A Semi-Automatic System for Non-Rigid Matching and Temporally Coherent 3D Shading of Animation Sequences

Dany Rybnikov

A Semi-Automatic System for Non-Rigid Matching and Temporally Coherent 3D Shading of Animation Sequences

Final Paper

Submitted in partial fulfilment of the requirements
for the degree of Master of Science in Computer Science

Dany Rybnikov

Submitted to the Senate of
the Technion — Israel Institute of Technology

Av 5773 Haifa August 2013

This final paper was done under the supervision of Prof. Craig (Chaim) Gotsman and Assist. Prof. Mirela Ben-Chen in the Computer Science Department.

The generous financial help of the Technion - Israel Institute of Technology is gratefully acknowledged

Contents

Contents

List of figures

List of figures

List of tables

ABSTRACT

Applying a depth effect or a 3D-like look to hand-drawn animations can be just as visually pleasing as the rendering of a full 3D animation character. In this work we present a method for computer-assisted shading of 2D animation sequences where the main character undergoes non-rigid deformations between the frames.

The system receives as input a sequence of vectorized 2D animation frames containing an outline sketch of the character and applies a 3D shading effect directly to these 2D frames. The system allows the animator to specify shading constraints on a specific frame and addresses the challenge of propagating this information to the consequent frames with minimal user intervention. Eventually, this produces a coherently shaded animation sequence, and significantly reduces the effort spent by the animator on the shading process.

In order to achieve visually appealing results and keep the shading coherent between the frames, we describe a graphical user interface and registration method that is able to propagate shading constraints through the animation sequence despite non-rigid character deformations. Finally, we describe a method for generating the 3D-like shading to animation frames based on the constraints.

Abbreviations and Notations

CPD	Coherent Point Drift registration algorithm
SVG	Scalable Vector Graphics file format
UI	User Interface
GMM	Gaussian Mixture Model

1 Introduction

1.1 Animation

Animation is widely used today in many areas, such as entertainment movies and visualization aids. Although, many of the animation sequences are fully computer generated there is still a place for a classic frame-by-frame hand-drawn animation approach.

The classic animation process usually consists of multiple stages. We will focus mainly on the stages which are relevant for this work: drawing the frames of the animated character outline and shading of the frames.

Traditionally, most of the animation stages are performed manually by animation artists. In big animation productions the lead artist would design the character key-frames outline and junior artists will create the in-betweening of the key-frames and shading for every frame.

In this work we focus on the existing hand-drawn sequences which were created before the computer animation era, or alternatively hand-drawn animation sequences created directly on a computer, but which use the same frame-by-frame sequence of character outline drawings. It should be noted that character outline drawing can change dramatically as the sequence advances, both in terms of contour visibility and overall non-rigid behaviour of the character shape, which makes the automation of this process even more challenging. The remaining challenge for the artist once such hand drawn outline sequence is created is to shade each frame.

In this work we will focus on creating the 3D-like shading based on the character outlines drawn manually by the artist. While shading can be done manually using regular 2D shading tools, the desired effect still needs to be carefully crafted by the artist and then recreated for each consequent frame. This time-consuming operation could be assisted by automated or semi-automated computer tools.

In this work we address the challenge of the 3D-like shading of the animated character sequences of a non-rigid nature.

1.2 Related Work

Creating coherent animation sequences or frames with depth or shading effects from 2D input will almost always requires some form of user hints. For example Sýkora et. al [1] show a method for animating and shading 2D characters using depth information provided by the user. This method can be extended to animating sequences using registration between the frames. As-rigid-as-possible image registration [2] can be used to propagate information from one frame to another when the animation character transformation can be perceived as locally rigid.

Keeping the animation sequence shaded or rendered coherently requires matching elements of the input frame such as curves or paths, and propagating information from one frame to another. Whited et. al [3] propose an interactive tool for in-betweening process by matching input curves based on arc length similarity and other curve properties. Other works, as by Juan and Bodenheimer [4] and Melikhov [5], rely on image texture similarities from the input to match curves between consecutive frames.

Another possible approach to create a 3D animation from 2D input is to use the animation frames to recreate a full 3D character, as shown by Ono and Nishita [6]. Igarashi et. al [7] uses similar idea to provide a 2D sketching interface for 3D models creation which then can be animated.

Correctly shading the resulting image or sequence of frames creates the desired 3D effect. This can be achieved by rendering the reconstructed 3D or 2.5D character under certain illumination conditions like in [8] and [9]. Alternatively, the 2D drawing can be shaded directly using a normal diffusion and shading method called ”Lumo”, first described by Johnson et. al [10]. This method is used not only as a visualization aid, but also to provide a depth and volume perception for the animation frames and images as described in [11] ,[12] and [13].

1.3 Background

Complex animation system requires a combination of different tools and approaches. In this section we give a short overview of the methods and algorithms used in this work.

1.3.1 Bezier Curves

Vector graphics is a set of geometric entities which can be represented by mathematical expressions. In this work we use Bezier [14] curves to represent an animator's work. However, in practice any combination of vector graphics components can be used.

We have chosen to use cubic Bezier curves as a representation for an input, since it is most commonly used in most vector graphics editing software and is supported by vector graphics file formats such as SVG [15]. To represent more complex curves a concatenation of cubic Bezier is used. Smoothness is achieved by aligning control polygons segments of the concatenated curves

A Bezier curve (Figure 1) is defined by its control polygon, and all the intermediate points can be calculated using the de Casteljau algorithm [14] or parametric formula.

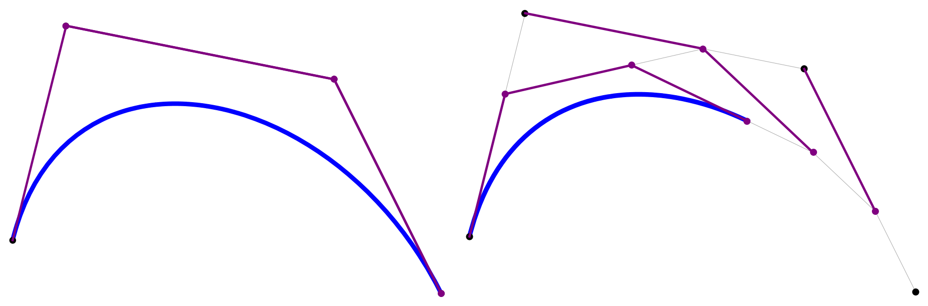

**Figure 1 – Left: cubic Bezier curve with its control polygon.
Right: curve point calculation using de Casteljau algorithm**

Since cubic Bezier curves in our system are used in point-based registration, we have to perform sampling that is proportional to the curve length, in a way that curves of different length will still have the same density of samples. We achieve this by approximating curve length by its control polygon length and then calculating an approximate number of samples using a predefined desired density value.

1.3.2 Point Set Registration

Point set registration is a crucial part of our system. Many registration methods exist, and these are usually divided into two groups: rigid and non-rigid. (Figure 2 and Figure 3 respectively)

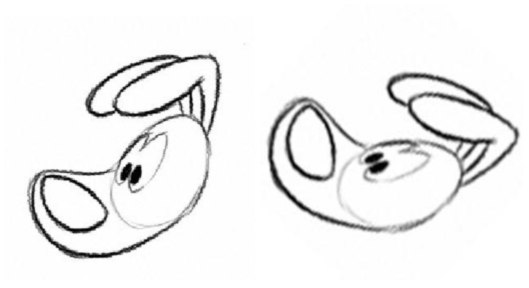

Figure 2 - Animation with rigid transformation

Rigid point set registration methods are well suited for solving problems where the point set does not undergo any non-rigid changes and both point sets are different instances of the same rigid body, possibly rotated or viewed from different directions.

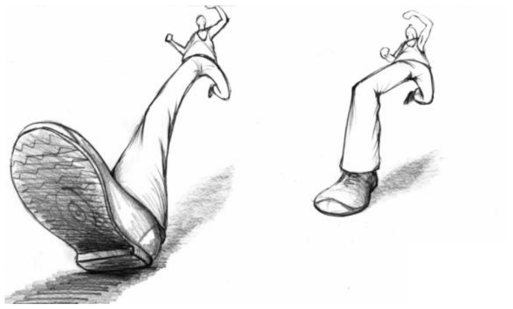

Figure 3 - Animation with non-rigid transformation

Non-rigid registration methods provide the ability to match points between sets which undergo some kind of non-rigid deformation, for example bending or non-uniform scaling. In animation from one frame to the next, characters are usually undergoing non-rigid deformations.

In our system we use the Coherent Point Drift [16] non-rigid registration method. The original method provides both rigid and non-rigid options. In our work we focus on the non-rigid option.

1.3.2.1 Non-rigid Coherent Point Drift

The main idea behind the non-rigid CPD algorithm is to rely on Motion Coherence Theory which states that points which are close to another tend to move coherently. In CPD, one point set, which is represented by Gaussian Mixture Model (GMM) of centroids, is placed on data points of the first set. The algorithm then maximizes the GMM posterior probability using the second set by iteratively moving the data points of the first set until both point sets are aligned and the posterior GMM probability reaches its optimum.

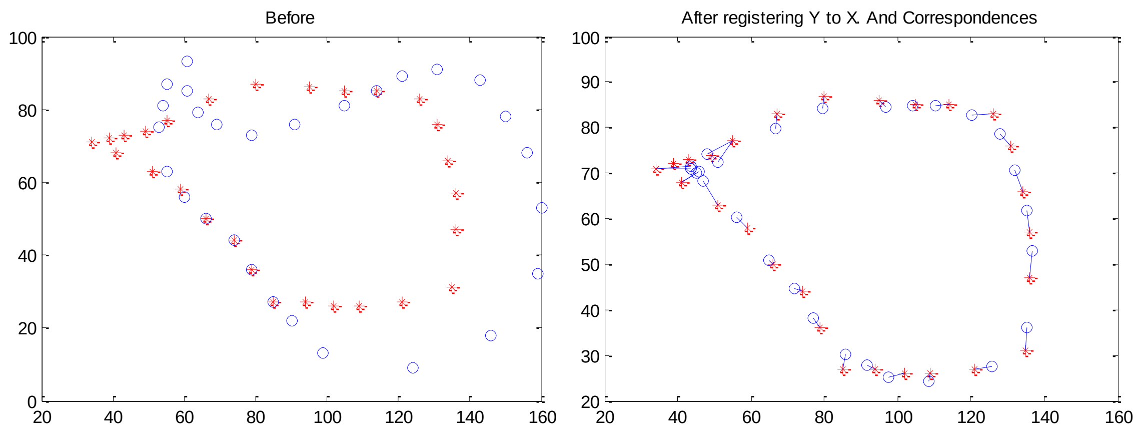

Figure 4 - CPD Non-Rigid registration

We use the following notations:

- X – target point set $\{y_1 \dots y_n\}$
- N – number of points in X
- Y – point set represented by the GMM centroids $\{y_1 \dots y_m\}$
- M – number of points in Y

For a point $x \in X$ we can define the probability density function depending on every point $y \in Y$as a sum of Gaussians. For the 2D case the posterior probability of the

GMM centroid is given by $p(x|m) = \frac{1}{2\pi\sigma^2}\exp^{\frac{\|x-y_m\|}{2\sigma^2}}$. Details of CPD algorithm are beyond the scope of this report.

