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

1 Introduction ............................................................................................................. 3
  1.1 Animation ........................................................................................................ 3
  1.2 Related Work ................................................................................................ 4
  1.3 Background ...................................................................................................... 5
    1.3.1 Bezier Curves .............................................................................................. 5
    1.3.2 Point Set Registration ..................................................................................... 6
      1.3.2.1 Non-rigid Coherent Point Drift ................................................................. 7
      1.3.2.2 CPD algorithm parameters ....................................................................... 8

2 Overview .................................................................................................................. 10
  2.1 Algorithm ......................................................................................................... 10

3 Implementation ....................................................................................................... 12
  3.1 The Animation Sequence .................................................................................... 12
  3.2 Registration Algorithm ...................................................................................... 12
  3.3 Manual adjustments .......................................................................................... 13
  3.4 Shading .............................................................................................................. 14

4 Software System Architecture and Design ......................................................... 16
  4.1 Python .............................................................................................................. 16
  4.2 System Overview .............................................................................................. 17
    4.2.1 User Interface ............................................................................................... 17
  4.3 Matlab ............................................................................................................... 19

5 Results and Conclusions ....................................................................................... 20
  5.1 Input creation .................................................................................................... 20
  5.2 Load and Display of the Sequence ..................................................................... 20
  5.3 Automatic Registration ..................................................................................... 21
  5.4 Manual Adjustments of the Shading Direction .................................................. 22
  5.5 Shading of the Sequence ................................................................................... 22
  5.6 Additional Examples ......................................................................................... 25
  5.7 Results Analysis and Conclusions ..................................................................... 28
    5.7.1 Quantitative analysis of the results ................................................................. 28
    5.7.2 System Limitations ....................................................................................... 28
      5.7.2.1 Too many small details or extreme difference of scale ......................... 29
      5.7.2.2 Occlusions or topological changes in character appearance .......... 29
## Contents

5.8 Summary and Future Work ................................................................. 30

6 References .......................................................................................... 31

Appendix I ............................................................................................. 33
List of figures

Figure 1 – Left: cubic Bezier curve with its control polygon. Right: curve point calculation using de Casteljau algorithm .................................................................5
Figure 2 - Animation with rigid transformation ..........................................................6
Figure 3 - Animation with non-rigid transformation .....................................................6
Figure 4 - CPD Non-Rigid registration ....................................................................7
Figure 5 – Effects of $\lambda$ parameter on CPD registration ......................................9
Figure 6 – Face and ear path tracking ......................................................................11
Figure 7 - Two point sets $\Psi$ and $\Psi + 1$ during the CPD registration...............12
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 ..................14
Figure 9 – Top: Shading of unedited paths. Bottom: Shading of adjusted and propagated paths ........................................................................................................15
Figure 10 - System Modules Overview ..................................................................17
Figure 11 - GUI mode selection. Examine frames, normal setup and matching options. .............................................................................................................................................18
Figure 12 - Examine the path match for the sequence of frames .............................18
Figure 13 - Normal selection. Each path normal can be changed in this mode, and will affect the final shading ..................................................20
Figure 14- Pixel frame (left) conversion to the vector graphics form (right) .......20
Figure 15 - Illustration of paths for a single frame ....................................................21
Figure 16 - Illustration for path registration and matching result. Dotted lines signify paths that were not matched by the automatic registration ................22
Figure 17 – Default path shading direction assignment. For many paths the direction has to be adjusted to give visually pleasant results in 3D shading ..................23
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 .................................................................24
Figure 19 - Unmatched paths in frames 4 and 5 have incorrect directions and must be manually adjusted by the animator .........................................................24
Figure 20 - Shaded dog animation sequence ............................................................25
Figure 21 – “Dog” animation with complex motion. .................................................26
Figure 22 – “Man with a Hat” animation. Complex animation with 18 frames.......27
List of figures

Figure 23 – “Running Man” animation ................................................................. 27
Figure 24 - Left: smaller scale “nose” is not matched correctly. Right: all the details are matched correctly ................................................................. 29
Figure 25 – Automatic matching of the face boundary path. ................................ 33
Figure 26 – Automatic matching of the nose path ............................................... 34
Figure 27- Failure in automatic eye path matching. In this case the path matching must be adjusted manually by the animator ........................................... 34
List of tables

Table 1 - System Effectiveness (in # clicks)................................................................. 28
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.
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPD</td>
<td>Coherent Point Drift registration algorithm</td>
</tr>
<tr>
<td>SVG</td>
<td>Scalable Vector Graphics file format</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>GMM</td>
<td>Gaussian Mixture Model</td>
</tr>
</tbody>
</table>
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](image)

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](image)

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](image)

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](image)

We use the following notations:

- $X$ – target point set $\{y_1 \ldots y_n\}$
- $N$ – number of points in $X$
- $Y$ – point set represented by the GMM centroids $\{y_1 \ldots y_m\}$
- $M$ – 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 \( p(x|m) = \frac{1}{2\pi \sigma^2} \exp\left(-\frac{||x-y_m||^2}{2\sigma^2}\right) \). 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 \( \beta \) parameter and the regularization is controlled by the \( \lambda \) 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)
- \( \lambda, \beta \) - Parameters which affect smoothness, sparsity and rigidity of the registration. \( \lambda \) is a regularization value that affects the rigidness of the point set motion, \( \beta \) controls the width of the GMM centroid (\( \sigma^2 \)).

