Seuz canal University

Faculty of Computers and Informatics

On Some Solution by Using Optimization Model for Laplacian Mesh

A Thesis Submitted to department of Computer Science, Faculty of Computers and Informatics- Suez canal University
In partial fulfillment of the requirement for Master of Science in Computer Science

By

Mohamed Soliman Elkomy

B.Sc in Electrical Engineering, Ain Shams Univ. 2001

Supervised by

Prof. Dr.

Mohamed Elsayed Waheed

Professor of Computer Science
Faculty of Computers and Informatics,
Suez Canal university

Dr.

Mohamed Abdullah

Lecturer of Information Systems
Faculty of Computers and Informatics
Suez Canal university

2016

Seuz canal University

Faculty of Computers and Informatics

On Some Solution by Using Optimization Model for Laplacian Mesh

A Thesis Submitted to department of Computer Science, Faculty of Computers and Informatics- Suez canal University
In partial fulfillment of the requirement for Master of Science in Computer Science

By

Mohamed Soliman Elkomy

B.Sc in Electrical Engineering, Ain Shams Univ. 2001

Examination Committee

	Prof. Dr. / Alsayed Mohamed Alsayed Alherbety Professor of Computer Science Faculty of Computers and Informatics, Ain Shams University	ا.د/ السيد محمد السيد الهربيطي أستاذ بكلية الحسابات و المعلومات جامعة عين شمس.
	Prof. Dr. / Mohamed Amin Abdelwahed Professor of Computer Science, Mathematics and Computer Science department Monofia University	ا.د / محمد امين عبد الواحد أستاذ بكلية العلوم جامعة المنوفية.
	Prof. Dr. / Mohamed Alsayed Wahed Professor of Computer Science Faculty of Computers and Informatics, Suez Canal university	أ.د / محمد السيد وحيد أستاذ بكلية الحاسبات جامعة قناة السويس.

Acknowledgements

I do indeed thank **Allah** (The Al-mighty); without whose mercy and guidance this work never has been started nor completed.

I would like first to thank my supervisor Prof. Dr. Mohamed Wahed for his help and guidance and support and for giving me the opportunity to work with him. I would also like to thank Dr. Mohamed Abdallah for his encouragement and help.

I would also like to thank my family for the love and support that they provided to me.

Contents

List of Tables

List of Figures

Chapter 2

Chapter 3

Abbreviations

FBS	**Feature Based Smoothing**
VS	**Variational subdivision**
LNO	**Lagrange-Newton Optimization**
SQO	**Sequential Quadratic Optimization**
LNO	**Lagrange Newton Optimization**
LCF	**Local Cost Function**
GCF	**Global Cost Function**
PP	**Parallel Processing**
CG	**Conjugate Gradient**
FFBP	**Feed Forward Back Propagation**

List of Publications

1- [WAE14] Wahed M., Abdallah M., Elkomy M. : Non Superfluous Time Solutions of the Laplacian Framework. The International Journal Of Computer Science and Network Security 2014 Vol. 14 No. 5 pp. 8-13

2- [WAE14] Wahed M., Abdallah M., Elkomy M. : Simulation of the detection of noxious breast cancer. The International Journal Of Computer Science and Network Security 2014 Vol. 14 No. 9 pp. 1-4

Abstract

Breast cancer is the development of cancer from breast tissue. BreastCancerCare.org.uk states that a small cancer may grow very quickly or a larger cancer may have been growing slowly over a longer time. Due to the sensitivity of the breast area, women are reluctant in going through the examination process. In This thesis a simulation of the detection of breast cancer is developed. The goal is to reach a system that can detect the presence of breast cancer through 3D scans. This system can decide if the patient has breast cancer or not. We will present fast methods to calculate the Laplacian processing framework and differential representations which is used to facilitate realistic models.

This topic is related to Surface representation and processing which is one of the key topics in computer graphics.

A Comparison between local and global optimization is presented that shows that global optimization is superior but not helpful in our case because we are interested in making the small area of tumor blend with the a section of the breast not the complete body.

Sometime the models that are subject to geometric processing are huge in size and take too much time for processing. Our results shows time saving methods to calculate the Laplacian framework.

It is shown through the variant methods that are proposed saves time in calculating the Laplacian Framework. We show that by dividing the Laplacian Framework problem to several smaller problems that can reduce the time required to solve the complete problem.

We then discuss the Perturbation of the Laplacian framework and we answer what if the model subject to our problem changes in shape question. Do we need to solve the whole problem from scratch or is there a short cut that we

can use as part of our previous solution. We use well known techniques in operations research.

The Laplacian Framework is used to generate models that represent female human bodies of both healthy and breast cancer bodies.

We use the methods discussed that deal with Perturbation of an object and the solution of the Laplacian framework to generate multiple objects of both cancer and cancer free bodies.

We then use a feed forward neural network and train it using these models. We latter on generate more models and test the trained neural network against these models and see if it has learned from our training and if it is capable of distinguishing between the cancer and the healthy objects.

Table of Contents

CHAPTER 1 Introduction

Introduction

The Following introduction is inspired by many of the references mentioned in the references section, as is most of the text in the different chapters.

3D Laser Scanning is a non-contact, non-destructive technology that digitally captures the shape of physical objects using a line of laser light. 3D laser scanners create "point clouds" of data from the surface of an object. In other words, 3D laser scanning is a way to capture a physical object's exact size and shape into the computer world as a digital 3-dimensional representation.

Triangle meshes produced from optically scanned point clouds. Scanning is measurement, and any measurement is subject to noise. Apart from detracting from the visual quality of the model, noise can also be a problem for other geometric algorithms. Therefore, removing noise from the "signal" in a triangle mesh is an important concern.

Smoothing is removing the high frequency noise. This Thesis shows methods to divide the smoothing problem into smaller problems. Hence the solution is reached in less time. Also the technique used in smoothing allows variations to be introduced in the model. The thesis also shows that these varied models can be calculated from the smoothing of one model in less time required to smoothing each varied model separately. This method is then used to generate several models that will be used in a simulation experiment.

1.1 Problem Definition

Breast Cancer patients suffer from pain and embarrassment to reveal private parts from their body. This Thesis is an attempt to provide using scientific advances a method of diagnosing the illness without humiliating the patients.

We use mathematical tools to generate models of healthy and breast affected bodies, and we use a neural network to detect that illness.

If this was realized, the patients will not be exposed.

1.2 Objectives

The objective of this thesis is to:

1. Help breast cancer patients

2.2 Thesis Outlines

The rest of this thesis is organized as follows: Chapter two shows the neural network training of the healthy and ill models. The ill models were generated by super imposing lumps on healthy models. That is why we needed soothing to make the ill models as realistic as possible. Chapter three shows a survey of smoothing techniques, what was selected as a technique of smoothing is smoothing by optimization. It is also demonstrated that difference between local and global optimization. The Choice was made was to select local optimization. Chapter four shows contemporary methods to reach more time saving solutions to the Laplacian framework. Chapter five concludes with conclusion

C H A P T E R 2 Simulation of the detection of noxious breast cancer

Simulation of the detection of noxious breast cancer

Two of the Signs and Symptoms of breast cancer that Doctors Hospital at Renaissance (http://www.dhr-rgv.com/) relay on are:

» A lump, mass, or thickening in the breast

» Change in the size or shape of a breast

In this chapter the Laplacian framework is used to generate variation of the same 3D model that represent the female bust. In some samples lumps are introduced to the breast area to simulate the presence of cancer.

A neural network is trained using some of the samples and then that trained neural network can classify any new sample to either have breast cancer or represent a healthy patient.

Our method will be very effective in identifying such changes that happen in the breast area.

A tumor is going to be introduced on a normal human female breast. This tumor is going to be changed in location and in size in several samples. All theses samples are going to be generated with the aid of the Laplacian framework and the delta changes.

In section 2.1 we talk about the delta changes that are added to the healthy model, and In section 2.2 we talk about the conjugate gradient method which is the algorithm used to train the neural network, and In section 2.3 we talk about feed forward back propagation Neural networks and the mathematics behind the back propagation is revealed. In section 2.4 we talk about the results obtained from training the neural network and how it could successfully classify the different models to healthy or having breast cancer.

2.0 Least squares Perturbation and Laplacain Mesh Processing(The Effect of delta Changes on Relative Error)

As can be seen in Figure 2.1 the left image shows a female bust with a tumor, and the right image shows a healthy bust.

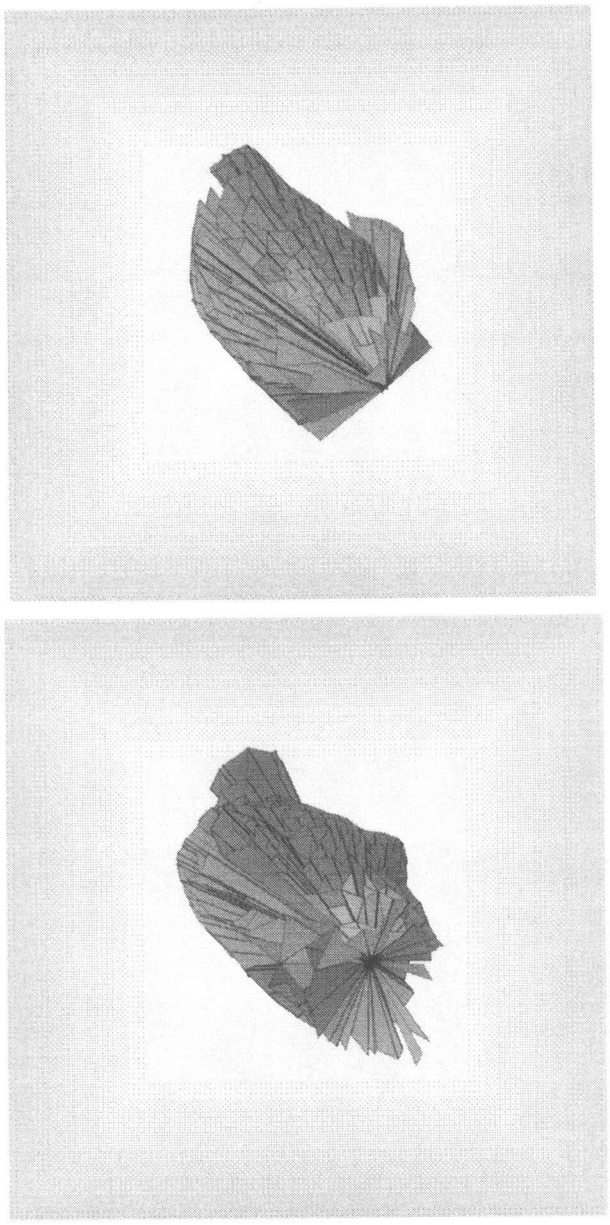


Figure 2.1: On the top we find a breast with a tumor representing breast cancer, on the bottom a healthy breast

As indicated in table 2.1, the difference between values before applying the changes to the model and after Appling the changes to the model. The changes appear in the z component of the breast affected area of the model. The increase is noticed in the columns containing the words Values with delta changes.

