the SVM technique may fail if it were applied to very early animation from the 1920's and 1930's, which contain significant amounts of noise, and no color information. It is likely that other automatic or semi-automatic segmentation techniques would fail as well. Figure VII.2 shows an example of a black and white cartoon from 1936 that exhibits lighting changes and a bright white spot in two sequential frames. Noise and artifacts like these are common throughout the entire cartoon and are typical of the earlier animations.

VII.2.3 Incorporating Principles of Animation

While some of the principles of animation [Thomas and Johnston 1981] may be maintained by re-sequencing existing traditional animation, others such as "timing" and "slow in slow out" cannot

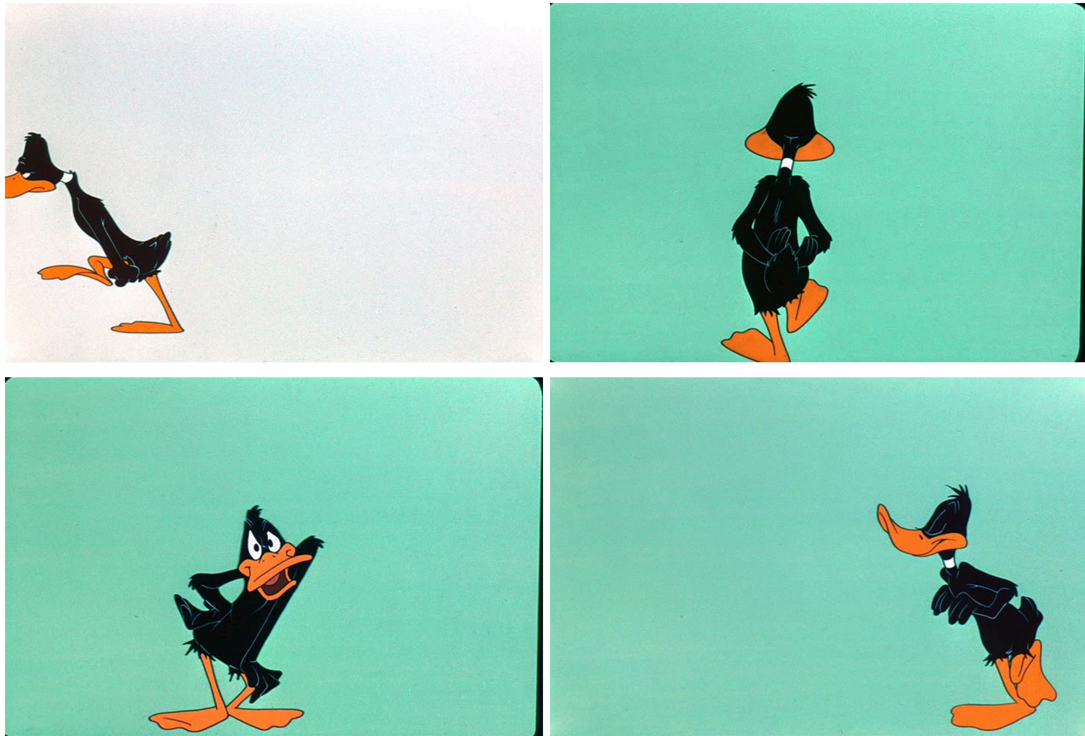


Figure VII.1: Two pairs of images at scene cuts from the *Daffy* data set, illustrating that these images would result in a high transition cost because of the dissimilarity of the character in the images. The top row is the first scene cut, the bottom row is the second scene cut. *Daffy Duck* is ^{TM}& ©*Warner Bros. Entertainment Inc.*

