Exploring interactive surface-based mesh deformation

UNIVERSITY OF BATH

DEPARTMENT OF COMPUTER SCIENCE: CS50245
NAME: Jacob Haynes
CANDIDATE NUMBER: 02151
PROGRAM: EngD Digital Entertainment
03/04/2018

Surface-based deformation involves deforming a mesh by manipulating its vertices. This report discusses the basic theory behind Igarashi's et al. As-Rigid-As-Possible (ARAP) algorithm [1]. A two dimensional implementation of the algorithm in Matlab is also presented and used to create a collection of key-frames for use in an animation sequence.

1 Introduction

Image deformation is commonly seen in real time applications such as image editing, graphical user interfaces, and games. Common approaches to shape deformation include the use of skeletons [2] and free-form deformation (FFD) [3]. However, these methods have drawbacks, it can be hard to define a skeleton for many objects and FFD can be a tedious process involving heavy manipulation from the user to maintain control of the object.

An alternative, presented by Igarashi et al. involves modelling the shape with a simple triangular mesh and allowing the user to select several points as handles to move [1]. The system allows the overall shape to move, rotate, and rescale to match the new handle positions while minimizing the overall distortion of the shape. The system is built on a two step algorithm which will be implemented and discussed in this report.

The original paper presents the algorithm by representing the distortion of each triangle face of the mesh. Igarashi later published an alternative aproach to the algorithm discussing the use of an edge based representation [4]. It is this representation that will be used in this implementation. Both face and vertex based implementations are equally valid although harder to implement. A vertex based implementation is also of particular use in three dimensions due to it maintaining local surface geometry and bumps through the deformation.

The remainder of this report will first discuss the problem and the basic theory and algorithm. This will then be followed by a review of a Matlab implementation of the algorithm with results and the generation of a collection of key-frames. Findings will then be summarised and the work concluded.

2 Theory & Methods

The overall pipeline is fairly straightforward and s of three steps. First, the user selects a two dimensional triangular mesh. Many algorithms exist to generate two dimensional meshes from simple closed polygons such as Markosian's work [5], and will not be discussed here, it will be assumed the user has a two dimensional triangular mesh at hand. Secondly, the user selects an arbitrary amount of handles. The handles act as constraints for the deformation, and can be manipulated by the user to deform the shape. This pipeline is shown in figure 1.

The input to the algorithm is simply the vertex coordinates and the output is the new coordinates of the deformed mesh vertices. This ideally is achieved by minimizing a single error metric, however it is concluded that a single quadratic error function that represents

the overall distortion is impossible [1]. Instead the problem is broken down into a two step problem, one for rotation and one for scaling with each part handled by an independent error function. The result is then found by sequentially solving the two least-squares problems.

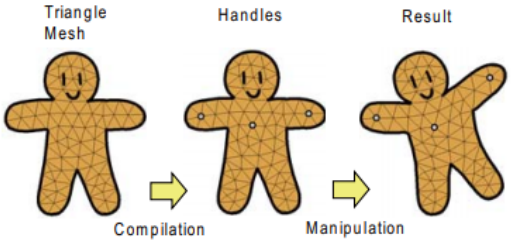

Figure 1: An overview of the problem. A two dimensional triangular mesh has handles selected and then manipulated to deform the shape. [4]

2.1 Base Algorithm

Firstly the base algorithm shall be discussed in terms of an edge based representation before discussing the two individual steps of the solution as both rely on this method. The goal is to find the deformed vertex coordinates that minimise the distortion of edge vectors given the handles as constraints. As such we solve the following via a least-squares method:

$$v'_j - j'_i = v_j - v_i(\{i,j\} \in E) \tag{1}$$

where v_i and v'_i are the vertex coordinates of the rest and deformed shape respectively. E is the set of all edges. This is solved subject to a set of constrained vertices C as $v'_i = C_i(i \in C)$. This combined as a cost function leads to the a least-squares minimization problem:

$$\begin{aligned} arg\min_{v' \in V} \Bigg\{ & \sum_{\{i,j\} \in E} ||(v'_j - v'_i) - (v_j - v_i)||^2 \\ & + w \sum_{i \in C} ||v'_i - C_i||^2 \Bigg\} \end{aligned} \tag{2}$$