Table 2.1: Changes in Readings of z component of the breast area with and without tumor percentage increase

Values with delta changes	Values without delta changes	Values with delta changes	Values without delta changes	Values with delta changes	Values without delta changes
0.60824	0.60711	0.14036	0.14	0.14029	0.13966
0.54745	0.54623	0.070222	0.069529	0.31594	0.31536
0.51604	0.51516	-0.017041	-0.017617	0.44149	0.44075
0.48689	0.48571	0.30394	0.30427	0.56147	0.56086
0.35449	0.35398	0.39946	0.39955	0.48003	0.47956
0.41426	0.41339	0.47032	0.46925	0.29299	0.29268
0.27682	0.2761	0.50747	0.50657	0.64017	0.63964

2.1 Conjugate Gradient Algorithm

The objective of this section is to develop algorithms to optimize a performance index F(x). For our purposes the word "optimize" will mean to find the value of x that minimizes F(x). In this section [GKTTLMC06] lead the mathematical derivation of steepest descent and Newton's method. All of the optimization algorithms we will discuss are iterative. We begin from some initial guess, x_o, and then update our guess in stages according to an equation of the form

$$x_{k+1} = x_k + \alpha_k \boldsymbol{p}_k \tag{2.1}$$

Or

$$\Delta x_k = (x_k - x_{k+1}) = \alpha_k \boldsymbol{p}_k \tag{2.2}$$

Where the vector represents a search direction p_k, and the positive scalar is the learning rate α_k, which determines the length of the step.

2.1.1 Steepest Descent

When we update our guess of the optimum (minimum) point using the previous equation, we would like to have the function decrease at each iteration.

$$F(x_{k+1}) < F(x_k) \tag{2.3}$$

Consider the first-order Taylor series expansion

$$F(x_{k+1}) = F(x_k + \Delta x_k) \approx F(x_k) + g_k^T \Delta x_k \tag{2.4}$$

Where g_k is the gradient evaluated at the old guess x_k:

$$g_k \equiv \nabla F(x)|_{x=x_k} \tag{2.5}$$

So the second term on the right-hand side of the Taylor expansion must be negative:

$$g_k{}^T \Delta x_k = \alpha_k g_k{}^T \boldsymbol{p}_k \tag{2.6}$$

We will select α_k that is small, but greater than zero. This implies:

$$g_k{}^T \boldsymbol{p}_k < 0 \tag{2.7}$$

Any vector p_k that satisfies this equation is called a descent direction. The function must go down i f we take a small enough step in this direction. This brings up another question. What is the direction of steepest descent? This will occur when:

$$g_k{}^T \boldsymbol{p}_k < 0 \tag{2.8}$$

is most negative. The inner product between the gradient and the direction vector will be most negative when the direction vector is the negative of the gradient.

$$\boldsymbol{p}_k = -g_k \tag{2.9}$$

$$x_{k+1} = x_k - \alpha_k g_k \tag{2.10}$$

For steepest descent there are two general methods for determining the learning rate. One approach is to minimize the performance index F(x) with respect to the learning rate at each iteration. In this case we are minimizing along the line

$$x_k - \alpha_k g_k \tag{2.11}$$

The other method for selecting α_k is to use a fixed value (e.g., $\alpha_k = 0.02$), or to use variable, but predetermined, values (e.g., $\alpha_k = 1/k$).

2.1.2 Newton's Method

The general form of a quadratic function is:

$$\mathbf{F(x)} = \frac{1}{2}\mathbf{x}^T\mathbf{A}\,\mathbf{x} + \mathbf{d}^T\mathbf{x} + \mathbf{c} \tag{2.12}$$

We can now compute the gradient of F(x) as:

$$\nabla\mathbf{F(x)} = \mathbf{A}\,\mathbf{x} + \mathbf{d} \tag{2.13}$$

and in a similar way we can find the Hessian:

$$\nabla^2\mathbf{F(x)} = \boldsymbol{A} \tag{2.14}$$

The derivation of the steepest descent algorithm was based on the first-order Taylor series Expansion. Newton's method is based on the second-order Taylor series:

$$F(\mathbf{x}_{k+1}) = F(\mathbf{x}_k + \Delta\mathbf{x}_k) \approx F(\mathbf{x}_k) + g_k{}^T \Delta\mathbf{x}_k + \frac{1}{2}\Delta\mathbf{x}_k{}^T A_k \Delta\mathbf{x}_k \tag{2.15}$$

The principle behind Newton's method is to locate the stationary point of this quadratic approximation to F(x). If we take the gradient of this quadratic function with respect to and set it equal to zero, we find

$$g_k + A_k \Delta \mathbf{x}_k = \mathbf{0} \tag{2.16}$$

Newton's method is then defined:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - A_k^{-1} g_k \tag{2.17}$$

This method will always find the minimum of a quadratic function in one step. This is because Newton's method is designed to approximate a function as quadratic and then locate the stationary point of the quadratic approximation. If the original function is quadratic (with a strong minimum) it will be minimized in one step.

If the function F(x) is not quadratic, then Newton's method will not generally converge in one step.

While Newton's method usually produces faster convergence than steepest descent, the behavior of Newton's method can be quite complex. In addition to the problem of convergence to saddle points (which is very unlikely with steepest descent), it is possible for the algorithm to oscillate or diverge. Steepest descent is guaranteed to converge, if the learning rate is not too large or if we perform a linear minimization at each stage.

Any general function can be approximated by quadratic functions in small neighborhoods, especially near local minimum points

2.1.3 Conjugate Gradient

[HDBD14] and [She05] shows the following about Conjugate Gradient. Newton's method has a property called quadratic termination, which means that it minimizes a quadratic function exactly in a finite number of iterations. Unfortunately, it requires calculation and storage of the second derivatives. When the number of parameters n, is large, it may be impractical to compute all of the second derivatives. (Note that the gradient has elements, while the Hessian has n squared elements.) This is especially true with neural networks, where practical applications can require several hundred to many

thousand weights. For these cases we would like to have methods that require only first derivatives but still have quadratic termination.

A set of search directions that will guarantee quadratic termination is conjugate directions.

A set of vectors are mutually conjugate with respect to a positive definite Hessian matrix if and only if

$$\mathbf{p}_k{}^T \mathbf{A}\, \mathbf{p}_j = \mathbf{0} \qquad k! = j \tag{2.18}$$

As with orthogonal vectors, there are an infinite number of mutually conjugate sets of vectors that span a given -dimensional space. One set of conjugate vectors consists of the eigenvectors of A.

It is not surprising that we can minimize a quadratic function exactly by searching along the eigenvectors of the Hessian matrix, since they form the principal axes of the function contours. Unfortunately this is not of much practical help, since to find the eigenvectors we must first find the Hessian matrix. We want to find an algorithm that does not require the computation of second derivatives.

If we make a sequence of exact linear searches along any set of conjugate directions, then the exact minimum of any quadratic function, with parameters n, will be reached in at most n searches

First, we want to restate the conjugacy condition, without use of the Hessian matrix. Recall that for quadratic functions. We find that the change in the gradient at iteration k+1 is:

$$\Delta \mathrm{g}_k = \mathrm{g}_{k+1} - \mathrm{g}_k = (A_k \mathbf{x}_{k+1} + \mathbf{d}) - (A_k \mathbf{x}_k + \mathbf{d}) = A \Delta \mathbf{x}_k \tag{2.19}$$

$$\Delta \mathbf{x}_k = (\mathbf{x}_{k+1} - \mathbf{x}_k) = \alpha_k \mathbf{p}_k \tag{2.20}$$

We can now restate the conjugacy conditions

$$\alpha_k \mathbf{p}_k{}^T A \mathbf{p}_j = \Delta \mathbf{x}_k A \mathbf{p}_j = \Delta \mathrm{g}_k \mathbf{p}_j = 0 \qquad k! = j \tag{2.21}$$

Note that we no longer need to know the Hessian matrix. We have restated the conjugacy conditions in terms of the changes in the gradient at successive iterations of

the algorithm. The Search directions will be conjugate if they are orthogonal to the changes in the gradient.

Note that the first search direction, p_o , is arbitrary, and can be any vector That is orthogonal to Δg_o. It is common to begin the search in the steepest descent direction:

$$\mathbf{p_o} = -g_o \tag{2.22}$$

At each iteration: we need to construct a vector p_k that is orthogonal

$\{\Delta g_o, \Delta g_1, \dots, \Delta g_{k-1})$

$$\mathbf{p_k} = -g_k + \beta_k \mathbf{p_{k-1}} \tag{2.23}$$

The conjugate gradient algorithm is something of a compromise between steepest descent and Newton's method, it does not require the calculation of second derivatives, and yet it still has the quadratic convergence property. (It converges to the minimum of a quadratic function in a finite number of iterations.) [DBJH 14]

We define a quantitative measure of network performance, called the performance index, which is small when the network performs well and large when the network performs poorly.

The backpropagation algorithm for multilayer networks is a generalization of the LMS algorithm and both algorithms use the same performance index: mean square error. The mean squared error performance index for multilayer networks is not quadratic; therefore the algorithm would not normally converge in iterations. The development of the conjugate gradient algorithm does not indicate what search direction to use once a cycle of iterations has been completed. There have been many procedure s suggested, but the simplest method is to reset the search direction to the steepest descent direction.

2.2 Feed Forward Back propagation Neural Network

In "feed forward", neurons are connected foreword. Each layer contains connections to the next layer. The term back propagation describes how the neural network is trained. It is a supervised training method, the network must be provided with both inputs and anticipated outputs, and the back propagation training algorithm then takes the calculated error and adjusts the weights of various layers backwards from the output layer to the input layer [HJ08].

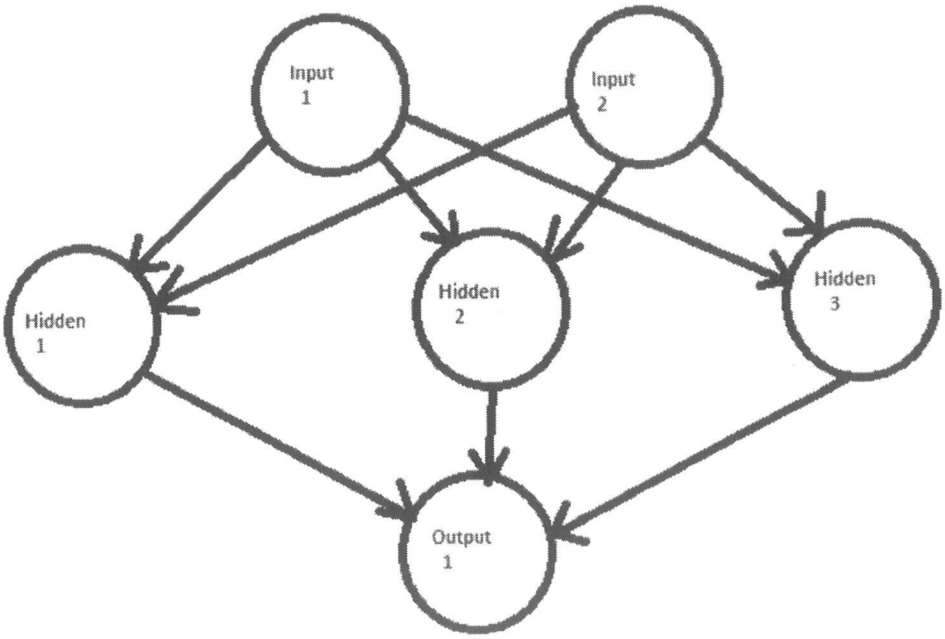


Figure 2.2 A Feed Forward Neural Network with one Hidden layer

2.2.1 Back Propagation Algorithm (BP Algorithm)

Description of BP Algorithm in Mathematics

[LCJF12] shows in this section a mathematics description of the BP algorithm.

The ideology guiding the learning rules of BP network is: the modification to the weight value and threshold value of network shall be done along the negative gradient direction reflecting the fastest declining of function.

$$x_{k+1} = x_k - \eta_k g_k \tag{2.24}$$

In the formula mentioned above, xk represents the matrix of current weight value and threshold value; gk represents the gradient of current function; ηk represents the learning rate. Here, the three-layer BP network is taken as an example to describe the BP algorithm in details.