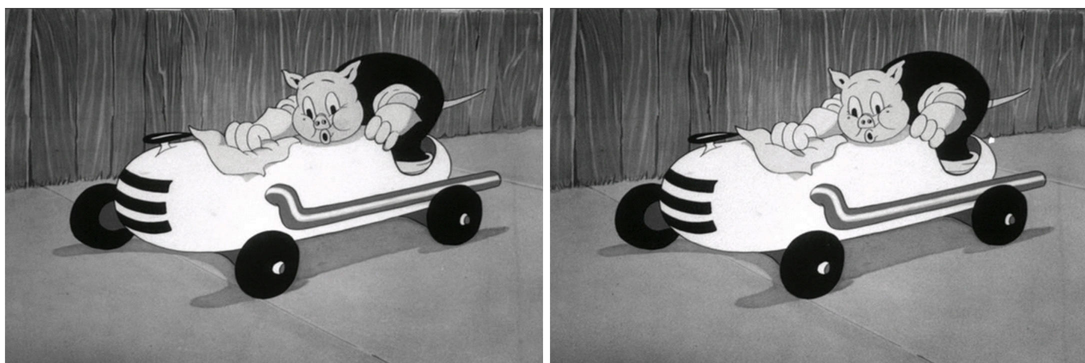


Figure VII.2: Two sequential images from the 1936 animated short "Porky's Road Race," produced by Leon Schlesinger. Notice the lighting changes on *Porky's* face and the large white spot that appears in the second frame behind the car. Also, one of *Porky's* dimples is missing in the first frame. *Porky Pig* is ^{TM}& ©*Warner Bros. Entertainment Inc.*

be maintained by simply traversing a low-dimensional manifold of the data using a shortest-cost method. Finding a traversal method that incorporates timing information would be an interesting future direction. Another approach would be to incorporate high-level information from the animation principles into the distance metric, for example to keep certain frames close together if they exhibit "squash and stretch" or "anticipation." To accomplish that would require a fair amount of user intervention in annotating the cartoon images with which principles of animation are exhibited in each of the frames.

VII.2.4 Inbetweening Revisited

We developed a method for producing inbetweens for traditional animation that produced good results. An alternative method for generating inbetweens would be to create a mesh with the color information from the key images as the texture on the mesh. A volumetric texture for the mesh can be generated and used for slicing, and a slice of that mesh would be the resulting inbetween image. We recently examined a method for creating and slicing a mesh to extract interpolated color or texture information: Mean Value Coordinates (MVC) by [Ju et al. 2005]. The MVC method shows promise as a future direction in inbetweening, specifically in creating a high-resolution mesh where each vertex represents a pixel in the key images, and using that mesh for slicing. By using a mesh to represent the character layers entirely (color and shape information together), the remaining issue of silhouette changes can be addressed.

The idea behind MVC is to find a function that can interpolate a set of values at the vertices of a mesh smoothly into its interior. MVCs have been used for closed 2D polygons, and the work by [Ju et al. 2005] generalizes the method to closed triangular meshes, leading to interesting applications such as our interest in volumetric textures. Since mean value coordinates work on arbitrary data associated with each mesh vertex, MVC can take either color information or texture coordinates. What the algorithm provides is an interpolation of color or texture coordinate data to the interior of the mesh volume, creating a volumetric texture.

Here we present results showing the viability of using the MVC method for inbetweening. The first step in using MVC is to create the mesh. Creating a mesh manually is a difficult and time consuming task, usually left to expert modelers or animators. To reduce the burden on the user of creating a mesh, we modify the RBF interpolation method described in Chapter VI.4.2. Instead of creating a mesh from a pair of contours, we use all of the color information in both key images. Essentially the two color images are on parallel planes separated by a small distance, and the idea is to find a plane between the parallel ones with the interpolated color values and use that result as an inbetween image. Setting up the images to create a mesh is done as follows. Each key image is viewed as a slice of planar data points, with each point having (x,y) pixel position data and their associated R,G,B color values. Two "slices" (on parallel planes) are separated along the z-axis by some small amount. The input to the FastRBF toolbox are the (x,y) pixel coordinates that represent the character, and their associated R,G,B values for each image, which are now in 3D space since each plane of pixel coordinates and color values have an associated z coordinate value.