**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
As Figure 5 shows, choosing the value of the $\lambda$ 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 $\lambda$ create rigid-like transformations which might be not flexible enough for matching non-rigid animations. In this work we used the value $\lambda=3$ 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.
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 the normal 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](image)
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](image)

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)](image)

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](image)

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.
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](image)

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>
<thead>
<tr>
<th>Sequence name</th>
<th>#Frames</th>
<th>Without propagation</th>
<th>With propagation</th>
<th>Improvement Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple Dog</td>
<td>9</td>
<td>97</td>
<td>22</td>
<td>4.4</td>
</tr>
<tr>
<td>Complex Dog</td>
<td>9</td>
<td>91</td>
<td>25</td>
<td>3.6</td>
</tr>
<tr>
<td>Men with a Hat</td>
<td>20</td>
<td>223</td>
<td>23</td>
<td>9.7</td>
</tr>
<tr>
<td>Running Men</td>
<td>7</td>
<td>61</td>
<td>22</td>
<td>2.8</td>
</tr>
</tbody>
</table>

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 fail-
ures 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](image)

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


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.](image)

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.
לאחר הצלחתו של המערכת, עדין קריים מספרים במיליםائه של חד אינטליגנטי בלחמה.

אטרופיתת ולק נורשה עבורה קושי ייחוד פיתוח, על אף שניצלה wcześniej. נוגבва יוחנו

מופיעה חסרהordo מכסום בלעון התאמה התחנה, אשר כולם מספרים גורלם הצפרדעים שלבי

 kısıית קיצוניים אזולים יש פרוסה קצינים רבים,╫אחרים המעוררים מעבשות על כולי הצבעים על פנת חפשי

אנשה את הצלחתו וול, יש סיכום שלא היו מILogger הב(vpועות יונgnore והמשימה על הפרים הקטנים בכק

לחול התחמיית את הצלחתו בריאר, בנפש, אוכלי של השתרטרים על יעコピー מצורפו בתוכלי האירופיות.

øjplement ל организация, אך המסגרת האורחית מצחירה את חיות ממותרות והארים ולא מצורף העציה.

אנני מסכל.mas בודא, ואם מסגרת התחנה כמות הבהיר של אמצעים,"ןורתיה של אינטליגנטית האורחית.

אפור השתרטר הצלחת נעלמה, מובן שלアップ ת בעיקר את הצלחת האורחית. האורחית אמס שיש עדשות, אויר

מופיעה באחונת מזワイン מרום, המ런ת החובסה על המשמשי, אשר יודור לאינטליגנטית עדין לעכיפה

האינטליגנטית.

לפרוסה, בעבורה וחברה מערבות המאפרשית לא限り הגבשה הלעימה להצלחתו שלינולדה

לאחר של הצלחתו של התור, אך א담צעות המהמה אומאפרשית הזוочка פעולות יונgnore והתונה

גירטו ל٬בחלות יונgnoreפלס המשמשי אומאפרשית יונgnore ה镏יונה שלתפת התור.

לפרוסת המ Deadpool הוא אטרופית ברים במקלל עבורה, ונמצאו מלקים שנילדרת התנועה עדינ של

