

3D MESH ANIMATION SYSTEM TARGETED FOR MULTI-TOUCH ENVIRONMENTS

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BİLKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By

Duygu Ceylan

August, 2009

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Tolga K. apın (Supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Blent zg

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Veysi şler

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet Baray
Director of the Institute Engineering and Science

ABSTRACT

3D MESH ANIMATION SYSTEM TARGETED FOR MULTI-TOUCH ENVIRONMENTS

Duygu Ceylan

M.S. in Computer Engineering

Supervisor: Asst. Prof. Dr. Tolga K. Çapın

August, 2009

Fast developments in computer technology have given rise to different application areas such as multimedia, computer games, and Virtual Reality. All these application areas are based on animation of 3D models of real world objects. For this purpose, many tools have been developed to enable computer modeling and animation. Yet, most of these tools require a certain amount of experience about geometric modeling and animation principles, which creates a handicap for inexperienced users. This thesis introduces a solution to this problem by presenting a mesh animation system targeted specially for novice users. The main approach is based on one of the fundamental model representation concepts, Laplacian framework, which is successfully used in model editing applications. The solution presented perceives a model as a combination of smaller salient parts and uses the Laplacian framework to allow these parts to be manipulated simultaneously to produce a sense of movement. The interaction techniques developed enable users to carry manipulation and global transformation actions at the same time to create more pleasing results. Furthermore, the approach utilizes the multi-touch screen technology and direct manipulation principles to increase the usability of the system. The methods described are experimented by creating simple animations with several 3D models; which demonstrates the advantages of the proposed solution.

Keywords: Laplacian mesh editing, mesh segmentation, volume preserving mesh editing, mesh animation, direct manipulation, multi-touch interaction.

ÖZET

ÇOKLU DOKUNMATİK ORTAMLAR İÇİN 3 BOYUTLU MODEL CANLANDIRMASI

Duygu Ceylan

Bilgisayar Mühendisliği, Yüksek Lisans

Tez Yöneticisi: Yard. Doç. Dr. Tolga K. Çapın

Ağustos, 2009

Bilgisayar bilimindeki hızlı gelişmeler çoklu ortam, bilgisayar oyunları, sanal gerçeklik gibi çeşitli uygulama alanlarının doğmasını sağlamıştır. Bütün bu uygulama alanları 3 boyutlu geometrik modellerin biçimlendirilip canlandırılması prensibiyle çalışmaktadır. Bu nedenle, bilgisayar modellemesini ve canlandırmasını sağlayan çeşitli araçlar geliştirilmiştir. Fakat, bu araçların çoğu modelleme ve canlandırma konularıyla ilgili belli bir seviyede deneyim gerektirmektedir. Bu durum, deneyimsiz kullanıcılar açısından bir engel oluşturmaktadır. Bu çalışmada, bu probleme çözüm oluşturmak amacıyla, özellikle amatör kullanıcılar için tasarlanmış bir canlandırma sistemi sunulmaktadır. Benimsenen ana yaklaşım model biçimlendirme uygulamalarında oldukça önemli kabul edilen Laplace yöntemini kullanmaktır. Sunulan çözüm, bir modelin göze çarpan daha küçük parçalardan oluştuğunu kabul ederek bu parçaların Laplace yöntemiyle aynı anda biçimlendirilmesini sağlamaktadır. Böylece, modele hareket ediyor hissi kazandırılmaktadır. Hem yerel düzenlemeleri hem de modelin toplu hareketini sağlayacak etkileşim teknikleri geliştirilerek daha memnun edici sonuçlar elde edilmektedir. Son olarak, belirtilen yaklaşım, çoklu dokunmatik ekran teknolojisini ve direkt manipülasyon tekniklerini kullanarak sistemin kullanılabilirliğini arttırmayı amaçlamaktadır. Belirtilen metodlar, önerilen çözümün faydalarını göstermek amacıyla farklı modeller için basit canlandırmalar yaratılarak test edilmiştir.

Anahtar sözcükler: Laplace model biçimlendirmesi, hacim korumalı model biçimlendirmesi, model animasyonu, direkt manipülasyon, çoklu dokunmatik etkileşim.

Acknowledgement

First and foremost, I would like to thank my advisor Asst. Prof. Dr. Tolga K. Çapın for his endless support during my whole Master's education. He did not only guided me to choose an interesting research topic, but also provided me with all the equipment I needed to accomplish this work. Besides helping me to overcome the problems I encountered in my work, he always encouraged me at the times I felt desperate. Finally, his valuable advice about academic life and assistance in graduate study applications helped me to acquire a chance of pursuing my academic studies in Switzerland.

Many thanks to my other thesis committee members, Prof. Dr. Bülent Özgüç and Assoc. Prof. Dr. Veysi İşler for reading and reviewing this thesis. They also gave me the opportunity to study different applications of Computer Graphics in the courses they had taught and supported my graduate study applications.

I would also like to thank my friend Sezen Erdem for helping me to solve various problems, such as setting up the working environment, fixing certain code bugs, and formatting this thesis. My other friends, Ertay Kaya and Yusuf Saran were mental supporters for me, especially towards the end of my studies. Thanks to Ertay, I did not feel lonely at all during the tough working period and Yusuf always found a way to cheer me up so that I can work with more enthusiasm.

Finally, I am grateful to my parents and my sister, who always believed in me. They always showed respect for my decisions and motivated me throughout my education life.

Contents

1	Introduction	1
1.1	Contributions	4
1.2	Organization of the Thesis	5
2	Background	6
2.1	Differential Representations for Editing Operations	6
2.1.1	Overview of Differential Representations	7
2.1.2	Using Discrete Forms in Editing	7
2.1.3	Gradient Field Based Editing	8
2.1.4	Laplacian Surface Editing	9
2.2	Mesh Segmentation	14
2.3	Volume Preserving Shape Deformations	16
2.4	User Interface Design	17
2.4.1	3D Interaction Design	17

2.4.2	Multi-touch Environments	19
3	Approach	20
3.1	Overview of the System	20
3.2	Mesh Processing Component	22
3.2.1	Model Loading	22
3.2.2	Mesh Partitioning	23
3.3	Animation Component	30
3.3.1	Laplacian Framework	32
3.3.2	Volume Preservation	42
3.4	User Interface	46
3.4.1	Multi-touch Environment	47
3.4.2	Interaction Techniques	48
3.4.3	Direct Manipulation Principle	53
4	Results and Discussion	55
4.1	Mesh Partitioning	55
4.2	Laplacian Framework	59
4.3	Volume Preservation	61
4.4	Usability Evaluation	64
5	Conclusions and Future Work	70

List of Figures

1.1	A jump action can be represented with a series of drawings of the individual states of the action. Displaying these drawings one after another creates an illusion of movement. [40].	1
2.1	A simple mesh.	10
3.1	Components of the system.	21
3.2	Flow of processes from the users' point of view.	22
3.3	A sample mesh and its data structure.	23
3.4	Mesh data structure.	23
3.5	Principle curvature directions, T1 and T2, of vertex v of a simple mesh is shown. The normal vector of the vertex is denoted by N. . .	29
3.6	Mesh partitioning process illustrated for a sample model (a) Nodes obtained after pre-partitioning process, each circle contains the node number and the sign of the average mean curvature. (b) Nodes 0,5,16,10, and 13 are merged to nodes 1,6,9,12, and 15 as a result of Rule-1. (c) Nodes 1,6,9,12, and 15 are merged to nodes 2,7,8,11, and 14 as a result of Rule-1. (d) Nodes obtained after the merging process (e) Model partitioned without merging step (f) Model partitioned with merging step	31

3.7	ROI Specification - Blue vertex is the handle selected by the user. Red vertices constitute the boundary and the in between part is the freely moving part.	39
3.8	Flowchart explaining ROI determination and computation of new vertex positions.	40
3.9	Cursor positions are projected to the z-plane on which the selected handle lies.	41
3.10	Flowchart explaining ROI determination with more than one handle.	42
3.11	Vector field is defined from the vertices of a mesh part to the closest points on the bounding box.	45
3.12	Laplacian framework uses the result of the previous Laplace operations, whereas the volume preservation component compares the final mesh with the initial mesh.	46
3.13	Flowchart of the distinguish operation between single and double clicks.	49
3.14	An improved Arcball widget.	50
3.15	An example arc between two cursor points, P_0 being the cursor down point and P_i being the cursor up point.	51
3.16	Translation vector obtained from cursor points. (a)Cursor moves from the inner region to outer region. (b)First contact with the widget is in the outer region.	52
3.17	Non-responsive region in the widget and different translation vectors are shown.	53
4.1	Mesh partitioning results for a dragon model ((a) and (b)) with different choices of the interval number, k (c) k=5 (d) k=6 (e) k=7 (f) k=8	57

4.2	Mesh partitioning results for a camel model ((a) and (b)) with different choices of the interval number, k (c) k=6 (d) k=7 (e) k=8 (f) k=9	58
4.3	A plane model with many details produces a large number of mesh parts.	59
4.4	Tail of the fish is manipulated with a single handle shown by the circle. (a) Original model (b) Manipulated model	61
4.5	Arms of the starfish are manipulated with appropriate handles shown by the circles. (a) Original model (b) Manipulated model .	62
4.6	Each wing of the dragon is manipulated with double handles shown by the circles. (a) Original model (b) Manipulated model	62
4.7	Models manipulated by disabling ((a) and (c)) and enabling ((b) and (d)) volume preservation component. Red circles denote the handle positions.	63
4.8	User interface of our system	64
4.9	Animation of a dragon model. Transformation widget is visible. .	68
4.10	Animation of a starfish model. Transformation widget is visible in (b), (c), (d), and (e).	69
5.1	Prototype design of the extended transformation widget. Orientation of the z translation component is adjusted according to the orientation of the index finger shown by an arrow inside the circles.	72

List of Tables

4.1	Partitioning time for different size of meshes	56
4.2	Computation times for matrix multiplication, factorization, and back substitution processes of Laplacian framework, given in seconds.	60

Chapter 1

Introduction

Human beings have always had the tendency to make representations of the things they see around them [35]. Man-made drawings found in the caves were the results of this tendency. Yet, single drawings or sculptures were capable of representing only a particular moment in life [35]. Therefore, the search for a means of representing objects or living things in a particular time interval continued. Finally, the invention of motion picture camera and roll film have met this request. By projecting photographs of an action onto a screen, the two instruments have given rise to a new art form. This new art form was called animation [35].

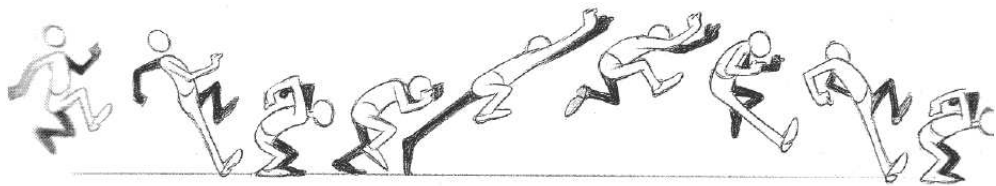


Figure 1.1: A jump action can be represented with a series of drawings of the individual states of the action. Displaying these drawings one after another creates an illusion of movement. [40].

Animation was considered as a very powerful form of art; because it gave the opportunity to represent emotions and thoughts without being limited to a certain set of actions [35]. With the development of color technology, the

effect of animation was increased further. Finally, with the use of computers as modeling and animation tools, the applications of animation art continued to extend, making it a powerful tool for expressing motion, thoughts, and feelings.

Today computer animation is used in many applications, such as computer games, animated web content, and simulators [24]. The variety of these applications give rise to research areas such as realistic object modeling and real-time computer animation. As an illustration, commercial geometric modeling tools have been developed such as 3Ds Max [3] and Blender [8]. These tools focus on creation of 3D objects as well as manipulation of these objects. However, most of these tools require an advanced knowledge or training about the concept, and they are hard to use for inexperienced users. Frequent use of object modeling and animation in computer applications makes the topic of shape modeling and animation an active research area. The deficiency of the commercial animation tools in providing an easy-to-use interface necessitates further research.

Related work to our system can be classified as object creation and object manipulation systems. A number of object creation systems are based on a sketching interface. As an example, Teddy [14] is one of the leading freeform modeling systems, where users are able to model 3D objects within a short amount of time by drawing 2D sketches. Similar easy-to-use object modeling systems have triggered the study of object manipulation. The approaches presented for this purpose fall into a variety of categories. Free form deformation (FFD) is one of the popular categories. In FFD applications, an object is surrounded by a lattice, and editing actions that are applied to the lattice manipulate the object in response. Another important category of modeling solutions is the differential representation. Differential solutions aim to encode shape features of a model and preserve them during manipulation. Obviously, this property of differential approaches makes them popular for realistic and shape preserving editing operations. Because of this, several systems have been developed based on this approach. Some of these system are also sketch-based, in which users define editing commands either by silhouette sketching as in the work of Nealen et al. [23] or gesture sketching as in FiberMesh [22]. There are also systems that involve a direct manipulation interface, where users interact directly with

the object being edited. Examples of this kind include the work of Sorkine et al. [33] and Lipman et al. [19].

The majority of the examples given above focus on building easy-to-use tools for object modeling and local deformations of 3D models. The problem of building animation systems for novice users still exists as an active research area. Traditional methods in computer animation production, such as key-frame animation or motion capture, are not applicable for novice users. These methods either require profound experience about drawing, or complicated and expensive setups. On the other hand, tools designed for inexperienced users should be easy-to-use and should have simple interfaces. In this thesis, we aim to develop a solution for this problem, by building an animation system via enhancing current manipulation techniques. We present a tool which allows intuitive creation of animations.

Our main approach in this work is based on improving differential methods used for model editing operations. We accomplish this improvement by automatically defining editing regions on a model and adding a volume preservation mechanism. Both of these improvements have an obvious effect on creating realistic animations. We also explore the use of the emerging technology of multi-touch screens. This new technology not only presents a more appealing environment for current desktop interfaces, but also triggers the development of new widgets and gestures. The reasons for using a multi-touch screen in our case are easing animation production and providing a more involving interface. Using multiple fingers instead of only a single mouse cursor enables the object of interest to be edited at several regions simultaneously, thus creating a more realistic animation. Moreover, the multi-touch screen gives the users the feeling of manipulating objects by hand.

1.1 Contributions

In this work we present a novel 3D mesh animation system. We can list our main contributions as follows:

- **An automatic scheme for definition of animated model parts.** In our system, each 3D model is segmented into semantically meaningful parts, before any editing operation takes place. When the user selects a few vertices of the mesh to manipulate, the corresponding parts of the mesh are automatically defined as the animating parts. Therefore, the user does not have to specify which parts of the model are to be affected from the manipulation. The segmentation process is based on the fact that humans perceive objects as a collection of smaller parts. As a result, the parts marked as animating coincide with the users' expectations from the animation of the object of interest.
- **Volume preserving mechanism.** When physical constraints in 3D animation are considered, one of the most important facts that arise is conservation of mass. When the density of an object remains constant during animation, this implies preservation of volume. For this reason, in our system we combine this physical constraint with our interactive and direct editing framework. Volume preservation is especially important for creating stretch and squash effects that result in a more expressive animation.
- **An interface that enables editing operations with direct control.** The interface that we are presenting benefits from the growing technology of multi-touch interaction. We combine direct manipulation principles with this technology to give the users direct control over the system. Being able to manipulate models as if holding in hand increases the feeling of immersion.

1.2 Organization of the Thesis

This thesis is organized as follows. Chapter 2 presents related work about the important concepts for our work. These concepts include differential representations for mesh editing operations, mesh segmentation techniques, 3D user interface design, and multi-touch environments. In addition, fundamental notations about these concepts are introduced to enable a better understanding of our work. Chapter 3 describes our approach for designing a mesh animation system targeted for multi-touch environments. This chapter first gives a brief overview of our system; then details the mesh processing, animation, and user interaction methods that we have applied. Chapter 4 provides the results of our system and discusses both advantages and disadvantages of our work. Finally, Chapter 5 concludes the thesis by summarizing what has been done and what can be considered as a future extension of our work.

Chapter 2

Background

In this chapter, we briefly review primary mesh editing techniques based on differential representations and introduce the fundamentals of Laplacian surface editing. Furthermore, we examine approaches regarding mesh segmentation, volume preserving shape deformations, and user interface design, other dominant notions for our work.

2.1 Differential Representations for Editing Operations

Mesheres are widely used to represent models in computer graphics. A mesh is a collection of vertices, edges, and faces that define the shape of the model being represented. Several different surface representation schemes exist, each exhibiting different features of a mesh model. To illustrate, simple triangular representations based on global cartesian coordinates are best for examining the topological structure of the model [31]. However, these simple representations are not adequate to carry modeling operations such as mesh editing, coating, resampling etc. Lately, differential representations of surfaces have gained popularity due to their advantages in modeling operations. These representations encode the shape

features of a surface and enable these features to be preserved in a modeling operation. In the following sections, we give a more detailed overview of the use of differential representations in one of the most frequent modeling operations, mesh editing. Mesh editing operations are also at the core of our system. We specifically focus on the Laplacian framework, a popular differential representation that we have based our work on.

2.1.1 Overview of Differential Representations

Preserving the shape and geometric details of a model is vital for editing operations. For this purpose, local surface modeling representation approaches have been proposed. Preserving geometric details means minimizing the difference between the deformations of local features of a model. These local features can be encoded via differential representation techniques such as (i) discrete forms, (ii) gradient fields, and (iii) Laplacian coordinates. These encodings are defined by considering each vertex of a mesh with its neighboring vertices. Since differential methods represent local features of a model, minimizing the difference between local features before and after a deformation operation means minimizing the difference between the differential coordinates.

In differential mesh deformation systems, users control the editing process by updating the positions of a few vertices called handles. The new positions of the remaining vertices are derived by considering the handle positions as constraints. The core of this process is to recover global Cartesian coordinates from the deformed differential coordinates, which requires solving a sparse linear system [33].

2.1.2 Using Discrete Forms in Editing

Discrete form based approaches in mesh editing define a local frame at each vertex, and represent the transition between the adjacent frames in terms of discrete forms. To define a local frame at a vertex, the one-ring neighborhood of the vertex is examined. Each edge connecting the vertex with a neighbor

is projected onto the tangent plane of the mesh at the given vertex. The first discrete form is used to encode the lengths of the projected edges and the angles between the adjacent projected edges, whereas the second discrete form encodes the normal directions. In other words, these two forms represent the geometry of a vertex and its one-ring neighborhood up to a rigid transformation [20]. During manipulation operations, local frames at each vertex are reconstructed from the discrete forms. This problem is expressed as a sparse linear system of equations. Once the local frames are reconstructed, global coordinates are also reestablished by integration of the local frames. This is also expressed as a linear system of equations.

In editing operations, users define a handle vertex to move freely and some boundary vertices which remain fixed to limit the deformation. The positions of these vertices are added to the system of equations to make it over determined. Finally, the over determined system of equations is solved in a least squares manner. The details of the approach are out of scope of this thesis. Readers can refer to the work of Lipman et al. for further information [20].

2.1.3 Gradient Field Based Editing

Gradient field based mesh editing is based on the Poisson equation. The Poisson equation is an alternative to the least squares problem. To apply Poisson equation to mesh editing, the coordinates of the target mesh are represented as scalar fields on the input mesh. These scalar fields are used to form the gradient field component of the Poisson equation [42]. In editing operations, the gradient fields are transformed by means of transforming triangles. Resulting vector fields are no longer gradients of a scalar function. Therefore, they are considered as guidance vector fields of the Poisson equation and the equation is solved to reconstruct the desired mesh coordinates. Finally, a boundary condition for a mesh is defined by considering the set of connected vertices on the mesh, the set of vertex positions, the set of frames that define local orientations of the vertices, the set of scaling factors, and a strength field. Strength field represents how much a vertex is affected by the boundary conditions and is computed based on

the distance between the initial and final vertex positions. During editing, scale and local frame changes of the constrained vertices in the boundary condition are propagated to the rest of the mesh. For detailed examples of applying this scheme to mesh editing operations, the work of Yu et al. can be examined [42].