As for the three-layer BP network, suppose its input node is xi, the node of hide layer is yj, and the node of output layer is zl. The weight value of network between the input

node and node of hide layer is wji, and the weight value of network between the nodes of hide layer and output layer is vlj. When the expected value of the output node is tl, f(·) is the active function. The computational formula of the model is expressed as follows:

Forward propagation: output of computer network

Output of the node of hide layer

$$y_j = f\left(\sum_i w_{ji}x_i - \theta_j\right) = f(net_j) \tag{2.25}$$

$$net_j = \sum_i w_{ji}x_i - \theta_j \tag{2.26}$$

Computational output of the output node

$$z_l = f\left(\sum_j v_{lj}y_j - \theta_l\right) = f(net_l) \tag{2.27}$$

$$net_l = \sum_j v_{lj}y_j - \theta_l \tag{2.28}$$

Error of the output node

$$E = \frac{1}{2}(t_l - z_l)^2 = \frac{1}{2}\left(t_l - f\left(\sum_j v_{lj}y_j - \theta_l\right)\right)^2$$

$$= \frac{1}{2}\left(t_l - f\left(\sum_j v_{lj} f\left(\sum_i w_{ji}x_i - \theta_j\right) - \theta_l\right)\right)^2 \tag{2.29}$$

Back propagation: the gradient descent method is adopted to regulate the weight value of all layers, and the learning algorithm of weight value is expressed as follows:

Modification of Weight Value

$$\frac{\partial E}{\partial v_{lj}} = \sum_{k=1}^{n} \frac{\partial E}{\partial z_k}\frac{\partial z_k}{\partial v_{lj}} = \frac{\partial E}{\partial z_l}\frac{\partial z_l}{\partial v_{lj}} \tag{2.30}$$

E is a function containing several z_k, but only one z_l is related with v_{lj} and all the z_k, are independent from each other, in this formula,

$$\frac{\partial E}{\partial z_l} = \frac{1}{2}\sum_k -2\left[(t_k - z_k).\frac{\partial z_k}{\partial z_l}\right] = -(t_l - z_l) \tag{2.31}$$

$$\frac{\partial z_l}{\partial v_{lj}} = \frac{\partial z_l}{\partial net_l}\frac{\partial net_l}{\partial v_{lj}} = f'(net_l).y_j \tag{2.32}$$

$$\frac{\partial E}{\partial v_{lj}} = -(t_l - z_l).f'(net_l).y_j \tag{2.33}$$

Suppose the error of input node is

$$\delta_l = (t_l - z_l).f'(net_l) \tag{2.34}$$

$$\frac{\partial E}{\partial v_{lj}} = -\delta_l.y_j \tag{2.35}$$

Deviation of the node of hide layer by error function

$$\frac{\partial E}{\partial w_{ji}} = \sum_l \sum_j \frac{\partial E}{\partial z_l}.\frac{\partial z_l}{\partial y_j}.\frac{\partial y_j}{\partial w_{ji}} \tag{2.36}$$

E is a function containing several z_l; it is targeted at certain w_{ji}, corresponding to one y_j, and related to all z_l, in this formula,

$$\frac{\partial E}{\partial z_l} = \frac{1}{2}\sum_k -2\left[(t_k - z_k).\frac{\partial z_k}{\partial z_l}\right] = -(t_l - z_l) \tag{2.37}$$

$$\frac{\partial E}{\partial w_{ji}} = -\sum_l (t_l - z_l).f'(net_l).v_{lj}.f'(net_j).x_i = -\sum_l \delta_l v_{lj} f'(net_j)x_i \tag{2.38}$$

Suppose the error of node of hide layer is

$$\delta_j' = f'(net_j)\sum_l \delta_l v_{lj} \tag{2.39}$$

$$\frac{\partial E}{\partial w_{ji}} = -\delta_j' x_i \tag{2.40}$$

As the modification of weight Δv_{lj} and Δw_{ji} is in proportion to the error functions and descends along the gradient, the formula showing the modification of weight of hide layer and output layer is expressed as follows:

$$\Delta v_{lj} = -\eta \frac{\partial E}{\partial v_{lj}} = \eta \delta_l y_j \tag{2.41}$$

In this formula, η represents the learning rate. The formula showing the modification between the input layer and hide layer is expressed as follows:

$$\Delta w_{ji} = -\eta' \frac{\partial E}{\partial w_{ji}} = \eta' \delta_j' x_i \tag{2.42}$$

$$\delta_j' = f'(net_j).\sum_l \delta_l v_{lj} \tag{2.43}$$

In this formula, η' represents the learning rate; $\sum_l \delta_l v_{lj}$ in the node error of hide layer δ_j' expresses that the error δ_l of output node z_l is back propagated through the weight value v_{lj} to the node y_j to become the error of node of the hide layer.

Modification of Threshold Value

The threshold value θ is also a variation value and it also needs to be modified while the weight value is modified; the theory applied is the same as that used in the modification of weight value.

Derivation of the threshold of output node by error function

$$\frac{\partial E}{\partial \theta_l} = \frac{\partial E}{\partial z_l}\frac{\partial z_l}{\partial \theta_l} \tag{2.44}$$

$$\frac{\partial z_l}{\partial \theta_l} = \frac{\partial z_l}{\partial net_l}\frac{\partial net_l}{\partial \theta_l} = f'(net_l).(-1) \tag{2.45}$$

In this way

The formula expressing the modification of threshold value is

$$\Delta\theta_l = \eta\frac{\partial E}{\partial \theta_l} = \eta\delta_l \tag{2.46}$$

$$\theta_l(k+1) = \theta_l(k) + \eta\delta_l \tag{2.47}$$

Derivation of the threshold of node of hide layer by error function

$$\frac{\partial E}{\partial \theta_j} = \sum_l \frac{\partial E}{\partial z_l}\cdot\frac{\partial z_l}{\partial y_j}\cdot\frac{\partial y_j}{\partial \theta_j} \tag{2.48}$$

$$\frac{\partial y_j}{\partial \theta_j} = \frac{\partial y_j}{\partial net_l}\frac{\partial net_l}{\partial \theta_j} = f'(net_l).(-1) \tag{2.49}$$

$$\frac{\partial E}{\partial \theta_j} = \sum_l (t_l - z_l).f'(net_l).v_{lj}.f'(net_j) = \sum_l \delta_l v_{lj} f'(net_j) = \delta_j' \tag{2.50}$$

$$\Delta\theta_j = \eta'\frac{\partial E}{\partial \theta_j} = \eta'\delta_j \tag{2.51}$$

$$\theta_j(k+1) = \theta_j(k) + \eta'\delta_j \tag{2.52}$$

2.3 Choosing the Network Structure and Producing Results

In this section we talk about how we conducted the Experiment. A two layer feed forward neural network can be trained to react to a given input pattern with a prescribed output response. The hidden layer maps an input vector onto one of the vertices of a unit hyper cube. The output neuron realizes a hyper plane to separate vertices according to the different class that they belong to [TK09].

We used Matlab as a pattern recognition tool to train a two layer neural network.

The input to the Neural Network shown in Figure 2.2 is 161 vertex points representing the breast area. We take only the z component representing the height of the breast.

We use the Laplacian model with delta changes to generate 16 models that have cancer. Omega ω that will appear in future chapters is selected to take the values (2, 1.5, 1, 0.7), also we divide the breast region into four quarters. So we have different 16 models representing the different combination of the four quarters and the omega values. As shown in figure 2.4 to Figure 2.6

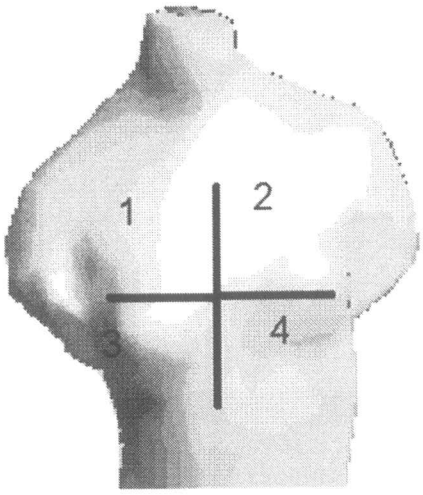


Figure 2.3 Dividing the breast area into four sections to introduce a tumor in each section

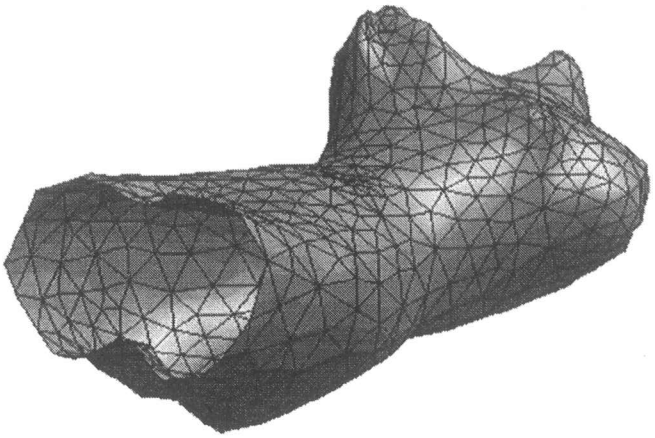


Figure 2.4 Exaggerated tumors in the upper left section of the breast

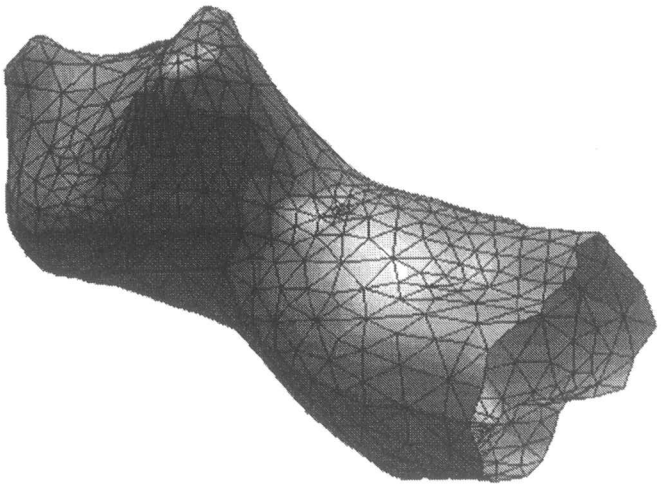


Figure 2.5 Exaggerated tumor in the bottom part of the breast

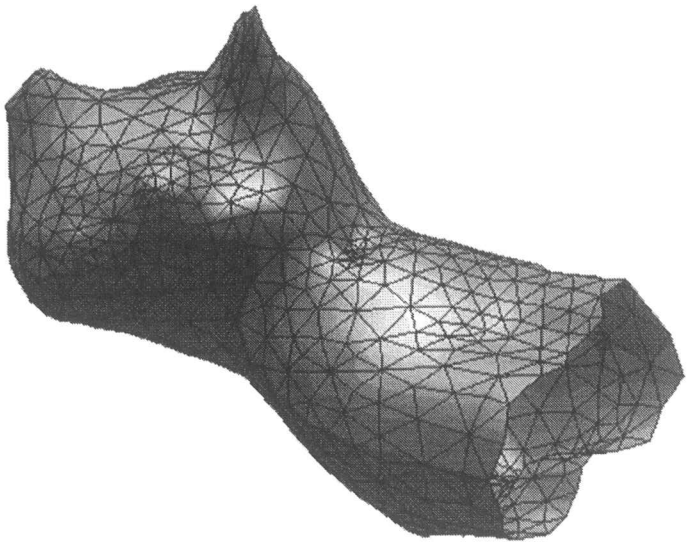


Figure 2.6 Exaggerated tumor in the upper section of the right side of the breast

