Exploring Shape Interpolation

UNIVERSITY OF BATH

DEPARTMENT OF COMPUTER SCIENCE: CS50245
CANDIDATE NUMBER: 02151
30/05/2018

Generating intermediate shapes between a source and target shape is known as shape interpolation and is a key technique for use in computer animation. This report will explore interpolation between a source and target mesh via two methods; linear interpolation and transformation-based interpolation. The two methods will be explained and implemented with the results compared. The report will close with a short animation sequence made using the best technique.

1 Introduction

Algorithms that interpolate between shapes or images are crucial in modern graphics and animation tasks. Applications are broad and include tasks such as morphing, pose transfer, and deforming objects. Grouped together as morphing or blending techniques, they involve the smooth transition from one shape to a target shape.

Shape interpolation begins with a pair of data primitive shapes, e.g. a mesh, line segments, or points. These primitive landmarks can be used to point to features on a more detailed image or model or be the object of interest them selves. The primary goal is to blend between these two objects with minimal user interaction wile retaining as much of their original appearance between steps as possible.

Figure 1: Example of shape interpolation between two 3D textured mesh objects [1]

In many cases, morphing effects such as these are carried out using image-based techniques [2]. Here, there is a requirement for the source and target to be similar otherwise a large amount of manual interaction is required to identify matching features and points. Often issues arise with the background of image-based morphs. This report will focus on the alternative object-space morphing approach.

With an object-space morphing approach, instead of using an image the objects are polygons or polyhedra meshes. The morphing then takes place between corresponding vertices. Identifying corresponding vertices has been tackled extensively in the literature and is known as the vertex correspondence problem. Solutions to vertex correspondence include merging meshes [3, 4], and fuzzy vertex correspondence [5] which aims to maximise a membership function to select a corresponding shape.

The vertex correspondence problem is of little interest in this report, the focus here is the paths of the vertices. Two methods of interpolation will be implemented and compared; the first is that of linear interpolation, the second an As-Rigid-As-Possible shape interpolation transform-based approach implemented from the work of Alexa, Cohen-Or and Levin [6].

The remainder of this paper will first discuss the two methods to be implemented. This will lead on to a pseudo code explanation of the implementation and a demonstration of results, including an example animation generated using the techniques. Finally the methods will be compared and conclusions on differences between the methods will be drawn.

2 Theory & Methods

Morphing between two shapes can lead to unusual deformation and lead to highly distorted in-between shapes. The standard method of this morphing is via using linear interpolation, discussed first. It is possible however to reduce this intermediate distortion by determining rigid parts of the transform. Maintaining an As-Rigid-As-Possible interpolation is the second method to be discussed.

2.1 Linear Interpolation

Linear interpolation is a straightforward method, simply take the start and end position of each vertex and divide the linear path between each position by a number of steps. Each step then identifies a position of the vertex at a frame of the animation. The position of a point, p, at a time, t, is given simply as:

$$p(t) = (1-t) * p_1 + t * p_0 \tag{1}$$

where the initial position at $t = 0$ is p_0, the finally position at $t = 1$ is p_1 and the time ranges between 0 and 1. This process is carried out for all corresponding vertices and clearly moves each vertex in the shortest possible path to its destination. Unsurprisingly, this can lead to unusual distortions, shown in figure 2, particularly when the initial and final shapes are significantly different.

Figure 2: An example of linear vertex interpolation between an elephant and a giraffe [6]. The intermediate steps clearly show distortion that ideally would be minimised for a smoother transition.

2.2 ARAP Shape Interpolation

The As-Rigid-As-Possible shape interpolation method aims to determine a vertex path that is smoother and maintains the rigidity of each triangle as much as possible between time steps. For each corresponding triangle the affine transformation matrix, A, is computed:

$$Ap_i + l = \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} p_i + \begin{pmatrix} l_x \\ l_y \end{pmatrix} = q_i \tag{2}$$

where p and q represent the source and target vertices respectively. The translation vector, l, is ignored as it does not describe any changes to the shape itself; only the position. The affine matrix A is computed for each pair of corresponding triangles via solving the linear system

$$ax = b \tag{3}$$

which corresponds to the following:

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & \cdots & & & & \\ \vdots & & & & & \\ 0 & 0 & 0 & x_3 & y_3 & 0 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ \vdots \\ a_{23} \end{bmatrix} = \begin{bmatrix} x'_1 \\ y'_1 \\ \vdots \\ y'_3 \end{bmatrix} \tag{4}$$

where x, y and x', y' correspond to the source and target vertices. The elements a_x then correspond to the 6 elements of the matrix A, however, as the translation component is ignored only elements a_11, a_12, a_21, & a_22 are used.