2.1.4 Laplacian Surface Editing

Laplacian coordinates are a form of differential representation used widely in mesh editing. Conversion from Cartesian coordinates to Laplacian coordinates requires only a linear operation, whereas the reconstruction of Cartesian coordinates is accomplished by solving a linear system. This linearity of the approach makes it very efficient and it is the main reason why we have also used Laplacian editing in our work. In this section, we look at how these coordinates are defined and used in editing operations.

2.1.4.1 Laplacian Representation

Laplacian coordinates focus on the difference between a vertex and its neighborhood. An overview report by Sorkine about the subject describes Laplacian representation in detail [31].

Let global Cartesian coordinates of a vertex i of the connected mesh M be denoted by v_i . The Laplacian coordinate (δ -coordinate) of this vertex is defined as the difference between v_i and the average of its neighbors [33]:

$$\delta_i = v_i - \frac{1}{d_i} \sum_{j \in N(i)} v_j \quad (2.1)$$

where $N(i)$ is the set of neighbors of vertex i , and d_i is the number of neighbors. Here the δ -coordinate is defined with uniform weights. As stated in the work of Sorkine, uniform weights work sufficiently well in most editing scenarios [33]. However, cotangent weights can be used as well, especially when the considered

mesh is not sufficiently regular [32].

The operation of transforming Cartesian coordinates to δ -coordinates can also be represented with matrices. Let A be the adjacency matrix of the mesh where A_{ij} is 1 if (i, j) represents an edge, 0 otherwise. Similarly, let D be the diagonal matrix such that $D_{ii} = d_i$. Then, the Laplacian matrix, L , which converts cartesian Coordinates to δ -coordinates, is defined as follows:

$$L = I - D^{-1}A \quad (2.2)$$

Usually, the symmetric of the Laplacian matrix is used in computations. This matrix can be called as L_s :

$$L_s = D - A \quad (2.3)$$

L_s should be normalized before being used to define the transformation of coordinates. We can illustrate this process with a simple example. Assume that we have the simple mesh given in Figure 2.1.

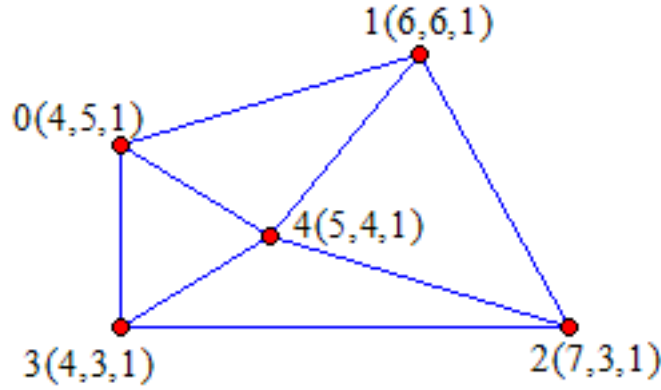


Figure 2.1: A simple mesh.

The symmetric Laplacian matrix for this mesh is:

$$L_s = \begin{pmatrix} 3 & -1 & 0 & -1 & -1 \\ -1 & 3 & -1 & 0 & -1 \\ 0 & -1 & 3 & -1 & -1 \\ -1 & 0 & -1 & 3 & -1 \\ -1 & -1 & -1 & -1 & 4 \end{pmatrix}$$

This matrix can be formed by just examining each vertex and its neighbors. For example, the first row of this matrix corresponds to v_0 and is computed as follows. Since v_0 has 3 neighboring vertices, the diagonal element becomes 3. Then, the column elements corresponding to the neighbors of v_0 are assigned the value -1 . Rest of the column elements are given the value 0.

Once this matrix is computed, it is normalized by setting the diagonal elements in each row to 1. Then it is substituted into the transformation equation as shown below. (This example uses x coordinates, transformations of y and z coordinates are similar.):

$$L_s * v = \delta$$

$$\begin{pmatrix} 1 & -0.33 & 0 & -0.33 & -0.33 \\ -0.33 & 1 & -0.33 & 0 & -0.33 \\ 0 & -0.33 & 1 & -0.33 & -0.33 \\ -0.33 & 0 & -0.33 & 1 & -0.33 \\ -0.25 & -0.25 & -0.25 & -0.25 & 1 \end{pmatrix} * \begin{pmatrix} 4.0 \\ 6.0 \\ 7.0 \\ 4.0 \\ 5.0 \end{pmatrix} = \begin{pmatrix} -1.0 \\ 0.666 \\ 2.0 \\ -1.333 \\ -0.25 \end{pmatrix}$$

In conclusion, constructing δ -coordinates from Cartesian coordinates is a straightforward process which requires only a linear operation.

2.1.4.2 Laplacian Editing

The basic idea behind using Laplacian coordinates in mesh editing operations is to perform manipulations on the mesh represented by these coordinates and

reconstruct the global Cartesian coordinates afterwards.

As explained in the previous section, definition of Laplacian coordinates of a mesh is not difficult. However, reconstructing cartesian coordinates from δ -coordinates is more complicated. The immediate reaction to take is to invert the L_s matrix defined previously. However, this matrix is singular. The singularity can be shown as follows. The sum of the elements in each row sum up to zero and since this matrix is symmetric, sum of the elements in each column is also zero. In other words, when row vectors of the matrix are added, a zero vector is obtained. Therefore, the last row of the matrix is in fact the sum of the other rows negated. This means, the last row is dependent on the other rows and the matrix rank is less than the matrix size. Since a matrix of size $n \times n$ and rank of $r, r < n$ is singular, the expression $x = L_s^{-1}\delta$ is not defined. Therefore, Cartesian coordinates can be recovered by defining at least one vertex as a constraint and then solving the linear system. We can define this set of vertices as C . These constraints are added to the previously defined system of equations. To illustrate, we assume v_0 is a constraint vertex for our mesh and compute the new L_s matrix:

$$\tilde{L} = \begin{pmatrix} 1 & -0.33 & 0 & -0.33 & -0.33 \\ -0.33 & 1 & -0.33 & 0 & -0.33 \\ 0 & -0.33 & 1 & -0.33 & -0.33 \\ -0.33 & 0 & -0.33 & 1 & -0.33 \\ -0.25 & -0.25 & -0.25 & -0.25 & 1 \\ \textcolor{red}{1} & \textcolor{red}{0} & \textcolor{red}{0} & \textcolor{red}{0} & \textcolor{red}{0} \end{pmatrix}$$

This new matrix is named as $\tilde{L}$. Each constraint added to the system may have a different weight, w_i . In our example we have used the weight of 1.0.

Adding constraints to the system makes it over-determined, which means it may not have an exact solution. However, the system can be solved in a least-squares sense. Therefore, an error metric is defined for this purpose:

$$E(v') = \sum_{i=1}^n \|\delta_i - L(v'_i)\|^2 - \sum_{i \in C} w_i \|v'_i - c_i\|^2 \quad (2.4)$$

The solution to this least squares problem can be expressed as matrix operations. $\tilde{L}$ is an $(n + m) \times n$ matrix, where n is the number of the vertices in the mesh, and m is the number of constrained vertices.

$$\begin{aligned} v' &= (\tilde{L}^T \tilde{L})^{-1} \tilde{L}^T b \\ (\tilde{L}^T \tilde{L}) v' &= \tilde{L}^T b \end{aligned} \quad (2.5)$$

In the above equation, b denotes the right-hand side of the linear system. It is a $(n + m) \times 1$ vector where the first n elements are the δ -coordinates of the mesh vertices, and the remaining m elements are the Cartesian coordinates of the constrained vertices. If we return back to our example, the above equation becomes:

$$(\tilde{L}^T \tilde{L}) v' = \tilde{L}^T \begin{pmatrix} -1.0 & 0.666 & 2.0 & -1.333 & -0.25 & \textcolor{red}{4.0} \end{pmatrix}^T$$

$$\begin{pmatrix} 2.2847 & -0.6042 & 0.2847 & -0.6042 & -0.3611 \\ -0.6042 & 1.2847 & -0.6042 & 0.2847 & -0.3611 \\ 0.2847 & -0.6042 & 1.2847 & -0.6042 & -0.3611 \\ -0.6042 & 0.2847 & -0.6042 & 1.2847 & -0.3611 \\ -0.3611 & -0.3611 & -0.3611 & -0.3611 & 1.4444 \end{pmatrix} v' = \begin{pmatrix} 3.2847 \\ 0.3959 \\ 2.2847 \\ -1.6041 \\ -0.3611 \end{pmatrix}$$

The product of a matrix and its transpose is a positive, semi-definite matrix and can be factorized via Cholesky factorization. Therefore, the product $(\tilde{L}^T \tilde{L})$ can also be written as the product of an upper triangular matrix and its transpose, $M = (\tilde{L}^T \tilde{L}) = (\tilde{R}^T \tilde{R})$. This factorization is used to solve the above equation. Remember that an appropriate linear system should be formed for each x,y, and z coordinate to obtain the full solution.

$$\begin{aligned}
(\tilde{L}^T \tilde{L})v' &= \tilde{L}^T b \\
Mv' &= \tilde{L}^T b \\
(\tilde{R}^T \tilde{R})v' &= \tilde{L}^T b \\
\tilde{R}v' &= x \\
\tilde{R}^T x &= \tilde{L}^T b
\end{aligned}
\tag{2.6}$$

In editing operations, users choose some vertices on the mesh as handles and manipulate them to achieve desired deformations. The manipulations of the handles are propagated to the rest of the mesh until a boundary region is reached. In other words, there is also a set of boundary vertices which remain fixed during manipulation. In order to correctly reconstruct Cartesian coordinates, both the handles and the boundary vertices are defined as constraints to form the above over-determined system. The system matrix is factorized once at the beginning of the editing session and the above equation is solved repeatedly as the movement of the handle vertices change the right hand side vector.

2.2 Mesh Segmentation

As discussed in the previous section, developing different representation schemes of mesh models has been an ongoing research area. Just like differential representations are useful for encoding shape features, examining a model as a combination of simpler components is an attitude close to human perception [1]. This is the main reason why we have developed the approach of applying mesh editing operations to smaller parts of a model.

Mesh segmentation approaches can be analyzed in two main groups. The first group of approaches concentrate on partitioning a mesh based on some sort of geometric property such as curvature, whereas the second group of algorithms focus on the features of the model to obtain meaningful parts [2].

Among the approaches proposed for the second group of mesh segmentation, the work of Katz. et al. can be listed [17]. This approach follows a two-step algorithm. The first step decomposes the mesh into meaningful components keeping the boundaries fuzzy via a clustering algorithm. The second step finds the exact boundaries in accordance with the features of the model such as curvature and the angle between the normals of the adjacent faces of the mesh. Another solution presented by Katz et al. examines all models in a pose-insensitive representation using Multi-Dimensional Scaling (MDS) [16]. Feature points of the model used in segmentation are extracted in this representation. Another approach is Antini et al.'s work, which is based on the visual saliency of the mesh parts [1]. The first step of this algorithm is to compute, for each vertex, the sum of the geodesic distances between the given vertex and the remaining vertices. The sums computed are normalized to a certain range and divided into a constant number of levels, called nodes. The authors comment on the choice of this constant according to experimental results. The second step of the algorithm examines the nodes with respect to adjacency, curvature, and boundary information, and merges some of the nodes with similar properties. To state briefly, the merging step aims to combine adjacent nodes with the same sign of average curvature. Each node obtained in the final stage represents a meaningful part of the mesh.

As stated before, in our work, we are interested in applying mesh editing operations into smaller parts of a model. Since the human visual system perceives a model as a combination of visual features, the smaller parts used in our system should be the meaningful parts of the model. Therefore, the second group of mesh segmentation solutions is more suitable for our system. Among the existing solutions given above, our system is based on the approach presented by Antini et al. [1] because it is based on visual saliency. The details of this approach and how we have made use of it are given in the following chapter.

2.3 Volume Preserving Shape Deformations

Animation of an object can be considered as a sequence of manipulation actions applied to it, such as bending, stretching, and compressing. To obtain plausible results, physical constraints should be taken into account during these manipulations. Preservation of volume is one of these important animation constraints, which follows from the conservation of mass principle. If the density of an object remains constant during animation, volume is also preserved. For this reason, applications that preserve volume are more advantageous in creating realistic animations.

Zhou et al. [43] present a 3D mesh deformation system, which focus on preserving volume during large deformations. Their solution is based on building a volumetric graph from a 3D mesh, which contains both the original mesh vertices and the points of a lattice constructed inside the mesh [43]. Volumetric features are represented by the Laplacian coordinates of the nodes of this graph. Volume is preserved by defining an energy function and minimizing it. Work of Hirota et al. [12] is another example system which tries to preserve volume during free form deformations. This approach also defines an energy function similar to potential energy functions for elastic solids and tries to minimize it. This energy function is defined so as to measure deformation and volume preservation constrains the minimization process [12].

In contrast to the examples mentioned above, Funck et al. [38] present a more straightforward mechanism to preserve volume. Even though, this work focuses on mesh skinning applications, authors claim that the volume of a model in an arbitrary deformation can be preserved similarly. This solution first defines a displacement vector field and applies a volume correction step after each deformation. The correction step adds the displacement field to the deformed mesh coordinates with appropriate scaling factors in order to preserve the volume before and after deformation.

We base the volume preservation component of our work on the approach presented by Funck et al. [38] because this approach does not require additional

volumetric structures or control components. Therefore, it can easily be adapted to triangular meshes. It is also sufficient to add a volume correction step after Laplacian editing computations so that the overall complexity of the system is not increased.

2.4 User Interface Design

Many graphics applications involve 3D scenes and require users to interact with the scene. Most of these applications work on desktop environments with traditional input devices such as a 2D mouse, whereas some work with devices with more degrees of freedom (DOF). Cubic Mouse [9], ShapeTape [10], and Control Action Table [11] are some of the interesting examples of high DOF devices. Another input device that is gaining high popularity is the multi-touch screen. Different interaction techniques have been developed for these different input instruments. In this section we review two important examples of these techniques, namely widget-based methods and direct manipulation based techniques. We also discuss recent work on multi-touch environments, which is of our interest in this work.

2.4.1 3D Interaction Design

Interaction design for a 3D application is more challenging compared to the 2D case, because manipulating 3D objects on a computer is a less familiar experience for novice users [13]. Moreover, most of the commonly used input devices supply only 2D information. Therefore, techniques to map 2D input to 3D manipulation tasks should be developed. Another user need related to 3D interaction is that the interaction with the scene should be intuitive. The direct manipulation mode of interaction best suits this second need, because it allows users to directly select objects and apply actions on them. Meanwhile, the display immediately shows the results of the user actions. This is especially important in model design and editing systems. In conclusion, the challenges in 3D graphics applications have

given rise to the design of more effective user interfaces which adopt direct manipulation approach, and the use of gestures and widgets to enhance interaction with the objects in the scene.

Common 3D manipulation tasks include rotation, translation, and scale. Performing these operations simultaneously is a challenging task with only a 2D input device, since these operations have more than 2 DOF. Forcing the user to constantly switch between different operation modes makes the interface distracting. The importance of gestures and widgets arises at this point because they are a powerful tool for mapping the 2D input to 3D. Therefore, they are frequently used in different applications. As an illustration, the work presented by Draper et al. shows how gestures can be used in Free Form Deformation (FFD), a common model editing scheme [7]. In traditional FFD interfaces, users have to manipulate control points on the lattice of the object, which is a challenging task because it is hard to foresee the result. To overcome this difficulty, Draper et al. [7] present gestures for common operations such as bending, twisting, stretch, and squash. Commands are given by drawing strokes directly on the model itself instead of interacting with the lattice. Similarly, the work of Nealen et al. introduces sketch-based gestures for mesh editing via differential representation [23]. The user sketches a stroke to define the silhouette of the mesh part to be edited and performs the editing action by sketching the new shape of the silhouette. This system also enables direct interaction with the object.

3D widgets are as important as gestures in describing 3D editing tasks. They are used to place controls directly in a 3D scene with the objects. Generally each widget connected to an object is responsible for a small set of manipulation operations [5]. As an illustration, a common widget called Virtual Sphere enables the users to rotate an object about an arbitrary 3D axis [6]. The object is assumed to be surrounded by a virtual sphere ball and the user rotates this ball instead of directly rotating the object. Similarly, the technique presented by Houde et al. surrounds an object with a bounding box in the shape of a rectangular prism and places handles at specific positions of the box to perform transformation operations [13]. For example, rotation handles are placed at the corners of the box where as lifting handle is found on the top face of the box.

In conclusion, in the examples given above, users deal directly with the objects in a scene via either a gesture or a widget. This feature enables them to feel more involved in the actions they are taking; making the interface more easy-to-learn.

2.4.2 Multi-touch Environments

As discussed in the previous section, the ability to manipulate objects directly on a screen is appealing for many users, because they feel more involved in the task. Touch screens, especially multi-touch screens, enhance this ability by enabling users to touch objects on a screen so that they feel like controlling objects directly. For this purpose, these instruments are increasingly used in graphical applications.

The multi-point input feature of multi-touch environments makes them suitable for gestural interfaces. Therefore, a variety of related work focuses on building gestures and more efficient interaction styles for these environments. As an illustration, Wu et al. presents different gestures for single finger, multi-finger, single hand, and multi-hand usage especially for tabletop displays [41]. The work of Benko et al. is another example which introduces new widgets and interaction methods to enhance clicking and selection operations [4]. Similarly, Rekimoto presents both a multi-touch sensor architecture and new interaction techniques that are hard to implement with a traditional mouse [27].

The nature of multi-touch environments is also well suited for direct manipulation interfaces. One of the best examples to illustrate this is the mesh editing system presented by Igarashi et al. [14], which works both in traditional desktop environments and multiple-point input device SmartSkin [27]. In this system, users are able to directly manipulate 2D meshes displayed on a multi-touch screen by touching and moving them.

On the whole, we can expect multi-touch environments to provide a good means to improve the previously presented approaches in effective interface design with their primary feature of providing multi-point input.

Chapter 3

Approach

In this chapter, we present our approach to the problem of interactive mesh animation. Our aim is to develop not only an editing but a simple 3D mesh animation system. The system is designed for multi-touch environments and targeted especially for novice users. The method we have adopted for mesh editing, the additional work done to enable animation, and the principles we have followed in user interface design are discussed in further detail.

3.1 Overview of the System

Our mesh animation system is composed of different components: mesh processing, animation, and user interface components. To briefly explain, the mesh processing component loads a model into the system and partitions it into meaningful parts. The animation component is responsible of mesh manipulation operations. Finally, the user interface component includes user interaction techniques. This architecture is summarized in Figure 3.1 and each component is described in detail in further sections.