The healthy models come from only the variation of omega values. As shown in figures from figure 2.9 to figure 2.10

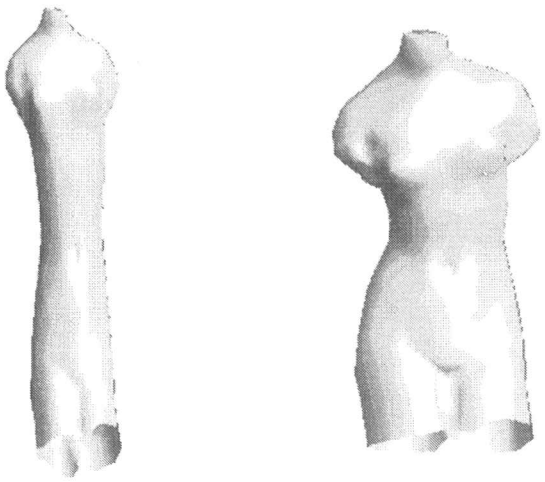


Figure 2.7 To the left a model with very small omega, on the right original model

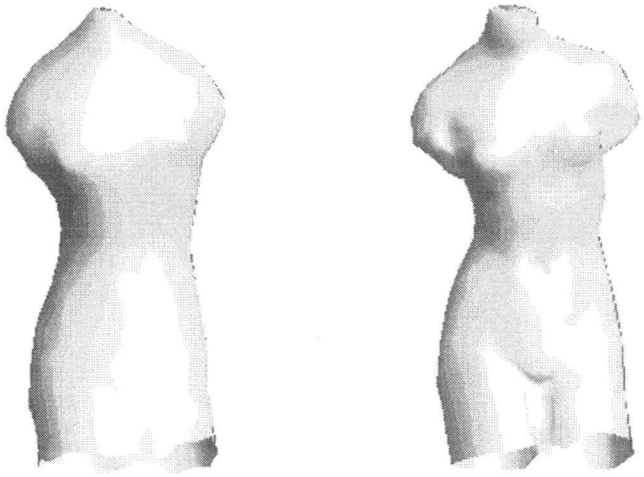


Figure 2.8 on the left omega equals .7, on the right original model

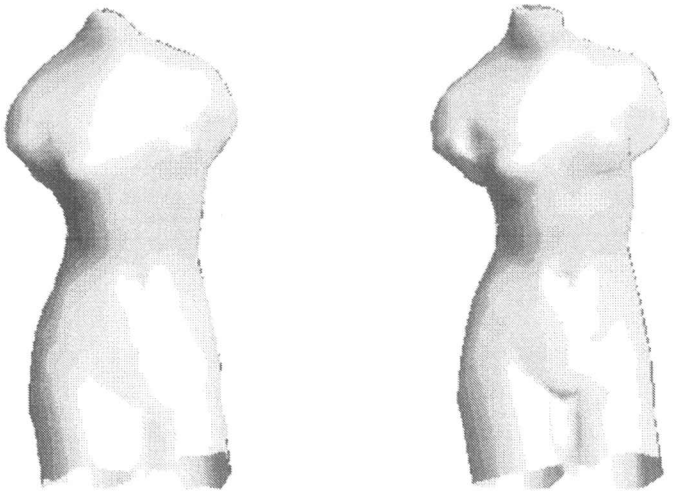


Figure 2.9 on the left omega equals 2, on the right original model

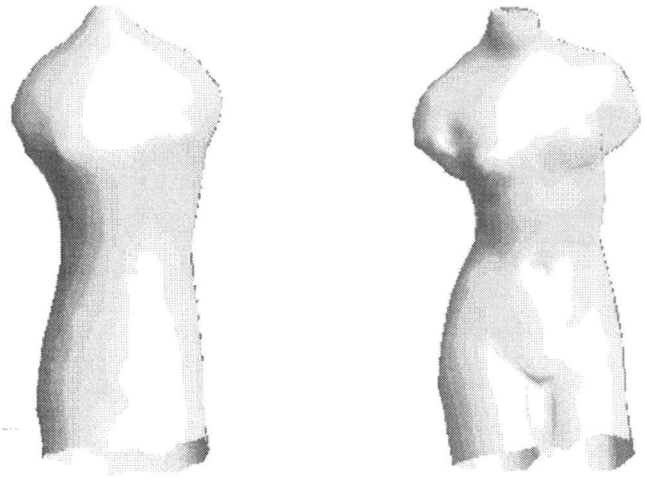


Figure 2.10 on the left omega equals .5, on the right original model

The delta change applied to the different four regions equals thirty percent extra of the actual breast z component value. The network output is a layer composed of one output. The output is one if there is tumor, zero otherwise.

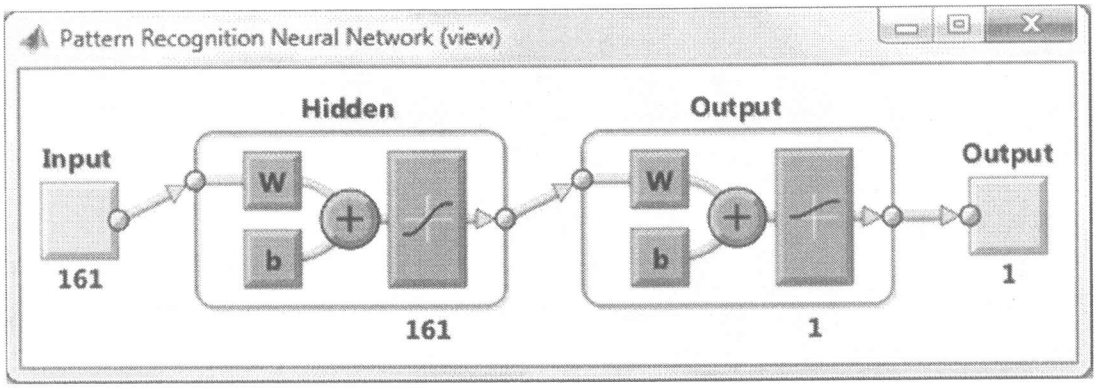


Figure 2.11: The Neural network used to detect breast cancer

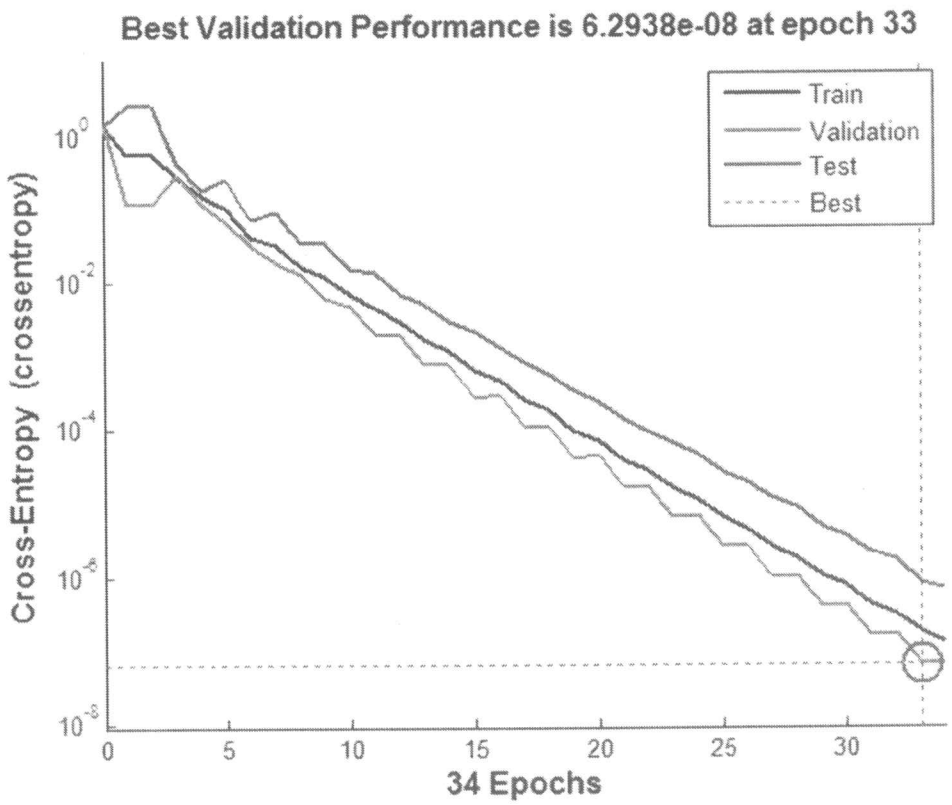


Figure 2.12: Training of neural network

Choice of Training Algorithm

For large networks, the Scaled Conjugate Gradient Algorithm of [Moll93] is very efficient. This method is also attractive for pattern recognition problems.

Stopping Criteria

For most applications of neural networks, the training error never converges identically to zero. The error can reach zero for the perceptron network, assuming a linearly separable problem, .However, It is unlikely to happen for multilayer networks. For this reason, we need to have other criteria for deciding when to stop the training.

We can stop the training when the error reaches some specified limit. However, it is usually difficult to know what an acceptable error level is. The simplest criterion is to stop the training after a fixed number of iterations

Another stopping criterion is the norm of the gradient of the performance index. If this norm reaches a sufficiently small threshold, then the training can be stopped. Since the

gradient should be zero at a minimum of the performance index, this criterion will stop the algorithm when it gets close to the minimum. Unfortunately, the performance surface for multilayer networks can have many flat regions, where the norm of the gradient will be small. For this reason, the threshold for the minimum norm should be set to a very small value

2.4 Validation

Pattern Recognition

The confusion matrix is a table whose columns represent t the target class and whose rows represent the output class. The correctly classified inputs show in the diagonal cells of the confusion matrix. The off-diagonal cells show misclassified inputs. The lower left cell shows that four inputs from Class 1 were misclassified by the network as Class2. If Class 1 is considered a positive outcome, then the lower left cell represents *false negatives*, which are also called Type II errors. The upper right cell shows that one input from Class 2 was misclassified by the network as Class 1. This would be considered a *false positive* or a Type I error.

ROC Curve

Another useful tool for analyzing a pattern recognition network is called the *Receiver Operating Characteristic (ROC) curve*. To create this curve, we take the output of the trained network and compare it against a threshold which ranges from -1 to +1 (assuming a tansig transfer function in the last layer). Inputs that produce values above the threshold are considered to belong to Class 1, and those with values below the threshold are considered to belong to Class 2. For each threshold value, we count the fraction of true positives and false positives in the data set. This pair of numbers produces one point on the ROC curve.

The ideal point for the ROC curve to pass through would be (0,1), which would correspond to no false positives and all true positives.

Overfitting and Extrapolation

The total data set is divided into three parts: training, validation and testing. The training set is used to calculate gradients and to determine weight updates. The validation set is used to stop training before overfitting occurs.

The test set is used to predict future performance of the network. The test set performance is the measure of network quality. If, after a network has been trained, the test set performance is not adequate, then there are usually four possible causes:

- The network has reached a local minimum,
- The network does not have enough neurons to fit the data,
- The network is overfitting, or
- The network is extrapolating.

The local minimum problem can almost always be overcome by retraining the network with five to ten random sets of initial weights. The network with minimum training error will generally represent a global minimum. The other three problems can generally be distinguished by analyzing the training, validation and test set errors.

For example, if the validation error is much larger than the training error, then overfitting has probably occurred. Even though early stopping is used, it is possible to have some overfitting, if the training occurs too quickly, in this case, we can use a slower training algorithm to retrain the network.