In the non-rigid CPD algorithm the Gaussian kernel width is controlled by the β parameter and the regularization is controlled by the λ parameter which balances the maximum likelihood and regularization terms. Both parameters are used to tune the smoothness, sparsity and rigidity of the registration result.

1.3.2.2 CPD algorithm parameters

Input:

X - First point set

Y - Second point set (the GMM centroids)

λ , β - Parameters which affect smoothness, sparsity and rigidity of the registration. λ is a regularization value that affects the rigidness of the point set motion, β controls the width of the GMM centriod (σ^2).

Output:

W- Non-rigid transformation coefficients matrix. This matrix is used with the Gaussian kernel matrix G to compute the aligned point set $T = Y + GW$

c - Correspondence vector, such that $X[c_i]$ corresponds to $Y[i]$. This is computed by finding the pairs of points which have maximum probability to be aligned with a corresponding centroids

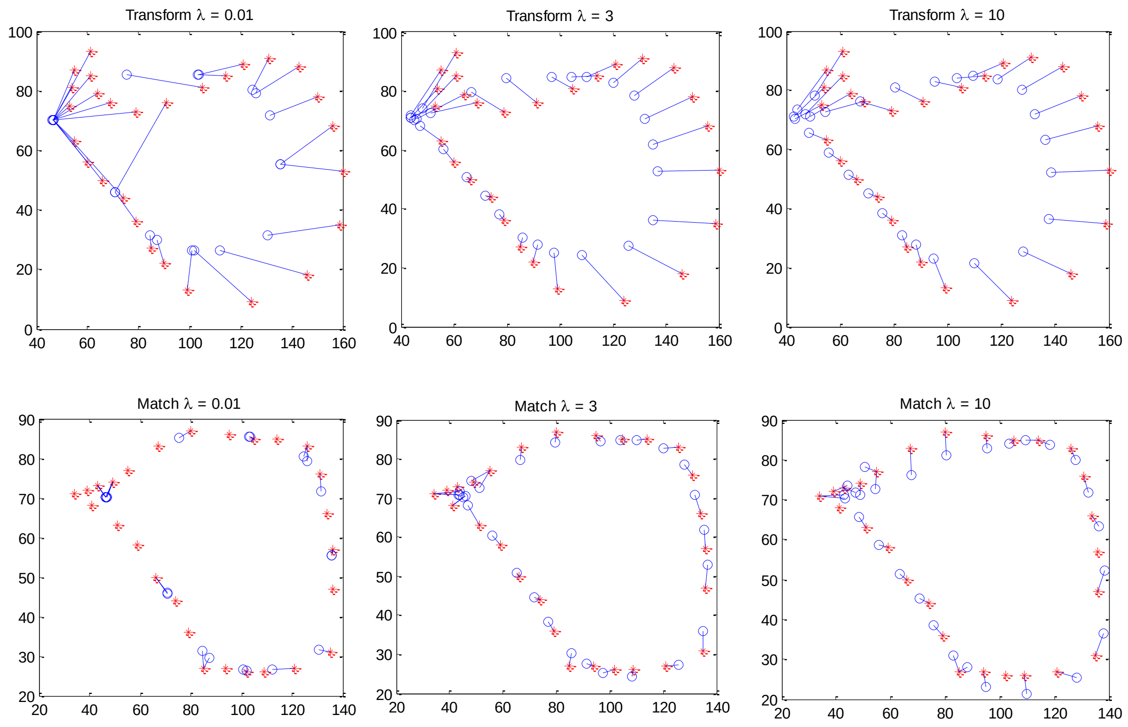

Figure 5 – Effects of λ parameter on CPD registration

As Figure 5 shows, choosing the value of the λ parameter is critical for successful non-rigid transformation. When its value is too low the registration becomes extremely non-rigid, resulting in higher sensitivity to outliers and over-fitting. High values of λ create rigid-like transformations which might be not flexible enough for matching non-rigid animations. In this work we used the value λ=3 as it produces good behavior for different kinds of animations.

We also use $\beta = 1$ since it is the best option for equally distributed normalized point sets.

2 Overview

The input to our system is a sequence of outline drawings created by an animator using vector graphics software. A drawing consists of a set of “paths”. Matching is done between paths in consecutive frames and allows path-related information to be propagated throughout the sequence. For example the path identified as a boundary on one of the frames is set as such on all others. The same applies for paths with crease points. Finally the frames are shaded using the Lumo [10] algorithm which results in a coherently shaded animation sequence.

2.1 Algorithm

- Stage 1: The animation sequence is created by the animator using Bezier curves.
- Stage 2: Path registration:
  - Match similar paths for every two consecutive frames in the sequence
    1. Sample all the paths in both frames to get two point sets
    2. Register two point sets using the non-rigid Coherent Point Drift registration algorithm [16]
    3. Use “path of origin” for every point to perform a voting algorithm (inspired by [17]) to match the paths between the frames (Figure 6).
- Stage 3: Manual adjustments
  - Paths are manually set to boundary or crease by the animator (Figure 8)
  - Information is automatically propagated to other frames using the registration
  - Incorrect or missing registration can be adjusted manually
  - Lumo intensity can be set manually for each path
- Stage 4: The sequence is shaded using an efficient implementation of Lumo algorithm (Figure 9)

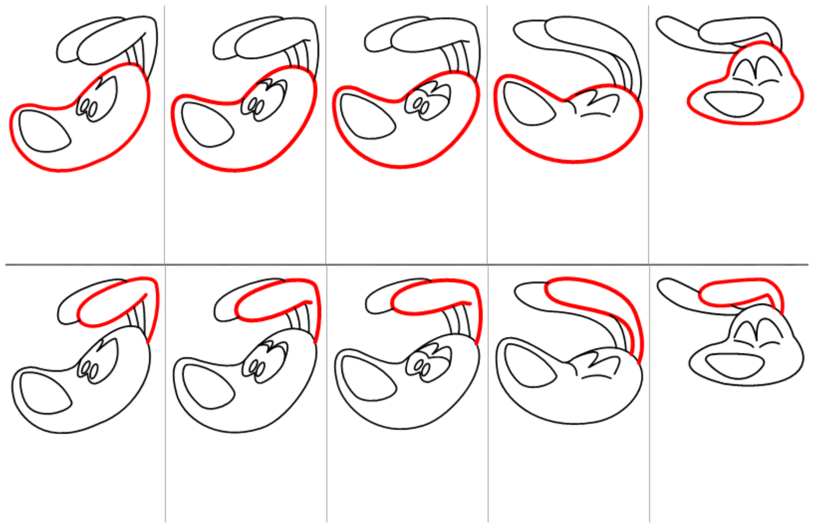

Figure 6 – Face and ear path tracking

3 Implementation

3.1 The Animation Sequence

The input animation sequence is created using a vector drawing application. In each frame the animator creates a set of Bezier curves which composes the drawing outline. The frames are saved in SVG vector graphics format with a file naming scheme coding their position in the sequence.

3.2 Registration Algorithm

1. Two SVG frames (F_i, F_{i+1}) are parsed and converted into parametric Bezier curve representation.
2. Each curve in a frame $c \in F_i$ is designated by its frame number $0 \leq i \leq n$ and the curve id $0 \leq j \leq m_i$ as $c_{i,j}$, where n is number of frames and m_i is number of paths in a frame i.
3. Every path is sampled to a sufficient number of sample points (the set of sample points for path j of the frame i is $P_{i,j}$).
   The number of samples is chosen so that it is proportional to the curve length.
4. Every point $p \in P_{i,j}$ is provided with "path of origin" data – a pair (i, j)
5. Registration is performed between every two point sets PS_i and PS_{i+1}.
   While $PS_i = \bigcup P_{i,j}$ and $PS_{i+1} = \bigcup P_{i+1,j}$ (**Error! Reference source not found.**)

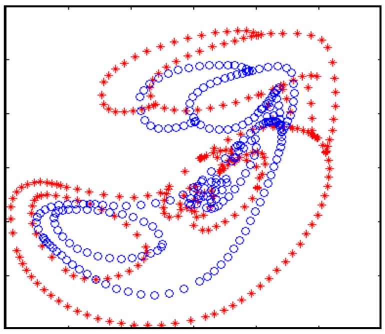
Figure 7 - Two point sets PS_i and PS_{i+1} during the CPD registration

a. The registration uses the non-rigid Coherent Point Drift (CPD) algorithm implementation in Matlab (provided by its author).
b. Two point sets PS_i and PS_{i+1} are provided as input.
c. **Non-rigid** registration is performed from PS_i to PS_{i+1} and from PS_{i+1} to PS_i
d. The registration order with the minimal error (MSE of the distance between closest points) is chosen.
e. Now every point $p \in PS_i$ has a matching point $p` \in PS_{i+1}$

6. In order to match paths between the two frames we employ the voting scheme:
   f. Build a voting matrix $V_{m_i \times m_{i+1}}$ and initialize it to zero.
   g. For every path $c_j \in F_i$ and $c_k \in F_{i+1}$ count the number of matches between samples of c_j and c_k according to the point sets registration and store the result in the voting matrix V
   h. To get the most probable matches between the paths, only c_j and c_k which have maximum number of votes with respect to each other, rather than other paths, are matched: $\max\left(row_j(V)\right) = \max\left(col_k(V)\right)$ A threshold check is also performed on an absolute number of votes to filter out poorly-matched paths.
   i. The result is a mapping between the paths of frame F_i to the paths of F_{i+1}

3.3 Manual adjustments

The animator manually chooses a shading behavior of the path in the first frame, e.g. selects a path to be a boundary or a crease and adjust the direction and strength of the shading. The selection is automatically propagated to subsequent frames using the path registration data (Figure 8).