Our mesh animation system is targeted for a multi-touch screen, but it can work just as well on a traditional desktop computer. The flow of the processes in

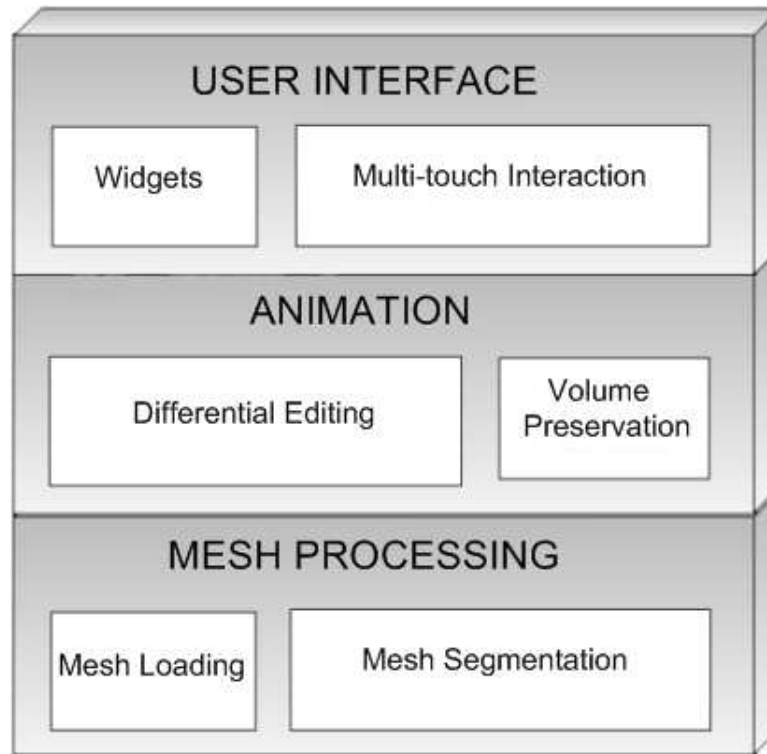


Figure 3.1: Components of the system.

this system from the users' point of view is as follows. Once a 3D model is loaded to the system, users can start animating it by first switching to the "Animation" mode. Otherwise, the model can only be globally translated, rotated, or scaled. When the "Animation" mode is active, users select a handle by just touching (clicking) the desired location of the model. In the background, the part of the model that will be affected from the editing operation is calculated automatically. The manipulation is based on the "drag-and-drop" principle, which means users move their fingers (or the mouse) which in turn move the handle. Meanwhile, the rest of the affected part is deformed accordingly. By taking advantage of the multi-point feature of the multi-touch screen, users can select as many handles at different locations of the model as they want and deform different parts of the model simultaneously. During the deformation process, the model can also be globally transformed. Users select this command by double touching (clicking) the model which enables a transformation widget. With this widget, users can translate or rotate a model while deforming certain parts of it. This scenario is summarized in the flow chart given in Figure 3.2.

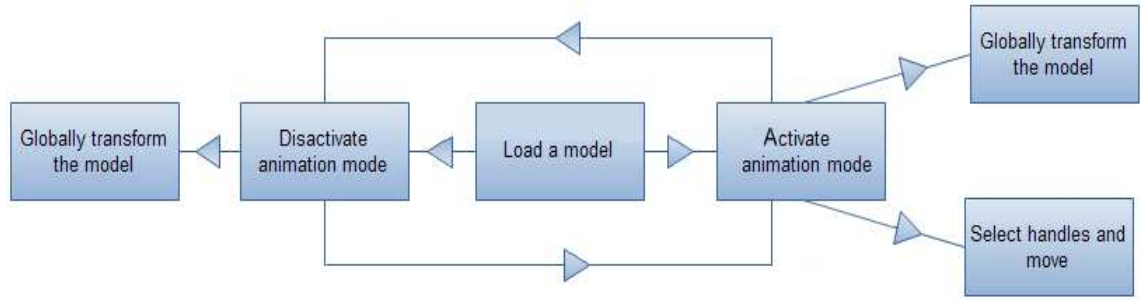


Figure 3.2: Flow of processes from the users' point of view.

3.2 Mesh Processing Component

In this section, we focus on mesh processing component of our system, which forms the base of the architecture. This component includes two smaller parts, which are responsible for model loading and mesh partitioning. These two parts are described in the coming sections.

3.2.1 Model Loading

The first action to take in order to use our system is to load a 3D model. 3D models loaded should be triangular meshes since further processing operates on triangles. The system currently supports 3D file formats, in which the coordinates of vertices and indices of the vertices forming a face are included. In addition, if the model includes textures, or special material and lighting effects, related information is also read from the model file.

When a model is loaded, the system forms a mesh data structure. The main components of this data structure are vertices and faces. The neighboring relations between these sets are also stored. For each vertex, lists of neighbor vertices and neighbor faces are defined. A face is considered as a neighbor to a vertex, if it contains that vertex. Similarly, for each face, lists of vertices forming the face and neighbor faces are defined. A sample mesh and the corresponding data structure is given in Figure 3.3.

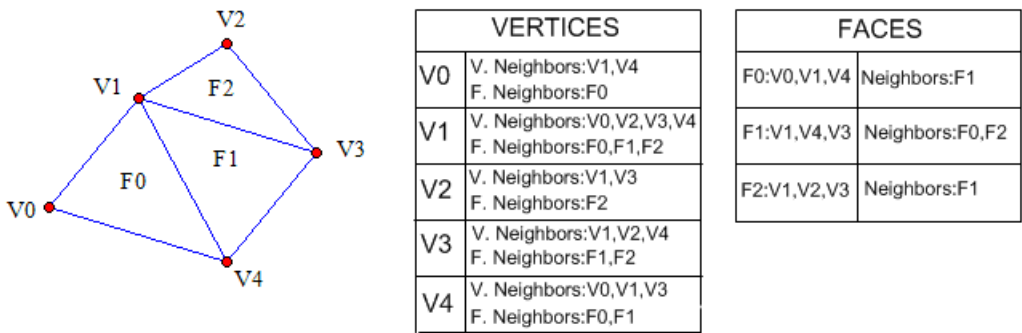


Figure 3.3: A sample mesh and its data structure.

The mesh data structure also contains a graph of nodes that represent the partitioned parts of the model. This graph is completed as a result of the partitioning process. Each node mainly contains the vertex indices that are found in the part of the mesh represented by the node. Details of the mesh partitioning process are discussed in the next section.

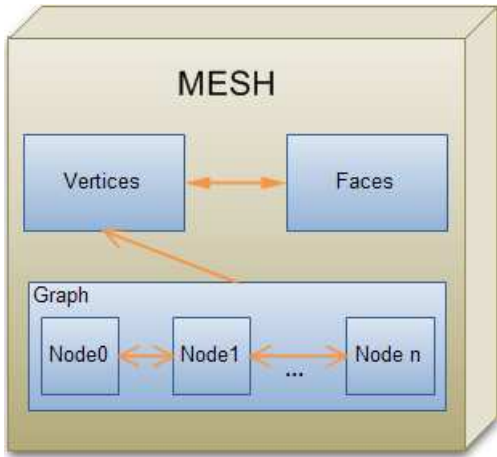


Figure 3.4: Mesh data structure.

3.2.2 Mesh Partitioning

One of our goals in this work is to extend current mesh editing approaches to enable simple animations. Our main observation is that users tend to animate salient parts of a model. This can be the tail of a fish, the ear of a dog, or the arm of a character. This is due to the fact that the human visual system perceives a

model as a combination of smaller parts. Therefore, a model partitioning scheme should be close to human perception. Based on this observation, in our system, a 3D model is partitioned into meaningful parts. The approach we follow is the method described by Antini et al. [1], which is based on the visual saliency of the parts of a model. This approach can be summarized in two steps. The first step identifies different regions of vertices on a mesh based on distances between them. The second step analyzes these regions and merges some regions according to their adjacency and curvature features. The details of the algorithm are discussed below.

3.2.2.1 Mesh Pre-partitioning

The mesh partitioning algorithm first defines a function on a mesh M with vertices v based on geodesic distance:

$$g(v_i) = \sum_{v_j \in M} \psi(v_i, v_j) \quad (3.1)$$

The above function computes a geodesic distance value for each vertex by summing the geodesic distances between the given vertex and any other vertex denoted by $\psi(v_i, v_j)$. Dijkstra's shortest path algorithm is used to compute the geodesic distance between two vertices [39]. On a mesh, this distance represents the minimum length between the vertices connecting them through the edges of the mesh.

After all the $g(v_i)$ values are computed, smallest (g_{min}) and largest (g_{max}) of these values are used to normalize all the values to the range $[0,1]$:

$$g_{normalized}(v_i) = \begin{cases} \frac{g(v_i) - g_{min}}{g_{max} - g_{min}} & \text{if } g_{min} \neq g_{max} \\ 0 & \text{otherwise} \end{cases} \quad (3.2)$$

The normalized values of $g(v_i)$ are divided into a certain number of intervals. If the number of intervals is n , then k^{th} interval contains values in the range of

$[(k-1)/n, k/n)$. This number of intervals affects the number of regions obtained as a result of partitioning and can be fixed or determined with a heuristic. Antini et al. [1] discuss that $n = 7$ is a reasonable choice according to the experimental results. In our system, users can optionally change this number with a slider. The possible values are chosen as 5, 6, 7, 8, and 9.

Once the intervals are determined, adjacent vertices falling into the same interval are grouped. Each group formed this way is called a node. These nodes form a graph, where two nodes are connected if there exists an edge between two vertices, one from each node. The number of node neighbors of each node can be denoted as e_i .

3.2.2.2 Mean Curvature Computation

Several properties are defined for each node in the graph. First of all, boundary vertices of each node are determined. These are the vertices that have no neighbor or have a neighbor in another node. Secondly, the average value of the mean curvature is calculated for each node. For this purpose, the mean curvature for each vertex should be computed. The approach described by Taubin [34] is used for this computation, because it is linear in time and includes simple and direct operations. The details of this algorithm are as follows.

The faces of the triangular mesh M defined before can be called f . For each vertex v_i of this mesh, the set of neighboring vertices N_i and incident faces I_i are defined. Further, the number of neighboring vertices and the number of incident faces at v_i are denoted by d_i and b_i .

The curvature computation begins with the estimation of the normals for each vertex as the weighted average of the normals of the incident faces. The weights are proportional to the surface area of the faces.

$$n_{v_i} = \frac{\sum_{f_j \in I_i} |f_j| n_{f_j}}{\left\| \sum_{f_j \in I_i} |f_j| n_{f_j} \right\|} \quad (3.3)$$

In the above equation, n_{v_i} and n_{f_j} denote the normals of v_i and f_j whereas $|f_j|$ is the surface area of the face.

Next, for each vertex, a 3 x 3 matrix is formed which has n_{v_i} as an eigenvector corresponding to the eigenvalue 0. The formulation of this matrix is as follows:

$$\tilde{M}_{v_i} = \sum_{v_j \in N_i} w_{ij} \kappa_{ij} T_{ij} T_{ij}^T \quad (3.4)$$

In the above expression, T_{ij} is defined as the projection of the vector $v_j - v_i$ onto the tangent plane $\langle n_{v_i}^T \rangle$:

$$T_{ij} = \frac{(I - n_{v_i} n_{v_i}^T)(v_j - v_i)}{\|(I - n_{v_i} n_{v_i}^T)(v_j - v_i)\|}. \quad (3.5)$$

κ_{ij} is the directional curvature in the direction of T_{ij} and is computed as:

$$\kappa_{ij} = \frac{2n_{v_i}^T(v_j - v_i)}{\|v_j - v_i\|^2}. \quad (3.6)$$

Finally, the weight w_{ij} is chosen proportional to the sum of the areas of the triangles that are neighbors of both v_i and v_j . The number of such triangles is 1 if one of the vertices lies on the boundary of the mesh, 2 otherwise. In addition, for each vertex v_i sum of these weights is set to 1.

$$\sum_{v_j \in N_i} w_{ij} = 1 \quad (3.7)$$

As mentioned before, n_{v_i} is an eigenvector of the constructed matrix corresponding to the eigenvalue of 0. Other eigenpairs should be computed to estimate the principle curvature directions. Taubin advises to form a Householder matrix for this purpose [34]:

$$Q_{v_i} = I - 2W_{v_i}W_{v_i}^T \quad (3.8)$$

The steps for computing W_{v_i} given in the above expression are not complicated. To begin with, a first coordinate vector $E_1 = (1, 0, 0)^T$ is defined. If $\|E_1 - n_{v_i}\| < \|E_1 + n_{v_i}\|$, W_{v_i} is equal to the difference between the first coordinate vector and the normal vector. Otherwise, it is equal to the sum of the two vectors.

$$W_{v_i} = \begin{cases} \frac{E_1 - n_{v_i}}{\|E_1 - n_{v_i}\|} & \text{if } \|E_1 - n_{v_i}\| < \|E_1 + n_{v_i}\| \\ \frac{E_1 + n_{v_i}}{\|E_1 + n_{v_i}\|} & \text{otherwise} \end{cases} \quad (3.9)$$

The first column of the Householder matrix is equal to either positive or negative n_{v_i} . The other two columns called $\tilde{T}_1$ and $\tilde{T}_2$ are used to estimate the principal curvature directions. Since n_{v_i} is an eigenvector of $\tilde{M}_{v_i}$ with the eigenvalue 0, the following equality can be written:

$$Q_{v_i}^T \tilde{M}_{v_i} Q_{v_i} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & \tilde{M}_{v_i}^{11} & \tilde{M}_{v_i}^{12} \\ 0 & \tilde{M}_{v_i}^{21} & \tilde{M}_{v_i}^{22} \end{pmatrix} \quad (3.10)$$

When the 0-row and 0-column are discarded from the above matrix, a 2 x 2 matrix remains. This matrix is diagonalized with Given's rotation to obtain an angle θ [26].

$$\theta = \frac{\tilde{M}_{v_i}^{22} - \tilde{M}_{v_i}^{11}}{2\tilde{M}_{v_i}^{12}} \quad (3.11)$$

The angle obtained is used together with the second and third columns of the Householder matrix Q_{v_i} , $\tilde{T}_1$ and $\tilde{T}_2$, to find the other two eigenvectors of $\tilde{M}_{v_i}$.

$$\begin{aligned} T_1 &= \cos(\theta)\tilde{T}_1 - \sin(\theta)\tilde{T}_2 \\ T_2 &= \sin(\theta)\tilde{T}_1 + \cos(\theta)\tilde{T}_2 \end{aligned} \quad (3.12)$$

These two eigenvectors are the principle curvature directions, the curvature values are computed from the corresponding eigenvalues, e_1 and e_2 .

$$\begin{aligned} e_1 &= T_1^T M_{v_i} T_1 \\ e_2 &= T_2^T M_{v_i} T_2 \\ \kappa_1 &= 3e_1 - e_2 \\ \kappa_2 &= 3e_2 - e_1 \end{aligned} \quad (3.13)$$

Once the principal curvatures are computed, the mean curvature is simply the average.

$$\kappa_{mean} = (\kappa_1 + \kappa_2)/2.0 \quad (3.14)$$

Figure 3.5 illustrates this process by demonstrating the normal vector and the principle curvature directions at a vertex of a simple mesh. The tangent plane, $\langle N_{v_i}^T \rangle$ is also shown.

3.2.2.3 Merging of Nodes

After the mean curvature for each vertex is computed as described, the average mean curvature for each node is easily calculated. This property together with the boundary properties is used to join adjacent nodes because the mesh could be over-segmented at this stage. In other words, after all of the nodes and their

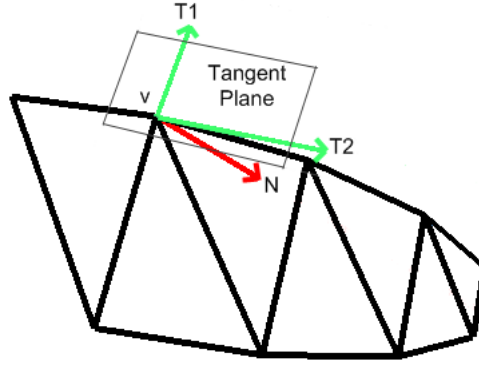


Figure 3.5: Principle curvature directions, $T1$ and $T2$, of vertex v of a simple mesh is shown. The normal vector of the vertex is denoted by N .

properties are defined, the second stage of the segmentation algorithm begins, in which some nodes are merged together. Anitini et al. [1] have defined some rules to carry out this process. These rules that should be applied in order are listed below.

- As long as there is a node, i , with only one neighbor, that single neighbor, called node j , should be examined. If the number of neighbors of node j is less than or equal to 2 and the sign of the average mean curvature of both node i and node j are the same, the two nodes are merged.
- As long as there is a node, i , with two neighbors, both of the neighbors should be examined. If for a neighbor, called node j , the number of neighbors is less than or equal to 2 and the sign of the average mean curvature of both node i and node j are the same, the two nodes are merged.
- As long as there is a node, i , with more than or equal to two neighbors, each of the neighbors are examined. If for a neighbor, called node j , the number of neighbors is greater than or equal to 2 and the sign of the average mean curvature of both node i and node j are the same, the two nodes are merged.

After the second phase of the algorithm is completed, a reasonable amount of mesh parts are obtained. This algorithm is summarized in Figure 3.6. As a final note, the segmentation task explained takes a reasonable amount of time especially when the mesh size grows. For this reason, the partitioning is done as a preprocessing step, and the results are stored as a file in our system so that when the same model is to be segmented with the same choice of interval number, the result can be loaded automatically from the file.

3.3 Animation Component

In this section, we primarily focus on how the meaningful parts of a model obtained as described in the previous section are used in animation production. Our main objective in this work is to provide users with the opportunity to create simple realistic animations by enhancing mesh editing techniques. As stated previously, users tend to animate salient parts of a model. For this reason, our system allows editing operations to be applied to the meaningful parts of the models. In addition, we provide both an editing and global transformation interface to enable users to translate and rotate a model while deforming. This results in more realistic animations. Finally, the use of a multi-touch screen has an evident effect in animation, since being able to deform different parts of a model simultaneously creates more expressive results. Even though all the stated features have a positive effect on building an animation tool, the mesh editing technique used has a special importance since it directly affects the results. Therefore, the core of the animation component is the mesh editing process. As discussed in the previous chapter, the method we have adopted for this purpose is based on differential representations that have gained a high popularity lately. The logic behind this choice is to be able to benefit from the advantages of differential methods. These methods are best for encoding shape features of a model and thus suitable for detail-preserving mesh deformations [31]. We particularly focus on Laplacian framework because conversion from absolute Cartesian coordinates to Laplacian coordinates is a straightforward operation, whereas the reconstruction of the Cartesian coordinates can be achieved by solving a sparse linear system

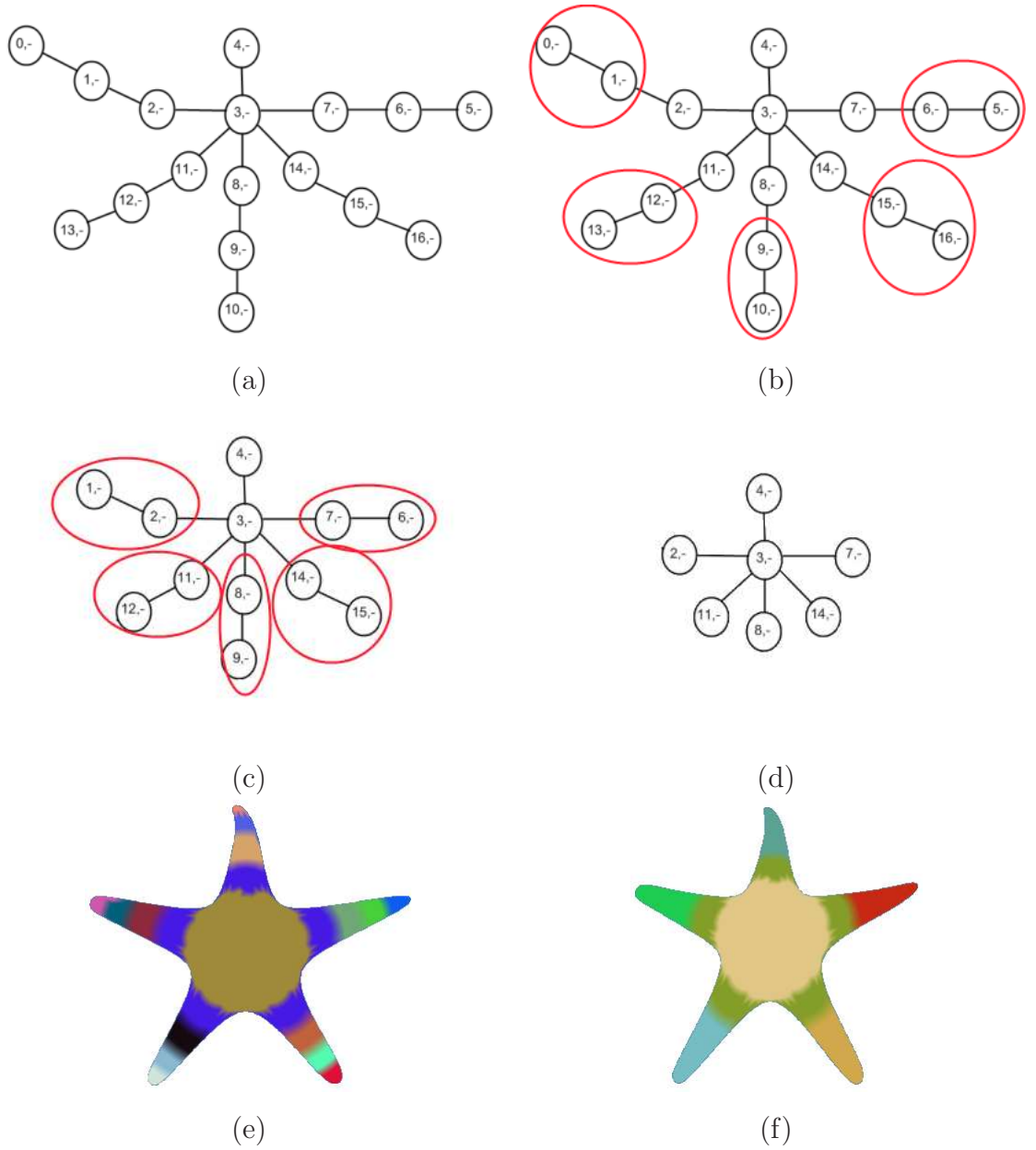


Figure 3.6: Mesh partitioning process illustrated for a sample model (a) Nodes obtained after pre-partitioning process, each circle contains the node number and the sign of the average mean curvature. (b) Nodes 0,5,16,10, and 13 are merged to nodes 1,6,9,12, and 15 as a result of Rule-1. (c) Nodes 1,6,9,12, and 15 are merged to nodes 2,7,8,11, and 14 as a result of Rule-1. (d) Nodes obtained after the merging process (e) Model partitioned without merging step (f) Model partitioned with merging step

[33]. The linearity of this approach makes it appealing for real-time use and the ability to preserve geometric properties in the reconstruction step is a necessity for many modeling operations. On the other hand, Laplacian surface editing does not focus on volume preservation, which is an equally important concept for realistic animations. Thus, our animation component also includes a volume preservation unit. The following sections detail both of these parts and underline the improvements accomplished.