If the validation, training and test errors are all similar in size, but the errors are too large, then it is likely that the network is not powerful enough to fit the data. In this case, we should increase the number of neurons in the hidden layer and retrain the network

If the validation and training errors are similar in size, but the test errors are significantly larger, then the network may be extrapolating. This indicates that the test data fall outside therange of the training and validation data. In this case, we need to get more data. You can merge the test data into the training/validation data and then collect new test data. You should continue to add data until the results on all three data sets are similar.

If training, validation and test errors are similar, and the errors are small enough, then we can put the multilayer network to use. However, we still need to be careful about the possibility of extrapolation. If the multilayer network inputs are outside the range of the

data with which it was trained, then extrapolation will occur. It is difficult to guarantee that training data will encompass all future uses of a neural network. One method for detecting extrapolation is to train a companion competitive network to cluster the input vectors in the multilayer network training set. Then, when an input is applied to the multilayer network, the same input is applied to the companion competitive network. When the distance of the input vector to the nearest prototype vector of the competitive network is larger than the distance from the prototype to the most distant member of its cluster of inputs in the training set, we can suspect extrapolation. This technique is referred to as *novelty detection*.

In the All Confusion Matrix the upper left cell shows that 2 of the 2 Cancer patients in the test set were classified correctly, while the 2,2 cell shows that 12 of the 12 Cancer patients were classified correctly. A total of 100% of the test data were classified correctly.

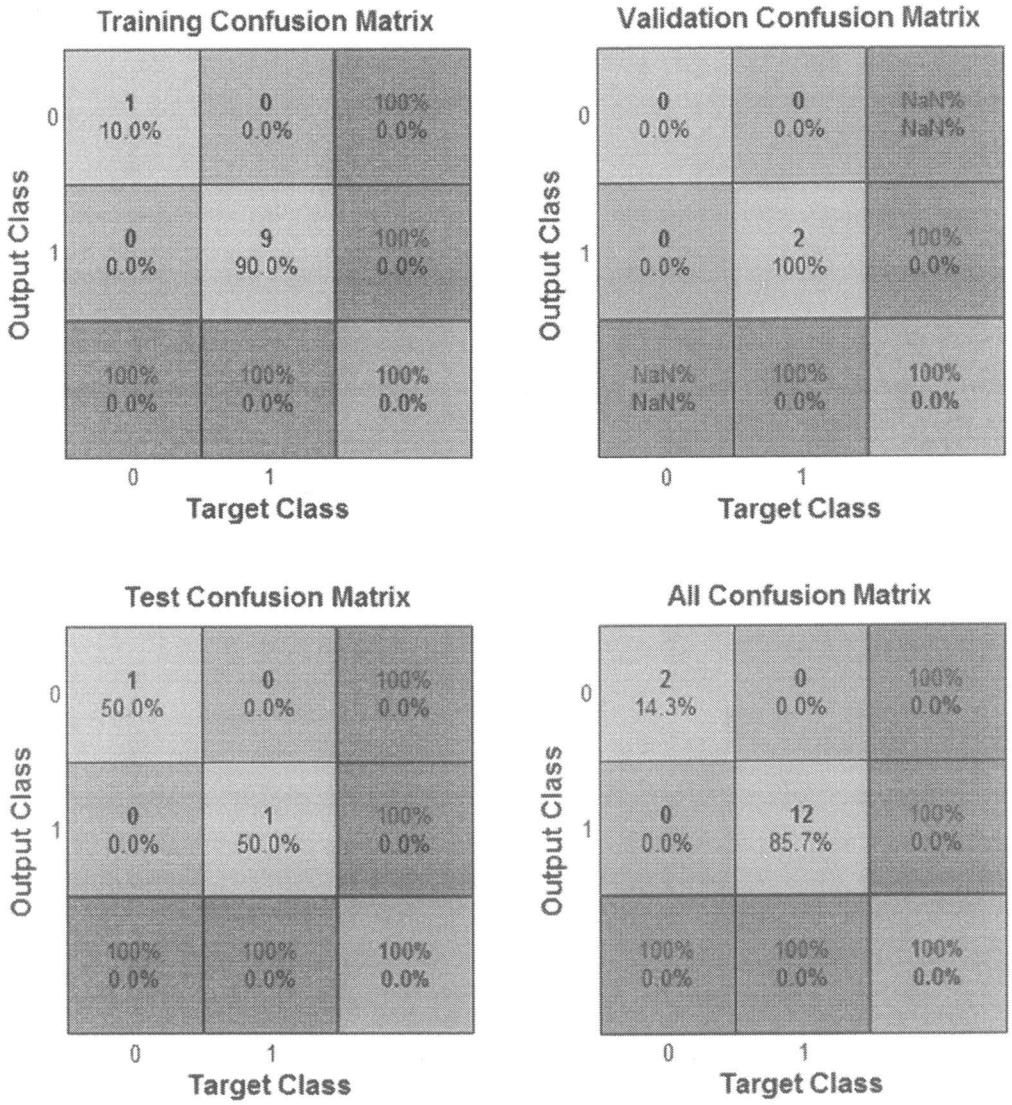


Figure 2.13: Confusion Matrix

Another useful validation tool for pattern recognition problems is the Receive Operating Characteristic (ROC) curve. The ideal curve would follow the path from 0,0 to 0,1 and then to 1,1. The curve for this test is identical to ideal.

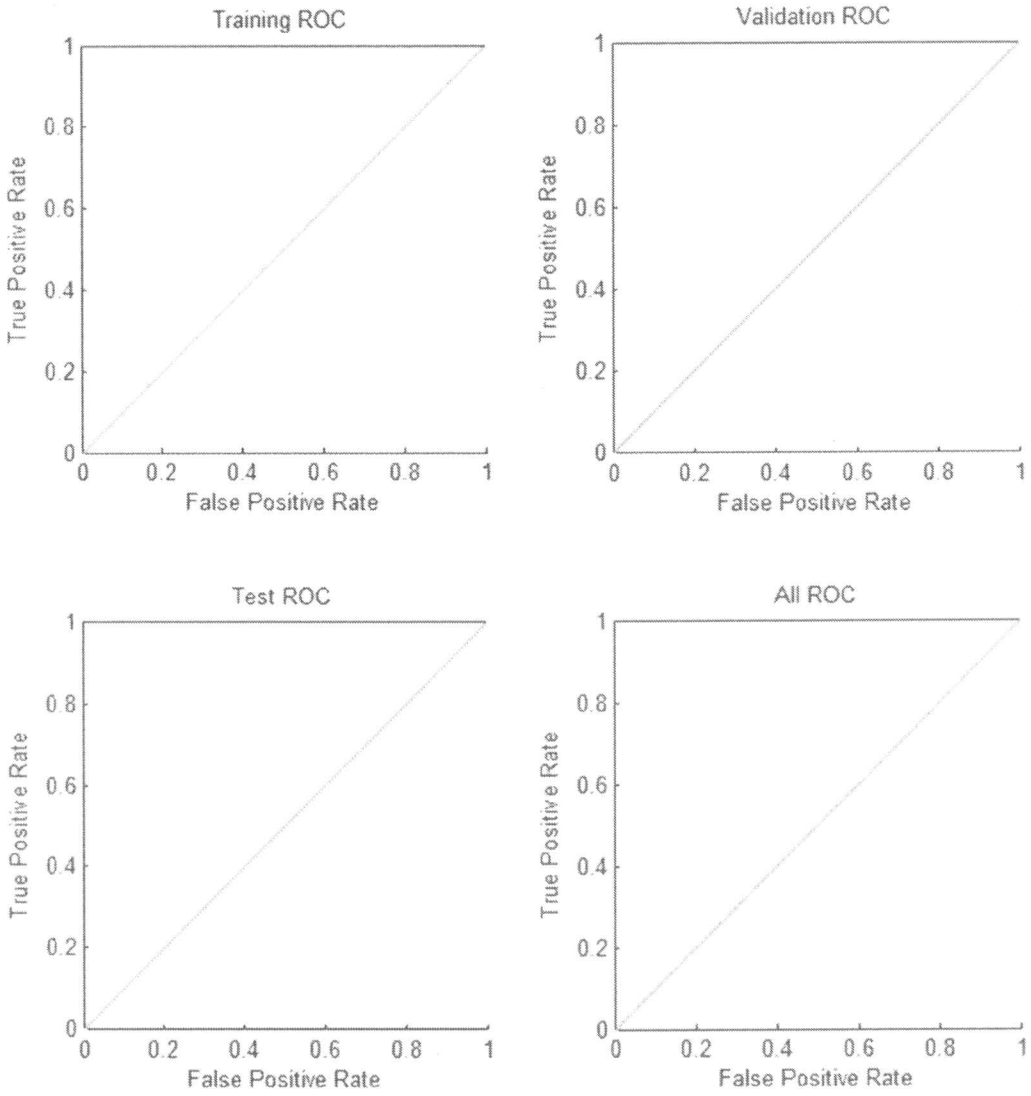


Figure 2.14: Receive Operating Characteristic (ROC) curve

Summary:

We have trained a neural network with breast cancer and healthy models of female bust. Our Neural network was able to classify successfully between new models that have cancer and other that are healthy.

CHAPTER 3 Background and Related Work

BackGround And Realted Work

Chapter three contains derivation and theoretical background. We will present different variants to achieving smoothing of a 3D model. A survey of different methods is presented to achieve the goal of smoothing.

Also a conversion from continuous to discrete models is presented along with the derivation of discrete differential operators that are used in the Laplacian Framework to achieve optimization.

A survey of different methods is presented to achieve the goal of smoothing is presented in section 3.1. In section 3.2 Surface curvatures and parametric representations of 3D models using a parameterization function to map a surface from 3D to 2D is presented. Section 3.3 deals with surface curvatures and shows the relationship between the Laplace operator and the curvature. Section 3.4 shows the discrete differential operator and contains a derivation for the discrete Laplace-Beltrami operator.

Section 3.5 describes the basics of a constrained optimization, Sequential Quadratic Optimization, Lagrange Newton Optimization, and connections between these two methods for general meshes. Section 3.6 discusses cost functions followed by differences between global and local constraint optimizations. In Section 3.7 we then present the difference between applying the cost functions mentioned in this chapter – both global and local- on our sample data set to show that global optimization is superior to local optimization.

Digital Geometry Processing

Recent innovation in 3D acquisition technology, such as computer tomography, magnetic resonance imaging, 3D laser scanning, ultrasound, radar, and microscopy has enabled

highly accurate digitization of complex 3D objects. Digital geometry processing: a relatively new field of computer science that is concerned with mathematical models and algorithms for analyzing and manipulating geometric data.

3.1 Surface Smoothing

Section 3.1 references to [BGAA12].

3D models obtained from laser scanning are usually stored and rendered as surfaces represented by triangular meshes. In all stages of the mesh construction process, noise is inevitably introduced. Surface smoothing, or denoising, adjusts vertex positions so that the overall surface becomes smoother while keeping mesh connectivity, or topology, unchanged, and it is an active area of research [DMSB99] .

We need to deal with acquired geometry: for instance, triangle meshes produced from optically scanned point clouds. Scanning is measurement, and any measurement is subject to noise. Apart from detracting from the visual quality of the model, noise can also be a problem for other geometric algorithms. Therefore, removing noise from the "signal" in a triangle mesh is an important concern, and, in fact, the analogy is excellent since we can construe the vertex positions of a triangle mesh as a discrete signal on an irregular grid.

The solution to this problem is by smoothing. The simplest type of mesh smoothing is the Laplacian smoothing, where a vertex is replaced by the average of its neighbors. Laplacian smoothing is effective but results in a great deal of shrinkage. Taubin smoothing, which is described next, does a better job of preserving the coarse features, but both Taubin and Laplacian smoothing also have a tangential component, causing the shape of the triangles to be smoothed as well as the geometry of the shape they represent.