If the registration fails to register matching paths correctly, the animator can register the paths manually using the UI. The information then is again automatically propagated for newly matched paths.

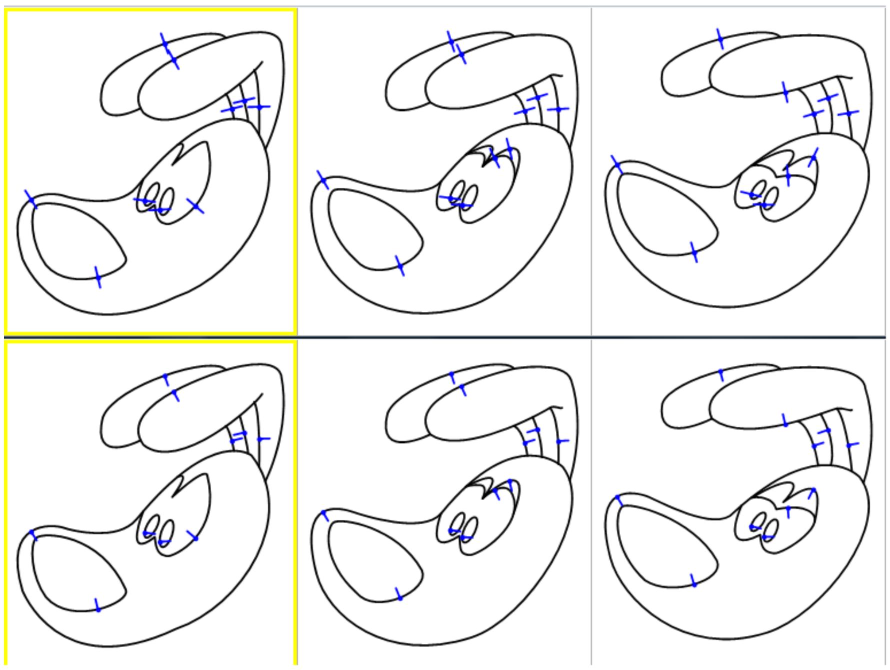

Figure 8 – Top: All unedited paths are assigned to be creases by default. Bottom: After editing, the first frame path information is propagated to the following frames. Blue marks represent each path normal presence and direction

3.4 Shading

When the animator decides the sequence is ready, it can be shaded using a Lumo-like algorithm.

Each path is sampled to the desired pixel resolution and for each pixel on the path we assign a normal direction (n_x, n_y) according to the animators input from the UI at the earlier stage.

The normal data is interpolated into the interior pixels by solving the homogeneous Laplace equation $\nabla^2 f = 0$, where f is x or y component of the normal with the normal vectors on both sides of the path used as boundary conditions. These should be given as Dirichlet boundary conditions when the normal direction is defined or Newman boundary conditions if no normal vector is set by the animator. The z component is calculated afterwards as $n_z = \sqrt{(1 - n_x^2 - n_y^2)}$. The final images are created using Lambertian shading, where every pixel color is determined according to the light source position and the computed normal (n_x, n_y, n_z).

There is a clear difference between the frames shaded with normals allowed to point in both directions of all the paths in the frames and frames shaded where the path normals were adjusted according to the animator desire to emphasize a certain feature or to create a specific depth effect. Figure 9 shows an example of shaded frames with and without editing of the normal direction.

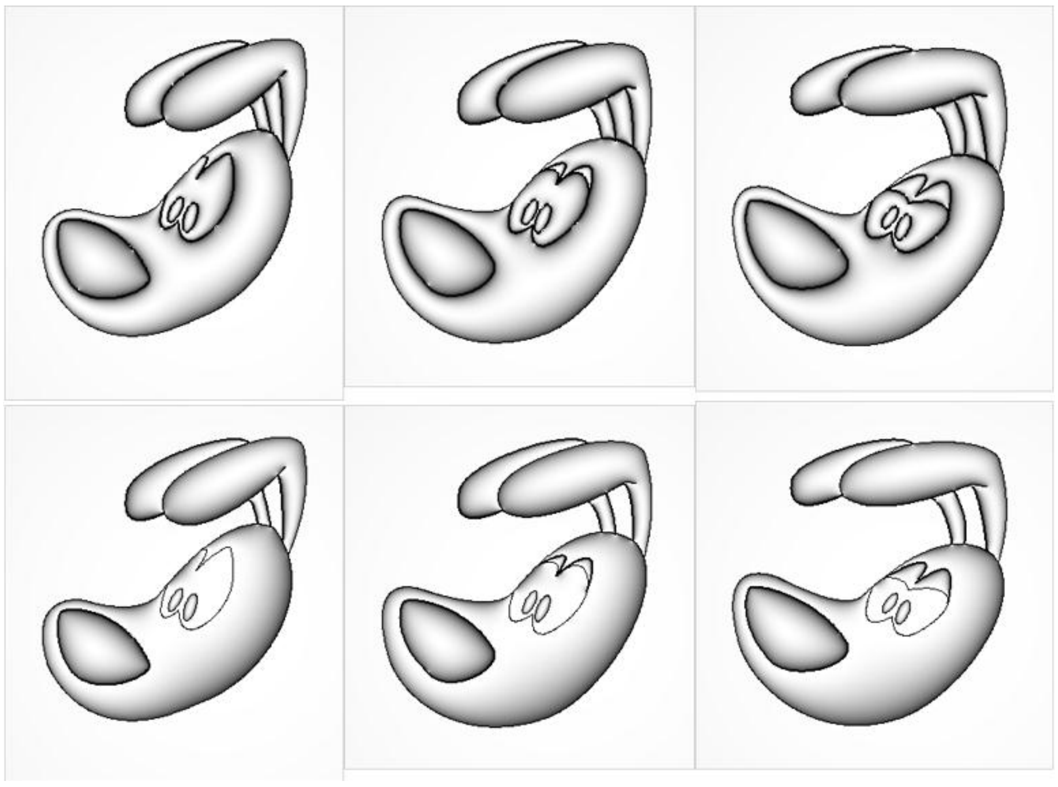

Figure 9 – Top: Shading of unedited paths. Bottom: Shading of adjusted and propagated paths

4 Software System Architecture and Design

Our system is a tool to assist an animator. The complete software application was designed and implemented to allow loading, editing and producing the output animations. The system was implemented in Python with portions of it (e.g. CPD) use existing Matlab implementations.

4.1 Python

Python is the language of choice for implementation of a large part of this application. It is an interpreted, interactive, object-oriented programming language. Python combines power with very clear syntax. It has classes, exceptions, very high level dynamic data types, dynamic typing and modules which can be written in C or C++. The language comes with a large standard library that covers areas such as string processing, software engineering and operating system interfaces.

Version 2.7 was used due to availability of some storage-related libraries and better compatibility than Version 3.x

4.2 System Overview

The system consists of a Graphic User Interface (GUI), logic, input/output, matching and registration and shading modules, as described in Figure 10.

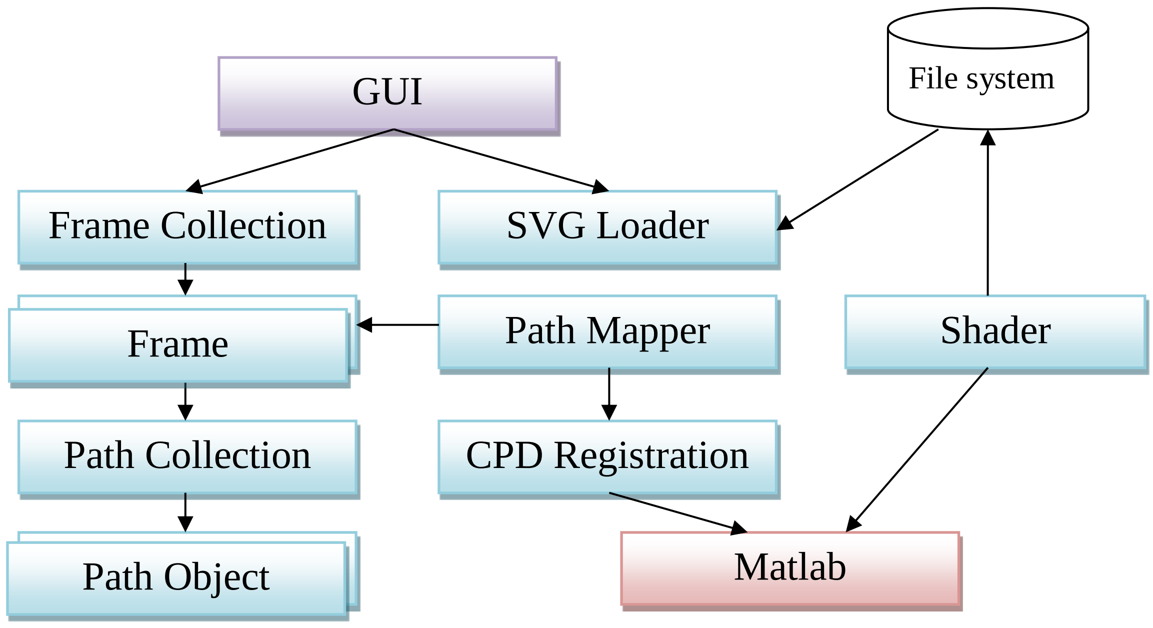

Figure 10 - System Modules Overview

4.2.1 User Interface

The GUI module assists the animator to visually inspect the animation sequence and validate the results of the automatic matching algorithm. When the automatic matching fails, the animator can adjust correspondences and select the direction of normals for the shading part. To support this usage model, the GUI has three major modes of operation (Figure 11):

- Examine frames – visually surveying if the path matching was successful. Hovering over each path will highlight all the matched paths in the sequence. (Figure 12)
- Manual path matching and propagation. If the animator discovers the path matching between two consecutive frames is incorrect, he can modify the match manually by clicking on the correctly matched paths. The match will be automatically propagated for all consequent frames (Figure 8).