3.3.1 Laplacian Framework

Laplacian framework is best suited for encoding the shape features of a model because each vertex of a mesh is considered within its neighborhood. This property of the framework enables reconstruction of Cartesian coordinates as outlined in Chapter 2, while preserving geometric details of the model. On the other hand, the main practical problem with Laplacian coordinates is their variance to scaling and rotation operations. This leads to the fact that reconstructed Cartesian coordinates once a mesh undergoes a linear transformation are erroneous. For this reason, some sort of a scheme should be developed to compensate for this deficiency. Different solutions have been proposed for this problem. In Lipman et al.'s work [19], explicit rotations are estimated by defining a local frame for each vertex. The system is solved again considering these estimations. The process is repeated until a smooth result is obtained. Yu et al.'s work [42] also uses explicit assignment of rotations, but the rotations of the handles are defined by the user and propagated to the other vertices proportional to geodesic distance. In contrast to explicit methods, Sorkine et al.'s work [33] presents an approach where the transformation of each vertex is computed implicitly. This approach has been applied to 2D mesh editing in the system presented by Igarashi et al. [14].

Yet, not all the example solutions given above are applicable in our case. To illustrate, the rotation estimation method discussed by Lipman et al. [19] requires many iterations to achieve smooth results for complex models [31]. Similarly, the approach of Yu et al. [42] requires users to define explicit rotations which can be

considered as a very challenging task for novices, the target group of our system. Regarding these facts, the solution based on implicit computation of rotations presented by Sorkine et al. [33] is more suitable for our work.

Recall that editing operations are applied to a mesh M with vertices v . The main idea of computing implicit rotations is to define a similarity transformation matrix, T_i , and to represent this matrix as a function of the unknown vertex coordinates. In other words, the least squares formulation to reconstruct Cartesian coordinates provided in the previous chapter becomes:

$$E(v') = \sum_{i=1}^n \|T_i(v')\delta_i - L(v'_i)\|^2 - \sum_{i \in C} \|v'_i - c_i\|^2 \quad (3.15)$$

Although both T_i and v' are unknown in the above function, when T_i is represented as a linear combination of v' there is only one unknown left. The logic for defining T_i also lies in considering each vertex within its neighborhood. The corresponding formulation is given in the work of Sorkine et al. [33] as follows:

$$E(v') = \min_{T_i} (\|T_i v_i - v'_i\|^2 + \sum_{j \in N_i} \|T_i v_j - v'_j\|^2) \quad (3.16)$$

Here each T_i transforms the 1-ring neighborhood of a vertex to its new location. Sorkine et al. [33] claim that each transformation should be constrained to allow free rotation and isotropic scaling in order to avoid shearing. True 3D rotations cannot be expressed linearly in 3D; a linear approximation can be developed, however. The approximation proposed by Sorkine et al. [33] is as follows:

$$T_i = \begin{pmatrix} s_i & -h_{i3} & h_{i2} \\ h_{i3} & s_i & -h_{i1} \\ -h_{i2} & h_{i1} & s_i \end{pmatrix} \quad (3.17)$$

As shown with this matrix, finding T_i means finding the unknown coefficients which can be represented by the vector $(s_i, h_i)^T$. The transformation $T_i v_i$ can also be represented as a multiplication of the below matrix with this vector.

$$P_i = \begin{pmatrix} v_x & 0 & v_z & -v_y \\ v_y & -v_z & 0 & v_x \\ v_z & v_y & -v_x & 0 \end{pmatrix} \quad (3.18)$$

$$T_i v_i = P_i(s_i, h_i)^T \quad (3.19)$$

The minimizer function for T_i given above can be rewritten as

$$E(v') = \min \|A_i(s_i, h_i)^T - b_i\|^2 \quad (3.20)$$

where the $3m \times 4$ matrix A_i (where m is the number of neighbors of v_i plus 1) is formed by writing the P matrices for the corresponding vertex and its neighbors one below the other. Similarly, b_i is the vector formed by writing the x,y, and z coordinates of v'_i and its neighbors. An example of these variables corresponding to the vertex v_0 in Figure 2.1 is given below:

$$A_0 = \begin{pmatrix} 4.0 & 0 & 1.0 & -5.0 \\ 5.0 & -1.0 & 0 & 4.0 \\ 1.0 & 5.0 & -4.0 & 0 \\ 6.0 & 0 & 1.0 & -6.0 \\ 6.0 & -1.0 & 0 & 6.0 \\ 1.0 & 6.0 & -6.0 & 0 \\ 4.0 & 0 & 1.0 & -3.0 \\ 3.0 & -1.0 & 0 & 4.0 \\ 1.0 & 3.0 & -4.0 & 0 \\ 5.0 & 0 & 1.0 & -4.0 \\ 4.0 & -1.0 & 0 & 5.0 \\ 1.0 & 4.0 & -5.0 & 0 \end{pmatrix}$$

The minimization problem for each T_i is solved in a least squares manner:

$$(s_i, h_i)^T = (A_i^T A_i)^{-1} A_i^T b_i \quad (3.21)$$

In the general least squares problem given in Equation 3.15, there is an expression of $T_i(v')\delta_i$. Just like defining $T_i v_i$ as the multiplication $P_i(s_i, h_i)^T$, this expression can also be redefined. Letting F_i be the matrix

$$F_i = \begin{pmatrix} \delta_x & 0 & \delta_z & -\delta_y \\ \delta_y & -\delta_z & 0 & \delta_x \\ \delta_z & \delta_y & -\delta_x & 0 \end{pmatrix} \quad (3.22)$$

which contains Laplacian coordinates, δ -coordinates, of v_i , $T_i(v')\delta_i$ can be rewritten as $F_i(s_i, h_i)^T$. Moreover, the expression for $(s_i, h_i)^T$ can be replaced with the equality given in Equation 3.21. (Since b_i contains the positions of v'_i and its neighbors, it is replaced with v' .)

$$F_i(A_i^T A_i)^{-1} A_i^T v' \quad (3.23)$$

The expression $F_i(A_i^T A_i)^{-1} A_i^T$ represents a $3 \times 3m$ matrix, where m is the number of neighbors of vertex v_i plus 1(for itself). The first three columns of this matrix corresponds to the vertex itself, while rest of the columns correspond to the neighboring vertices. This matrix is multiplied by b_i which contains the positions of v'_i and its neighbors. The positions are placed in the same order of the columns of the matrix. If the positions of all v'_i are written one below the other, a $3n \times 1$ vector is obtained. This vector can be multiplied with a $3n \times 3n$ matrix formed by writing the $3 \times 3m$ matrices one below the other. However, in order to obtain a matrix with $3n$ columns, for each vertex matrix, the columns corresponding to the vertices that are not a neighbor should be filled with a 0. This process is illustrated for the example mesh given in Figure 2.1. In the example given below, c_{ij} represents the 3×1 vector corresponding to the columns for the neighbor v_j in the $3 \times 3m$ constructed matrix for v_i .

$$F_0(A_0^T A_0)^{-1} A_0^T v' = \begin{pmatrix} c_{00}^x & c_{00}^y & c_{00}^z & c_{01}^x & c_{01}^y & c_{01}^z & c_{03}^x & c_{03}^y & c_{03}^z & c_{04}^x & c_{04}^y & c_{04}^z \\ v'_{0x} & v'_{0y} & v'_{0z} & v'_{1x} & v'_{1y} & v'_{1z} & v'_{3x} & v'_{3y} & v'_{3z} & v'_{4x} & v'_{4y} & v'_{4z} \end{pmatrix}' T$$

$\Downarrow$ overall equation

$$\begin{pmatrix} c_{00}^x & c_{00}^y & c_{00}^z & c_{01}^x & c_{01}^y & c_{01}^z & 0 & 0 & 0 & c_{03}^x & c_{03}^y & c_{03}^z & c_{04}^x & c_{04}^y & c_{04}^z \\ \vdots & & & & & & & & & & & & & & \end{pmatrix} * \begin{pmatrix} v'_{0x} & v'_{0y} & v'_{0z} & v'_{1x} & v'_{1y} & v'_{1z} & 0 & 0 & 0 & v'_{3x} & v'_{3y} & v'_{3z} & v'_{4x} & v'_{4y} & v'_{4z} \end{pmatrix}^T \quad (3.24)$$

The $3n \times 3n$ transformation coefficients matrix shown above can be represented as M . After substituting M into the general least squares equation, only one unknown, v' , is left.

$$\begin{aligned} E(v') &= \sum_{i=1}^n \|T_i(v')\delta_i - L(v')\|^2 - \sum_{i \in C} \|v'_i - c_i\|^2 \\ &= \sum_{i=1}^n \|Mv' - L(v')\|^2 - \sum_{i \in C} \|v'_i - c_i\|^2 \\ &= \sum_{i=1}^n \|(M - L)v'\|^2 - \sum_{i \in C} \|v'_i - c_i\|^2 \end{aligned} \quad (3.25)$$

As shown above, in order to compensate for the general linear transformations, the Laplacian matrix found in the least squares problem is replaced with $M - L$, called A . From this point on, the solution is just like the problem in the case that ignores linear transformations. The difference is that, instead of solving 3 ($n \times n$) matrix systems corresponding to x,y, and z coordinates, a single ($3n \times 3n$) system is solved.

$$\begin{aligned}
(\tilde{A}^T \tilde{A})v' &= \tilde{A}^T b \\
Mv' &= \tilde{A}^T b \\
(\tilde{R}^T \tilde{R})v' &= \tilde{A}^T b \\
\tilde{R}v' &= x \\
\tilde{R}^T x &= \tilde{A}^T b
\end{aligned} \tag{3.26}$$

Finally, Laplacian framework is ready to reconstruct Cartesian coordinates without a problem. Next, we outline how we use this framework in our mesh editing system.

3.3.1.1 Computation of Region of Interest

A typical editing animation scenario for our system is as follows. Once a 3D model is loaded, users activate the deformation process by switching the application to “Animation” mode and selecting handle vertices on the model by touching them. (The design of the interaction techniques are further discussed in related sections.) As explained already, our system perceives a 3D model as a combination of smaller parts which are defined as main deforming regions. Therefore, when a vertex is chosen as a handle, the part containing it is determined as the region of interest (ROI). All the vertices in this part except the handle are assumed to be unconstrained freely moving vertices. However, the transition between the ROI and the rest of the model should be smooth, meaning that there must be a set of boundary vertices that remain fixed and constrain the deformation process. In our system, this set is composed of vertices that are the first-order neighbors of the vertices in the ROI and not belonging to the same part of the mesh. This process is given in Algorithm 3.1.

Once the ROI with the handle and the boundary vertices are defined, the detailed solution steps explained previously are applied to the sub-mesh defined by these vertices. To begin with, Laplacian matrix corresponding to the sub-mesh is formed and normalized. Then, for each vertex in the sub-mesh, the $3 \times 3m$

Algorithm 3.1 Compute ROI for v_i

```

 $v_i$  = vertex selected as handle
 $n$  = node containing  $v$ 
 $f$  = freely moving vertices list
 $b$  = fixed boundary vertices list
 $N_{v_i}$  = neighbors of vertex  $v_i$ 
for all  $v_n \in n$  do
     $f \leftarrow f + v_n$ 
end for
for all  $v_n \in f$  do
    for all  $v_j \in N_{v_n}$  do
        if  $v_j \notin n$  then
             $b \leftarrow v_j$ 
        end if
    end for
end for

```

matrix as shown in Equation 3.24 is constructed. Later, these $3 \times 3m$ matrices are combined together to form the overall $3n \times 3n$ transformation coefficients matrix M (n is the number of vertices in the sub-mesh.) given in Equation 3.24. This matrix is substituted to the general least squares problem provided in Equation 3.25. Finally, the difference between the matrix M and the Laplacian matrix is computed and the result is multiplied with its transpose to form a positive definite, symmetric matrix which needs to be factorized. The matrix multiplication operation involves two $3n \times 3n$ matrices and is of $O(n^3)$. This computation time of this operation increases dramatically as the sub-mesh of interest grows. To overcome this obstacle, we use a linear algebra package called Lapack++ [25]. The multiplication routines provided by this library optimizes the matrix multiplication process and decreases the computation time to reasonable values. (For computation times of this operation, refer to the Results chapter.) Once the multiplication process is complete, the resulting matrix is factorized and this factorization is used to solve the least squares problem. The factorization step is the main computational core of this process and is computed with a sparse linear

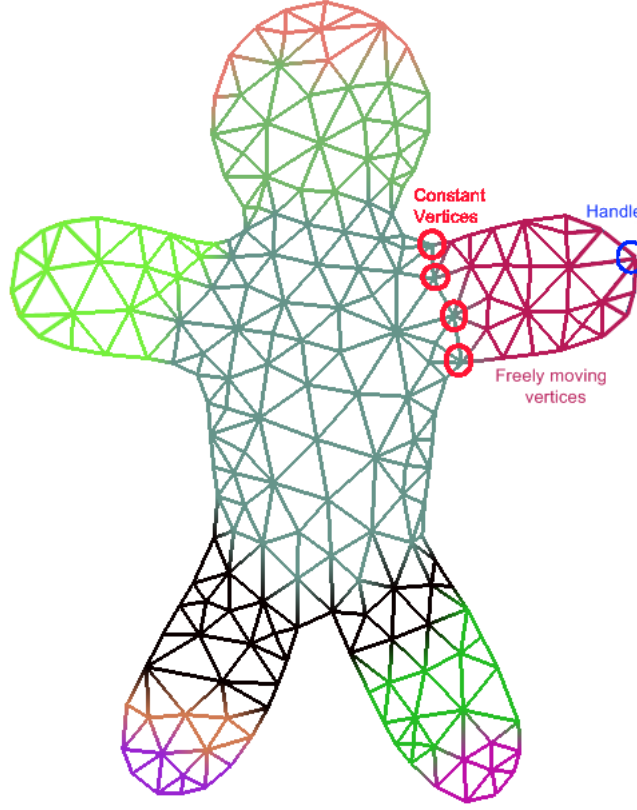


Figure 3.7: ROI Specification - Blue vertex is the handle selected by the user. Red vertices constitute the boundary and the in between part is the freely moving part.

solver library, Taucs [36]. The factorization routine provided by this library is multi-threaded which makes the code faster. After the factorization is computed and saved, the new positions of the vertices of the sub-mesh are calculated by substituting the updated positions of the handles to the right hand side of the Equation 3.26. Solving the equation by back-substitution is sufficiently fast and the manipulation actions are carried interactively. This whole process is summarized in the flow chart given in Figure 3.8.

Users have a direct control on the handle positions in our system. Once a handle is specified via a cursor (a traditional mouse or a finger), its position is constantly updated based on the cursor position. However, the cursor has only x and y coordinates, which correspond to its location on the screen, whereas the handle positions are in 3D. Therefore, a projection of the cursor coordinates to

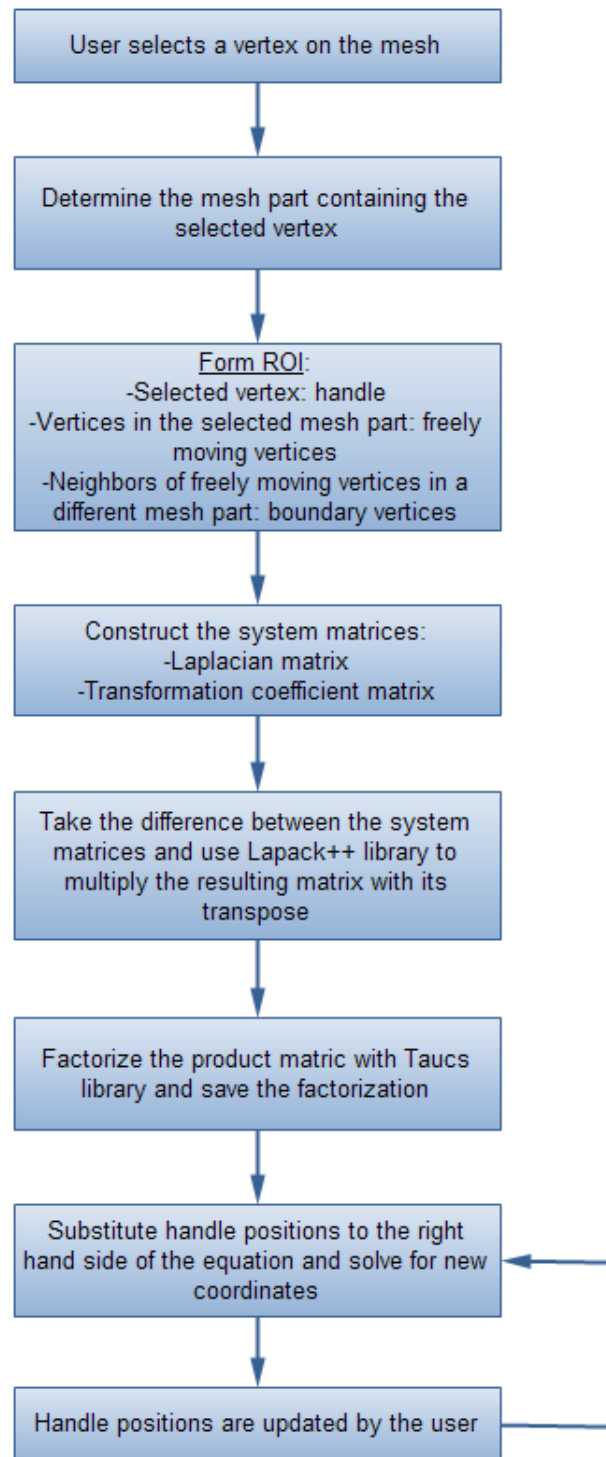


Figure 3.8: Flowchart explaining ROI determination and computation of new vertex positions.