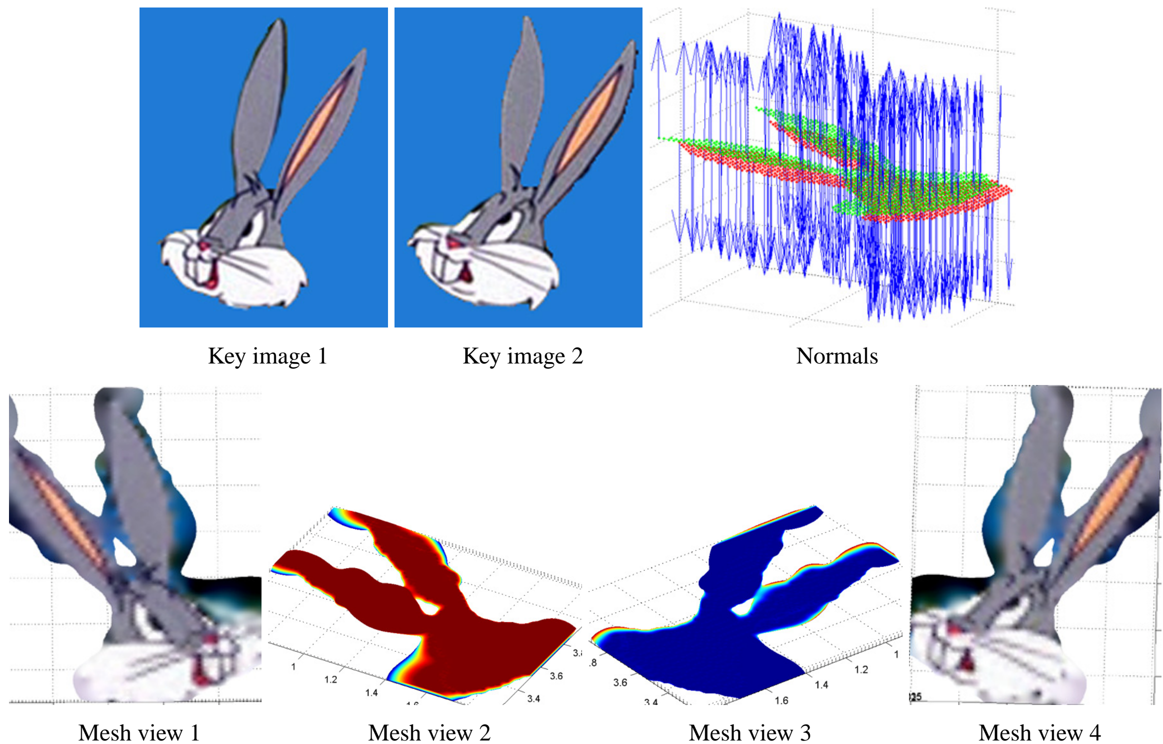


Figure VII.3: An example of how to set up a mesh for use with the MVC slicing method. On the top row the two input images and the normals for the data points, red points indicate one image and green points indicate the second image. On the bottom row, the resulting mesh shown from four viewpoints. *Bugs Bunny* is ^{TM}& ©*Warner Bros. Entertainment Inc.*

One requirement is that the key images used are already segmented and partitioned into layers. As such, none of the background pixels are used, i.e., if a pixel at (x,y) has color equal to the known background color, it is excluded from the input data used to create the mesh. Normals for each data point are calculated using a pre-defined FastRBF function, and bi-harmonic basis functions are used for the RBFs. Both pixel locations and color data are interpolated by the RBFs to create an isosurface. Finally, a mesh is created by using a fast marching-cubes algorithm, and the color values for each point are interpolated across the surface of the mesh. Each vertex in the mesh has an associated color value, allowing for easy interpolation of color when the mesh is sliced using MVC. Figure VII.3 shows the steps involved in creating the mesh using RBF interpolation.

Nine slices are generated at $0.1,\ldots,0.9$ distance between the top and bottom slice of the mesh from figure VII.3. For proof-of-concept, they are created at a low resolution 128 x 128. Figure VII.4 shows the results of slicing the RBF mesh. There are several interpolation defects, such as the blurring of color features in the middle, for example, at the 0.5 slicing plane. This blurring is inevitable because we are doing interpolation on the colors. The overall loss in quality is a result of several interpolation steps, including the RBF interpolation of color to create the mesh, down-sampling the mesh to run efficiently using MVC, and also the MVC interpolation itself. The run-