We replace the Laplacian with the Laplace–Beltrami operator and arrive at mean curvature flow, which does a much better job of preserving the shape of the triangles while smoothing the geometry represented by the mesh. The Laplace–Beltrami operator for a triangle mesh can be written as a matrix as we shall discuss, and we can use the

eigenvectors of this matrix to perform a spectral analysis of the shape completely analogous to the discrete Fourier transform. This can be used for smoothing although for large meshes it is quite challenging to implement efficiently.

We will also discuss some smoothing methods that preserve sharp edges and corners. Such features are treated as noise by other smoothing algorithms. Finally, smoothing combined with an up-sampling of the triangle mesh leads to a simple technique for generating very smooth surfaces. This technique is known as variational subdivision.

3.1.1 Laplacian and Taubin Smoothing

We can smooth the vertex positions, $\mathbf{p}_i$, of our mesh using the Laplacian:

$$\mathbf{p}_i \leftarrow \mathbf{p}_i + \lambda L(\mathbf{p}_i)$$

Returning to the low-pass filter outlook, Taubin observed that it is possible to define notions of frequency and vibration modes for a triangle mesh, and these concepts are strongly linked to the Laplacian. Laplacian smoothing has the effect that it attenuates all frequencies (except 0). This indiscriminate attenuation leads to severe shrinkage, and Taubin designed a filtering process with less shrinkage. The result is the λ|μ algorithm, which is identical to Laplacian smoothing except that for every other iteration

$$\mathbf{p}_i \leftarrow \mathbf{p}_i + \lambda L(\mathbf{p}_i)$$

and for every remaining iteration

$$\mathbf{p}_i \leftarrow \mathbf{p}_i - \mu L(\mathbf{p}_i)$$

If the constants λ and μ are chosen according to certain guidelines, this procedure will attenuate large frequencies and actually enhance low frequencies slightly.

The λ|μ algorithm is effective in that it does not shrink nearly as much as Laplacian smoothing while it does remove high frequency noise. The theoretical explanation for this is that it is a better approximation of a low-pass filter: unlike in Laplacian smoothing, the low frequencies are not attenuated. In fact they are boosted slightly. A more intuitive

explanation is that it alternately shrinks and expands the model. Thus, the total shrinkage is far less than in plain Laplacian smoothing

3.1.2 Spectral Smoothing

As discussed in [KCVS98], [LZ10], [VL08] and [ZHe11], smoothing attenuates the high frequency details in the mesh, but it is difficult to filter out some frequencies while leaving other frequencies unscathed using methods such as Laplacian or Taubin smoothing. Thus, methods which allow us to deal more precisely with the frequency content of meshes are of great interest. As observed by Taubin, the basis functions of the Fourier transform are the eigenfunctions of the Laplace operator [Tau95]. On a finite discrete grid, we can represent the Laplace operator as a matrix, and the eigenvectors of this matrix form a basis in which we can represent functions on the grid. This generalizes to triangle meshes where we would use the eigenvectors of the Laplace–Beltrami operator as our basis. Thus, we need to express the Laplace– Beltrami operator.

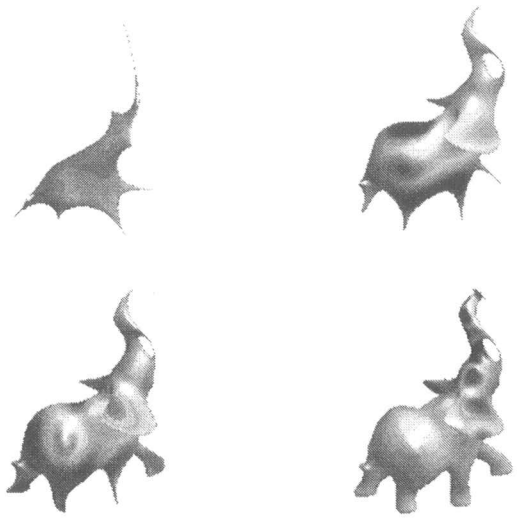


Figure 3.1 Reconstructing the Elephant mesh using different number of eigenvectors.

3.1.3 Feature-Preserving Smoothing

The method proposed here is related to Taubin's approach in that it also consists of two phases: face normal smoothing and subsequent vertex position update using the LSE method. However, instead of rotating the face normal based on a weighted sum of its neighborhood, we compute the smoothed face normal as the fuzzy vector median (FVM). Median based filters have been widely used in signal processing, due to their ability to reject outliers while preserving visually important cues, such as fine structures, edges, and monotonic regions. Fuzzy ordering theory allows for more effective filtering while better preserving desired features.

All of the methods discussed so far indiscriminately attenuate high frequency content whether it corresponds to noise or to actual fine details such as corners and edges. An effective method for feature-preserving smoothing, known as FVM (Fuzzy Vector Median) filtering, is due to Shen and Barner [SB04]. Their approach is to perform first a filtering of the face normals and to then fit the mesh to this new set of normals. Naively taking the average of a face normal and the normals of adjacent faces would not preserve features. Instead, the authors propose to use a median filter. Initially, for each face, f, we form a set of faces which are incident on f. The median normal is the face normal with the smallest angle to the other normals in that set. We obtain the smoothed normal of face f by computing the weighted average of all the normal in the incident face set, where the weight is computed using a Gaußian function of the angular distance to the median normal.

In the second step, the vertices are moved so as to minimize the difference between the filtered normal and the actual face normal [SB04].

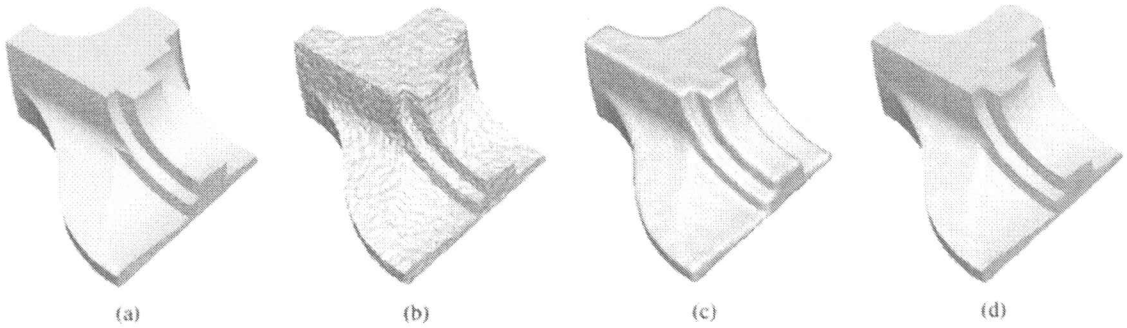


Figure 3.2 (a) original mesh, (b) noise added, (c) Taubin smoothing, (d) FVM smoothing [SE03]

3.1.4 Varitional Subdivision

In engineering applications it is often desirable to represent surfaces as the outer skin of a solid object. Such objects are typically constructed by combinations of simple basic shapes like spheres, cones and boxes (Constructive Solid Geometry). However, the lacking shape flexibility of CSG objects makes this technique inappropriate for sophisticated freeform modeling applications [Kob00].

The two major difficulties with discrete and explicit surface representations are the lacking infinitesimal smoothness and the high complexity. Polygonal models with several millions of triangles have become commonplace since moderately priced 3D scanning devices are available. In order to be able to handle the complexity of such meshes on standard PCs and workstations we have to apply mesh decimation algorithms which reduce the number of triangles in a given model while minimally changing its geometric shape. As a byproduct such algorithms also generate hierarchical representations for highly detailed meshes and thus enable to dynamically adapt the level of detail to the available hardware resources and the application-dependent quality requirements.

In the next section we will present a more precise definition of discrete curvature. Smooth meshes are then characterized by low discrete curvature. The operators by which

curvatures can be computed on triangle meshes lead to simple filter algorithms that improve the smoothness of an existing mesh by moving the vertices in order to minimize the discrete curvature or its variation. Applying such discrete fairing algorithms on different levels of a hierarchical mesh representation significantly accelerates their convergence.

In the preceding sections, we have thought of smoothing mostly as a form of lowpass filtering, but we can also see it as a diffusion process or as energy minimization which is the outlook we will adopt in the rest of this chapter: the surface is changed to reduce some energy, and if we combine this type of energy minimizing smoothing with refinement of the mesh, we obtain what has been called variational subdivision [Kob00].

Using variational subdivision, we iteratively place new vertices on the midpoints of edges to introduce more detail and then move these new vertices to minimize energy. Since we want the mesh to interpolate the original points, these are never moved—only inserted points are moved. This means that the method is relatively indifferent to the size of the input, but is very sensitive to the size of the output since the amount of work done is proportional to the number of output vertices in the final triangle mesh.

In this section, we discussed the situation when a fine triangle mesh is given and discrete fairing techniques are applied to improve its quality. A rather different setup in freeform modeling is the scattered data interpolation problem where only few points in space are given and a smooth interpolating surface is sought. The topology of the interpolating surface is usually part of the input data. Hence the given data points usually come as the vertices of a coarse triangle mesh and we want to generate a refined mesh with the same topology which interpolates or approximates the original points.

3.1.4.1 Subdivision operators

The most effective technique in the context of geometric modeling with polygonal meshes is subdivision schemes. A subdivision scheme is given by a set of rules to refine a polygonal control mesh. By iteratively applying these rules, we generate a sequence of

meshes which eventually converge to a smooth limit surface. This technique is very similar to the knot-insertion operation for spline surfaces. Inserting a knot-line into the spline representation of a surface requires the computation of new control vertices according to simple rules (linear combinations of old control vertices). It is well-known that the control meshes converge to the spline surface if the knot-lines eventually become dense.

Subdivision schemes are a generalization of knot-insertion in the sense that the refinement rules of a subdivision scheme can be applied to arbitrary meshes, i.e., the tensor-product regularity of spline control meshes is not required. This enables the generation of surfaces with arbitrary topology and not only triangular or quadrilateral patches.

The subdivision operator always consists of two parts. The first is a topological split operation by which new control vertices are inserted into the mesh. The split operation is chosen uniform such that all new vertices are regular and the number of extraordinary vertices in the refined meshes is constant (subdivision connectivity). The second part is a smoothing operator which moves the control vertices according to weighted averages of neighboring vertices. The weights of the smoothing rule usually depend on the valence of the vertices only.

There are two major classes of subdivision schemes. One class is the interpolatory schemes which do not change the position of the old vertices (and hence the refined mesh always interpolates the coarser one). For interpolatory schemes, the limit surface interpolates all intermediate control vertices. The other class is the noninterpolatory schemes where all vertices are shifted by the smoothing operator. In this case the limit surface only approximates the control vertices (cf. cubic spline curves and their control polygon).

We can combine the two techniques, i.e., the high quality surface generation by energy minimization (variational modeling, fairing) and the generation of surfaces with arbitrary topology by subdivision schemes, in the following way: Instead of using the fixed smoothing rules of stationary subdivision schemes, we place the vertices in the refined mesh such that a global energy functional is minimized. Since otherwise the algorithmic structure of subdivision schemes is preserved, we call such algorithms: variational subdivision schemes.