3D is necessary. This projection is computed based on a simple assumption that cursor movements are limited to the z-plane on which the handle lies. In other words, handles move on a plane parallel to the screen. Other approaches are also possible to accomplish this mapping. As an illustration, handles can be moved along a vector field as described in Section 3.3.2.

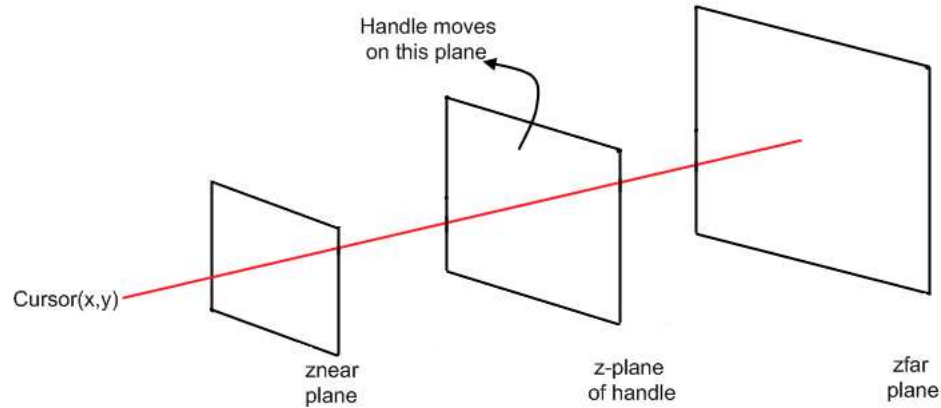


Figure 3.9: Cursor positions are projected to the z-plane on which the selected handle lies.

Users can specify different handles on different parts of the model since the system works on a multi-touch screen. If that is the case, the parts containing each of the handles are determined. If two parts are neighbors, they are combined to a single ROI. The boundary vertices are reconfigured to be the neighbors not belonging to any of the parts. Similarly, system matrices are updated according to the new ROI with the handles being the vertices chosen by the user. On the other hand, if the parts are not neighbors, then they are considered as two different ROI. For each ROI, its own least squares problem is defined and the systems are solved independently. These steps are summarized in the flow chart given in Figure 3.10. As a result of this explained approach, users manipulate a model as individual parts or as a combination of parts.

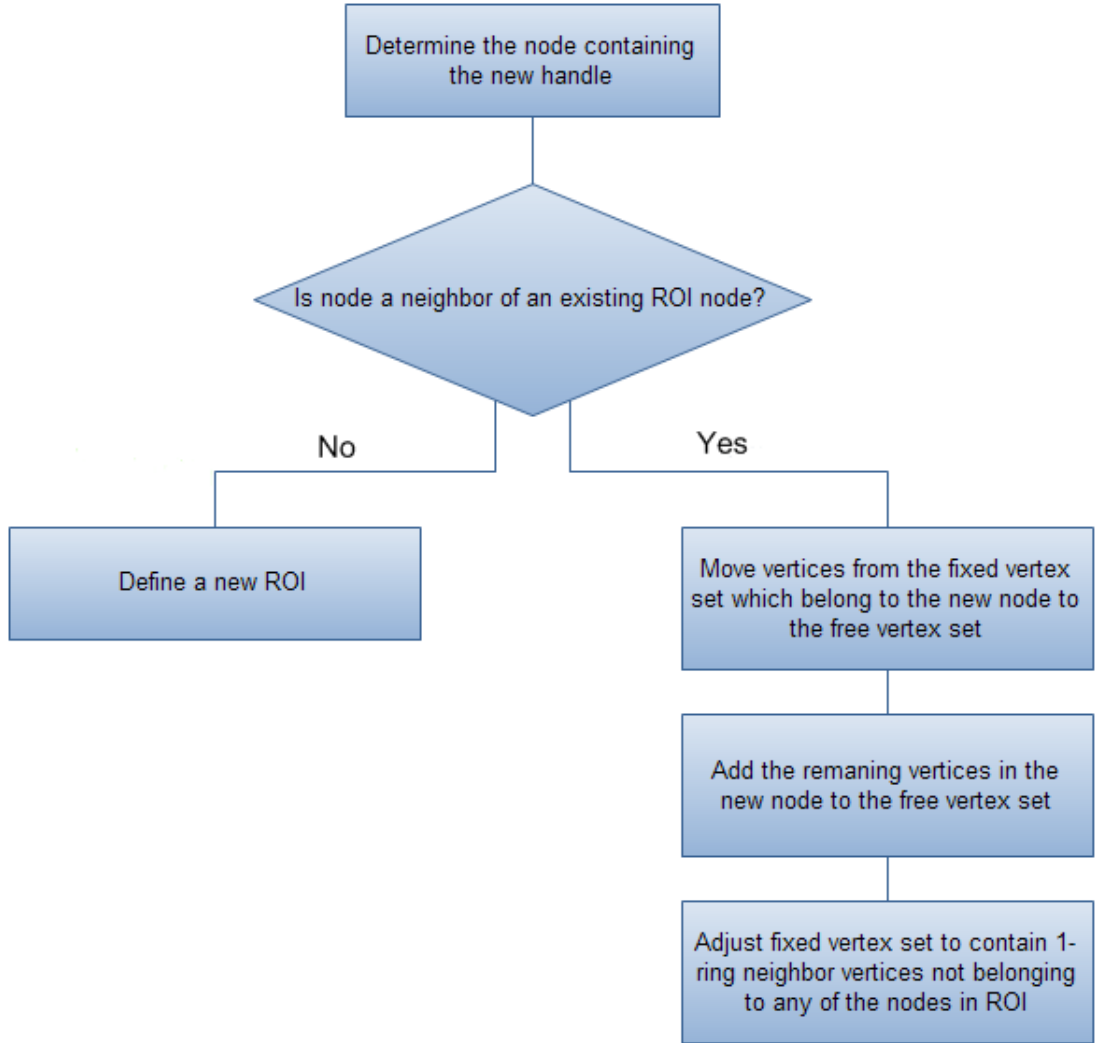


Figure 3.10: Flowchart explaining ROI determination with more than one handle.

3.3.2 Volume Preservation

Preservation of volume during manipulation and animation of objects is an important feature for obtaining plausible and realistic results. However, during manipulation sessions based on the Laplacian framework, loss of volume can be observed due to large deformations. To overcome this problem, we include a volume preservation component in our system. Users are free to enable or disable this component during animation production. This component is designed based on the approach presented by Funck et al. [38] and aims to correct the

coordinates of mesh vertices to preserve the volume of the model.

The logic behind the volume preservation operations is to define a displacement vector field for the mesh. This field defines in which direction and how strong each vertex should be moved after a manipulation action to preserve the volume of the model. This means that the volume correction operations are applied after each manipulation step.

The topology of a triangular mesh, M , with a vertex set of V can be represented by a triangulation set T , which contains a triple of vertex indices included in each face. The volume of the mesh can be approximated by the volume of the tetrahedra formed by each face and the origin [12]:

$$volume(M) = \frac{1}{6} \sum_{(i,j,k) \in T} v_i \cdot (v_j \times v_k) \quad (3.27)$$

In the above equation, $\cdot$ and $\times$ represent dot and cross products respectively. If we represent the displacement field of the mesh by F , the volume preservation step assumes that by adding this field to the manipulated mesh with an appropriate scale factor, λ , the volume can be kept constant:

$$volume(M' + \lambda \cdot F) = volume(M) \quad (3.28)$$

In the above equation, M' denotes the manipulated mesh coordinates. If the expression for the volume of a mesh is substituted into this equation, the scaling factor can be represented in terms of the original and manipulated mesh coordinates.

$$\begin{aligned} \frac{1}{6} \sum_{(i,j,k) \in T} (v'_i + \lambda f_i) \cdot ((v'_j + \lambda f_j) \times (v'_k + \lambda f_k)) &= \frac{1}{6} \sum_{(i,j,k) \in T} v_i \cdot (v_j \times v_k) \\ \sum_{(i,j,k) \in T} (v'_i + \lambda f_i) \cdot ((v'_j + \lambda f_j) \times (v'_k + \lambda f_k)) &- \sum_{(i,j,k) \in T} v_i \cdot (v_j \times v_k) = 0 \\ c_0 \lambda^3 + c_1 \lambda^2 + c_2 \lambda + c_3 &= 0 \end{aligned} \quad (3.29)$$

where

$$\begin{aligned}
c_0 &= \sum_{(i,j,k) \in T} f_i \cdot (f_j \times f_k) \\
c_1 &= \sum_{(i,j,k) \in T} v'_i \cdot (f_j \times f_k) + f_i \cdot (v'_j \times f_k) + f_i \cdot (f_j \times v'_k) \\
c_2 &= \sum_{(i,j,k) \in T} v'_i \cdot (v'_j \times f_k) + v'_i \cdot (f_j \times v'_k) + f_i \cdot (v'_j \times v'_k) \\
c_3 &= \sum_{(i,j,k) \in T} v'_i \cdot (v'_j \times v'_k) - v_i \cdot (v_j \times v_k)
\end{aligned} \tag{3.30}$$

In the above expressions, f_i denotes the component of the displacement field, F , applied to the vertex v_i . λ is simply the solution of the cubic equation given. A cubic equation has up to 3 real solutions. Since the manipulated mesh should be changed as little as possible, the solution with the smallest absolute value is chosen. Cardano's method is used to solve the cubic equation [21].

The key step of the volume preservation component is the definition of the displacement field. As manipulation actions are applied to the partitioned mesh parts in our application, preserving the volume in these parts results in a global volume preservation. Therefore, we define displacement fields separately for each mesh part. The fields should be defined so as to change the volume of the mesh. A suitable choice is to define displacement vectors pointing away from the mesh. In order to obtain such vectors, we first compute the axis aligned rectangular bounding box of each mesh part. For each vertex in a part, we find the closest point to the vertex on the closest face of the bounding box. The vector joining the vertex and the closest point constitutes the displacement vector. We can denote these vectors by d_i . A simple mesh part and its displacement vectors are shown in Figure 3.11.

Finally, we have to define how much a vertex should be affected by the defined displacement vectors. It is natural for the vertices close to the handle selected inside the mesh part to be affected more than the vertices close to the boundary of the part. For this reason, the maximum distance between the handle and any

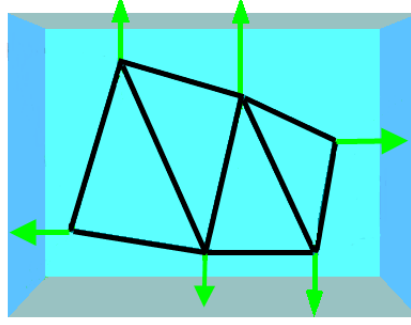


Figure 3.11: Vector field is defined from the vertices of a mesh part to the closest points on the bounding box.

other vertex inside the mesh part is computed. This distance corresponds to the distance between the handle and one of the boundary vertices. This vertex is given a weight of 0, whereas the handle is given a weight of 1. All the other vertices in between are assigned a proportional weight:

$$w_i = 1 - \frac{\text{distance}(i, \text{handle})}{\text{maximum distance}} \quad (3.31)$$

In the above equation, w_i is the weight assigned to vertex i . Once the displacement vectors and the weight of these vectors are computed, volume preservation component becomes active. After each Laplacian manipulation cycle, the transformations defined in Equation 3.17 are computed explicitly for each vertex. These transformations are applied to the sum of the mesh vertices and the displacement vectors. The difference between the two set of deformed positions are scaled with the corresponding weights. The result is the final displacement field.

$$f_i = w_i(T_i(v'_i + d_i) - T_i v_i) \quad (3.32)$$

By substituting the values for f_i into the Equation 3.29, the scaling factor, λ can be computed. The final step is to update the deformed vertex coordinates accordingly.

$$v_i'' = v_i' + \lambda f_i \quad (3.33)$$

v_i'' denotes the final corrected vertex positions. There is an important drawback of this approach. Correcting the deformed vertex positions to preserve volume changes the Laplacian coordinates of the vertices. If we keep applying the Laplacian framework to the corrected positions, we will end up losing the surface features of the model. For this reason, as long as the volume preservation component is enabled, displayed vertex coordinates are chosen as the corrected vertex positions. However, Laplacian operations are applied to the result of the operations obtained in the previous cycle of the Laplacian framework. In other words, Laplacian operations are independent from the volumetric calculations. Each volume correction cycle compares the volumes of the initial model and the model obtained at the last Laplacian cycle.

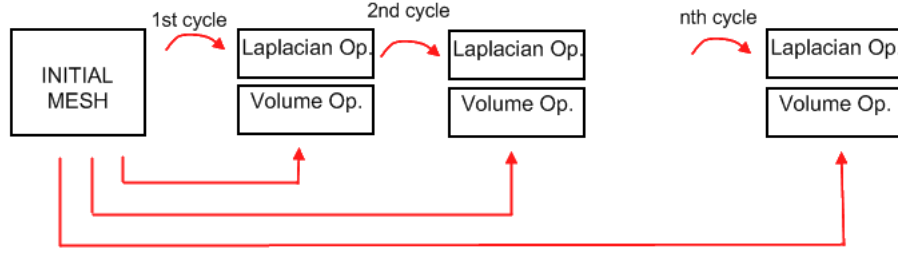


Figure 3.12: Laplacian framework uses the result of the previous Laplace operations, whereas the volume preservation component compares the final mesh with the initial mesh.

3.4 User Interface

As discussed in the previous chapter, designing an intuitive interface for applications that require user interaction is very important, especially for 3D applications. Consequently, the user interface component of our system focuses on techniques to enhance user interaction. In this section, we describe these techniques in detail. More specifically, we discuss the effects of using a multi-touch

screen in our application and the interaction methods developed to ease mesh animation operations.

3.4.1 Multi-touch Environment

A notable characteristic of the system presented in this thesis is its applicability to multi-touch screens. There are several reasons behind this design decision. To begin with, multi-touch environments have begun to gain high popularity in a variety of applications due to the potential advantages they present in user interaction. The multi-point feature of these environments increases the possible interactions that can be developed. This work aims to explore these advantages in a very common graphics application, 3D modeling operations. Clearly, the most significant benefit of using a multi-touch environment in our work is the ability to manipulate several regions of a model simultaneously. This feature has an unquestionable effect in animation creation.

The multi-touch screen used in our system is a product of Stantum Technologies [15]. This is a display device capable of detecting more than one contact point at a time. Each contact point is represented as a “cursor” and given a unique id. Event messages belonging to the current cursors are constantly generated. Once a user touches the screen, a new cursor with a new id is created and an appropriate message is generated. From then on, messages about any position change or click events are received with the corresponding id. Although this device is not designed to give pressure feedback, it is able to distinguish between single and multi clicks. This is the main feature used to group user actions in our system.

As described already, our system accepts two kinds of user actions. The first kind is related to local editing of a model, while the second kind corresponds to global transformations. The single clicks on a model are used to define local editing actions in the animation mode of the system. Handle vertices are specified by directly touching the desired regions of a model. The number of handles can be as many as the points of contact on the screen. After a selection is performed, the handles follow the movements of the corresponding cursor, in other words,

the finger of the user. Meanwhile, the user may want to globally transform the same model. If that is the case, the transformation widget should be activated by double clicking on any region of the model. The details of distinguishing between single and double clicks are provided in the next section.

In summary, multi-point property of the multi-touch screen enables the users to manipulate many regions of the 3D model and perform global transformations at the same time. Thus creating simple animations become easier. For example, in our system a dog model can be animated easily as if it is barking while walking.

3.4.2 Interaction Techniques

Defining 3D manipulations like rotation or translation is not a straightforward process with 2D input. For this purpose, several interactions techniques or gestures are used to enhance user interfaces. These techniques become even more valuable if the target users are novices, as in our case. The nature of our application and the multi-touch screen shape the development of the most suitable interaction techniques.

To begin with, the most basic interaction technique in our application is based on single and double clicks, which are used to define operation modes of the system. Obviously, with a traditional mouse no additional work is needed, since these events are automatically distinguished. However, with the multi-touch screen, some basic steps should be taken. As explained in the previous subsection, the multi-touch screen represents points of contact as cursors and assigns them a unique id. The messages generated for these cursors are down, move, and up messages, which correspond to the actions of touching the screen with a finger, moving the finger over the screen, and lifting it up respectively. When a new contact point is detected on the multi-touch screen, a certain amount of time delay begins. If a new down event is raised on the same point (or in a small region about the point) before a move message during this delay, instead of assigning a new id to the cursor, the cursor with the previous id is said to be double clicked. If no other down message is generated during the delay, the cursor

is single clicked. In conclusion, in our application, the number of down messages for a cursor are adjusted to distinguish between single and double clicks. This simple process is illustrated in Figure 3.13.

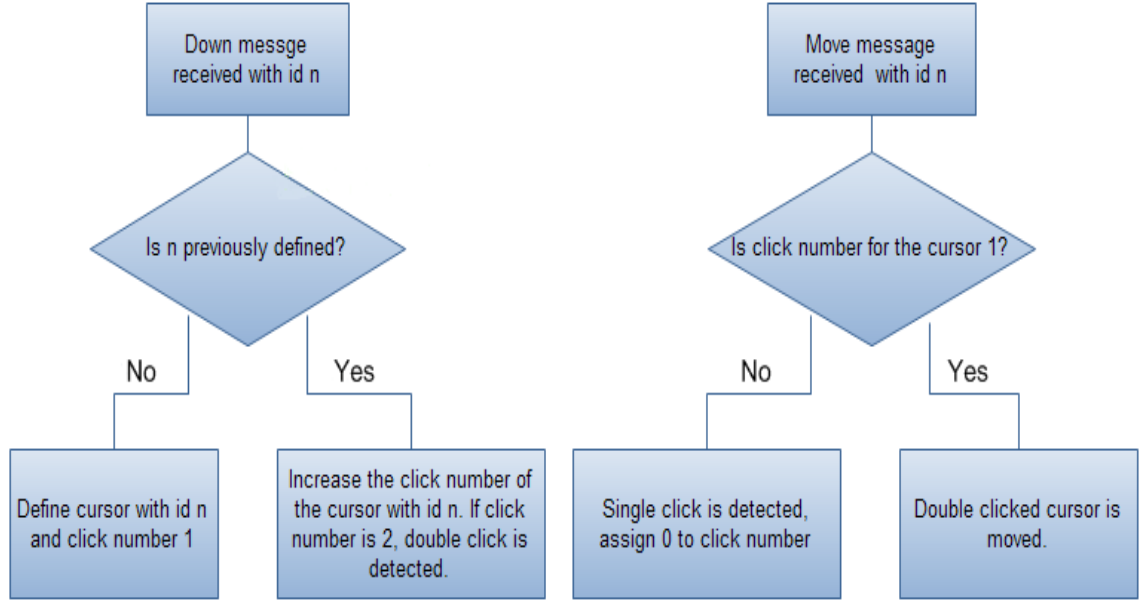


Figure 3.13: Flowchart of the distinguish operation between single and double clicks.