where w is a weight factor. This can be rewritten in matrix form as

$$arg\min_{v'} ||Av' - b||^2 \tag{3}$$

This can be solved separately for the x and y components with v' being the column vector of the deformed vertices, b being a combined column vector of the i edge vectors between connected vertices, e, and weighted constraints, wC. Matrix A is a more is the same for both x and y components and identifies which vertices are connected to which with a -1 for the start of the edge vector and a 1 for the end of the edge vector in the relevant matrix coordinate. The result of the minimisation is then found by solving the following normal equation:

$$A^T A v' = A^T b \tag{4}$$

2.2 Step One: Similarity Transformation

In this step, a method developed by Sorkine et al. is used to rotate the edge vector (right hand side of equation 2) by a rotation matrix T_k [6]. This rotation maps the neighbouring vertices around the edge to their deformed locations. The rotation matrix T_k is actually a similarity transform that allows free rotation and uniform scaling. The full derivation can be found in Igarashi's work [4] but will be briefly discussed here.

The rotation matrix T_k is a transformation that maps neighbouring vertices around the edge to new positions minimised in a least-squares sense. Igarashi proposes using four neighbours to get a good result, however, in this implementation three were used for simplicity. Here T_k is computed as follows:

$$\begin{aligned} T_k &= arg\min_{T_k} \sum_{v \in N(e_k)} ||T_k v - v'||^2 \\ &= \begin{bmatrix} c_k & s_k \\ -s_k & c_k \end{bmatrix} \end{aligned} \tag{5}$$

where $N(e_k)$ represents the neighbouring vertices (v_i, v_j, v_l) to the edge e_k. The elements of the rotation matrix are the resultant sine and cosine functions of the rotation to be found.

For the three neighbours in N this can be

expanded as

$$\{c_k, s_k\} = arg\min_{c_k,s_k} \left\| \begin{bmatrix} v_{ix} & v_{iy} \\ v_{iy} & -v_{ix} \\ v_{jx} & v_{jy} \\ \vdots & \end{bmatrix} \begin{bmatrix} c_k \\ s_k \end{bmatrix} - \begin{bmatrix} v'_{ix} \\ v'_{iy} \\ v'_{jx} \\ \vdots \end{bmatrix} \right\|^2$$
$$= arg\min_{c_k,s_k} \left\| G_k \begin{bmatrix} c_k \\ s_k \end{bmatrix} - \begin{bmatrix} v'_{ix} \\ \vdots \end{bmatrix} \right\|^2 \quad (6)$$

This can then be solved as a least-squares problem:

$$\begin{bmatrix} c_k \\ s_k \end{bmatrix} = (G_k^T G_k)^{-1} G_k^T \begin{bmatrix} v'_{ix} \\ \vdots \end{bmatrix} \quad (7)$$

This is then used in equation 2 transforming the left-hand summation into:

$$(v'_j - v'_i) - T_{ij}(v_j - v_i) \quad (8)$$

The same procedure can be followed as in section 2.1 to form the equivalent or equation 3 which can be solved the same way. This gives a deformed shape which looks mostly correct. However, the further the handle is moved from its original position the more the shape inflates. Step two is then used to adjust the scale.

2.3 Step Two: Scale Adjustment

This step takes the G and v' values calculated from step one and uses them to calculate c_k, s_k and thus T_k for each edge. This is then normalised as

$$T'_k = \frac{1}{c_k^2 + s_k^2} T_k \quad (9)$$

This is then inserted into equation 2 as show in equation 8 and again formed into a matrix equation similar to equation 3 but with each edge vector in b multiplied by the respective T_k. Again the normal equation can be solved to return the final deformed vertex positions.

3 Implementation & Results

This section will first outline the Matlab implementation in terms of pseudo code and a description of the user interface and functionality. A demonstration will follow including resultant key-frames for an animation sequence.

3.1 The Application

The user interface is simple and does not include a method for importing meshes. As such meshes must be manually imported in the code, a demonstration mesh is used of the .obj format. This file contains two fields, firstly vertex positions in 3D space and secondly a list of the vertex indices for each triangular face. The mesh is displayed on a plot and three buttons exist on the interface. The first is used to toggle on the selection of constraint handles and also activates the data cursor. The second is used to confirm a chosen constraint and save its position. The third and final button is used to toggle on the selection and movement of a deformation handle. This implementation does not support 'click and drag' functionality or the movement of multiple handles at the same time.