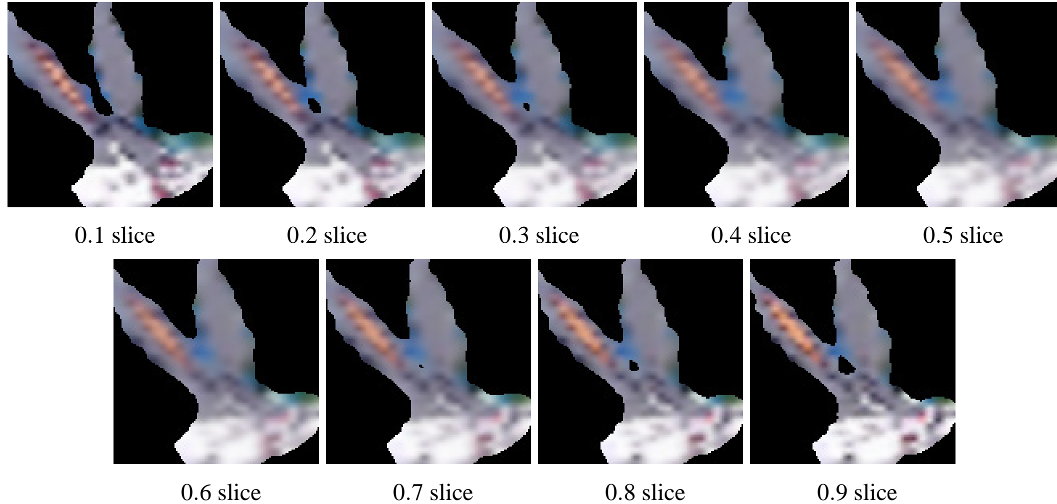


Figure VII.4: Slices of the mesh generated using RBF interpolation of pixel location and colors. In the top row, the slices at 0.1 distance to 0.5 distance from the top and bottom planes, and in the bottom row, the slices at 0.6 distance to 0.9 distance from the top and bottom planes. Images obtained courtesy of Dr. Tao Ju. *Bugs Bunny* is ™ & ©*Warner Bros. Entertainment Inc.*

time of the MVC algorithm is $Nverts * Npoints$, where $Nverts$ is the number of vertices in the mesh, and $Npoints$ is the number of points to be evaluated. The interpolation example shown in figure VII.4 took approximately 10 minutes for each slice. Each slice has 128 x 128 points, and the color at each point is interpolated from 18,622 mesh vertices. With nine slices extracted, the total time to generate the slicing results took 90 minutes. However, since in many cases only one or two slices would be required for the inbetweens, the run-time of this algorithm is reasonable on small meshes.

To yield the best results and eliminate losses in quality due to RBF interpolation, a high-quality mesh would have to be generated manually. Figure VII.5 shows an example of a mesh created manually with the texture information on each large surface plane applied from a partitioned key image layer. The idea is to generate an interpolation between two texture images using slices of a volumetric texture. Although all of the color information is preserved without any loss of detail, the MVC method can only use one texture applied to a closed mesh. Using the two key images as one texture will result in an interpolation across the texture coordinates and not produce a meaningful inbetween. One alternative to using two textures is to create a high resolution mesh with each vertex assigned a color value, similar to the RBF mesh discussed above. If it is possible to associate every mesh vertex with a specific color and not lose any detail from the image, then the mesh would have an extremely high vertex count, resulting in an unreasonable computational time to extract an intermediate slice. Another idea for improving the results is to have a correspondence between features on the two planes, and use MVC on smaller meshes segmented by those corresponding points. The difficulty with any of these improvements is that the generation of the mesh is no longer