As stated, double clicks are used to activate the global transformation widget. Several facts constrain the design of this widget. First of all, in animation creation, rotation and translation are the most common operations and they should be defined via a single widget to simplify the interface. In addition, since global transformations are carried out simultaneously with manipulation operations, most probably only a single hand or even a single finger will be used to interact with this widget. Considering these facts, we have designed a widget, which is an extension of Arcball [30]. Arcball is a technique to define 3D rotations with 2D input. We have improved this traditional widget to specify translation data as well. This widget is enabled and disabled with double clicks. Once activated, cursor down and move messages with a position inside the widget area are mapped to translation and rotation actions. The widget is deactivated only when another double click is recognized.

Arcball technique assumes that an object is enclosed in a sphere and the user rotates this object by moving the input device on the circle, which is the projection of the sphere on the screen. In other words, user interactions are constrained within a circle. In our design, we define an additional outer circle around this projection circle to produce the widget shown in Figure 3.14.

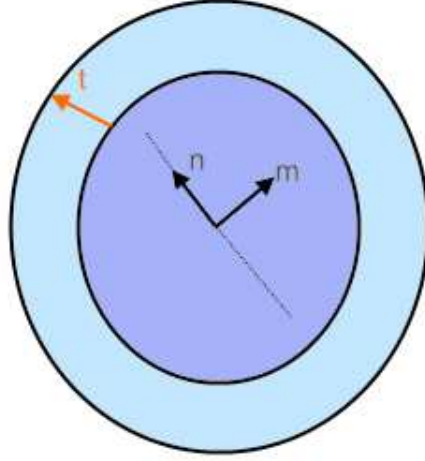


Figure 3.14: An improved Arcball widget.

The inner circle in Figure 3.14 is the original projection of the sphere and the cursor movements within this circle are mapped to 3D rotations. The logic behind this is to interpret cursor points as an arc and compute the rotation from this arc using quaternion representation. A quaternion $q(\mathbf{v}, w)$ is a four coordinate system, including a scalar part equal to $\cos(\theta/2)$ and a vector part equal to $\sin(\theta/2)$ times a unit vector in the rotation axis where θ is the angle of rotation [30]. Shoemake shows that rotation represented by the arc between two points P_0 and P_i can be computed as the product of the final point and the conjugate of the first point. This product produces a quaternion which represents the new orientation of the object enclosed in the sphere [29].

$$q(\mathbf{v}, w) = P_i P_0^* = (P_0 \times P_i, P_0 \cdot P_i) \quad (3.34)$$

In the above equation, $\times$ and $\cdot$ denote cross and dot products respectively. This equation follows from the quaternion multiplication. Each vector can be

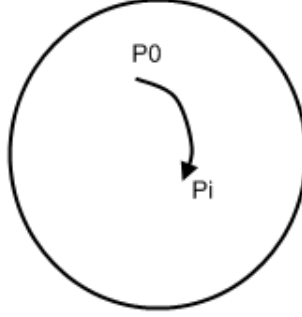


Figure 3.15: An example arc between two cursor points, P_0 being the cursor down point and P_i being the cursor up point.

considered as a quaternion with a scalar part of 0. Therefore, the cursor points are denoted by quaternions with a 0 scalar part and a quaternion multiplication is computed. This equation states that the arc between any two points can be represented with a quaternion. The new orientation is the product of the quaternion computed when cursor motion just started with the quaternion obtained in the i^{th} cycle [30]. Once the final quaternion is derived, the corresponding rotation matrix can be easily formed, as detailed in the work of Shoemake [29]. After the resulting quaternion $q((v_x, v_y, v_z), w)$ is normalized to satisfy the equality $v_x^2 + v_y^2 + v_z^2 + w^2 = 1$, the corresponding rotation matrix is constructed as follows:

$$M = \begin{pmatrix} 1 - 2v_y^2 - 2v_z^2 & 2v_xv_y + 2wv_z & 2v_xv_z - 2wv_y \\ 2v_xv_y - 2wv_z & 1 - 2v_x^2 - 2v_z^2 & 2v_yv_z + 2wv_x \\ 2v_xv_z + 2wv_y & 2v_yv_z - 2wv_x & 1 - 2v_x^2 - 2v_y^2 \end{pmatrix}$$

In our application, the interactions of the user with the transformation widget is limited to the region denoted by the outer circle. If the cursor movements are inside the inner circle, above computations are carried to calculate rotations. Once the cursor moves to the region between the inner and outer circles, a translation component is added to the final orientation of the object. This translation is denoted by the vector between the final cursor point and the last point on the inner circle the cursor has passed when moving to the outer region. On the

other hand, if the first contact with the widget is in the outer region, then the translation vector is computed by defining a vector from the center of the widget to the current cursor point. The intersection of this vector with the boundary of the inner circle constitutes the start of the translation vector.

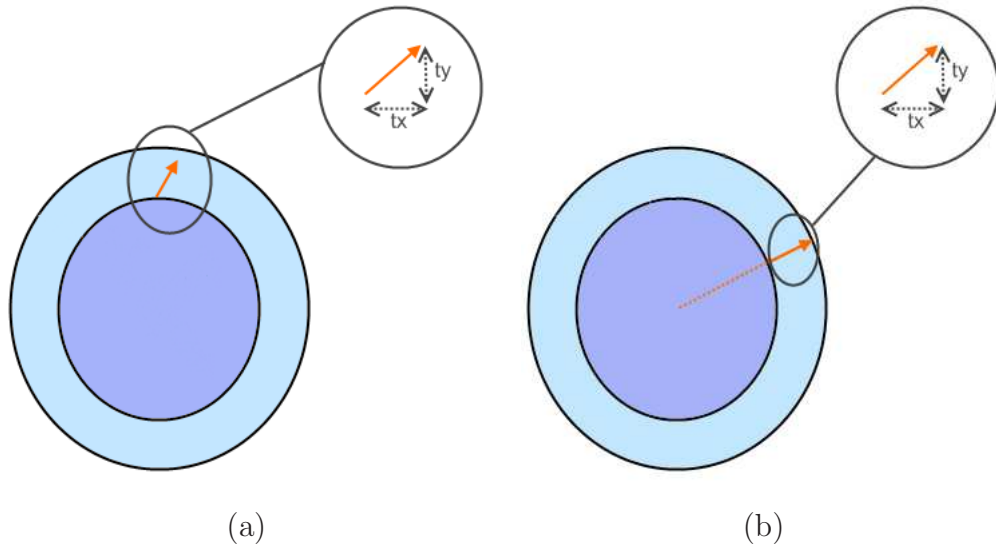


Figure 3.16: Translation vector obtained from cursor points. (a) Cursor moves from the inner region to outer region. (b) First contact with the widget is in the outer region.

As long as the cursor hovers inside the region between the inner and outer circles, the translation denoted by the most recent translation vector is added to the current translation component. This way, when the cursor moves back to the inner circle, the location of the object remains constant.

Finally, there is a thin non-responsive region between the two circles of the transformation widget. In other words, when the cursor is moved inside this region, no change is applied to either of the transformation components. The main advantage of this non-responsive region is to avoid undesired location and orientation changes due to noise data. Using a multi-touch screen for a long time may create the risk of hand fatigue. This risk becomes vital for precise interactions such as the transition between the inner and outer circles of the widget. The cursor may be moved between the circles unintentionally producing noise data. The non-responsive region discards the cursor movements close to

the boundary of the circles and reduces the effect of this noise data. In addition, this region is useful in changing the translation direction without affecting the location and orientation of the object. Users may want to move an object in an opposite direction after a certain transformation is reached without lifting the cursor. If that is the case, the cursor is moved to the desired translation region to reverse the translation vector. If this transition is done through the non-responsive region by moving the cursor close to the boundary between the widget circles, undesired transformation changes are avoided.

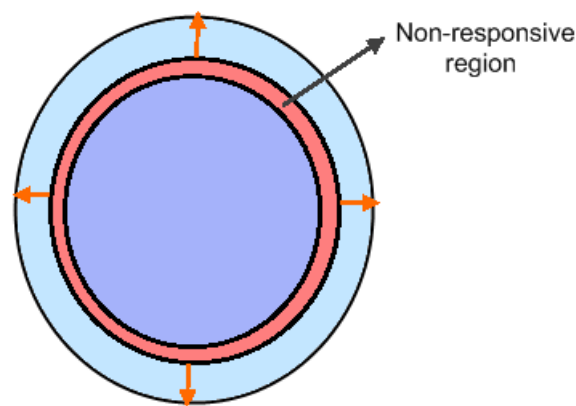


Figure 3.17: Non-responsive region in the widget and different translation vectors are shown.

3.4.3 Direct Manipulation Principle

All the interactions with our system obey the direct manipulation principles. Shneiderman explains the properties that a direct manipulation interface should have as the following [28]:

- Object of interest should be continuously represented.
- Physical actions and button presses are used instead of complex syntax.
- Operations are rapid and reversible actions, which are immediately visible.

Our system can be examined with respect to each of these items. As appropriate to the first property, in our application the object of interest remains on the screen during all operations performed and changes shape in response to user actions. The commands are specified by physical actions and simple button presses such as directly touching the model and moving fingers on the model. Moving fingers over the transformation widget also creates a direct interaction. Lastly, the object of interest immediately responds to these actions by changing shape, location, and orientation. These actions are also reversible. To illustrate, manipulations can be reversed by bringing the corresponding handles to their initial positions. In addition, when the handles are released, the object is interpolated to its initial shape.

The reason to base the interactions techniques used in our application on the principle of direct manipulation is to benefit from the advantages of it. The techniques proposed increase the usability of the interface, which is especially important since target users of our system are novices. The direct interaction methods give the feeling of manipulation objects by hand. This makes our system a more realistic animation tool.

Chapter 4

Results and Discussion

This chapter presents experimental results for the different components of our system in separate sections.

4.1 Mesh Partitioning

We begin with the mesh partitioning component. As stated in the previous chapter, the partitioning process can be time consuming especially for large meshes. For this reason, partitioning results for a particular mesh are stored in a file at the preprocessing stage. Table 4.1 includes partitioning times for different size of meshes. To provide a better comparison, partitioning process is repeated for different choices of interval number (See Section 3.2.2.1) and the average computation time is given in seconds.

As shown in Table 4.1, even for a medium size mesh of about 1000 vertices, the process takes about 2 minutes. The main computational core of this process is the calculation of the geodesic distances between every two vertices of the mesh. Geodesic distance computation is done based on Dijkstra's shortest path algorithm. The running time of this algorithm for a graph with n nodes, which finds the shortest distance between a source node and every other node, is $O(n^2)$

Table 4.1: Partitioning time for different size of meshes

Model	No of vertices	k=5 (sec)	k=6 (sec)	k=7 (sec)	k=8 (sec)	k=9 (sec)	avg (sec)
man	214	1.319	1.355	1.344	1.300	1.312	1.326
fish	478	14.613	14.047	13.973	13.974	14.102	14.141
dog	1030	146.625	153.307	148.018	146.956	146.678	148.316
starfish	1890	878.143	892.741	868.929	895.875	914.253	889.988

[39]. Since this algorithm is run for each vertex of a mesh with n vertices, the total running time in our case becomes $O(n^3)$. This is the reason for the dramatic increases in the computation times. The numbers provided in the above table show that storing segmentation results in a file for reuse is a very reasonable design decision.

Our application provides a slider to change the interval number used in the partitioning process. This number is used to group nodes during the process (See Section 3.2.2.1) and affects the overall number of parts obtained. Antini et al. comment that 7 is a reasonable choice for this number according to experimental results [1]. Therefore, the options provided in our application (5,6,7,8, and 9) are chosen around 7. Users are free to observe different partitioning results and use the desired one. To illustrate the effect of the interval number, Figures 4.1 and 4.2 provide the partitioning results for different models with different choices of this number.

Recall that mesh segmentation process works as a two-step algorithm. The first step groups vertices of a mesh based on their geodesic distance values. The second step defines the final model parts by examining curvature properties of each group and joining the adjacent groups with similar curvature properties. Models with smooth surfaces do not include rapid curvature changes. As a result, a larger number of vertex groups are joined producing a reasonable number of final mesh parts. The examples given in Figures 4.1 and 4.2 illustrate this fact. On the other hand, models with sharp features or many details contain sudden curvature changes, which constrain the merge phase of the segmentation process.

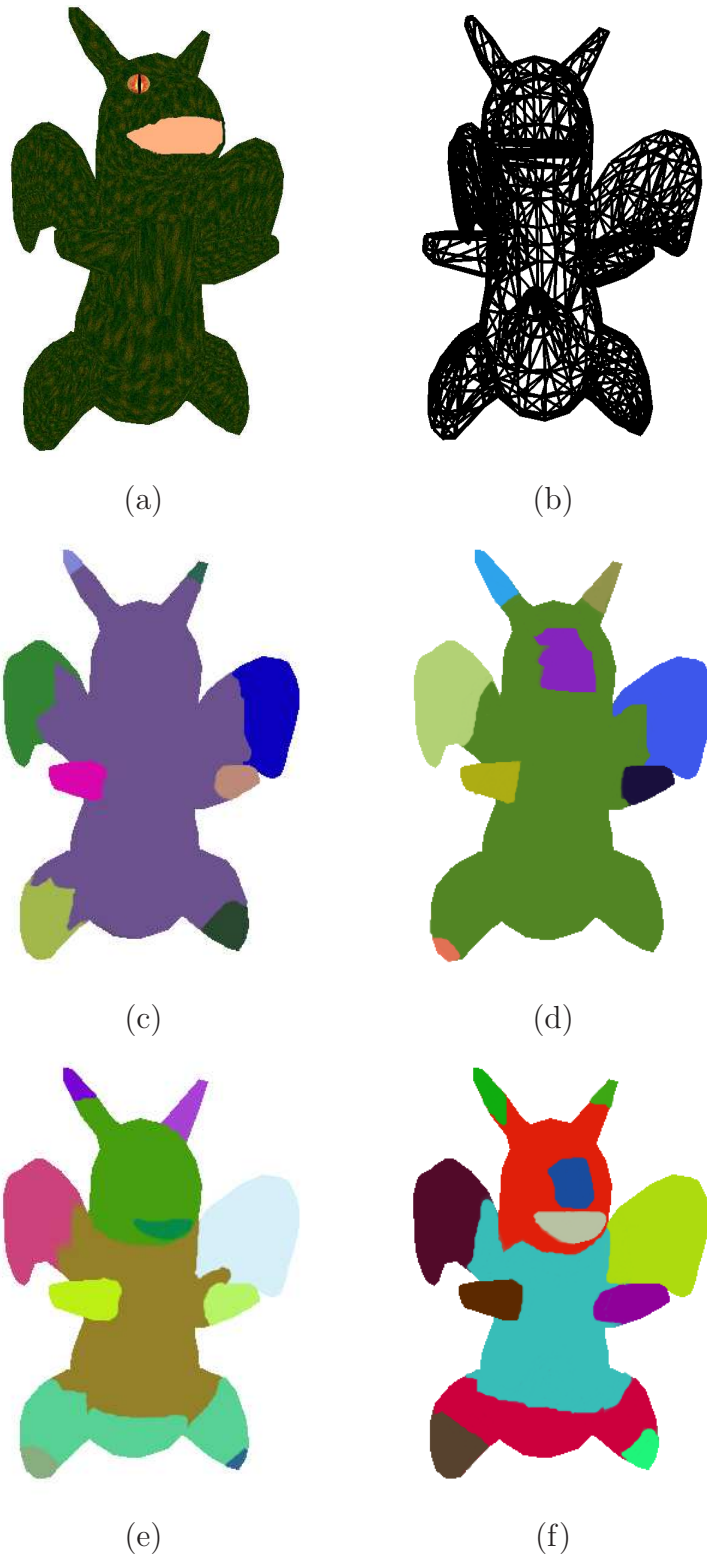


Figure 4.1: Mesh partitioning results for a dragon model ((a) and (b)) with different choices of the interval number, k (c) $k=5$ (d) $k=6$ (e) $k=7$ (f) $k=8$

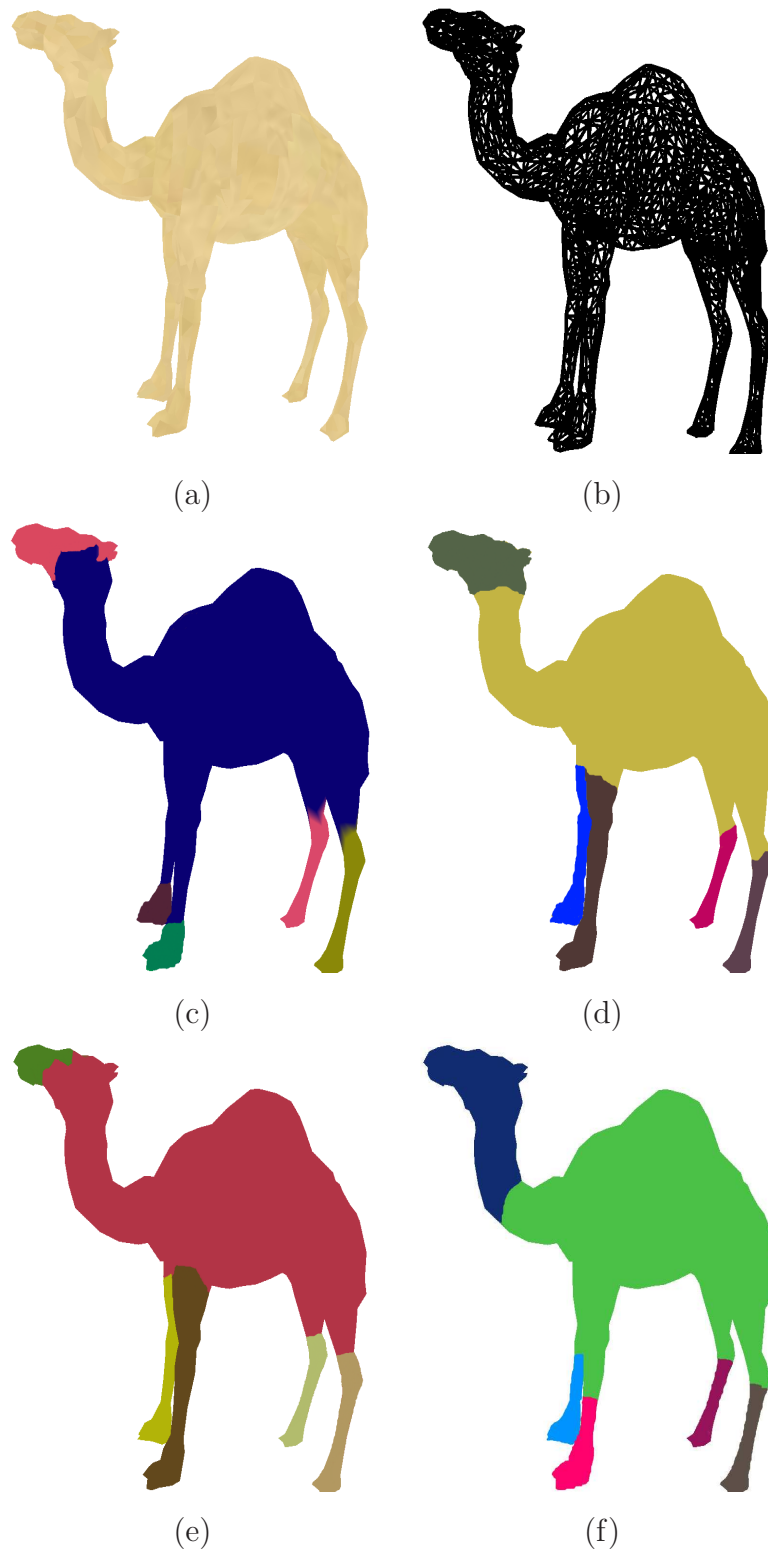


Figure 4.2: Mesh partitioning results for a camel model ((a) and (b)) with different choices of the interval number, k (c) $k=6$ (d) $k=7$ (e) $k=8$ (f) $k=9$

Consequently, a large number of mesh parts are obtained. An example of such a model is provided in Figure 4.3.