- Selection of normal direction for shading. The frame shading is affected by defining the direction and existence of the normal for the specific path. This will be eventually translated into boundary conditions for the path and transferred to the Lumo shader (Figure 13).

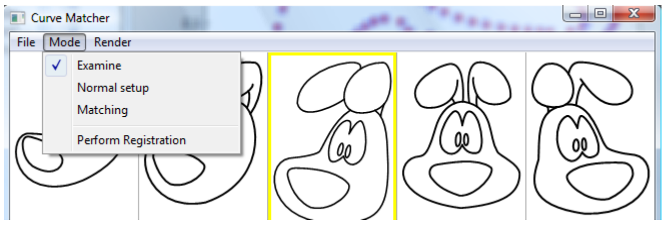

Figure 11 - GUI mode selection. Examine frames, normal setup and matching options.

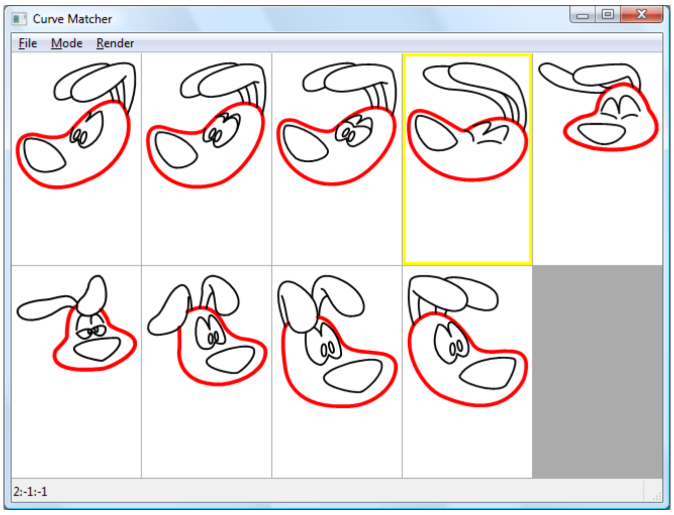

Figure 12 - Examine the path match for the sequence of frames

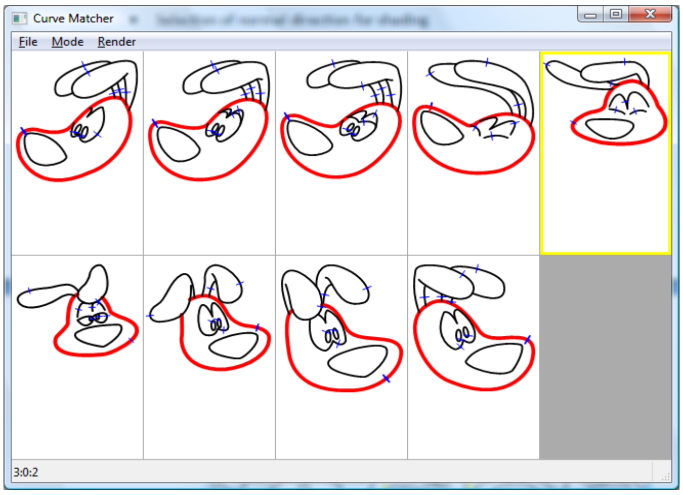

Figure 13 - Normal selection. Each path normal can be changed in this mode, and will affect the final shading.

4.3 Matlab

The Matlab implementation of the Coherent Point Drift (CPD) algorithm [18] was adopted from the author's site and used without significant changes. The algorithm was used in its faster form which makes use of the Fast Gaussian Transform and Gaussian kernel approximations.

The Lumo shading method was also implemented in Matlab. The core method behind the shader is the solution of the Laplace equation with Dirichlet boundary conditions. This method requires operation on the sparse matrices which represents the linear system required to solve the problem. While Python is good for application logic, GUI and input processing, Matlab performs much better in linear algebra related problems.

In order to transfer the data from the Python layer into Matlab we use Python COM object support. Matlab is accessed directly from Python through the "matlab.application" object and its interface.

5 Results and Conclusions

5.1 Input creation

In order to test the system we had to create or reuse existing animations. Because all the animations we used in this work originated in hand drawings, we had to translate all the animation frames into vector graphics format. This was achieved by tracing the pixel images of each frame manually and saving the resulting images in the SVG format.

Several tools were tested for that purpose. The only one that provided best tracing capability along with simple output format structure was the Inkscape graphics editor. The example of the frame transition shows that some small details were lost during the conversion; however general form and path complexities were preserved, as can be seen in Figure 14.

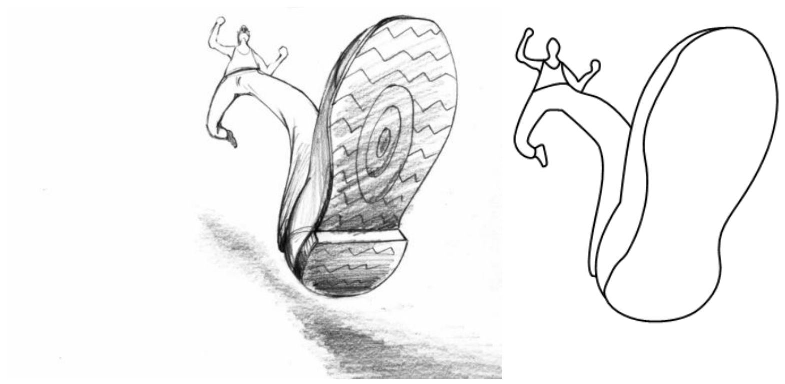

Figure 14- Pixel frame (left) conversion to the vector graphics form (right)

5.2 Load and Display of the Sequence

When the input is ready in SVG format, it is loaded by the application and displayed. At this stage every animation frame is displayed showing the path data from the SVG file. The paths are sampled according to screen resolution and rendered on the screen. The animator is able to explore each frame by hovering the mouse over paths. Since no path matching has been done yet, every path is highlighted separately.

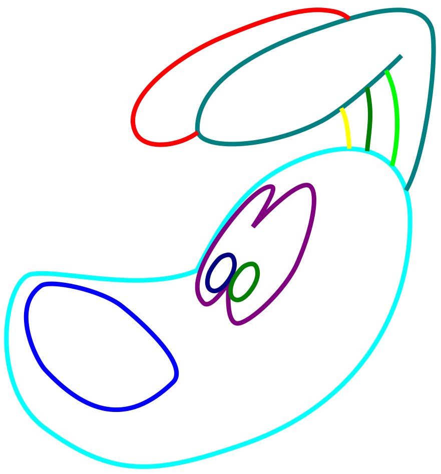

Figure 15 - Illustration of paths for a single frame

5.3 Automatic Registration

After the automatic registration and matching is completed the results are displayed. The matched paths could be explored by the animator if he hovers with the mouse over a specific path. Although most of the sequence will be matched successfully, some will still require manual adjustment by the user. Figure 16 demonstrates automatic registration results.

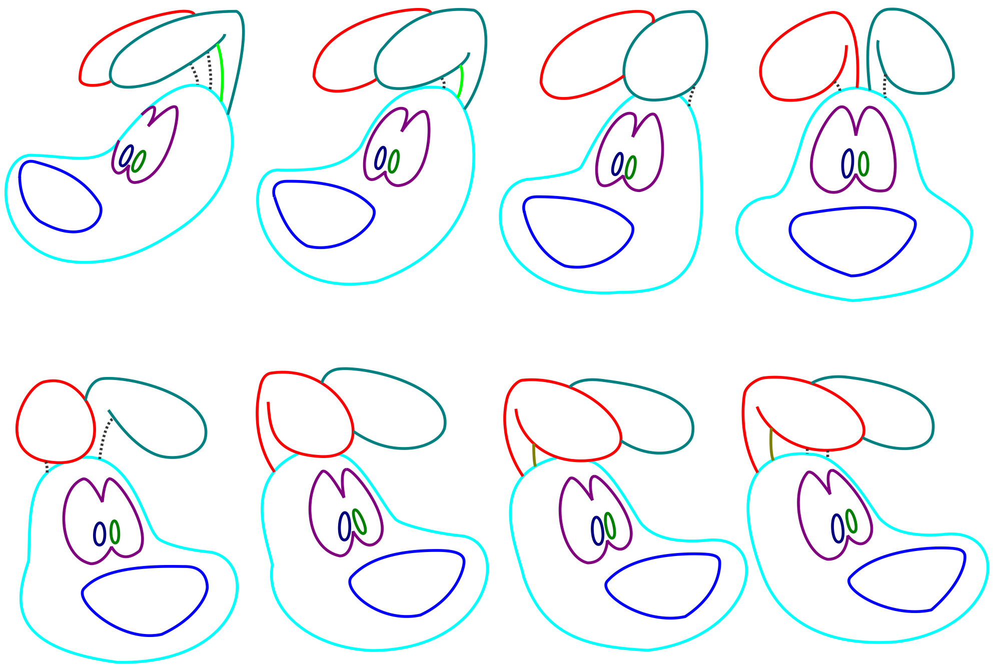

Figure 16 - Illustration for path registration and matching result. Dotted lines signify paths that were not matched by the automatic registration.

5.4 Manual Adjustments of the Shading Direction

Following registration, the animator specifies each path behavior in a shading process. While the default behavior chosen for every path is of the crease type (Figure 17), the animator can manually override it to create a more natural look of the shaded frame. The system will then automatically propagate the changes from the first frame to the subsequent ones (Figure 18). Some paths will not have a correct matching due to failure of the automatic matching (Figure 19) and their shading direction will have to be adjusted manually per frame.