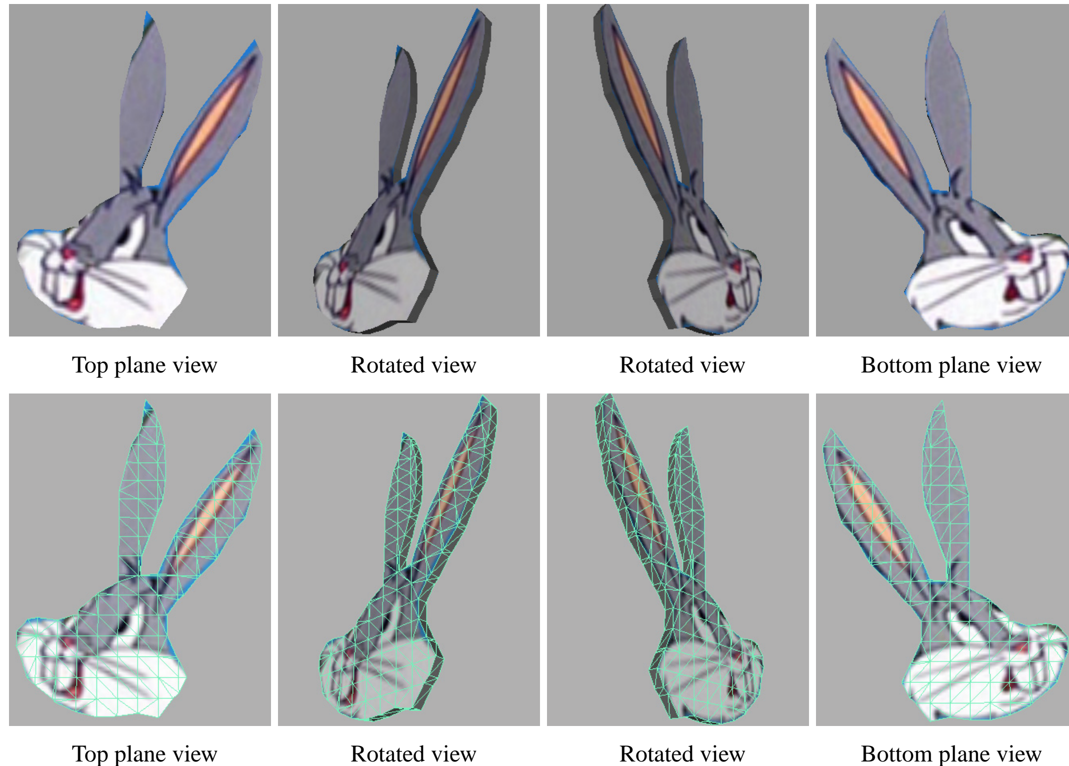


Figure VII.5: Several viewpoints of the manually created mesh. The top and bottom planes are the two key images. The bottom row shows the same mesh with the underlying wireframe highlighted. *Bugs Bunny* is ™& ©*Warner Bros. Entertainment Inc.*

automatic, and if correspondences need to be determined between features on the two image planes, the burden of both of these tasks is placed back with the user.

Although the major limitations of using MVC on a high-resolution mesh lie in the current state of technology, we believe that the MVC technique will become a viable resource once computing power increases. While we do not claim to have solved the two-dimensional inbetweening problem, we have overcome some of the hurdles that require large amounts of user-intervention. We provided a method of inbetweening that is model-free, and may have reached the limits of such methods. It may be possible to apply techniques such as view-dependent geometry [Rademacher 1999] to achieve improved texture filling on inbetweened contours by using a three-dimensional model of the character. Assuming one can extract a three-dimensional model from two or more images, the view-dependent geometry method may help when the character partitioning into layers is insufficient in resolving the self-occlusion problem (discussed in Chapter II.1.2). Sometimes layering alone will not solve the occlusion problem, and having a three-dimensional representation of the character may resolve any remaining issues, in particular the silhouette changes.

Another interesting avenue of research is in generating a high-quality volume mesh from a pair of keyframe images, or extending the method of MVC to be able to interpolate across multiple

textures on a mesh. The mesh shown in figure VII.3 is generated using RBFs. This represents an interesting first step towards automatic mesh generation, but suffers from loss of detail in the image due to interpolation across the mesh when fitting the RBF and then again when slicing the mesh using MVC. Once again, incorporating ideas from [Rademacher 1999] with implicit surface representations, such as RBF interpolation, could provide a reasonable mesh automatically. Having a model sheet (defined in Chapter II) of a character would help in the creation of a mesh of that character in a neutral pose.

VII.2.5 Stop-Motion Animation

Although this dissertation has focused on traditional hand-drawn animation and how to best merge it with computer animation techniques, another form of animation has not been influenced much by computational technology. Stop-motion animation is an extremely laborious process, more so than hand-drawn animation, and is an interesting form of art that is still popular in films today. Some research has been done on stop-motion animation, in particular how to add motion blur [Brostow and Essa 2001]. Like traditional animation, stop-motion does not exhibit any motion blur because both are series of still images that when played back create the illusion of a moving scene. Motion blur occurs when objects move while a camera shutter is open. Stop-motion animation, sometimes referred to as clay animation or "claymation" because of the pliable material such as clay or *Plasticine* used to create the characters, has a unique look that is not easily simulated by a computer. It would be interesting to apply the methods of this dissertation to clay animation characters like *Wallace* and *Gromit*, figure VII.6, and build a motion library out of clay characters instead of hand-drawn ones. One new challenge for re-using clay characters that does not arise in the hand-drawn ones, is in re-lighting the clay characters in a re-sequenced animation. Because the clay characters are actual three-dimensional models on a set with real lights, re-sequencing frames of those characters would exhibit changes in the shadows and lights that appear on them. Some of those shadows can be seen in figure VII.6 on *Gromit*.

