Interactive Surface-based Mesh Deformation

CM50245 - Computer Animation and Games II

Unit Leader: Dr. Yongliang Yang

Yongqiang Zhu (MSc Digital Entertainment)

March 22, 2019

1 Introduction

There are many computer graphics techniques developed for computer animation and games. One of the most important techniques has always been object deformation. In both animation and video games, objects would usually have their movements, and it is important to make sure that their shapes do not crack while moving. This could be achieved by artists working on it frame by frame, but that is not cost effective. Researchers have been developing techniques to deform objects, both in 2D and 3D, such as free form deformation (Sederberg & Parry, 1986) which involves fitting the object inside a bounding box, or surface based deformation (Igarashi, Moscovich, & Hughes, 2005).

This project is aiming to implement (Igarashi & Igarashi, 2009), a surface based mesh deformation technique which was established on top of (Igarashi et al., 2005). The program will be written C++, along with the graphics language OpenGL. The input is a 2D triangle mesh of a gingerbread man, which detail is stored in an '.obj' file. The target of this project is to place control points on the mesh, and drag a handle to deform the mesh with the gingerbread man still looking natural.

The structure of the report is shown as follow: section 2 will explain the input file, and how to load it and represent it visually. Section 3 will focus on manipulating control vertex. Section 4 is the main content of this implementation, which explains the process of the deformation. Section 5 is the guidance of how to use the program. Section 6 acknowledges the C++ library downloaded and used, and section 7 will conclude the report and outline some limitations of this implementation.

2 Mesh Loading

The input '.obj' file has the following characteristics: some comments which begin with '#'; some vertex coordinates which begin with 'v' follow by the x, y and z coordinates, in this project since it is a 2D mesh all the z values will be 0; some faces with 'f' and follow by the 3 corresponding vertex indices.

The input parsing function would first detect the first character in each line in the input file, if it is '#' it would just ignore the line and jump to the next line. If it is 'v' it would then store its x and y coordinates with the correct index. For the case of 'f' it would store its 3 vertex indices. Then the object will be drawn with the OpenGL function 'GL_LINE_LOOP', connecting the 3 vertices in each face. The result is shown in Figure 1, and it has a convincing gingerbread man shape.

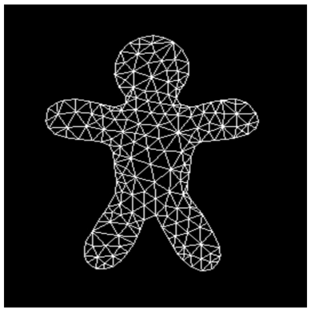
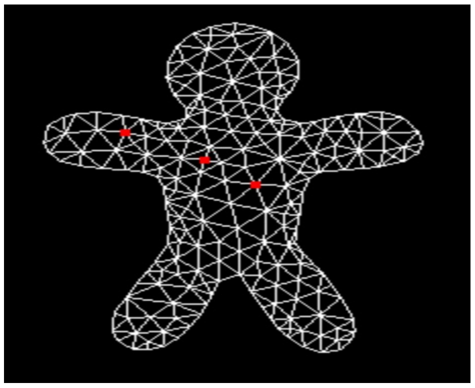


Figure 1: Loading and drawing the mesh: gingerbread man

3 Control Vertices

This implementation involves placing control points on mesh vertex, and one of it would act as the handle of the deformation. Hence the functions of placing control vertex and removing them are implemented.

The key 'Z' must first be pressed before placing a control point. Then use a mouse to click on the mesh, base on the clicking coordinate the program would find the closest vertex, and mark it with a red dot. If a click is performed near an existing control vertex, the clicking will be ignored. The result of placing control vertices is shown in Figure 2a.


(a) Placing control vertices

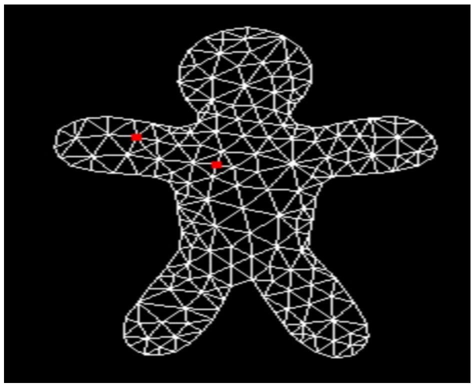

(b) Removing a control point

Figure 2: Placing (a) and removing (b) control points from the object mesh

To remove a control vertex (when there is one or more), press key 'C' then click on the mesh (ideally on the control point i.e. red dot), then the control vertex will be removed from the mesh, like it is represented in Figure 2b when comparing to Figure 2a.