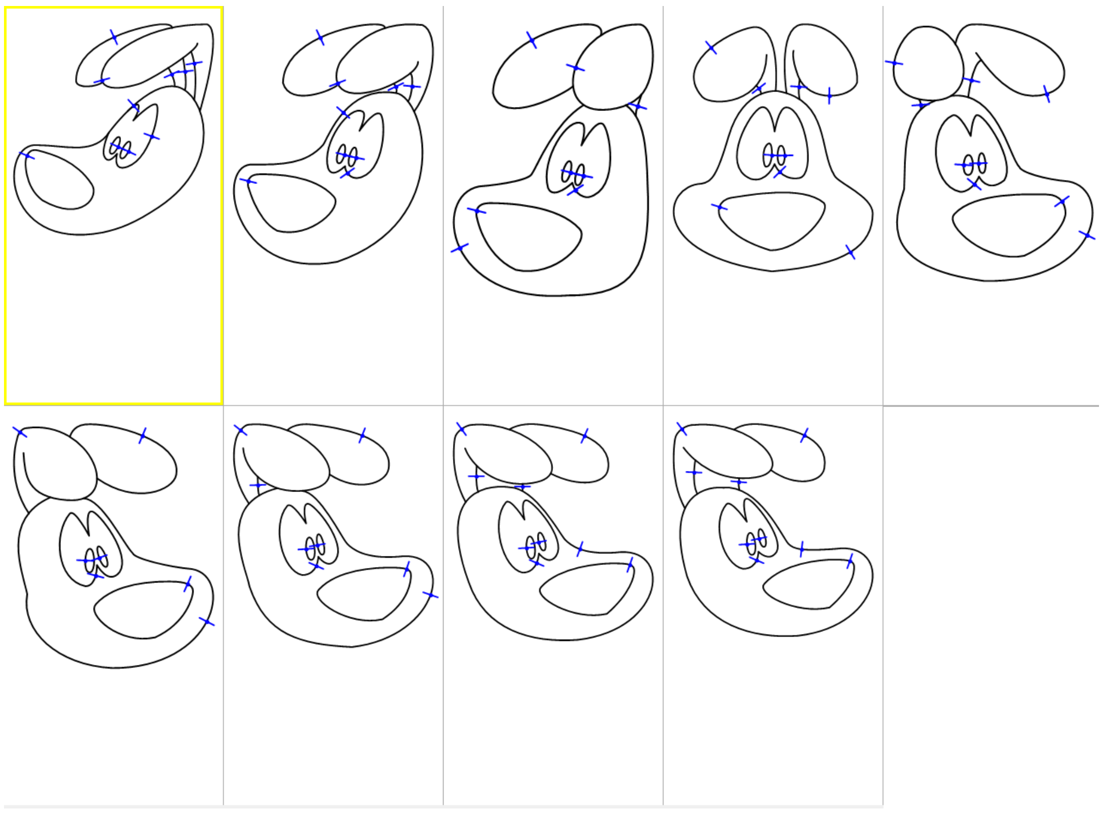

Figure 17 – Default path shading direction assignment. For many paths the direction has to be adjusted to give visually pleasant results in 3D shading.

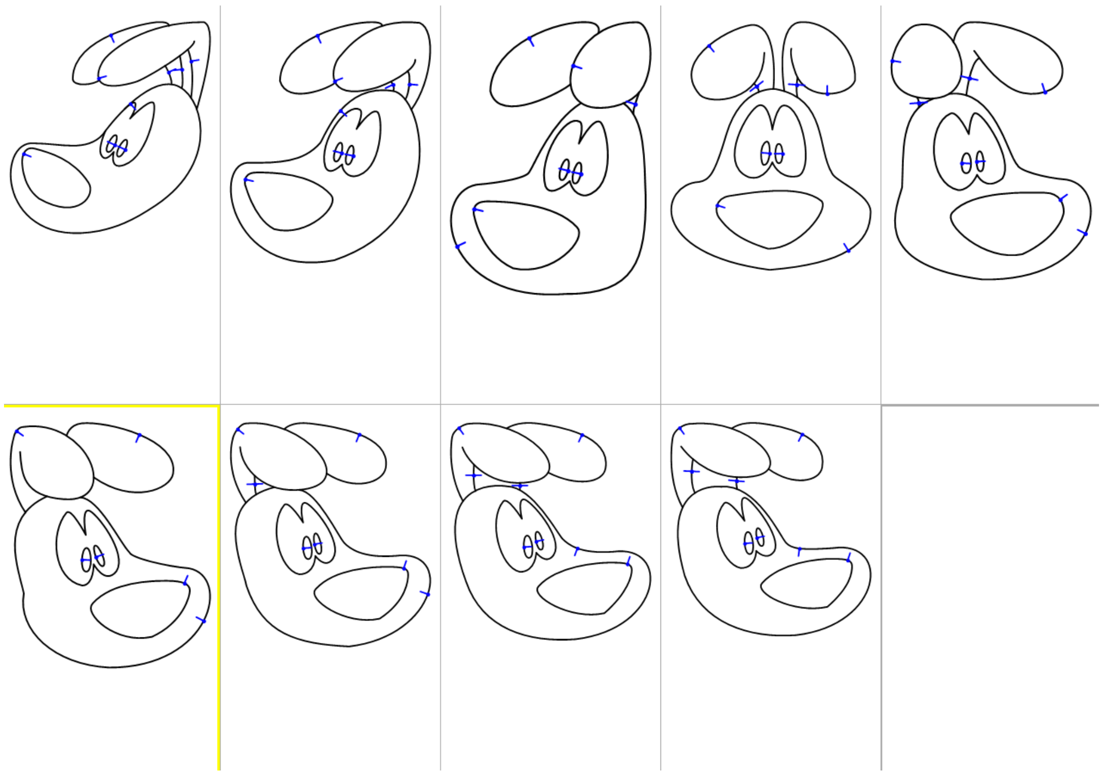

Figure 18 – Settings of the desired shading direction for the first frame only. Most paths will have a correct setting propagated from the first frame thanks to the automatic matching.

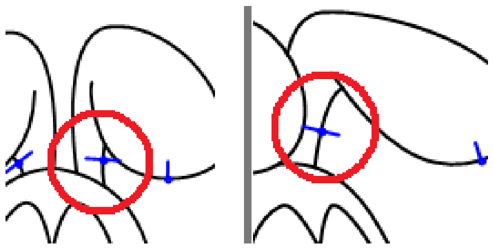

Figure 19 - Unmatched paths in frames 4 and 5 have incorrect directions and must be manually adjusted by the animator.

5.5 Shading of the Sequence

When the animator is satisfied with the adjusted sequence all the frames are shaded and rendered into a complete animation sequence.

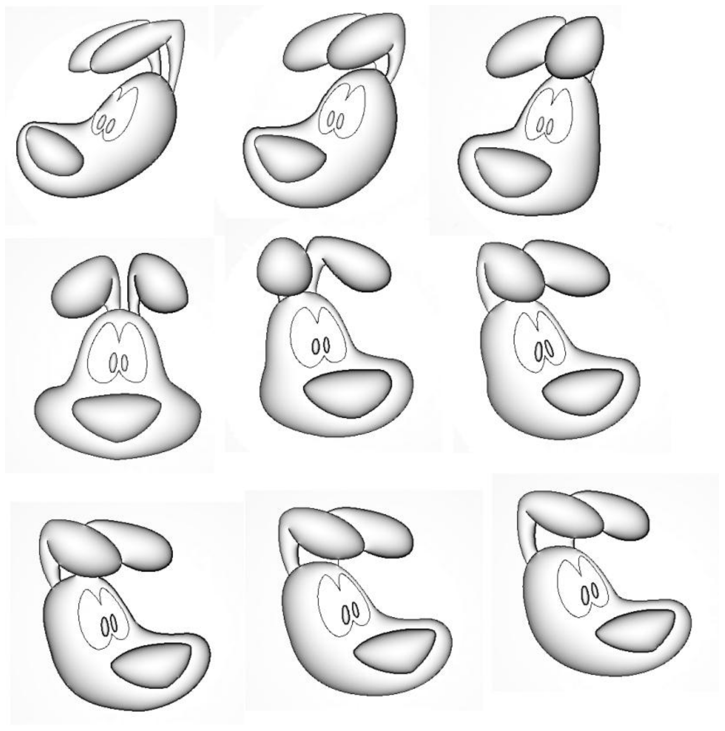
Figure 20 - Shaded dog animation sequence

The whole process can be repeated until the animator is fully satisfied with the result.

5.6 Additional Examples

The system can be used for shading a wide range of animation sequences. Figure 20 shows the shading result of a relatively simple dog animation. There are quite small changes over frames which the dog head undergoes. Most of the shape details change in a near-rigid way.

More complex examples include a second “Dog” animation (Figure 21) and the “Man with a Hat” animation (Figure 22). Both animations exhibit non-rigid behavior in various parts of the sequence. For example, the Dog animation contains highly non-rigid transformation of the dog’s ears along with rotational movement of the entire character, yet the system is very successful in matching the paths, even those that participate in the non-rigid transformation of the character.

The “Running Man” animation (Figure 23) has the most extreme non-rigid deformation, even between consecutive frames. In this sequence the automatic matching is less successful, and does only part of the work required by the animator. There are many incorrect or missing matches and we will discuss the reasons for those in the analysis section.

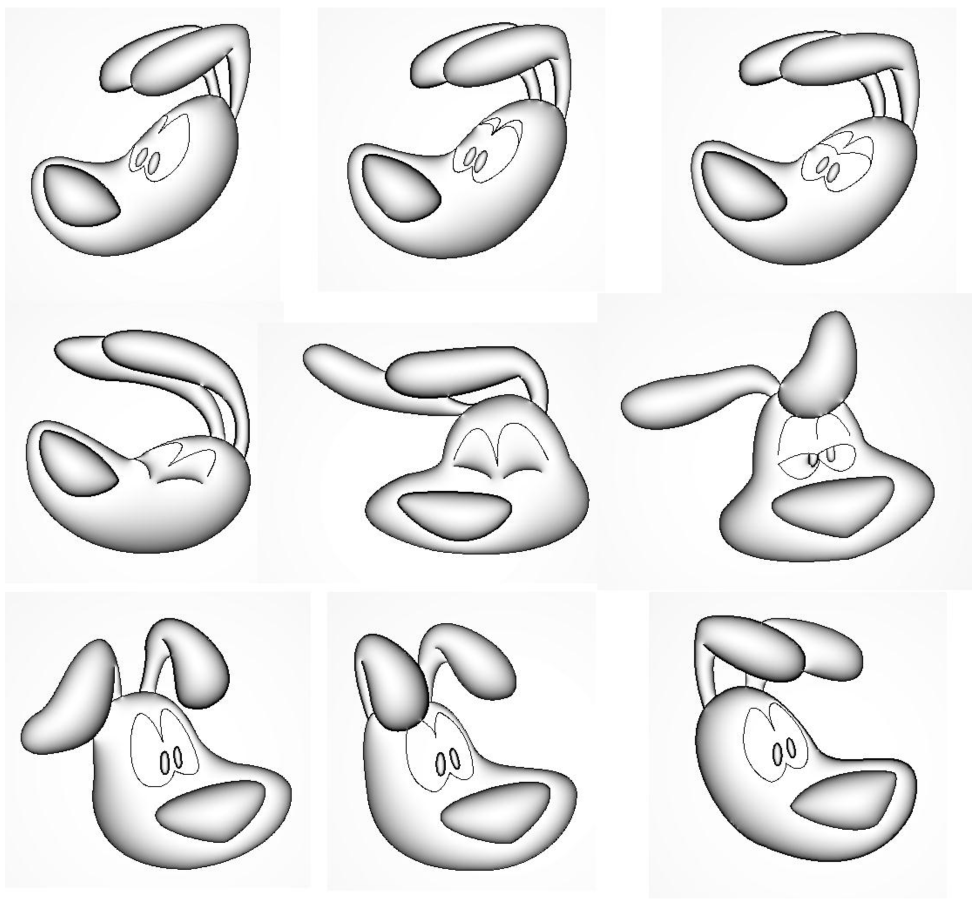

Figure 21 – “Dog” animation with complex motion.