Figure VII.6: Examples of a stop-motion animation character *Gromit*, from the film “The Wrong Trousers,” exhibiting changing lighting conditions. ©*Aardman / Wallace and Gromit Ltd. 1993.*

REFERENCES

[Alexa et al. 2000]ALEXA, M., COHEN-OR, D., AND LEVIN, D. 2000. As-rigid-as-possible shape interpolation. In *Proceedings of ACM SIGGRAPH 2000*, ACM Press/Addison-Wesley Publishing Co., 157–164.

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

[Beier and Neely 1992]BEIER, T., AND NEELY, S. 1992. Feature-based image metamorphosis. In *Proceedings of ACM SIGGRAPH 1992*, ACM Press, 35–42.

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

[Bloomenthal 1997]BLOOMENTHAL, J., Ed. 1997. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers, Inc.

[Brand 2003]BRAND, M. 2003. Charting a manifold. In *Advances in Neural Information Processing Systems*, MIT Press, vol. 15.

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

[Brostow and Essa 2001]BROSTOW, G. J., AND ESSA, I. 2001. Image-based motion blur for stop motion animation. In *Proceedings of ACM SIGGRAPH 2001*, ACM Press, 561–566.

[Burtnyk and Wein 1976]BURTNYK, N., AND WEIN, M. 1976. Interactive skeleton techniques for enhancing motion dynamics in key frame animation. *Communications of the ACM 19*, 10, 564–569.

[Canny 1986]CANNY, J. 1986. A computational approach to edge detection. *IEEE Trans. on Pattern Analysis and Machine Intelligence 8*, 6, 679–698.

[Cao and Dawant 2005]CAO, Z., AND DAWANT, B. M. 2005. Personal Communication, March 30, 2005 – August 16, 2005.

[Carr et al. 2003]CARR, J. C., BEATSON, R. K., MCCALLUM, B. C., FRIGHT, W. R., MCLENNAN, T. J., AND MITCHELL, T. J. 2003. Smooth surface reconstruction from noisy range data. In *GRAPHITE '03*, ACM Press, 119–126.

[Catmull 1978]CATMULL, E. 1978. The problems of computer-assisted animation. In *Proceedings of ACM SIGGRAPH 1978*, ACM Press, 348–353.

[Chang and Lin 2001]CHANG, C.-C., AND LIN, C.-J. 2001. *LIBSVM: a library for support vector machines*. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[Corrêa et al. 1998]CORRÊA, W. T., JENSEN, R. J., THAYER, C. E., AND FINKELSTEIN, A. 1998. Texture mapping for cel animation. ACM Press, 435–446.

[de Juan and Bodenheimer 2004]DE JUAN, C., AND BODENHEIMER, B. 2004. Cartoon textures. In *2004 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, R. Boulic and D. Pai, Eds., 267–276.

[Di Fiore et al. 2001]Di Fiore, F., Schaeken, P., Elens, K., and Reeth, F. V. 2001. Automatic in-betweening in computer assisted animation by exploiting 2.5D modelling techniques. In *Proceedings of Computer Animation*, 192–200.

[Donoho and Grimes 2003]Donoho, D., and Grimes, C., 2003. Hessian eigenmaps: new locally linear embedding techniques for high-dimensional data.

[Etzion and Rappoport 1997]Etzion, M., and Rappoport, A. 1997. On compatible star decompositions of simple polygons. *IEEE Transactions on Visualization and Computer Graphics 3*, 1, 87–95.