המשמשי. פתרון בטיעית הת사회 אמסר כלכל ותאמה זומת הבכרה המשמשי קמנד יונgnore וקרב

אותה והצג עזב עם המערבות המאפרשית שלינולדה מנסו שלום בצרכי אודומטיז תולומטי.
After the two images have passed the sampling, we will apply the CPD algorithm on the
cloud of points that the CPD algorithm described is based on. The result of the algorithm is the
matching of each point in the source image to a point in the target image. In addition to
this, we order the images and perform the inverse matching. After that, we compute the
cumulative error of each matching and choose the match with the smallest error. After
we have computed the matching of the samples, we need to translate it back to
match the curvatures in the images. This is
simple, because there may be topological changes in the objects between the matched
samples. This is not desirable. In order to deal with this problem, we perform a
procedure of matching between the samples.
Most of the samples in both images at the
image source and target will be declared as
matching. If we do not have enough votes for a certain curvature, we will not make
such a match. The process described above will be repeated for each pair of consecutive
frames in the animation. Finally, we will adjust the curvature matching in each pair of
images. From that point on, the system can create a sequence of matches for all
curvatures in all images. As long as this process is automatic, but in the current
stage, the animator will be required to intervene. We declare that some curvatures
are not matched and some of the votes received are incorrect. The system presents
the results of the matching on the graphical user interface, where the user can
review the condition of the curvature at the end of the automatic stage. The user can
revert matching between adjacent images or add matching between curvatures that
the system did not detect.
Here begins the third and final stage,
where the user will complete the animation. Since
most of the curvatures have already been
matched, you will only need to change direction
by marking the normal on each of the contours of the image. The marking will be done
on the first image and the decisions will be transferred without further intervention
to additional images in the series. It is understood that the animator will need to fill
in the rest of the curvature for new or added curvatures in the images. The number of
changes in most cases will not be large. In order to create the animation, we use
a tool that comes from the "Lumo" algorithm. The Lumo tool applies a
filtering algorithm to the image edges, until the desired result is obtained. We perform
the same tool by solving a Poisson equation. The solution of this equation uses
boundary conditions of Dirichlet type, where the normal is defined by the animator or
Newman if it is not defined.
The process in the end generates an animation image with features
of three-dimensional images and a smooth appearance.

II
The addition of depth or three-dimensional appearance to a two-dimensional sequence of images can provide a smooth experience for the eye, even without the figure of the three-dimensional character. In this work, we present a method and a system that allows the animator to perform coherent image holes in a computer environment, where the character in the scene can undergo shape changes on a flat screen.

Currently, animation is a widely used field for the creation of entertainment and educational materials. In this work, we focused on creating a sequence of images that provide a three-dimensional impression from an existing two-dimensional sequence of images. The images in the sequence can be hand-drawn on paper or, alternatively, on a computer and serve as the base of the animated character when animated. In most cases, the animated character will undergo changes on a flat screen. Examples of this can be ears of a dog, which change from one frame to the next, or a person walking in front of the camera, so that different body parts are getting closer and further away.

The main challenge in automatic hole detection is the choice of the depth for every part of the character and maintaining this choice throughout the sequence of images. There are several possible approaches for the hole detection technique of threedimensional imagery. One approach allows the user to define the depth order in the monographed character and thus perform image hole detection. Another approach attempts to reconstruct the three-dimensional character from a two-dimensional sequence of images and perform three-dimensional hole filling. However, the main difficulty of the above methods is dealing with significant changes in shape, especially if there is a non-rigid transformation between one image and another.

We present an algorithm and a semi-automatic system that generates coherent image holes throughout the animation sequence, even if there are non-rigid transformations of the character. The system is built from several steps:

- In the first step, we receive as input the images created by the animator on paper that went through a scanning and resizing process. It is possible that the animator already created the images in a format that contains only curves (Bezier). These curves are suitable for display in our system.
- After the input is completed, we match between every two images in the animation sequence. The base for the matching is an algorithm for correspondence points (CPD — correspondance points detection). The algorithm matches between two clouds of points, where the change they go through can be non-rigid. In order to reach a state in which we have a set of points that match each image, we sample the curves. Sampling is done according to the length of the curve in order to allow uniform sampling of the samplings in the image.

The system is composed of several modules:

- An analysis module to extract the three-dimensional sequence of shapes from the images.
- An estimation module to define the depth of the shapes. The estimation is performed by sampling:
  - Shapely: the analysis module creates the shape of the hole and the animation is created in a way that every image is the correct transformation of the previous image. The estimation module is performed according to the sequence of shapes and the motion of the objects in the scene. (Bezier). The estimation module is performed according to the sequence of shapes and the motion of the objects in the scene.

- The system is built on top of the following modules:
  - A feature matching module that extracts unique features from the images.
  - A matching module that matches the features between two images.
  - A motion estimation module that estimates the motion of the objects in the scene.

The system was tested on several images and was found to work well.
המעון נounge התוכנית פרויקט חניון בקמפוס ורפורMiami מירלה בון-
חן
בפקולטה למדעי המחשב.

אני מודה לטכניון על התמיכה הכספית בפיתוח בדוי-
מודי.

אני מודה לטכניון על התמיכה הכספית והנדיבת בדוי-
מודי.

אני מודה לטכניון על התמיכה הכספית והנידיבה בדוי-
מודי.
מערכת חצי-אוטומטיות לתחזיות מבנים

קשיות היצלחלות והирующיות

של רצף האנימציה

דני ריבניקוב

לשם ממילוי חלק של תדריך תקפלת התואר

מניסיון לפיתוח של פעלו-גרף משותף

DENI RIBINCIOB

הודש לסנט הטכניון - מכון טכנולוגי לישראל

坭שחט תStreamer

ספטמבר 2013
מערכת חצי-אוטומטית לחזרת
בלתי קשיחות והצללה קוהרנטית
של רצף האנימציה

דני ריבניקוב