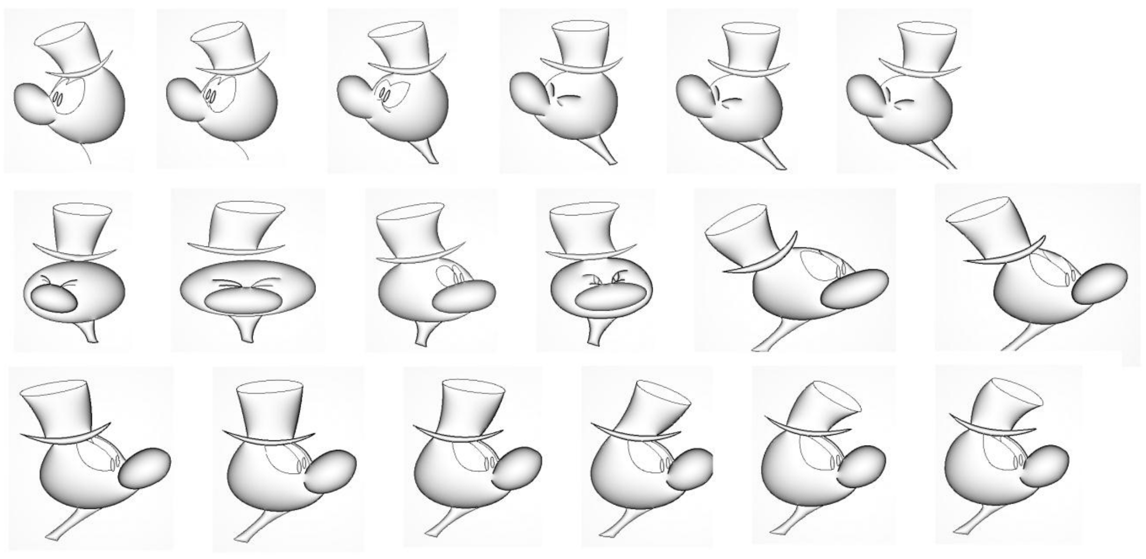

Figure 22 – "Man with a Hat" animation. Complex animation with 18 frames.

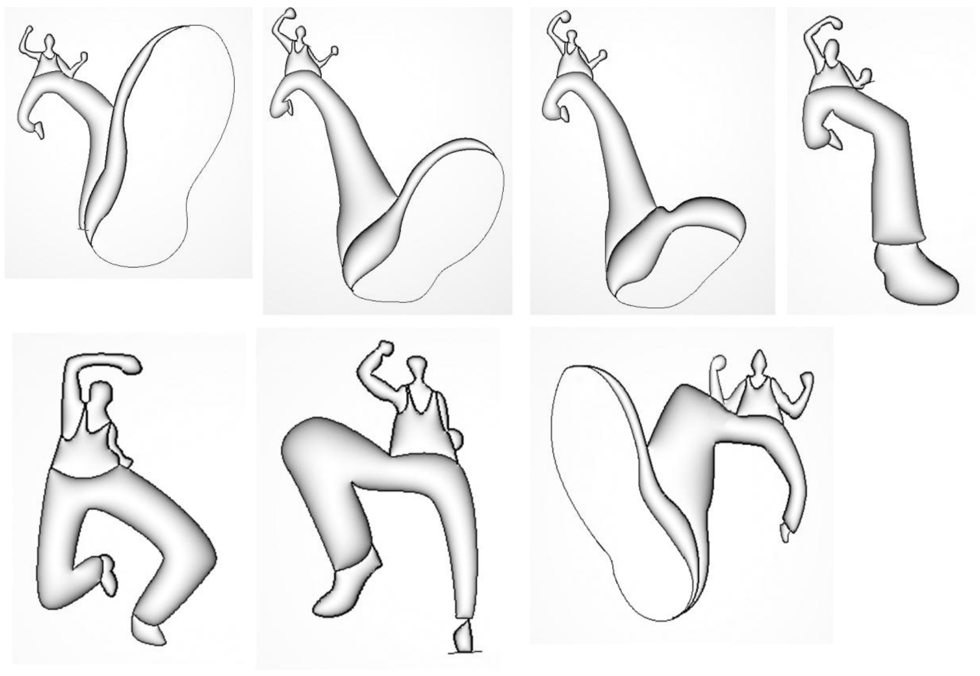

Figure 23 – "Running Man" animation

5.7 Results Analysis and Conclusions

5.7.1 Quantitative analysis of the results

As shown above, creating a Lumo-shaded animation without adjusting the directions of the path normals can result in sub-optimal shading quality. On the other hand, adjusting each normal separately for each frame is extremely tedious and impractical.

The implemented system makes creation of a coherently shaded animation from a sequence of frames a much easier task. To quantify the efficiency of the system, we counted the number of clicks an animator needs to perform on a single sequence in order to create a desired effect.

Table 1 - System Effectivness (in # clicks)

Sequence name	#Frames	Without propagation	With propagation	Improvement Ratio
Simple Dog	9	97	22	4.4
Complex Dog	9	91	25	3.6
Men with a Hat	20	223	23	9.7
Running Men	7	61	22	2.8

As shown in Table 1, the system improves x3 – x4 times the effort it takes to create a shaded sequence. For long sequences with a small number of registration failures, the improvement could be as high as x10.

5.7.2 System Limitations

Although the goal of the system is to automate the animator's work as much as possible, there are cases when the system fails to achieve that. Most of such failures are easily fixed during the manual adjustment stage, and the user interface allows the animator to examine and resolve the errors or misses introduced by the automatic matching. In some cases however the work might become as tedious as adjusting the whole sequence by hand. In this section we will examine the most frequent cases of automatic path matching failure.

5.7.2.1 Too many small details or extreme difference of scale

Small details such as eyes or ears of a figure or emphasis of minor boundaries and cusps are a challenge for the matching algorithm. The problem arises from the fact that in order to match two paths we require a certain number of voting samples to match. Although the paths are sampled according to arc length, for even distribution of the samples on the frame the actual path length is also taken into account. The reasoning behind this is to achieve more stable behavior for paths of significant length, than the shorter ones. However, the side effect is a lower number of samples on small features, which are usually represented by shorter paths.

Suppose that a significant transformation is dictated globally by the long paths, yet the short paths are also undergoing local, but somewhat different transformation. In this case the matching algorithm can fail to produce a transformation which satisfies the minimal number of votes required to correctly match the short paths. Figure 24 demonstrates that a small reduction in a feature size (of a "nose" in this case) may prevent it from being matched correctly.

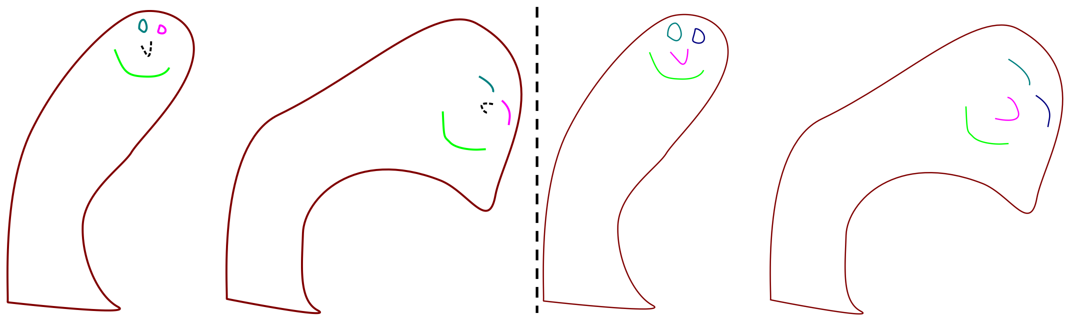
Figure 24 - Left: smaller scale "nose" is not matched correctly. Right: all the details are matched correctly

5.7.2.2 Occlusions or topological changes in character appearance

Occlusions and topological changes occur quite frequently during animation sequences. An example for those could be any of the following: the character moves an arm and it covers other part of its body, the character turns away from the camera and some of its features are not visible anymore or the character closes its eyes or clenches an open hand into a fist.

Our system does not handle occlusions or topological changes specifically, but rather avoids incorrect matches in these cases and leaves most of the work for the manual adjustment phase. The rational for this is that usually such changes do not take place sequentially over several frames. The animator then can connect two correctly matched subsequences simply by attaching one to another at a single frame.

5.8 Summary and Future Work

In this work we have described a system that allows animators to achieve visually pleasing results while saving a fair amount of manual work. The results can be examined and adjusted further, assisted by the system's user interface.

Although many challenges were addressed in the course of this work, even greater ones still remain in order to achieve a holy grail of fully automatic matching and shading. The most challenging is handling of occlusions. Resolving this limitation will vastly improve the usefulness of the system and increase the number of sequences that can be handled automatically.

6 References

[1] D. Sýkora, D. Sedlacek, S. Jinchao, J. Dingliana, and S. Collins, “Adding Depth to Cartoons Using Sparse Depth (In)equalities,” *Computer Graphics Forum*, vol. 29, no. 2, pp. 615–623, Jun. 2010.

[2] D. Sýkora, J. Dingliana, and S. Collins, “As-rigid-as-possible image registration for hand-drawn cartoon animations,” *Proceedings of the 7th International Symposium on NonPhotorealistic Animation and Rendering NPAR 09*, vol. 1, no. 212, p. 25, 2009.