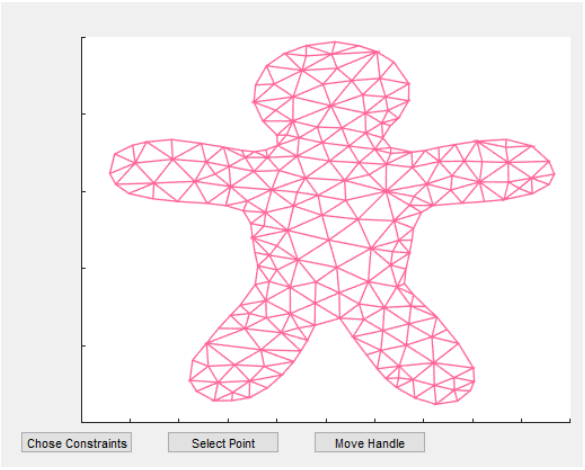

Figure 2: The implemented UI with the default mesh loaded.

3.2 Algorithm Implementation

When the user toggles the 'Chose Constraints' button to on the user is provided with instructions to select constraints and the data cursor is toggled on. While this mode is active the second button saves the position of the current selected point as a constraint.

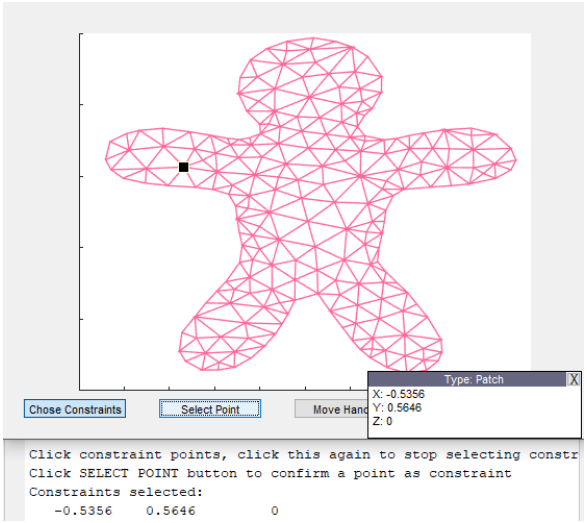

Figure 3: The UI while the user is selecting constraints. Note that a confirmed constraint position is broadcast back to the user.

Once all constraints have been selected and the mode toggled off, the chosen constraints are marked on the mesh with red markers and the position of all constraints broadcast back to the user.

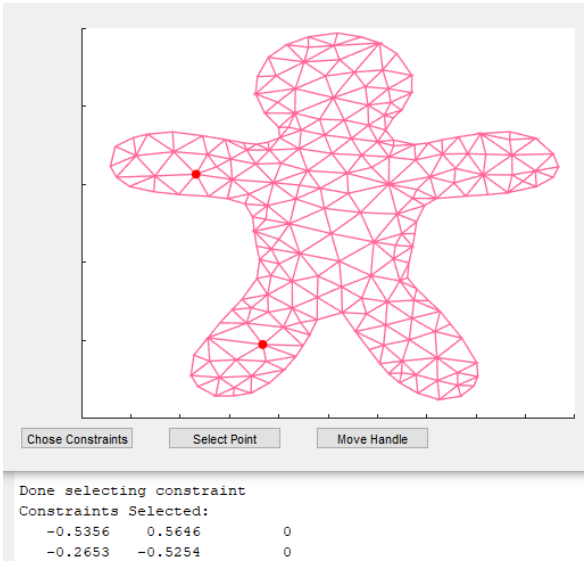

Figure 4: Chosen constraints are broadcast back to the user and marked on the mesh with red markers.

Once the user toggles the 'Move Handle' button, mouse clicks are used to select a handle and then a target location. Once the new handle location is chosen, the deformation algorithm (pseudo code bellow) can be used.

Data: User mouse clicks.
Result: Gets the index of the selected handle and its target location for use in deformation.
On user click;
if *'Move Handle' is toggled on* **then**
  if *No handle is selected* **then**
    Get mouse location;
    Get the aspect ratio of the plot;
    Compute the distance of the click to each mesh vertex;
    Select the closest vertex as the deformation handle;
  else
    Get mouse location;
    Set the mouse location as a constraint position for the handle;
    Perform the deformation algorithm;
  end
end

Algorithm 1: The behaviour of the system upon a user moving a handle.