[Fekete et al. 1995]Fekete, J., Bizouarn, É., Cournarie, É., Galas, T., and Taillefer, F. 1995. TicTacToon: A paperless system for professional 2-D animation. In *Proceedings of ACM SIGGRAPH 1995*, Addison Wesley, R. Cook, Ed., 79–90.

[Fischer and Modersitzki 1999]Fischer, B., and Modersitzki, J. 1999. *Numerical Algorithms*, vol. 22. Springer Science + Business Media B.V., March, ch. Fast inversion of matrices arising in image processing, 1–11.

[Foley et al. 1990]Foley, J. D., van Dam, A., Feiner, S. K., and Hughes, J. F. 1990. *Computer Graphics Principles and Practice*. Addison-Wesley Publishing Company, Inc.

[Gleicher 1998]Gleicher, M. 1998. Retargeting motion to new characters. In *Proceedings of ACM SIGGRAPH 1998*, 33–42.

[Gonzalez and Woods 2001]Gonzalez, R. C., and Woods, R. E. 2001. *Digital Image Processing*. Prentice Hall, Upper Saddle River, N.J.

[Huttenlocher et al. 1993]Huttenlocher, D. P., Klanderman, G. A., and Rucklige, W. J. 1993. Comparing images using the Hausdorff distance. *IEEE Trans. on Pattern Analysis and Machine Intelligence 15*, 9 (September), 850–863.

[Jenkins and Matarić 2003]Jenkins, O. C., and Matarić, M. J. 2003. Automated derivation of behavior vocabularies for autonomous humanoid motion. In *AAMAS '03: Proc. of the 2nd international joint conf. on Autonomous agents and multiagent systems*, ACM Press, 225–232.

[Jenkins and Matarić 2004]Jenkins, O. C., and Matarić, M. J. 2004. A spatio-temporal extension to isomap nonlinear dimension reduction. In *The International Conference on Machine Learning (ICML 2004)*, 441–448.

[Jolliffe 1986]Jolliffe, I. 1986. *Principal Component Analysis*. Springer-Verlag, New York.

[Ju et al. 2005]Ju, T., Schaefer, S., and Warren, J. 2005. Mean value coordinates for closed triangular meshes. *ACM Transactions on Graphics 24*, 3 (July), 561–566.

[Kort 2002]Kort, A. 2002. Computer aided inbetweening. In *NPAR 2002: Proc. of the 2nd international symposium on Non-photorealistic animation and rendering*, ACM Press, 125–132.

[Kovar and Gleicher 2004]Kovar, L., and Gleicher, M. 2004. Automated extraction and parameterization of motions in large data sets. *ACM Transactions on Graphics 23*, 3, 559–568.

[Kovar et al. 2002]Kovar, L., Gleicher, M., and Pighin, F. 2002. Motion graphs. *ACM Transactions on Graphics 21*, 3 (July), 473–482.

[Kowalski et al. 1999]KOWALSKI, M. A., MARKOSIAN, L., NORTHRUP, J. D., BOURDEV, L., BARZEL, R., HOLDEN, L. S., AND HUGHES, J. F. 1999. Art-based rendering of fur, grass, and trees. In *Proceedings of ACM SIGGRAPH 1999*, 433–438.

[Kruskal and Wish 1978]KRUSKAL, J. B., AND WISH, M. 1978. *Multidimensional Scaling*. Sage Publications, Beverly Hills.

[Lasseter 1987]LASSETER, J. 1987. Principles of traditional animation applied to 3d computer animation. In *Proceedings of ACM SIGGRAPH 1987*, 35–44.

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

[Li et al. 2006]LI, X., YANKEELOV, T. E., ROSEN, G., GORE, J. C., AND DAWANT, B. M. 2006. Multimodal inter-subject registration of mouse brain images. In *Medical Imaging 2006: Image Processing. Proceedings of the SPIE.*

[Lucas and Kanade 1981]LUCAS, B., AND KANADE, T. 1981. An iterative image registration technique with an application to stereo vision. In *Proc. of the 7th International Joint Conf. on Artificial Intelligence*, 674–679.

[O'Rourke 1994]O'ROURKE, J. 1994. *Computational Geometry in C*. Cambridge University Press.

[Osher and Sethian 1988]OSHER, S., AND SETHIAN, J. A. 1988. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics 79*, 12–49.