[3] B. Whited, G. Noris, M. Simmons, R. W. Sumner, M. Gross, and J. Rossignac, “BetweenIT: An Interactive Tool for Tight Inbetweening,” in *Computer Graphics Forum*, 2010, vol. 29, no. 2, pp. 605–614.

[4] C. N. de Juan and B. Bodenheimer, “Re-using traditional animation: methods for semi-automatic segmentation and inbetweening,” in *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 2006, pp. 223–232.

[5] K. Melikhov, *Frame Skeleton Based Auto-Inbetweening in Computer Assisted Cel Animation*. IEEE, 2004, pp. 216–223.

[6] Y. Ono and T. Nishita, “3D Character Model Creation from Cel Animation,” *International Conference on Cyberworlds*, pp. 210–215, 2004.

[7] T. Igarashi, S. Matsuoka, and H. Tanaka, “Teddy: a sketching interface for 3D freeform design,” in *ACM SIGGRAPH 2007 courses*, 2007, p. 21.

[8] J. L. Barron and H. Spies, “Quantitative Regularized Range Flow,” in *In Vision Interface*, 2000, pp. 203–210.

[9] A. Rivers, T. Igarashi, and F. Durand, *2.5D cartoon models*. New York, New York, USA: ACM Press, 2010, p. 1.

[10] S. F. Johnston, “Lumo: Illumination for cel animation,” in *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, 2002, p. 45.

[11] D. Bandeira and M. Walter, “Automatic Sprite Shading,” *2009 VIII Brazilian Symposium on Games and Digital Entertainment*, pp. 27–31, Oct. 2009.

[12] H. Bezerra, B. Feijo, and L. Velho, “An Image-Based Shading Pipeline for 2D Animation,” *XVIII Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI’05)*, pp. 307–314, 2005.

[13] H. Bezerra, E. Eisemann, D. Decarlo, and J. Thollot, “Diffusion constraints for vector graphics,” in *Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering*, 2010, vol. 1, no. 212, pp. 35–42.

[14] K.-U. Bletzinger, “‘Geometric Modeling with Splines’, by Cohen, E., Riesenfeld, R.F., Elber, G., A.K. Peters, Ltd., 2001, ISBN 1-56881-137-3.,” *Structural and Multidisciplinary Optimization*, vol. 24, no. 6, pp. 464–465, Dec. 2002.

[15] D. Jackson, “Scalable vector graphics (SVG),” in *ACM SIGGRAPH 2002 conference abstracts and applications on - SIGGRAPH ’02*, 2002, p. 319.

[16] A. Myronenko and X. Song, “Point set registration: coherent point drift.,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 12, pp. 2262–75, Dec. 2010.

[17] R. D. Kalnins, P. L. Davidson, L. Markosian, and A. Finkelstein, “Coherent stylized silhouettes.” In SIGGRAPH ’03: ACM SIGGRAPH 2003 Papers, pp. 856–861, 2003.

[18] A. Myronenko, “Coherent Point Drift (CPD) Matlab Implementation. https://sites.google.com/site/myronenko/research/cpd.” .

Appendix I

This appendix contains examples for curve matching results from the automatic registration of the animation frames. Figure 25 and Figure 26 show an example of correct matching throughout the entire sequence. Figure 27 is an example of the system's failure to match the character's right eye. This failure is a result of a sudden topological change caused by character closing his eyes.

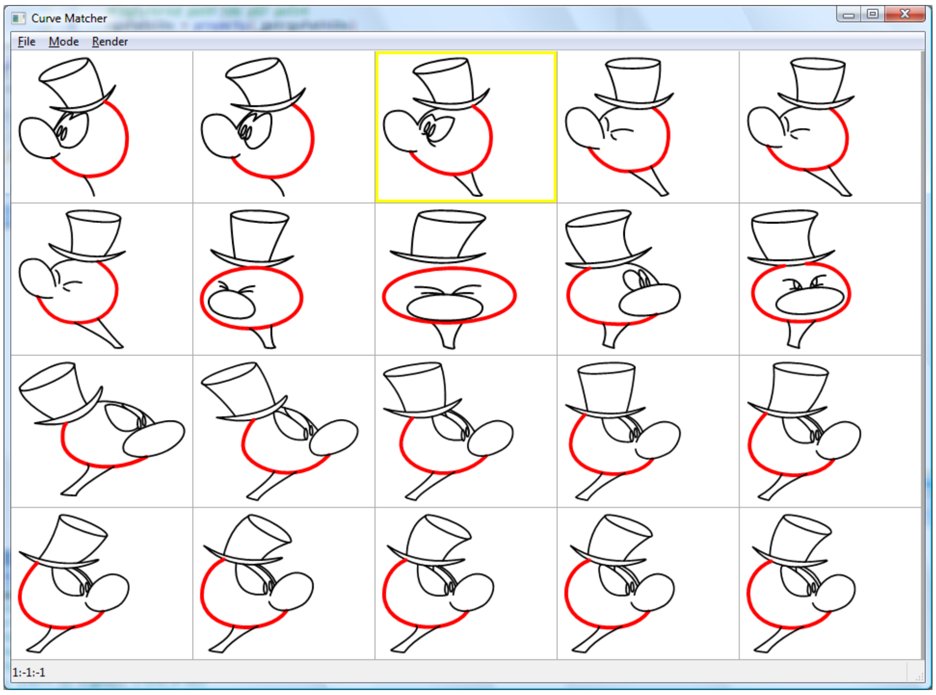

Figure 25 – Automatic matching of the face boundary path.

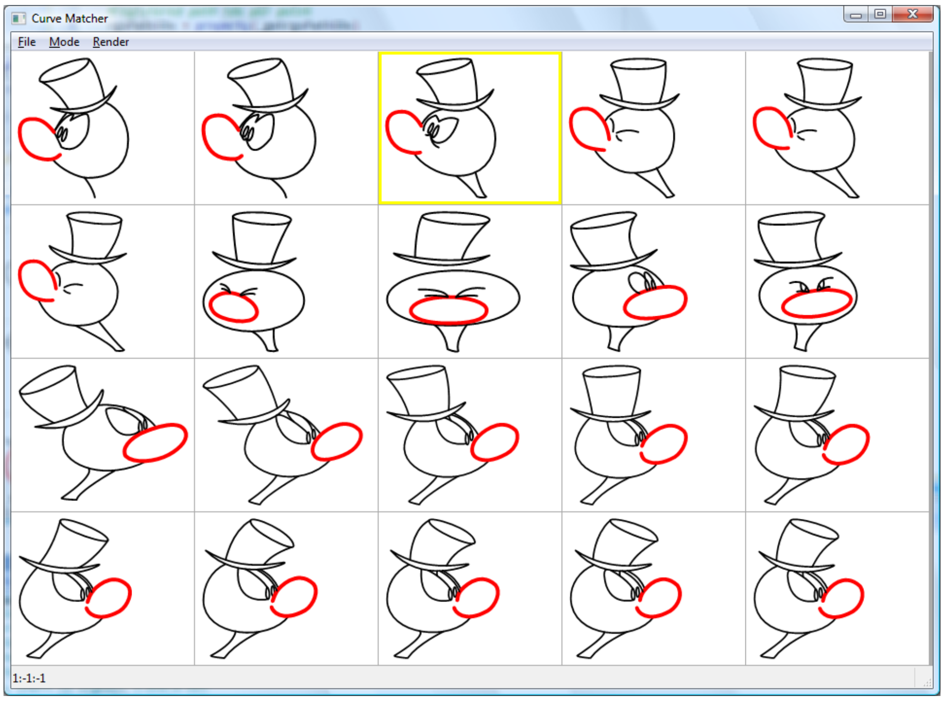

Figure 26 – Automatic matching of the nose path.

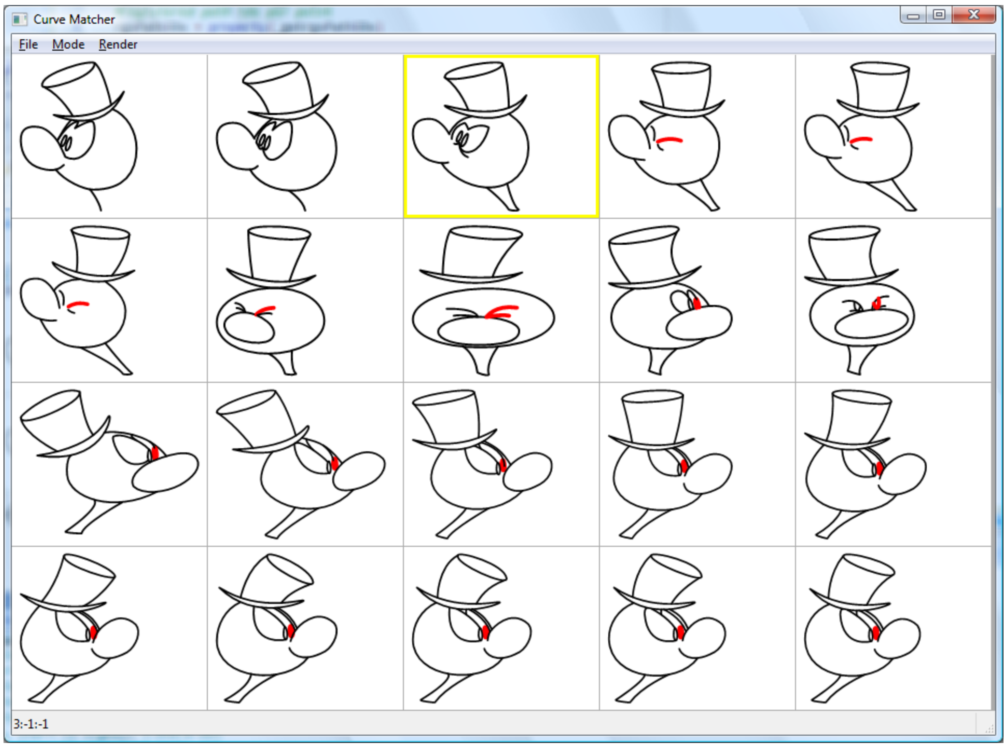

Figure 27- Failure in automatic eye path matching. In this case the path matching must be adjusted manually by the animator.