Data: Perform ARAP deformation in an edge representation of a mesh with constraints, C, handle, H.
Result: Returns the deformed mesh vertex positions.
Get the constraints indices;
for *Each face of the mesh* **do**
  Get the connected neighbour indices;
  Compute G for each edge;
  Compute edge vector e;
  Compute h (as in [4]);
  Form top half of matrix A with a pair of rows (x and y) for each edge, the column identifies the vertex for each element of h;
  Form bottom half of A with w for each constraint vertex;
  Concatenate top and bottom of A;
  Construct bottom of b from the constraint vertex positions and w;
  Concatenate bottom of b with a zeros matrix the length of the amount of edges;
end
Solve for v';
for *Each face of the mesh* **do**
  Use G for each edge to compute c and s for each edges T;
end
Perform scale adjustment;

Algorithm 2: The first step of the deformation algorithm.

Data: Performs the scale adjustment step using the computed T.
Result: Returns the scale adjusted deformed vertex positions.
for *Each face of the mesh* **do**
  Construct A from the vertex positions of the edge terminals;
  Compute the product of T and e for each edge;
end
Construct b;
Solve for v';

Algorithm 3: The scale adjustment step of the deformation algorithm.

3.3 Key Frames

A collection of meshes have been produced for use as key frames in animation. Here the mesh has had its left arm manipulated with constraints applied to the other limbs. A small amount of head distortion can be seen as the arm is pulled away from the head. This may be due to an error in the scale correction step not accurately correcting the scale to avoid this distortion.

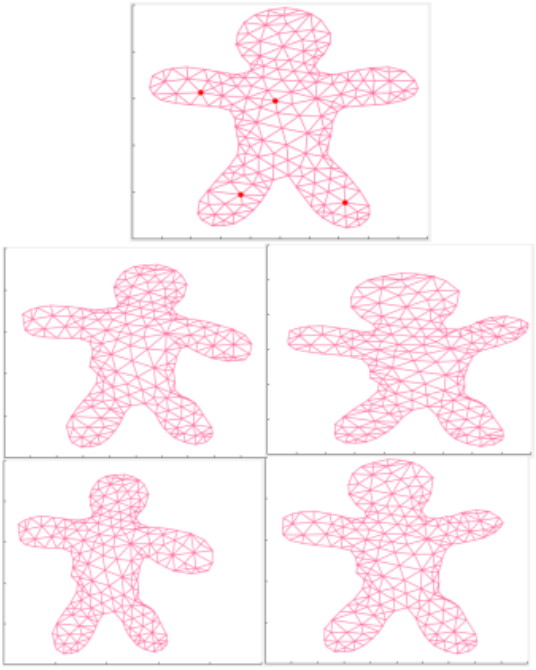

Figure 5: A collection of deformed meshes as a demonstration of potential key frames for a waving animation.

4 Conclusions

The aim of this report was to explore and implement an ARAP surface deformation algorithm. Igarashi's et al. method was discussed and implemented in Matlab for two dimensional triangular meshes. A collection of key frames were produced to demonstrate potential use in animation. The implementation works in real time with an unnoticeable time required to calculate the deformed mesh. It is noted that the mesh can deform slightly in scale and have rough edges at times, it is believed this is due to an error in the implementation of the scale adjustment step.

References

[1] Takeo Igarashi, Tomer Moscovich, and John F. Hughes. As-rigid-as-possible shape manipulation. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 1134–1141, New York, NY, USA, 2005. ACM.

[2] J. P. Lewis, Matt Cordner, and Nickson Fong. Pose space deformation: A unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 165–172, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[3] Ron MacCracken and Kenneth I. Joy. Free-form deformations with lattices of arbitrary topology. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 181–188, New York, NY, USA, 1996. ACM.

[4] Takeo Igarashi and Yuki Igarashi. Implementing as-rigid-as-possible shape manipulation and surface flattening. *J. Graphics, GPU, & Game Tools*, 14:17–30, 2009.

[5] Lee Markosian, Jonathan M. Cohen, Thomas Crulli, and John Hughes. Skin: A constructive approach to modeling free-form shapes. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pages 393–400, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[6] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H.-P. Seidel. Laplacian surface editing. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, SGP '04, pages 175–184, New York, NY, USA, 2004. ACM.