This gives the overall affine transformation from source to target. To acquire the intermediate shapes, $V(t)$, the affine matrix at that time step needs to be computed. Firstly A is decomposed via single value decomposition:

$$A = R_\alpha D R_\beta \tag{5}$$

Alexa et al. [6] demonstrate the best results are found by forming a single rotation matrix and a symmetric matrix, R_γ, as follows

$$\begin{aligned} A &= R_\alpha D R_\beta = R_\alpha (R_\beta R_\beta^T) D R_\beta \\ &= (R_\alpha R_\beta)(R_\beta^T D R_\beta) = R_\gamma S \end{aligned} \tag{6}$$

$A(t)$ is then computed by linear interpolation, as discussed earlier.

$$A_\gamma(t) = R_{t\gamma}((1-t)I + tS) \tag{7}$$

The matrix $A_\gamma(t)$ now exists for each triangle and for each time step. When considering all triangles in a mesh, its clear that many vertices correspond to more than one triangle. As such a single mapping of all the vertices could not conform with each individual ideal transformation that has already been computed. We now define the matrix $B_{i,j,k}(t)$ to be the affine transformation from previous point, $p_{i,j,k}$, to the next time step location $v_i(t)$, $v_j(t)$, & $v_k(t)$.

$$B_{i,j,k}p + l = v(t) \tag{8}$$

We define the intermediate time step shape as the configuration that minimises the following least squared problem:

$$\sum_{\{i,j,k\} \in \mathcal{T}} ||A_{i,j,k}(t) - B_{i,j,k}(t)||^2 \tag{9}$$

where A is known from before and B is an unknown 2×2 transformation matrix. Each element in B can be represented as a linear combination of the three intermediate vertex positions for the triangle of question.

$$b_x = \begin{bmatrix} u_1 & u_2 & \cdots & u_6 \end{bmatrix} \begin{bmatrix} x'_i \\ y'_i \\ \vdots \\ y'_k \end{bmatrix} \tag{10}$$

This is built from the affine transform described in equation 4, where the column of a entries in this case are the b_x entries. This identifies the row vector of u_x entries as the corresponding row from the

inverse of the 6×6 matrix. Apply this to all vertices in the mesh by extending the lengths of the two vectors as

$$b_x = \begin{bmatrix} 0 & \cdots & u_1 & u_2 & \cdots & u_3 & \cdots & 0 \end{bmatrix} \begin{bmatrix} x'_1 \\ y'_1 \\ \vdots \\ x'_i \\ y'_i \\ \vdots \\ x'_n \\ y'_n \end{bmatrix} = \mathbf{u}_x^T \mathbf{x} \tag{11}$$

Reviewing the least squares problem in equation 9, it can be seen that for each of the four pair of elements the following holds:

$$a_{11} - b_{11} = a_{11} - \mathbf{u}_1^T \mathbf{x} \tag{12}$$

This is then carried out for all triangles and concatenated into a matrix form of $Ax = b$. Where

$$A = \begin{bmatrix} \mathbf{u_1} & \mathbf{u_2} & \cdots \end{bmatrix}^T \tag{13}$$

and

$$b = \begin{bmatrix} a_{11} & a_{12} & \cdots \end{bmatrix}^T \tag{14}$$

The first vertex is then removed from each row in A due to it being fixed to the linear interpolation found earlier. The matrix problem is then solved for x which gives the coordinates of the vertices for the next time step. These are then used for the next time steps initial position and the process repeated.

3 Implementation & Results

In this section the implementation of the two methods will be discussed followed by examples of their operation. All implementation was carried out in MATLAB with no extra packages.

3.1 Implementation

Firstly, MATLAB has no inbuilt method of reading a '.obj' mesh file such as the one used. The file type is a simple CSV style file with each row identifying either a vertex location or a face's vertices. The type is identified with a 'v' or 'f' character and a function was implemented that reads each line of the file and forms a MATLAB struct data type for the file that has a field for the vertices and faces. This data struct is then used for the rest of the implementation. Both the source and target mesh are loaded this way.

3.1.1 Linear Interpolation Implementation

The linear interpolation follows simply performs equation 1 for all vertices for a specified number of frames.

Data: Source & target mesh, number of frames.
Result: Performs linear interpolation from a source mesh to a target mesh in a specified number of frames;
for *Number of frames* **do**
  t = frame/total frames;
  Position at frame = (1-t)Target + (t)Source;
end

Algorithm 1: Linear interpolation from source to target mesh.

The output is then a collection of vertex positions at each frame and can be displayed as an animation by plotting the mesh at each frame.

3.1.2 ARAP Shape Interpolation Implementation

The ARAP shape interpolation can be split into two steps. Firstly, the computation of matrix $A_\gamma(t)$ for each triangle. Secondly, solving the least squared problem in equation 9.