Animation actions are applied to model parts in our system and working with many small parts complicates these actions. The human visual system perceives a model as a combination of its main features at first glance, which makes it easier to animate a model partitioned into its main parts. As a result, we can conclude that, our system is more suitable for smoother models, such as cartoon style characters.



Figure 4.3: A plane model with many details produces a large number of mesh parts.

4.2 Laplacian Framework

As explained in the previous chapters, Laplacian model editing operations constitute an essential part of our system. These operations are used in a real-time

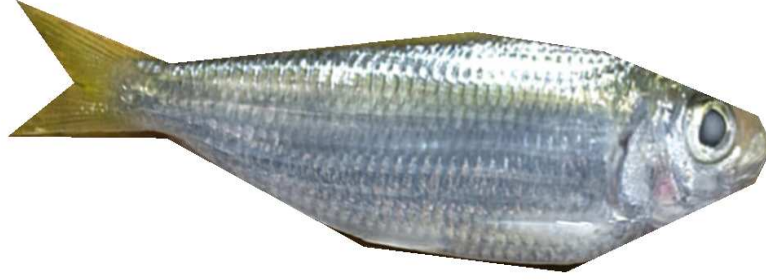
interactive system, which makes the computation time an important measure for evaluation. The main computational cores of the Laplacian framework are the construction of the system matrix via matrix multiplication operations, and the factorization of the product. Fortunately, these operations are carried only once per ROI definition. Updating the right hand side of the general linear equation given in Equation 3.26 with new handle positions and solving the system by back substitution is sufficient for the remaining editing session. Table 4.2 demonstrates the considerable difference between the running times of the multiplication, factorization, and back substitution processes.

Table 4.2: Computation times for matrix multiplication, factorization, and back substitution processes of Laplacian framework, given in seconds.

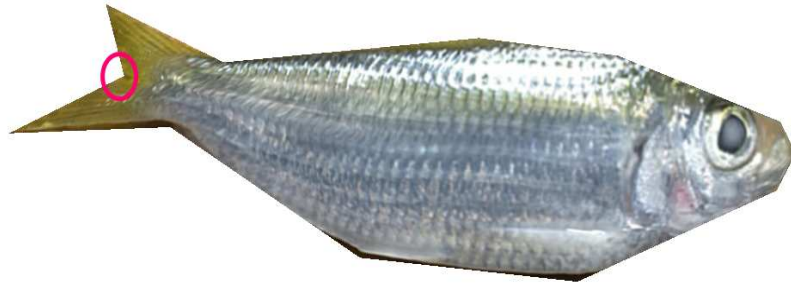
No of vertices in ROI	Matrix Multiplication (sec)	Factorization (sec)	Back Substitution (sec)
100	0.0312	0.0156	0.0020
149	0.0936	0.0312	0.0020
240	0.3740	0.0624	0.0060
576	2.9550	0.0936	0.0312
850	15.600	0.1404	0.0312
969	20.813	0.1716	0.0468

As the table shows, computation times for the matrix multiplication process increase in considerable amounts as the size of the ROI increases. On the other hand, in our system, mesh segmentation component partitions a model into smaller parts, which constitute the main ROI blocks. Therefore, in most of the cases, size of the ROI is limited to the size of one or two model parts. In other words, computation complexity of the multiplication operation does not constitute a serious problem. Moreover, as explained above, this operation is carried only once per ROI specification, having no major effect on subsequent the interactivity of the system. Furthermore, automatic generation of the ROI from the model parts discards the need for explicit definition of the ROI and the boundary vertices. For this reason, our system presents an easier interface compared to the traditional Laplacian editing systems. Below we provide some models which have been manipulated by the Laplacian framework. These examples include

cases where only a single handle, multiple handles in different model parts, and multiple handles in the same model part are selected.



(a)



(b)

Figure 4.4: Tail of the fish is manipulated with a single handle shown by the circle. (a) Original model (b) Manipulated model

4.3 Volume Preservation

As described earlier, manipulation of meshes with the Laplacian framework may result in loss of volume. The volume preservation component of our system, tries to update mesh vertices in order to preserve the initial volume. During this process, vertices are assigned automatic weights so that nearby regions of a handle are affected more by the volume correction operations compared to farther regions. Figure 4.7 shows example models manipulated by the Laplacian

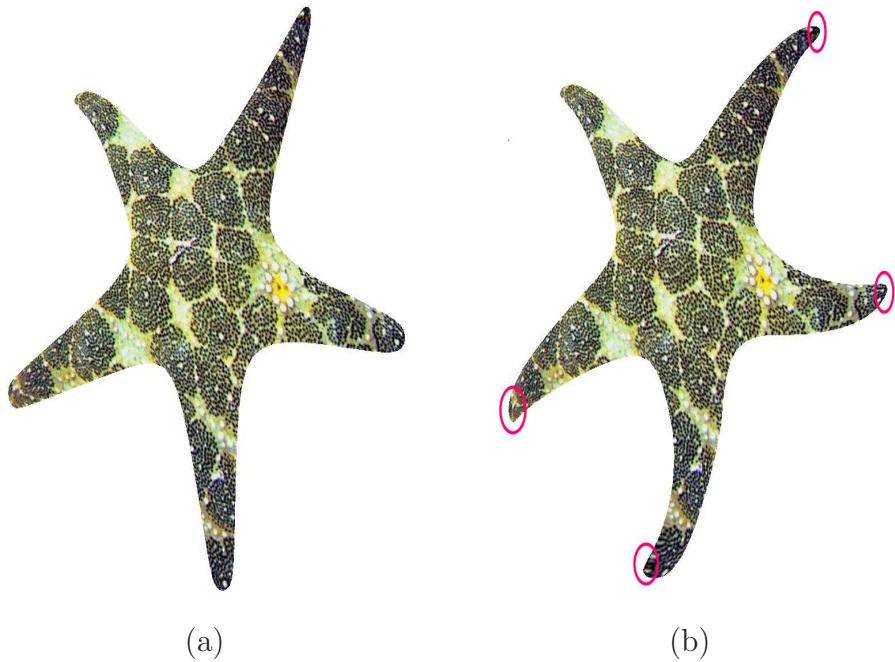


Figure 4.5: Arms of the starfish are manipulated with appropriate handles shown by the circles. (a) Original model (b) Manipulated model

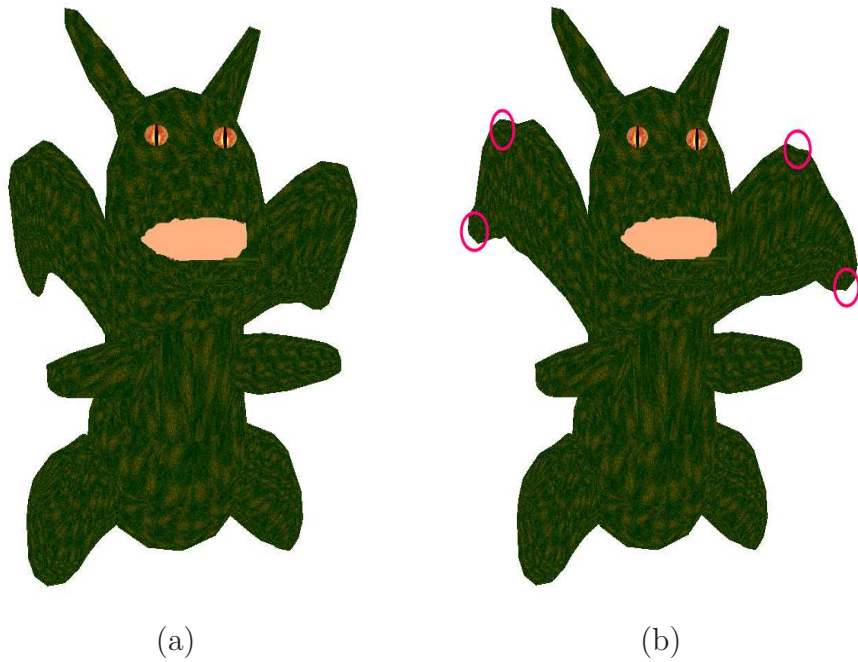


Figure 4.6: Each wing of the dragon is manipulated with double handles shown by the circles. (a) Original model (b) Manipulated model

framework with and without applying volume preservation.

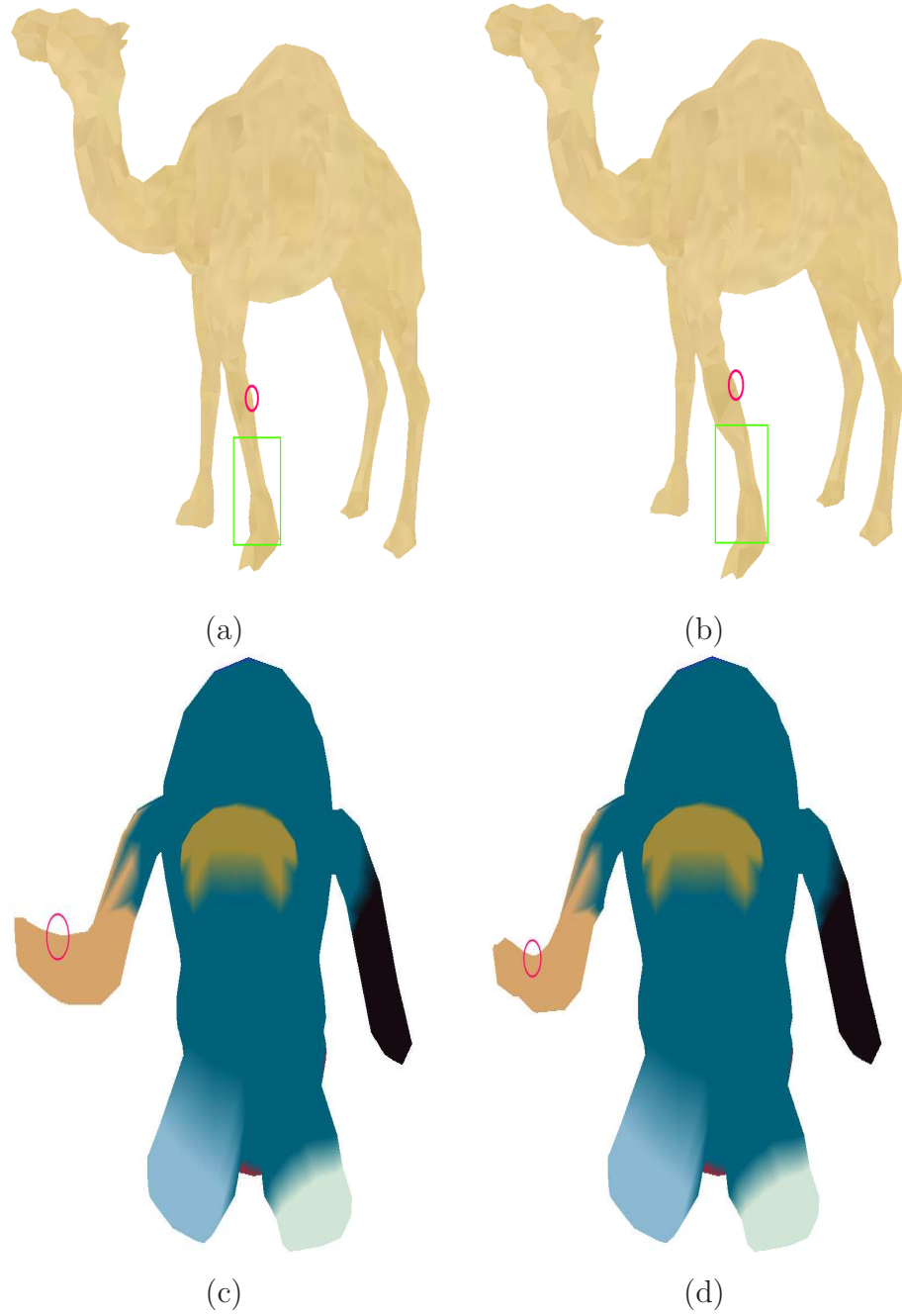


Figure 4.7: Models manipulated by disabling ((a) and (c)) and enabling ((b) and (d)) volume preservation component. Red circles denote the handle positions.

The automatic weighting scheme of the volume preservation component is sufficient to produce pleasing results. It also provides an easy interface for novice

users. However, this component can also be further extended to present an interface to define manual weights. Explicit weighting can be beneficial in creating complex and user-defined manipulations.

4.4 Usability Evaluation

In this section we analyze the usability of our system based on the subjective reactions of the users. Before discussing these reactions, we have to note that our system is based on a simple interface with only a few commands. Figure 4.8 provides an example snapshot. We have explored this interface and the general features of our system by defining several different tasks. After completing these tasks, users were asked to comment on the usability of our system and express their impressions. The designed tasks are classified into three groups, which focus on different components of our system.

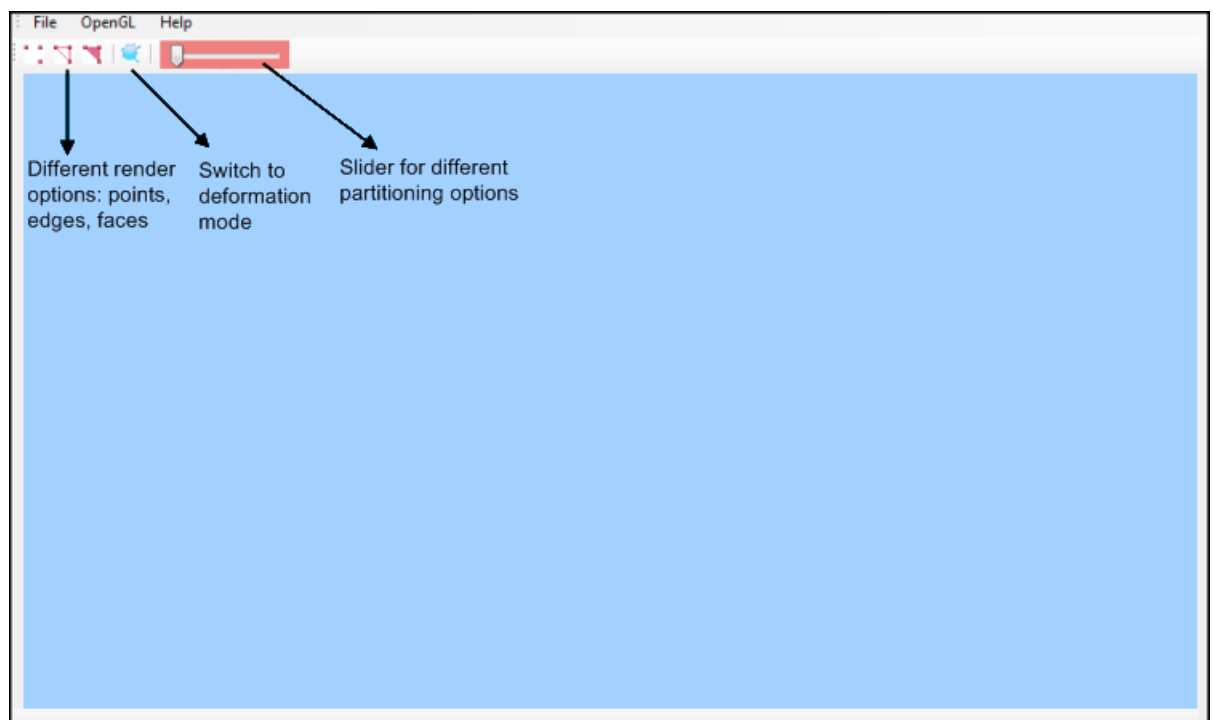


Figure 4.8: User interface of our system

The first group of tasks concentrate on global transformations of models by

examining the usability of the transformation widget. Users were required translate and rotate models to given positions and orientations. They were then expected to evaluate the capability of the widget in creating desired rotations and translations, and the transition between these two kinds of transformations. The general user feedback is that the individual usage of the transformation widget is not complicated. If two kinds of transformations are examined separately, we can say that applying translations is found to be easier than applying rotations. The widget displays a translation vector showing the direction of movement whenever the cursor is inside the translation region. The length of this vector varies proportional to the translation speed. This vector is found very useful in translating a model in a desired direction with a desired speed. On the other hand, users need a short practice time to understand the working principle of the rotation component. After producing rotations about different rotation axes with various kinds of arcs, users feel more comfortable in predicting the outcome of their actions. Thus, desired orientations can be achieved in a shorter amount of time. Finally, no serious problem has been reported about the transition between the two regions of the widget. Switching between the two modes of transformations is done with ease.

The second group of tasks investigate the mesh partitioning and the Laplacian framework components of our system by asking the users to manipulate different regions of a model. During these manipulations, users were especially requested to concentrate on the automatic generation of the manipulation regions and the consequences of using both hands at the same time. To begin with, in most of the cases, automatically chosen manipulation regions are consistent with the expectations of the users. In case these regions are smaller or larger than the desired manipulation regions, users try another partitioning result until the most suitable one is chosen. Therefore, we can conclude that the slider which presents different partitioning results proves to be beneficial. As for the other concern of this group of tasks, we can state that the users are comfortable in using both hands during manipulation. However, we have to note that, they prefer to use their thumbs and index fingers in each hand to manipulate more than a single region at the same time. In general, other fingers are used only in situations where

handles close to each other are specified. The reason behind this preference is that it is easier to synchronize the thumb and the index finger than any other pair of fingers.

Finally, the last group of tasks ask the users to create simple animations by applying global transformation and manipulation actions simultaneously. The aim of this task is to have an overall evaluation of the system. The most important observation of these tasks is that, users do not prefer to rotate and manipulate a model at the same time. The reason is obvious, in fact. When a model is rotated, the region, which contains the handle, is also relocated with the handle. As a result, unintended manipulations occur. Therefore, users first rotate the model to the desired orientation and then apply manipulation actions. Fortunately, this is sufficient in most cases because no strong insistence in rotating and manipulating a model together has been observed.

To better evaluate the style of translating a model during manipulation, users are presented with two different options. In the first option, the transformation widget stays constant while the model is being translated. Consequently, the user holds her finger constant to keep on the translation. On the other hand, the second option translates the widget with the model. Therefore, a user must move her finger with the widget to keep on the current translation. Both of these options have their own advantages and disadvantages. With the first option, users can hold their finger in one hand still while using their other hand freely. No synchronization is needed between the hands. However, if the users try to give translation and manipulation commands with the same hand, the distance between the fingers specifying each of the commands will increase as one finger is kept still, while the other one moves with the model. When this distance reaches a certain value, users will no longer be able to move their fingers. The second option eliminates the distance problem, since the finger specifying the translation command must be moved as well. Nevertheless, this solution brings up the necessity to synchronize fingers used in different operations. Moving a handle to desired locations with one finger and keeping the translation vector constant with the other finger creates a handicap for many users. The synchronization is established only after a certain amount of practice. In conclusion, it is found

easier to apply global transformation and manipulation commands with separate hands for both options. Using separate hands eliminates both the distance and the synchronization problems. However, no option is strongly preferred over the other one.

On the whole, some general comments can be noted about our system. First of all, users have a direct control over the selection and relocation of the handles during manipulation actions. This feature makes our system easier to use compared to other handle specification methods, such as silhouette sketching [23] and stroke drawing [7]. Moreover, the results of the actions are immediately displayed on the screen, eliminating the need of predicting the output prior to giving commands. Finally, users feel more comfortable using our system as they gain practice and are able to produce more pleasing animations.

Before concluding the section, we provide snapshots of simple animations created by our system in Figures 4.9 and 4.10. As a final note, the 3D models used in these experiments are obtained from the web site of TurboSquid [37].