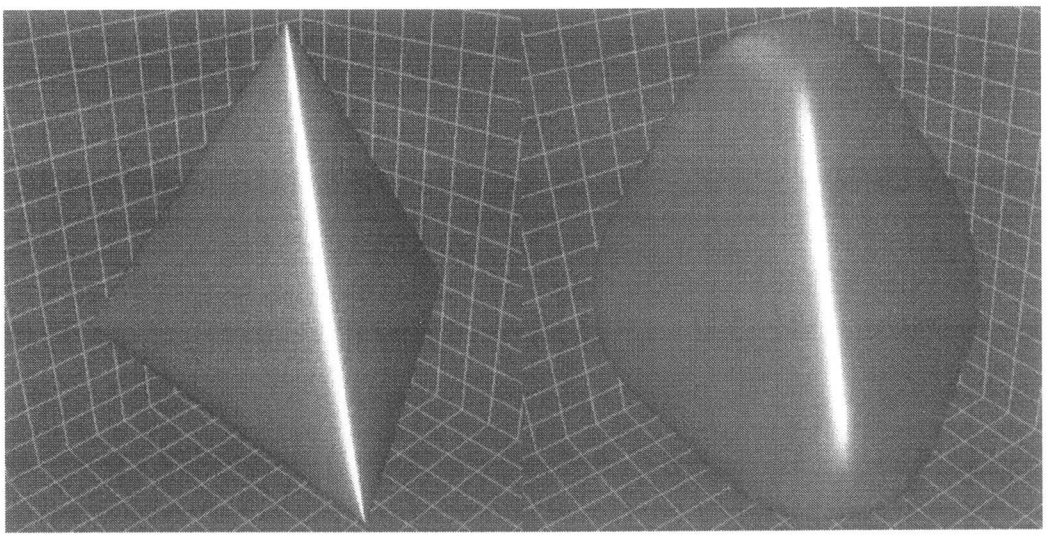


Figure 3.3 Subdivision schemes applied to a tetrahedron meshes [ZSS96]

3.2 Surface Representation

The following section references [BKPAL 10]. There are two classes of surface representations:

3.2.1 Parametric representation

Parametric surfaces are defined by a vector-valued parameterization function $f: \Omega \rightarrow S$ that maps a 2D parameter $\Omega \subset \Re^2$ domain to the surface $S = f(\Re^3)$

3.2.2 Isometric (Volumetric) surfaces

is defined to be a set of a scalar-valued function $F: \Re^3 \rightarrow \Re$

Metric Properties:

A continuous surface $S \subset \Re^3$ can be given in parametric form as:

$$X(u,v) = \begin{pmatrix} x(u,v) \\ y(u,v) \\ x(u,v) \end{pmatrix}, (u,v) \in \Omega \subset \Re^2 \tag{3.1}$$

Where x,y, and z are differentiable functions in u and v, and is the parameter domain. The scalars (u, v) are the coordinates in parameter space.

The metric of the surface is defined by the first derivative of the function X

$$X_u(u_o,v_o) := \frac{\partial X}{\partial u}(u_o,v_o) \quad \text{and} \quad X_v(u_o,v_o) := \frac{\partial X}{\partial v}(u_o,v_o) \tag{3.2}$$

The surface normal vector is orthogonal to both tangent vectors and can thus be computed as

$$n = \frac{X_u * X_v}{\|X_u * X_v\|} \tag{3.3}$$

In addition, we can define arbitrary directional derivatives of X. Given a direction vector $\bar{w} = (u_w, v_w)$ defined in parameter space, we consider the straight line parameterized by t passing through (u,v) and oriented by $\bar{w}$ given by $(u,v) = (u_o + v_o) + t\bar{w}$. The image of this straight line through X is the curve

$$C_w(t) = X(u_o + tu_w, v_o + tv_w) \tag{3.4}$$

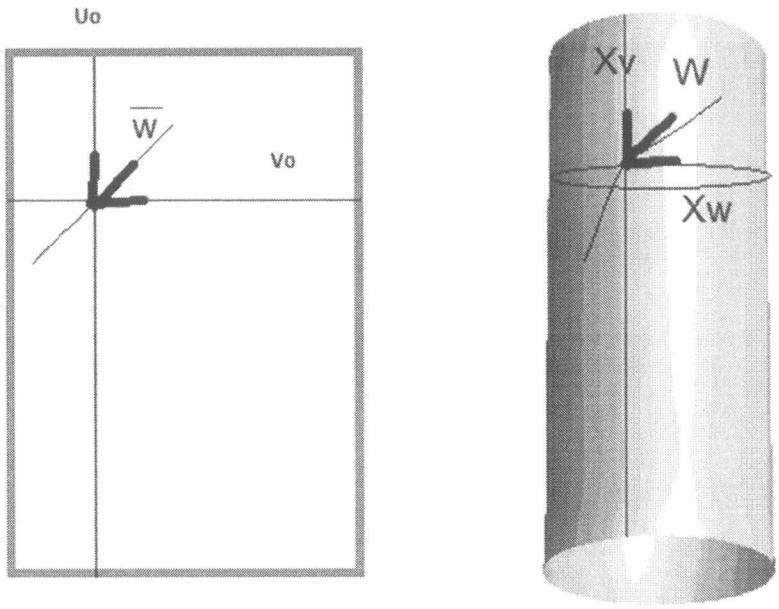


Figure 3.4: Transforming a vector $\overline{w}$ from parametric space into a tangent vector w of a surface

The Jacobian matrix of the parameterization function X corresponds to the linear map that transforms a vector $\overline{w}$ in parameter space into a tangent vector w on the surface

The directional derivative w of X at $(u_o + v_o)$ relative to the direction $\overline{w}$ defined to be the tangent to C_w at t = 0, given by

$$w = {}^{\partial C_w(t)}/_{\partial t} \qquad (3.5)$$

Applying the chain Rule it follows that $w = J\overline{w}$ where J is the Jacobian

$$J = \begin{bmatrix} \partial x/\partial u & \partial x/\partial v \\ \partial y/\partial u & \partial y/\partial v \\ \partial z/\partial u & \partial z/\partial v \end{bmatrix} = [X_u \quad X_v] \tag{3.6}$$

The Jacobian matrix encodes the metric of the surface in the sense that it allows measuring how angles, distances, and areas are transformed by the mapping from the parameter domain to the surface.

Let $\bar{w}_1$ and $\bar{w}_2$ be two unit vectors in parameter space. The cosine of the angle between these two vectors is given by the scalar product $\bar{w}_1^T \bar{w}_2$. The scalar product between the corresponding tangent vectors on the surface is given by

$$\bar{w}_1^T \bar{w}_2 = (J\bar{w}_1)^T (J\bar{w}_2) = \bar{w}_1 (J^T J) \bar{w}_2 \tag{3.7}$$

The matrix product $(J^T J)$ is known as the first fundamental form of X and it is written as:

Besides measuring angles, we can also determine the squared length of a tangent vector w as

$$I = J^T J = \begin{bmatrix} E & F \\ F & G \end{bmatrix} = \begin{bmatrix} X_u^T X_u & X_u^T X_v \\ X_u^T X_v & X_v^T X_v \end{bmatrix} \tag{3.8}$$

3.3 Surface Curvatures

The following section references [BKPAL 10]. Next we need to discuss curves on surfaces

Let $t = u_t X_u + v_t X_v$ be a tangent vector at a surface point $p \in S$ represented as a $\bar{t} = (u_t, v_t)^T$ in parameter space. The normal curvature $k_n(\bar{t})$ at p is the curvature of the planar curve created by intersecting the surface at p with the plane spanned by t and the surface normal n

$$k_n(\bar{t}) = \frac{\bar{t}^T И \bar{t}}{\bar{t}^T I \bar{t}} \tag{3.9}$$

Where И is the second fundamental form

$$И = \begin{bmatrix} e & f \\ f & g \end{bmatrix} = \begin{bmatrix} X_{uu}{}^T n & X_{uv}{}^T n \\ X_{uv}{}^T n & X_{vv}{}^T n \end{bmatrix} \tag{3.10}$$

Assuming $k_n(\bar{t})$ varies with $\bar{t}$ it can be shown that the rational quadratic function of the curvature equation has two distinct values, called the principal curvatures. We denote with k_1 the maximum curvature and withk_2 the minimum curvature.

If $k_1 != k_2$ we can identify two unique unit tangent vectors t_1 and t_2 are called the principle directions.

3.3.1 Euler theorem

An important theorem by Euler relates the normal curvature

to the principal curvatures:

$$k_n(\bar{t}) = k_1 \cos^2\varphi + k_2 \sin^2\varphi \tag{3.11}$$

Where φ is the angle between t and t_1

Euler's theorem also states that principal directions are always orthogonal to each other

The mean curvature H is defined as the average of the principal curvatures [MDSB03]:

$$H = \frac{k_1 + k_2}{2} \tag{3.12}$$

Laplace operator. The following chapters will make extensive use of the Laplace operator Δ and the Laplace-Beltrami operator Δ_S. In general, the Laplace operator is defined as the divergence of the gradient, i.e., $\Delta = \nabla . \nabla$. For a 2-parameter function $f(u, v)$ in Euclidean

space this second order differential operator can be written as the sum of second partial derivatives

The Laplace-Beltrami operator extends this concept to functions defined on surfaces. For a given function f defined on a manifold surface S, the Laplace-Beltrami operator evaluates to the mean curvature normal

$$\Delta_S = -2Hn \tag{3.13}$$

3.4 Discrete Differential Operators

The following section references [BKPAL 10]. Since polygonal meshes are piecewise linear surfaces. The following definitions of discrete differential operators are thus based on the assumption that meshes can be interpreted as piecewise linear approximations of smooth surfaces. The goal is then to compute approximations of the differential properties of this underlying surface directly from the mesh data.

Normal vectors for individual triangles $T = (x_i, x_j, x_k)$ can be computed as the normalized cross-product of two triangle edges:

$$n(T) = \frac{(x_j - x_i) * (x_k - x_i)}{\|(x_j - x_i) * (x_k - x_i)\|} \tag{3.14}$$

Computing vertex normal as spatial averages of normal vectors in a local one-ring neighborhood leads to a normalized weighted average of the (constant) normal vectors of incident triangles:

$$n(v) = \frac{\Sigma_{T \epsilon N_1(v)} n(T)}{\|\Sigma_{T \epsilon N_1(v)} n(T)\|} \tag{3.15}$$

3.4.1 Gradients

Since the Laplace-Beltrami operator is defined as the divergence of the gradient, we will first look at a suitable definition of the gradient of a function on a piecewise linear triangle mesh.

We assume a piecewise linear function f that is given at each mesh vertex as $f(v_i) = f(x_i) = f(u_i) = f_i$ and interpolated linearly within each triangle(x_i, x_j, x_k) :

$$f(u) = f_i B_i(u) + f_j B_j(u) + f_k B_k(u) \tag{3.16}$$

Were $u = (u, v)$ is the parameter pair corresponding to the surface point x in a 2D conformal parameterization induced by the triangle.