4 Deformation

As described in the paper (Igarashi & Igarashi, 2009) the deformation process includes two steps: the first one makes sure that the transformed object has the similar shape orientation compare to its original shape; the first step would incur a scale variance when deforming the object, therefore the second step would then adjust the scale to make sure the transformed object is at a similar scale compare to its raw shape.

Both of the steps involve using edges of the meshes. The algorithm is aiming to minimise the variance in every edge. The input file only contains face and vertex details, therefore it would require to first find all the existing edges. If loop over all the faces to find edges, some of the edges would be duplicated. This characteristic can be used to determine the neighbour vertices of every edge, if there are two neighbour vertices then this edge is an inner edge, it is a boundary edge if it has only one neighbour vertex.

4.1 Similarity Transformation

There are several matrices and vector which would need to be computed: the G matrix, the H matrix, the A matrix and the b vector. The G matrix is the collection of vertex coordinates of every edge and its neighbour(s). Matrix H can be derived from G, like in equation 1, where e is the edge vector.

$$H = \begin{bmatrix} -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} e_x & e_y \\ e_y & -e_x \end{bmatrix} (G^t G)^{-1} G^T \qquad (1)$$

The A matrix is combining all the H matrices for edges, as well as the w coefficient of the control vertices, where w is set to 1000 as it is suggested in the paper. The b vector is filled with corresponding edge value (all 0), and the products of w and the control vertices. The least square minimisation problem of edges can now be rewritten as the following calculation:

$$A^T A v' = A^T b$$
$$v' = (A^T A)^{-1} A^T b$$

The first attempt has resulted in failure like shown in Figure 3. It can be observed that the control vertices and the handle (right bottom red dot) are in correct positions (compare it with Figure 5b), but the shape of the gingerbread man is not expected. Without scale adjustment, according to the paper (Igarashi & Igarashi, 2009) if the handle is dragged further away from other control vertices, the size of the object would be scaled up visually. In Figure 3 it can be observed that the gingerbread man has not been scaled up, but rather scaled down.

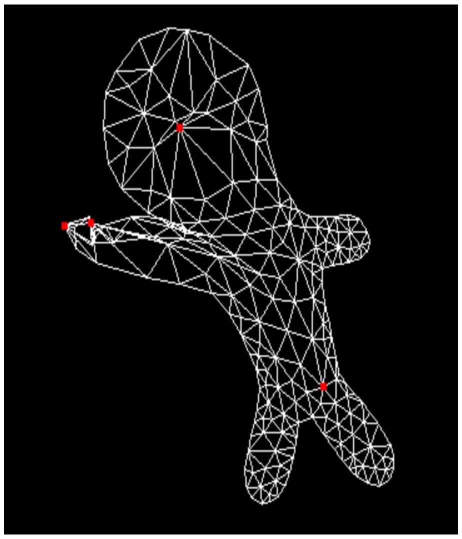
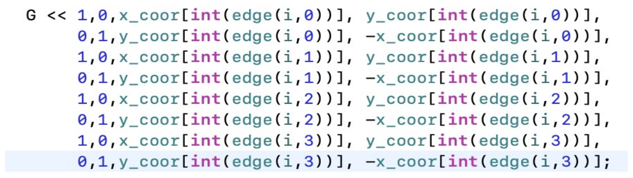


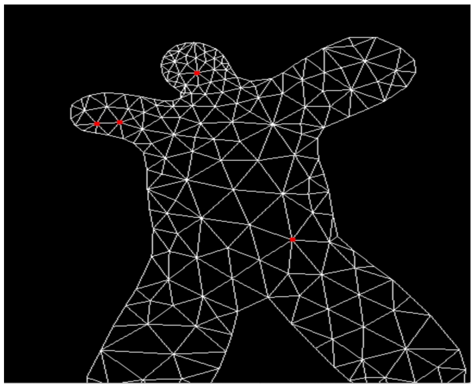
Figure 3: Result of incorrect matrix values

It has soon been discovered that it is the problem of setting up the G matrix. The 1s and 0s have been inserted into G on the left of the vertex coordinates, like shown in Figure 4, but actually they should be on the right. After amending the G matrix, the gingerbread man can now be deformed as expected, as shown in Figure 5a.



```
G << 1,0,x_coor[int(edge(i,0))], y_coor[int(edge(i,0))],
     0,1,y_coor[int(edge(i,0))], -x_coor[int(edge(i,0))],
     1,0,x_coor[int(edge(i,1))], y_coor[int(edge(i,1))],
     0,1,y_coor[int(edge(i,1))], -x_coor[int(edge(i,1))],
     1,0,x_coor[int(edge(i,2))], y_coor[int(edge(i,2))],
     0,1,y_coor[int(edge(i,2))], -x_coor[int(edge(i,2))],
     1,0,x_coor[int(edge(i,3))], y_coor[int(edge(i,3))],
     0,1,y_coor[int(edge(i,3))], -x_coor[int(edge(i,3))];
```