למרות ההצלחות של המערכת, עדיין קיימים מספר מקרים בהם היא נכשלת בהתאמה אוטומטית ולכן נדרשת עבודה קשה יותר מצד המשתמש, על מנת להגיע לתוצאה הרצויה. מגבלה אחת מופיעה כאשר מנסים לבצע התאמת תמונה, אשר כוללת מספר פרטים גדולים העוברים שינוי בלתי קשיח קיצוני ולצִדם יש פרטים קטנים רבים. מאחר והמערכת מתבססת על כללי הצבעה על מנת להתאים את העקומות זו לזו, יש סיכוי טוב שלא יהיו מספיק הצבעות עבור הדגימות של הפרטים הקטנים וכך לא תוכל להתאים את העקומות כראוי. בנוסף, כאשר יש הסתרה של עקומה מסוימת במהלך האנימציה, למשל, כאשר הדמות המצוירת מסתירה את ידיה מאחורי הראש או מסתובבת הצידה. האלגוריתם שלנו אינו מטפל בבעיה זו, אך מנסה לזהותה בהסתברות גבוהה ולמנוע התאמה שגויה/ מוטעת. אם בעקבות ההסתרה העקומה נעלמת, מובן שלא נתאים אותה לעקומות האחרות. אבל אם יש עקומה חדשה, אשר מופיעה באחת מתמונות הרצף, המערכת תתבסס על המשתמש, אשר יצטרך להתאימה ידנית לעקומות האחרות.

לסיכום, בעבודה זו הראנו מערכת המאפשרת לאמני ההנפשה להגיע לתוצאות הצללה הדומות לאלו של הצללה תלת-ממדית, אך באמצעות התאמה אוטומטית החוסכת פעולות ידניות רבות. התוצאה ניתנת לבחינה ויזואלית וממשק המשתמש מאפשר ביצוע תיקונים בקלות עד להגעה לתוצאה הרצויה. למרות התמודדות עם אתגרים רבים במהלך עבודתנו, נמצאו מקרים בהם נדרשת התערבות ידנית של המשתמש. פתרון בעיית ההסתרה יאפשר קבלת תוצאה דומה בהתערבות משתמש קטנה יותר ויקרב אותנו צעד נוסף למערכת המאפשרת הצללות מסוג לומו באופן אוטומטי לחלוטין.

לאחר ששתי התמונות עברו את הדגימה, נפעיל על שני ענני הנקודות את אלגוריתם ה-CPD. תוצאת האלגוריתם היא התאמה בין כל נקודה בתמונת המקור לנקודה בתמונת המטרה. נוסף לכך, נחליף את סדר התמונות ונבצע את ההתאמה ההפוכה. לאחר מכן, נשווה את השגיאה המצטברת של כל התאמה ונבחר את ההתאמה שעבורה השגיאה היא הקטנה ביותר. לאחר שהשגנו את ההתאמה של הדגימות נצטרך לתרגם אותה בחזרה להתאמת העקומות בתמונות. הדבר אינו פשוט, כי ייתכנו שינויים טופולוגיים של הדמות בתמונה וגם העקומות חולקות מספר דגימות שהותאמו ביניהם, לעתים בצורה לא רצויה. כדי להתמודד עם הבעיה אנו מבצעים תהליך של הצבעה בין דגימות העקומות. אם רוב הדגימות משתייכות לשתי עקומות שבתמונת המקור ובתמונת המטרה, אנו מכריזים עליהן כמתאימות. אם לא נמצא מספיק הצבעות עבור עקומה מסוימת, לא נמצא לה התאמה כלל.

התהליך שתואר קודם חוזר עבור כל זוג תמונות סמוכות ברצף האנימציה. לבסוף, נישאר עם התאמת העקומות בכל זוג של תמונות. מכאן ביכולתנו לגזור רצף התאמה עבור העקומות שבתמונה בודדת לכל התמונות האחרות ברצף. עד כאן פעולת המערכת היא אוטומטית לגמרי, אבל בסיום השלב הנוכחי נדרשת התערבותו של המשתמש, האנימטור, כיוון שחלק מהעקומות לא הותאמו כלל וחלק מהן קיבלו התאמה שגויה. המערכת מציגה את תוצאות ההתאמה על גבי ממשק משתמש גרפי, בו המשתמש יכול לעבור על כל העקומות של רצף התמונות ולקבל משוב מידי על מצב ההתאמה בפועל, לאחר סיומו של החלק האוטומטי. המשתמש יכול לבטל התאמות בין העקומות בתמונות סמוכות או להוסיף התאמה בין עקומות שהמערכת לא גילתה.

כאן מתחיל השלב השלישי והסופי, בו המשתמש יבצע את ההצללה של רצף האנימציה. כיוון שרוב רובן של העקומות כבר עברו התאמה, יצטרך לשנות את כיוון ההצללה ע"י סימון כיוון הנורמל על כל אחד מקווי המתאר של הדמות. הסימון יתבצע על גבי התמונה הראשונה וההחלטות יעברו ללא התערבות נוספות לכל תמונות הרצף הנוספות. מובן שהאנימטור יצטרך להשלים את כיווני ההצללה עבור עקומות חדשות או כאלה שנוספו בתמונות אחרות ברצף. מספרם של שינויים אלה ברוב רובם של המקרים לא יהיה גדול. על מנת לייצר את ההצללה נשתמש ברעיון אשר מגיע מאלגוריתם ה"לומו". לומו מציעה להפעיל על הנורמלים של התמונה את מסנן הממוצע באופן מחזורי, עד שמתקבלת התוצאה הרצויה. אנו מממשים ברעיון זהה ע"י פתרון משוואת Poisson (סוג של משוואת חום). בפתרונה של משוואה זו נשתמש בתנאי קצה מסוג Dirichlet במידה והנורמל הוגדר ע"י האנימטור או מסוג Newman אם אין הגדרה של כיוונו. תהליך זה מייצר בסופו של דבר תמונת הצללה בעלת מאפיינים של תמונה תלת-ממדית ומראה נעים לעיניים.

תקציר

הוספת אפקט עומק או מראה תלת-ממדי לרצף אנימציה דו-ממדית יכול לספק תוצאה נעימה לעין, גם ללא המרת הדמות לתלת-ממד מלאה. בעבודה זו נציג שיטה ומערכת המאפשרת לאנימטור לבצע הצללה קוהרנטית של תמונות באמצעות מחשב, כאשר הדמות המופיעה בהן יכולה לעבור שינויי צורה בלתי קשיחים.

כיום, אנימציה היא תחום רב שימושי ומשמשת לייצור סרטי בידור וחומרים חינוכיים רבים. בעבודה זו התמקדנו ביצירת רצף תמונות המספק את תחושת התלת-ממד מתוך רצף קיים של תמונות דו-ממדיות. התמונות ברצף יכולות להיות ציור שבוצע ידנית על הנייר או לחלופין על גבי מחשב ומהוות הנפשה של דמות מצוירת המבצעת תנועה מסוימת. ברוב המקרים הדמות המצוירת תעבור שינויים בלתי קשיחים במישור הדו-ממדי. דוגמאות לכך יכולות להיות אוזניים של כלב, אשר משנות את צורתן ושטחן מתמונה לתמונה, או אדם שהולך לכיוון המצלמה, כך שאיברי הגוף השונים שלו מתקרבים ומתרחקים אל הצופה.

הקושי העיקרי ביצירת אנימציה עם הצללה אוטומטית היא בחירת כיווני העומק שלה עבור כל חלק וחלק בדמות ושמירת הבחירה באופן אחיד לאורך רצף התמונות. קיימות מספר גישות אפשריות לביצוע ההצללה של דמוית תלת-ממדיות. גישה אחת מאפשרת למשתמש להגדיר את סדר העומקים בדמות המונפשת וכך לאפשר ביצוע הצללת התמונה כראוי. גישה נוספת מנסה לשחזר את הדמות התלת-ממדית מתוך רצף התמונות הדו-ממדי ולבצע הצללה תלת-ממדית ממשית. אך הקושי העיקרי של השיטות הנ"ל נעוץ בהתמודדות עם שינויים משמעותיים של צורת הדמות, במיוחד אם קיימת טרנספורמציה בלתי-קשיחה בין תמונה אחת לאחרת.

אנו מציעים אלגוריתם ומערכת חצי-אוטומטית המייצרת הצללות קוהרנטיות לאורך רצף תמונות אנימציה, גם אם קיימות טרנספורמציות בלתי קשיחות של הדמות. המערכת בנויה ממספר שלבים:

בשלב הראשון נקבל כקלט את התמונות אשר נוצרו ע"י האנימטור על גבי נייר רגיל ועברו תהליך של סריקה ווקטוריזציה. ישנה גם אפשרות שהאנימטור כבר יצר את התמונות בפורמט המתאים, אשר מכיל אך ורק עקומות בזייר (Bezier). עקומות אלו מתאימות לייצוג של תמונות האנימציה ונוחות לשימוש במערכת שלנו.

לאחר שהקלט מוכן, נתאים בין כל שתי תמונות ברצף האנימציה. בבסיס ההתאמה נשתמש באלגוריתם ה-CPD – נדידת נקודות קוהרנטית. האלגוריתם מתאים בין שני ענני נקודות, כאשר השינוי שהם עוברים יכול להיות בלתי קשיח. כדי להגיע למצב, בו יש לנו את ענני הנקודות המתאימים לכל תמונה, נדגום את העקומות. הדגימה מתבצעת לפי אורך הקשת כדי לאפשר פיזור אחיד של הדגימות בתמונה.

המחקר נעשה בהנחיית פרופ' חיים גוטסמן ופרופ'/מ' מירלה בן-חן בפקולטה למדעי המחשב.

אני מודה לטכניון על התמיכה הכספית הנדיבה בהשתלמותי.

מערכת חצי-אוטומטית להתאמה בלתי קשיחה והצללה קוהרנטית של רצף האנימציה

חיבור על עבודת גמר

לשם מילוי חלקי של הדרישות לקבלת התואר
מגיסטר למדעים במדעי המחשב

דני ריבניקוב

הוגש לסנט הטכניון - מכון טכנולוגי לישראל

תשרי תשע"ד חיפה ספטמבר 2013

מערכת חצי-אוטומטית להתאמה
בלתי קשיחה והצללה קוהרנטית
של רצף האנימציה

דני ריבניקוב