Data: Source & target mesh data, number of frames.
Result: Computes $A_\gamma(t)$;
for *Each triangle* **do**
  Form matrix a in equation 4 from source mesh.;
  Form vector b from target mesh;
  Solve linear system $ax = b$ for x;
  Keep only the rotation elements and form A;
  Perform SVD of A to get R_α, D, & R_β;
  Compute R_gamma;
  Compute S;
  for *Each Frame* **do**
    Compute $A_\gamma(t)$ via linear interpolation as in equation 1;
  end
end

Algorithm 2: Computation of $A_\gamma(t)$ by first computing A as in equation 4 and then performing SVD, rearranging, and linear interpolation.

This first algorithm computes A_γ for each frame and each triangle. This is then used in the second

algorithm.

Data: Source mesh and $A_\gamma(t)$ for each triangle.
Result: Computes vertex positions at each frame, $V(f)$;
for *Each frame, f* **do**
  for *Each triangle* **do**
    Form matrix a in equation 4 with each corner vertex;
    Compute inverse of a;
    Form vectors **u** for rows 1, 2, 4, 5 that correspond to the rotation elements of the affine matrix ;
    Remove first vertex from $mathbfu$. Form matrix b in equation 10 from the relevant $A_\gamma(t)$;
    Solve the linear system $ax = b$ for x;
    Concatenate the first vertex onto x from the linear interpolation to form $V(f)$ for that triangle;
    Concatenate $V(f)$ for each triangle;
  end
  Use $V(f)$ as the source mesh for the next frame;
end

Algorithm 3: Computation of vertex positions at each frame via the least squared minimisation in equation 9.

The vertex positions, $V(f)$, can then be plotted and used for animation.

All animations are produced by using MATLAB's 'patch' function to plot each triangle. The frame is held for a short period of time (for example 0.1 seconds) and the next frame is then loaded.

3.2 Results

The linear interpolation of the mesh was successful. Two animations where produced and can be seen in figure 3. The method does not maintain the shape of the object through the motion and some areas appear distorted. It proved to be adequate for small motions, but the larger the motion the more the model distorted through the intermediate poses.

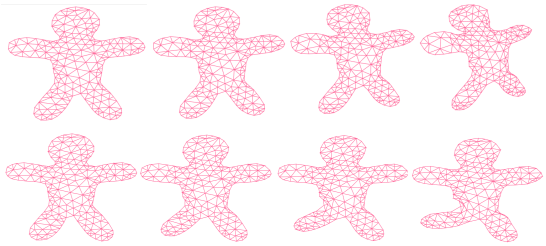

Figure 3: Four stills from two animations of the linearly interpolated mesh. The first is a stretch of the right hand side of the body. The second is a movement of the left side leg. The points move in a straight line, shortest distance path from source to target position.

The implementation of the ARAP method was less successful. The reasons for the failure are unknown but the animation can be seen in figure 4. The model does move and intermediate steps are formed, however they are significantly far away from the original mesh. This causes significant distortion as the steps proceed as the error builds on itself. One point appears to remain fixed; this is the point that is chosen to be fixed to the linear interpolation path.

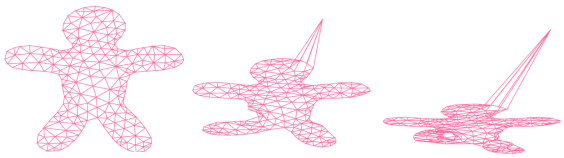

Figure 4: Three stills from the animation of the ARAP shape interpolated mesh. It can be seen the mesh distorts significantly. The near static point is the vertex chosen to be fixed to the linearly interpolated mesh.

4 Discussion

Within the results presented here, the only successful interpolation carried out was via linear interpolation methods. Unfortunately, the implementation error within the ARAP method was unable to be identified within the time frame available. As such the methods cannot be accurately compared, although it is clear that the linear interpolation method did experience issues with large movements where the mesh appears to shrink and then regrow as the vertices take the shortest path to their destination.

The exact issue with the implementation of the ARAP method could not be identified. It is unlikely to be within the first algorithm as $A_\gamma(t)$ fits the structure expected. In addition, R_γ and S are

of the correct structure, particularly with S having identical diagonal elements as anticipated, and can be decomposed back into the original A matrix with no issue. As such it is expected the problem lies somewhere within the indexing of the second algorithm although it is unsure as to where.

References

[1] Mesh morphing example. 29/05/2018.

[2] George Wolberg. Image morphing: a survey. *The Visual Computer*, 14(8):360–372, Dec 1998.

[3] James R. Kent, Wayne E. Carlson, and Richard E. Parent. Shape transformation for polyhedral objects. In *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '92, pages 47–54, New York, NY, USA, 1992. ACM.

[4] Eyal Carmel and Daniel Cohen-Or. Warp-guided object-space morphing. *The Visual Computer*, 13(9):465–478, Jan 1998.

[5] Yuefeng Zhang. A fuzzy approach to digital image warping. *IEEE Computer Graphics and Applications*, 16(4):34–41, Jul 1996.

[6] Marc Alexa, Daniel Cohen-Or, and David Levin. As-rigid-as-possible shape interpolation. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 157–164, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.