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

[Petrovic et al. 2000]PETROVIC, L., FUJITO, B., WILLIAMS, L., AND FINKELSTEIN, A. 2000. Shadows for cel animation. In *Proceedings of ACM SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 511–516.

[Pless 2003]PLESS, R. 2003. Image spaces and video trajectories: Using isomap to explore video sequences. In *Proceedings of International Conference on Computer Vision 2003*, ICCV, 1–8.

[Rademacher 1999]RADEMACHER, P. 1999. View-dependent geometry. In *Proceedings of ACM SIGGRAPH 1999*, ACM Press/Addison-Wesley Publishing Co., 439–446.

[Reeves 1981]REEVES, W. T. 1981. Inbetweening for computer animation utilizing moving point constraints. In *Proceedings of ACM SIGGRAPH 1981*, 263–269.

[Rose et al. 1998]ROSE, C., COHEN, M., AND BODENHEIMER, B. 1998. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications 18*, 5, 32–40.

[Roweis and Saul 2000]ROWEIS, S., AND SAUL, L. 2000. Nonlinear dimensionality reduction by locally linear embedding. *Science 290*, 2323–2326.

[Schödl and Essa 2002]SCHÖDL, A., AND ESSA, I. 2002. Controlled animation of video sprites. In *Symposium on Computer Animation*, ACM Press / ACM SIGGRAPH, 121–127.

[Schödl et al. 2000]SCHÖDL, A., SZELISKI, R., SALESIN, D. H., AND ESSA, I. 2000. Video textures. In *Proceedings of ACM SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 489–498.

[Seah and Lu 2001]SEAH, H. S., AND LU, J. 2001. Computer-assisted in-betweening of line drawings: Image matching. In *CAD/Graphics*, Kunming International Academic Publishers.

[Sederberg and Greenwood 1992]SEDERBERG, T. W., AND GREENWOOD, E. 1992. A physically based approach to 2-D shape blending. In *Proceedings of ACM SIGGRAPH 1992*, E. E. Catmull, Ed., 25–34.

[Sedgewick 2002]SEDGEWICK, R. 2002. *Algorithms in C: Part 5 Graph Algorithms*, 3 ed. Addison-Wesley.

[Shapiro and Stockman 2001]SHAPIRO, L. G., AND STOCKMAN, G. C. 2001. *Computer Vision*. Prentice Hall, Upper Saddle River, N.J.

[Sidenbladh et al. 2002]SIDENBLADH, H., BLACK, M. J., AND SIGAL, L. 2002. Implicit probabilistic models of human motion for synthesis and tracking. In *Computer Vistion — ECCV 2002 (1)*, Springer-Verlag, A. Heyden, G. Sparr, M. Nielsen, and P. Johansen, Eds., 784–800.

[Tenenbaum et al. 2000]TENENBAUM, J. B., DE SILVA, V., AND LANGFORD, J. C. 2000. A global geometric framework for nonlinear dimensionality reduction. *Science 290*, 2319–2323.

[Thomas and Johnston 1981]THOMAS, F., AND JOHNSTON, O. 1981. *The Illusion of Life: Disney Animation*. Hyperion.

[Turk and O'Brien 1999]TURK, G., AND O'BRIEN, J. F. 1999. Shape transformation using variational implicit functions. 335–342.

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

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

[Weinberger and Saul 2004]WEINBERGER, K. Q., AND SAUL, L. K. 2004. Unsupervised learning of image manifolds by semidefinite programming. In *Conference on Computer Vision and Pattern Recognition*, IEEE Computer Society, 988–995.

[Weng et al. 1993]WENG, J., HUANG, T., AND AHUJA, N. 1993. *Motion and Structure from Image Sequences*. Springer-Verlag, ch. 2: Image Matching.

[Wirtz et al. 2004]WIRTZ, S., FISCHER, G., MODERSITZKI, J., AND SCHMITT, O. 2004. Superfast elastic registration of histologic images of a whole rat brain for 3d reconstruction. In *Medical Imaging 2004: Physiology, Function, and Structure from Medical Images. Proceedings of the SPIE*, A. A. Amini and A. Manduca, Eds., vol. 5370, 328–334.

[Wolberg 1990]WOLBERG, G. 1990. *Digital Image Warping*. IEEE Computer Society Press, Los Alamitos, CA.