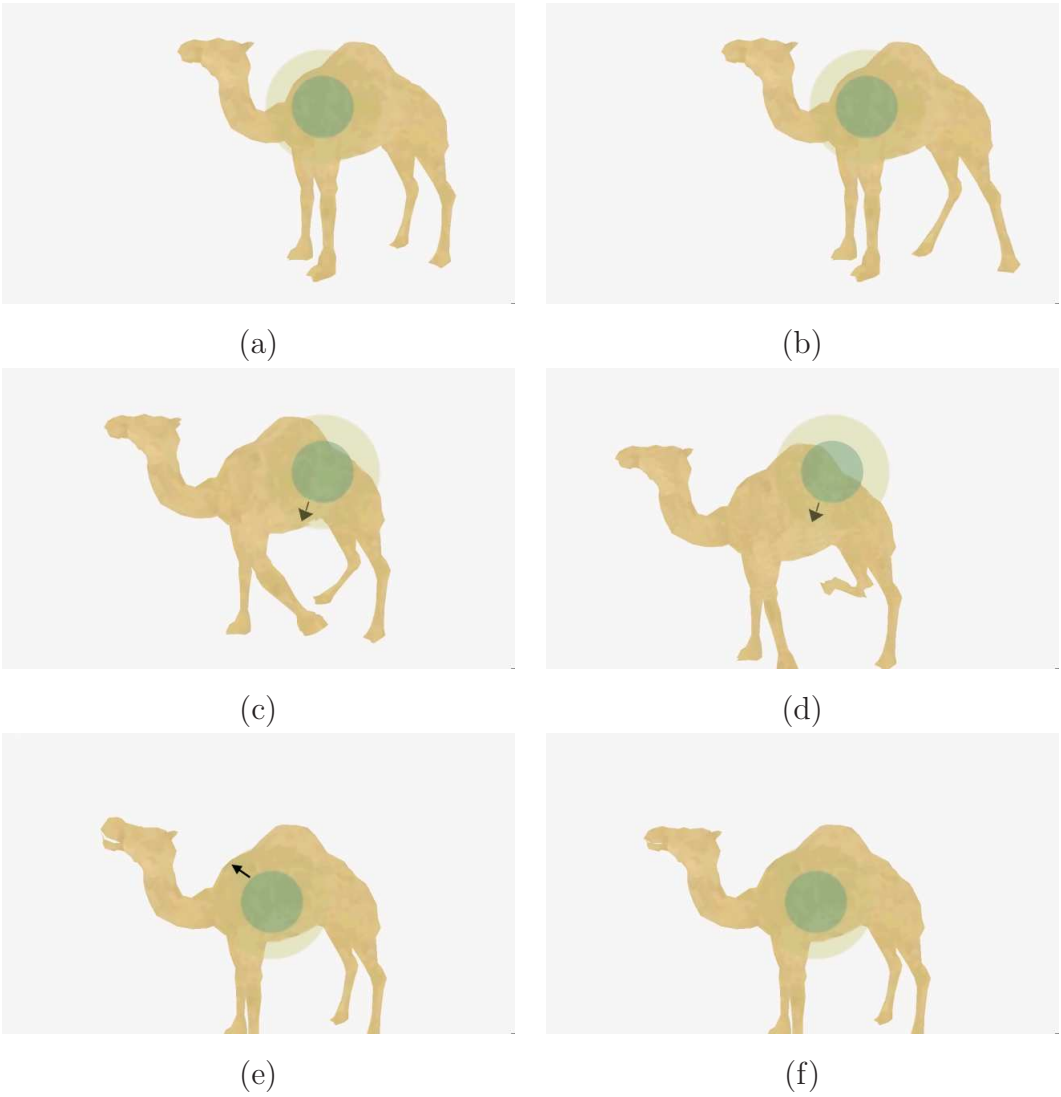


Figure 4.9: Animation of a dragon model. Transformation widget is visible.

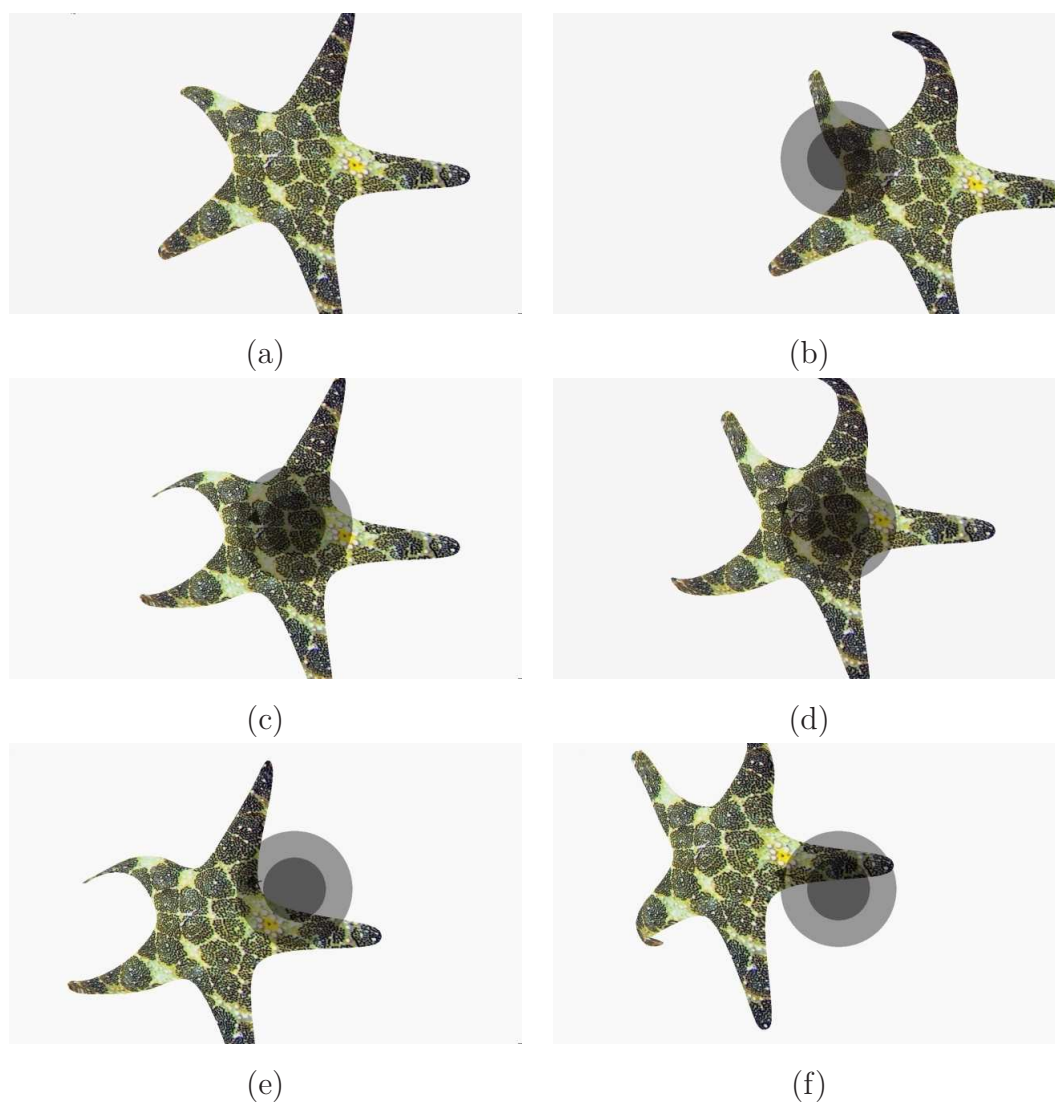


Figure 4.10: Animation of a starfish model. Transformation widget is visible in (b), (c), (d), and (e).

Chapter 5

Conclusions and Future Work

Creating simple computer animations for novice users is considered as a complicated task. Some of the commercial products are designed especially for experienced artists and require a reasonable amount of knowledge about geometric modeling and animation principles [3], [8]. Some other performance driven animation techniques such as motion capture require a complex and expensive setting [18].

In this thesis, we present a work, which proposes a solution to the problem of creation of simple animations by novice users. The approach is based on improving the current shape editing techniques, namely the Laplacian framework. We consider this framework as a suitable editing tool for our system, because it enables to manipulate desired regions of a model while preserving the shape features. In addition, it is sufficient to freely move a handle, which is a set of single or more mesh vertices in the given region, to obtain this manipulation. The most important disadvantage of this framework is the high computation time of the operations involved at the beginning of a manipulation session. We try to optimize these operations with the use of libraries that focus on linear algebra operations [25] and sparse linear systems [36]. Another drawback of this framework is the loss of volume observed as a result of the rotations involved during manipulations. In order to create more realistic results, we enhance the Laplacian framework to enable preservation of volume. We accomplish this feature by

adding a post-processing step, which updates mesh vertices to keep the volume constant, after the Laplacian operations.

In our system, we apply Laplacian editing operations to automatically generated regions of a model simultaneously in order to create an illusion of movement. These regions are obtained by decomposing the input model into meaningful parts. The human visual system perceives objects as a collection of salient parts and tends to move these parts in an animation. Therefore, by manipulating meaningful parts of a model, we produce results close to the expectations of the users. Additionally, we eliminate the need for explicit assignment of manipulation regions.

Finally, our system is targeted for multi-touch screens, an emerging technology in human-computer interaction. We make use of the multi-point feature of the multi-touch screen to develop simple interaction styles to increase the usability of our system. We obey direct manipulation principles in the design of these interaction techniques. By so, we aim to create the feeling of manipulating objects by hand.

Our current system provides expressive results, however it can be further improved. One of the most important extensions is the further development of the interaction methods. The widget we have designed to enable transformations currently supports 3D rotations and translations in x and y coordinates. Another component can be added to this widget to enable translations in the z direction as well. We have completed a prototype design for this component. Our design assumes that an additional rectangular component lies beside the original widget. The middle of this rectangle corresponds to the value of $z = 0$. Therefore, if the cursor inside this rectangle moves up, translation in -z direction is achieved. Similarly, down cursor movements result in translations in +z direction. We claim that, this additional component can be used with the thumb finger while the index finger is active inside the widget circles. To further ease the usage, we propose adjusting the orientation of the additional component according to the index finger orientation. Figure 5.1 illustrates this design.

Another possible important improvement of our system is the redesign of the

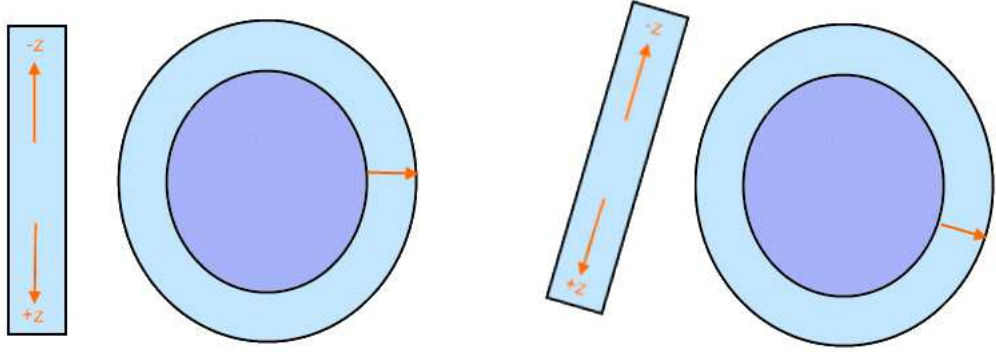


Figure 5.1: Prototype design of the extended transformation widget. Orientation of the z translation component is adjusted according to the orientation of the index finger shown by an arrow inside the circles.

volume preservation component. Instead of adjusting volume as a post-process, we suggest to enhance the minimization problem used in the Laplacian framework to consider volume preservation as well. Such an enhancement will enable to preserve volume using the Laplacian coordinates. Therefore, the problem of losing surface details during the volume correction process can be prevented.

A final useful extension is the application of the traditional animation principles to our current system. We believe that, different effects, such as exaggeration and stretch-and-squash, can be accomplished by adjusting the weights of the constraints used in the Laplacian framework and the weights of the displacement vectors used in volume preservation. We recommend to develop a heuristic to assign automatic weights according to the choice of the animation effect desired. More plausible results can be obtained as a result.

To conclude, we can say that we have gained hands on experience about a variety of subjects by completing this work. Differential methods constitute an important part of these subjects. These methods are commonly used in various modeling operations, such as transferring details between two models and mixing of these details, in addition to editing tasks. Therefore, our obtained knowledge will also be beneficial in other applications we develop. Finally, working with a multi-touch screen has been the most fascinating part of our system. Exploring

different features of this emerging technology and developing appropriate interaction techniques have been a valuable practice in human-computer interaction applications.

Bibliography

- [1] G. Antini, S. Berretti, and P. Del Bimbo, A. and Pala. 3d mesh partitioning for retrieval by parts applications. *Proceedings of the International Conference on Multimedia and Expo*, pages 1210–1231, 2005.
- [2] M. Attene, S. Katz, M. Mortara, G. Patane, M. Spagnuolo, and A. Tal. Mesh segmentation - a comparative study. *Proceedings of the IEEE International Conference on Shape Modeling and Applications*, pages 7–18, 2006.
- [3] Autodesk. 3ds max, 2009. <http://www.autodesk.com/3dsmax>.
- [4] H. Benko, A.D. Wilson, and P. Baudisch. Precise selection techniques for multi-touch screens. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1263–1272, 2006.
- [5] D.A. Bowman, E. Kruijff, LaViola J.J., and I. Poupyrev. *3D User Interfaces: Theory and Practice*. Addison-Wesley, 2005.
- [6] M. Chen, S. Mountford, and A. Sellen. A study in interactive 3-d rotation using 2-d control devices. *Computer Graphics*, 22(4):121–129, 1988.
- [7] G.M. Draper and E.K. Egbert. A gestural interface to free-form deformation. *Proceedings of Graphics Interface*, pages 113–120, 2003.
- [8] Blender Foundation. [blender.org](http://www.blender.org/blenderorg/blenderfoundation), 2009. <http://www.blender.org/blenderorg/blenderfoundation>.
- [9] B. Fröhlich and J. Plate. The cubic mouse: A new device for three-dimensional input. *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pages 526–531, 2000.

- [10] T. Grossman, R. Balakrishnan, and K. Singh. An interface for creating and manipulating curves using a high degree-of-freedom curves input device. *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pages 185–192, 2003.
- [11] M. Hachet, P. Guitton, and P. Reuter. The cat for efficient 2d and 3d interaction as an alternative to mouse adaptations. *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 205–212, 2003.
- [12] G. Hirota, R. Maheshwari, and M.C Lin. Fast volume-preserving free form deformation using multi-level optimization. *Proceedings of the ACM Symposium on Solid Modeling and Applications*, pages 234–245, 1999.
- [13] S. Houde. Iterative design of an interface for easy 3-d direct manipulation. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 135–142, 1992.
- [14] T. Igarashi, T. Moscovich, and J.F. Hughes. As-rigid-as-possible shape manipulation. *ACM Transactions on Computer Graphics*, 24(3):1134–1141, 2004.
- [15] Stantum Inc. Stantum:pioneer of multi-touch technologies, multi-touch development kit, multi-touch salut, 2009. <http://www.stantum.com>.
- [16] S. Katz, G. Leifman, and A. Tal. Mesh segmentation using feature point and core extraction. *The Visual Computer*, pages 865–875, 2005.
- [17] S. Katz and A. Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. *International Conference on Computer Graphics and Interactive Techniques*, pages 954–961, 2003.
- [18] J. Lee, J. Chai, P.S.A. Reitsma, J.K. Hodgins, and N.S. Pollard. Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics*, 21(3):491–500, 2002.
- [19] Y. Lipman, O. Sorkine, D. Cohen-Or, D. Levin, C. Rössl, and H.P. Seidel. Differential coordinates for interactive mesh editing. *Proceedings of Shape Modeling International*, pages 181–190, 2004.

- [20] Y. Lipman, O. Sorkine, D. Levin, and D. Cohen-Or. Linear rotation-invariant coordinates for meshes. *ACM Transactions on Computer Graphics*, 24(3):479–487, 2005.
- [21] Graeme McRae. Cubic formula: Cardano’s method of solving a depressed cubic, 2009. <http://2000clicks.com/mathhelp/FactoringCubic1.htm>.
- [22] A. Nealen, T. Igrashi, O. Sorkine, and M. Alexa. Fibermesh: Designing freeform surfaces with 3d curves. *ACM Transactions on Graphics*, pages 1142–1147, 2007.
- [23] A. Nealen, O. Sorkine, M. Alexa, and D. Cohen-Or. A sketch-based interface for detail preserving mesh editing. *ACM Transactions on Graphics*, pages 1142–1147, 2004.
- [24] R. Parent. *Computer Animation: Algorithms and Techniques*. Morgan Kaufmann Publishers, 2002.
- [25] R. Pozo. Lapack++: Linear algebra package in c++, 1992. <http://math.nist.gov/lapack++/>.
- [26] Stanford Exploration Project. Givens rotations, 1997. <http://sepwww.stanford.edu>.
- [27] J. Rekimoto. Smartskin: an infrastructure for freehand manipulation on interactive surfaces. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Changing Our World, Changing Ourselves*, pages 113–120, 2002.
- [28] B. Shneiderman. The future of interactive systems and the emergence of direct manipulation. *Behavior and Information Technology*, pages 237–256, 1982.
- [29] K. Shoemake. Animating rotation with quaternion curves. *ACM SIGGRAPH Computer Graphics*, pages 245–254, 1985.
- [30] K. Shoemake. Arcball: A user interface for specifying three-dimensional orientation using a mouse. *Proceedings of the Conference on Graphics Interface*, pages 151–156, 1992.

- [31] O. Sorkine. Differential representations for mesh processing. *Computer Graphics Forum*, 25(4):789–807, 2006.
- [32] O. Sorkine and M. Alexa. As-rigid-as-possible surface modeling. *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, pages 109–116, 2007.
- [33] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H.P. Seidel. Laplacian surface editing. *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pages 179–188, 2004.
- [34] G. Taubin. Estimating the tensor of curvature of a surface from a polyhedral approximation. *Proceedings of the International Conference on Computer Vision*, pages 902–908, 1995.
- [35] F. Thomas and O. Johnston. *The Illusion of Life - Disney Animation*. Walt Disney Productions, 1981.
- [36] S. Toledo. Taucs, a library of sparse linear solvers, year = 2003, note =.
- [37] Inc Turbo Squid. 3d models, 3d modeling textures and plugins at turbosquid, year = 2009, note =.
- [38] W. von Funck, H. Theisel, and H.-P. Seidel. Volume preserving mesh skinning. *Proceedings of Vision Modeling Visualization*, page to appear, 2008.
- [39] Inc Wikimedia Foundation. Dijkstra’s algorithm: Wikipedia, the free encyclopedia, 2009. <http://en.wikipedia.org/wiki/>.
- [40] R. Williams. *The Animator’s Survival Kit: A Manual of Methods, Principles, and Formulas for Classical, Computer, Games, Stop Motion, and Internet Animators*. Faber and Faber, 2002.
- [41] M. Wu and R. Balakrishnan. Multi-finger and whole hand gestural interaction techniques for multi-user tabletop displays. *Proceedings of the ACM symposium on User Interface Software and Technology*, pages 193–202, 2003.

- [42] Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, and H.Y. Shum. Mesh editing with poisson-based gradient field manipulation. *International Conference on Computer Graphics and Interactive Techniques*, pages 644–651, 2004.
- [43] K. Zhou, J. Huang, J. Snyder, X. Liu, H. Bao, B. Guo, and H.Y. Shum. Large mesh deformation using the volumetric graph laplacian. *Proceedings of ACM SIGGRAPH*, pages 496–503, 2005.