Figure 4: Filling in values for G matrix with incorrect order


(a) Result from step 1

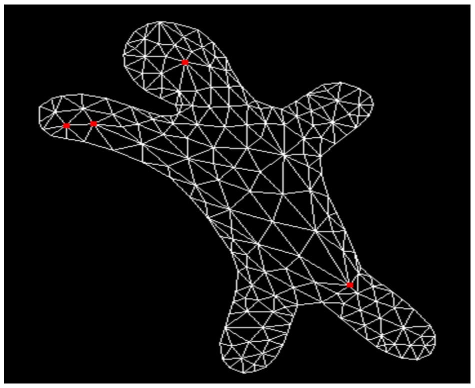

(b) Final result

Figure 5: Visualising the results of object deformation: step1(a) & step2(b)

4.2 Scale Adjustment

After completing the first step, we are now moving onto adjusting the scale of the gingerbread man, so that it would not be drastically enlarged or shrinked. Unlike the first step, where x and y coordinates of vertices can all be calcuated at once, at this step they would need to be calculated separately, but with more or less the same process.

The rotation matrix T are distinctly computed for every edge. It would also be normalise so that it does not scale the object when rotating. The T matrix is shown in equation 2, where c and s can be calculated by equation 3. V is the collection of vertex coordinates from every edge and its neighbour, and the coordinates are the updated values from the first transformation step.

$$T = \frac{1}{\sqrt{c^2 + s^2}} \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \tag{2}$$

$$\begin{bmatrix} c \\ s \end{bmatrix} = (G_k^t G_k)^{-1} G_k^T V \tag{3}$$

This step of transformation also involves an A matrix, and b vectors for both x and y. Matrix A contains the coefficients (1 and -1) for edge vectors, and coefficient w for control vertices. Vector b has the edge vector after rotation, and the product of w and the control vertices. The final step is to calculated the end pose vertices of the gingerbread man, and this is rather similar to the first step, the equation is shown below:

$$v' = (A^T A)^{-1} A^T b \tag{4}$$

Now the gingerbread man can be deformed with the appropriate orientation and scale, and the result can be visualised in Figure 5b. If compare it to the result from the first step Figure 5a, it can be observed that the gingerbread man in Figure 5b has relatively more uniform scale across the whole object, and this is the expected outcome from this deformation algorithm.

5 User Manual

- Mouse Left Button Click: Depends on the following key selections:
- Z: Mouse click to place control point on the mesh vertex;
- X: Select existing control vertex as control handle;
- C: Remove existing control vertex;
- V: Drag the handle to the mouse click coordinate;
- Q: Exit the program.

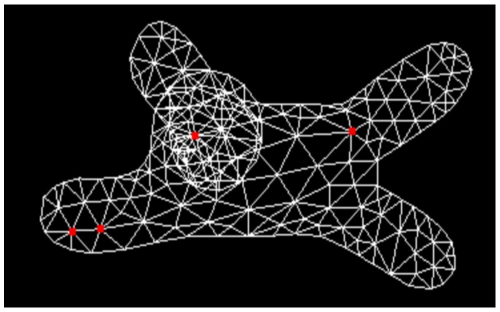


Figure 6: Object deformation with self-intersection

6 Acknowledgement

A C++ library ‘Eigen’ is used (https://bitbucket.org/eigen/eigen/), and this library helps to use matrices and do matrix operations.

7 Conclusion

In this report we have been looking at the algorithm of surface based mesh deformation, particularly for 2D objects. We can also see the visualised representation of this algorithm, suggesting that it can deform a 2D object while keeping its shape without drastically changing the scale.

However there is also limitation of this implementation. In the paper (Igarashi & Igarashi, 2009) it suggests that the process of deformation can be completed in realtime, but due to redundant code writing, the implementation in this project has relatively slow runtime, so it cannot be made into realtime interaction. There is also limitation in the algorithm. Since this algorithm focus on maintaining the shape and scale when deforming an object, it has not specifically set constraints to the boundary of the object, so the outcome could result in self-intersection like shown in Figure 6. This is an interesting aspect of object deformation, and it is worth diving into in the future.

References

Igarashi, T., & Igarashi, Y. (2009). Implementing as-rigid-as-possible shape manipulation and surface flattening. *journal of graphics, gpu, and game tools*, *14*(1), 17–30.

Igarashi, T., Moscovich, T., & Hughes, J. F. (2005). As-rigid-as-possible shape manipulation. In *Acm transactions on graphics (tog)* (Vol. 24, pp. 1134–1141).

Sederberg, T. W., & Parry, S. R. (1986). Free-form deformation of solid geometric models. *ACM SIGGRAPH computer graphics*, *20*(4), 151–160.