The gradient of f is given as

$$\nabla f(u) = f_i \nabla B_i(u) + f_j \nabla B_j(u) + f_k \nabla B_k(u) \tag{3.17}$$

Since the basis functions satisfy the barycentric condition of partition of unity

$$B_i(u) + B_j(u) + B_k(u) = 1 \tag{3.18}$$

So

$$\nabla B_i(u) + \nabla B_j(u) + \nabla B_k(u) = 0 \tag{3.19}$$

Hence

$$\nabla f(u) = (f_j - f_i)\nabla B_j(u) + (f_k - f_i)\nabla B_k(u) \tag{3.20}$$

The gradient of B_j is therefore given as

$$\nabla B_i(u) = \frac{(x_k - x_j)^{\perp}}{2A_T} \tag{3.21}$$

Where $^{\perp}$ denotes a counterclockwise rotation by 90 degrees in the triangle plane and A_T is the area of triangle T. Consequently, the gradient of the piecewise linear function f within a triangle T evaluates to the constant

$$\nabla f(u) = (f_j - f_i)\frac{(x_i - x_k)^{\perp}}{2A_T} + (f_k - f_i)\frac{(x_j - x_i)^{\perp}}{2A_T} \tag{3.22}$$

3.4.2 Discrete Laplace-Beltrami Operator

Uniform Laplacian

$$\Delta f(v_i) = \frac{1}{|N_1(v_i)|} \Sigma_{v_j \epsilon N_1(v)} (f_j - f_i) \tag{3.23}$$

Where the sum is taken over all one-ring neighbors $v_j \epsilon N_1(v)$

Cotangent formula

A more accurate discretization of the Laplace-Beltrami operator can be derived using a mixed finite element/finite volume method

The goal is to integrate the divergence of the gradient of a piecewise linear function over a local averaging domain $A_i = A(v_i)$

To simplify the integration we make use of the divergence theorem for a vector-valued function

$$\int_{A_i} \text{div } F(u) dA = \int_{\partial A_i} F(u).n(u)\, ds \tag{3.24}$$

Applied to the Laplacian

$$\int_{A_i} \Delta f(u) dA = \int_{A_i} \text{div } \nabla f(u) dA = \int_{\partial A_i} \nabla f(u).n(u)\, ds \tag{3.25}$$

$\nabla f(x)$ is constant within each triangle, the integral for a triangle T evaluates to

$$\int_{\partial A_i \cap T} \nabla f(u).n(u)\, ds = 0.5\, \nabla f(u).(x_j - x_k) \tag{3.26}$$

$$\int_{\partial A_i \cap T} \nabla f(u).n(u)\, ds = (f_j - f_i) \frac{(x_i - x_k)^{\perp}.(x_j - x_k)}{4A_T} + (f_k - f_i) \frac{(x_j - x_i)^{\perp}.(x_j - x_k)}{4A_T} \tag{3.27}$$

$$\int_{\partial A_i \cap T} \nabla f(u).n(u)\, ds = .5\, (\cot\gamma_k (f_j - f_i) + \cot\gamma_j (f_k - f_i)) \tag{3.28}$$

From figure 3.5 we can see that the he Cotangent Laplacian produces delta vectors with normal components only, unlike the uniform Lapacian which has tangential components and may be non-zero on planer n-rings[Sor05]

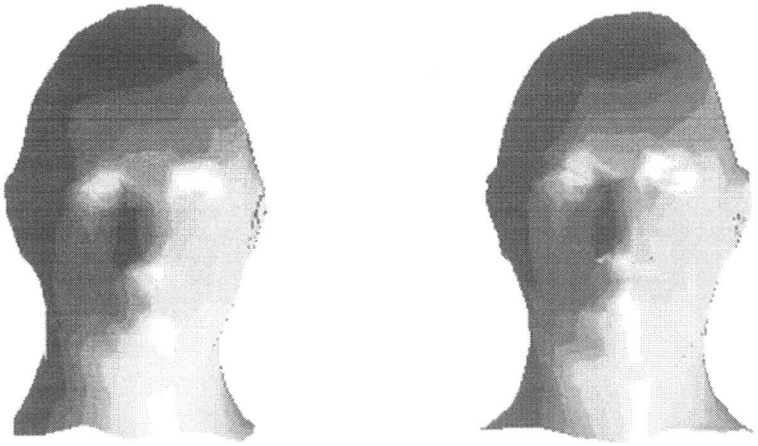


Figure 3.5: The difference between Uniform Laplacian (to the left) and Cotangent Laplacian (to the right)

3.5 Global Optimization with Mesh Smoothing

Mesh optimization methods are used in mechanical engineering applications areas such as solid and fluid dynamics, heat transfer, material science etc. The numerical investigation of these physical problems may require fine grained meshes over a single area of a physical model to resolve large solution variation. Even some commercial software still has problems with fine mesh generation, thus assuring effective and robust adaptive grid methods for such problems is still necessary. Currently, the majority of virtual reality applications use triangular meshes as their fundamental modeling and rendering is primitive. Such meshes can be the result of the modeling software, or may be an output of a scanning device. Their properties are often not considered as meshes with high quality. They have a non-ideal mesh elements and bad vertex connectivity.

With mesh optimization one can systematically achieve improved mesh quality that will provide a reliable analysis. The purpose of this chapter is to show effective mesh optimization technique that ensures better mesh structure and is applicable to general meshes. Improvements in mesh are obtained by just repositioning of vertices without changing connectivity. i.e., the position of the mesh vertices is modified, but the mesh topology remains unchanged. While internal mesh vertices are freely movable, external vertices must often remain fixed on boundaries. Moving (or shifting) vertices can have a drastic effect on the quality of a mesh and it is more efficient than refinement and collapsing vertices especially when the translation amplitudes are small. Such mesh relaxation can be regarded as mesh smoothing as it results in a visually pleasing mesh that follows some local or global rules.

[KKD2010] approach is based on the general theory of the Lagrange-Newton Optimization (LNO) applied to the finite element meshing problems. The LNO method is one of the iterative quadratic methods. In these methods, each iteration step includes a solution of a quadratic optimization problem. LNO is a non-linear optimization and originates from Sequential Quadratic Optimization (SQO). Comparable smoothing

methods are Laplacian. [KKD2010] smoothing technique differs from the above mentioned in several notable ways:

(a) It is not restricted to triangular meshes

(b) includes global and local optimization cost functions

(c) Preserves boundaries and can assure various geometrical constraints

(d) can be applied to 2D and 3D meshes as well as non-manifold ones

(e) cost function can be weighted and extended with additional functionals

(f) It pre-calculates average edge lengths at every iteration step.

This ensures very fast convergence and flexible vertex movement.

In practical implementation of general purpose LNO theory the input mesh structure is processed custom triangular and quad mesh grids . The source code is written in the Matlab program.

The motivation of the work presented is to propose and analyze reasonable cost functions and to compare how they perform in "benign" situations and to identify the most promising ones for application on difficult problems. This chapter considers only meshes in which the vertices are moved with a fixed connectivity with applied movement constraints on the boundary vertices.

3.5.1 Back Ground:

In a general optimization, the aim is to obtain the best result under given circumstances. The ultimate goal of an engineer's decisions is to minimize the effort required or to maximize the desired benefit. An optimization can be defined as a process of finding the conditions that give the maximum or the minimum value of a function. From Fig. 1 it can be seen that if the point x* corresponds to the minimum value of the function f(x), the same point also corresponds to the maximum value of the negative of the function.

Constrained optimization techniques can be classified into two main categories: the direct and the indirect methods. The constraints in the direct methods are handled in an explicit manner, whereas in most of the indirect methods, the constraint problem is solved in a sequence of unconstrained minimization problems. Our mesh optimization problem is an indirect optimization technique using the method of sequential quadratic programming.

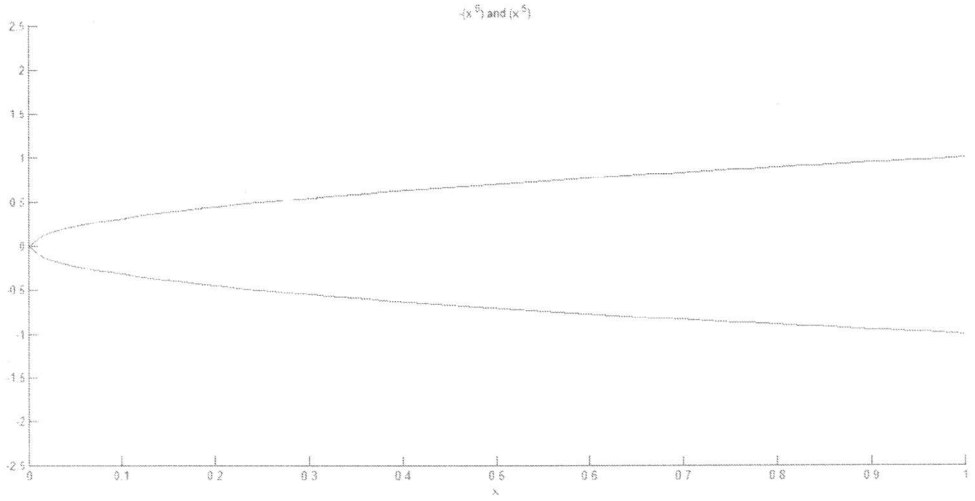


Fig. 3.6 The minimum of f(x) and the maximum of - f(x) are the same values of x*

Sequential Quadratic Optimization

The following derivation is presented by [KKD2010]. The SQO is considered as one of the best iterative optimization techniques. The method can be divided into two theoretical bases: (i) a set of nonlinear equations is solved by using the Newton's method, and (ii) a Lagrangian function formed with Kuhn-Tucker conditions. Find the solution of:

$$x^* = \arg\min_{x \in H} f(x),$$

$$H = \{x \in R^n | c(x) = 0\}.$$

Here c_i is the i^{th} component of a constrained function $c : R^n \rightarrow R^r$. Now the Lagrange's function can be written:

$$L(x, \lambda) = f(x) - \lambda^T . c(x),$$

with the gradient

$$L'(x, \lambda) = \begin{bmatrix} L'_x(x, \lambda) \\ L'_\lambda(x, \lambda) \end{bmatrix} = \begin{bmatrix} f'(x) - j_c^T \lambda \\ -c(x) \end{bmatrix},$$

and the Jacobian matrix of the constraint c is:

$$(J_c)_{ij} = \frac{\partial c_i}{\partial x_j}(x).$$

At the stationary point x_s is $L'(x_s, \lambda_s) = 0$, which satisfies the constraints and Kuhn-Tucker manifold ones, (e) cost function can be weighted and thus, a non-linear system of equations is obtained:

Find (x^*, λ^*) to satisfied $L'(x_s, \lambda_s) = 0$.

To solve this problem, the Newton-Raphson's method can be used. In each iteration step, the next iterate solution is found as $(x+h, \lambda+ \eta)$. Each step is determined by $L''(x,\lambda)\ [h\ \eta]T = -L'(x,\lambda)$, with:

$$L'' = \begin{bmatrix} L''_{xx} & L''_{x\lambda} \\ L''_{\lambda x} & L''_{\lambda\lambda} \end{bmatrix} = \begin{bmatrix} W & -J_c^T \\ -J_c & 0 \end{bmatrix}$$

Where $W = L''_{xx}(x, \lambda)$ and

$$L''_{xx}(x, \lambda) = f''(x) - \sum_{i=0}^{r} \lambda_i c''_i(x).$$

When one Newton-Raphson step is solved by $x = x+h$, $\lambda=\lambda+ \eta$, ,and elimination of η:

$$\begin{bmatrix} W & -J_c^T \\ -J_c & 0 \end{bmatrix} \begin{bmatrix} h \\ \lambda \end{bmatrix} = - \begin{bmatrix} f'(x) \\ -c(x) \end{bmatrix} \qquad (3.29)$$

Eq. (3.29) gives the solution h and the corresponding Lagrange multiplier vector λ to the following problem:

Find $h = \mathrm{argmin}_{e \in H}\{q(h)\}$.

In the next step, a constant $q(h) = 0.5.\, h^T W h + f'(x)^T h$ is defined and constraints:

$$H_{lin} = \{h \in R^n | J_c h + c(x) = 0\}.$$

Lagrange Newton Optimization

LNO is a kind of Sequential Quadratic Optimization. The name Lagrange-Newton comes from the two steps: in the first one, a Lagrangian function is optimized followed by the second, the Newton step when a new solution achieved. Currently, it is considered as the most efficient method.

The LNO method includes a soft line search with a special type of the penalty function. The description with an update method for the Hessian matrix can be concluded. This makes the method a Quasi-Newton with a good final convergence without having to implement second derivatives.

First, a quadratic model q of a cost function in neighborhood of x is considered,

$$f(x, \delta) \approx q(\delta) = f(x) + \delta^T f'(x) + 0.5\delta^T W(x)\delta,$$

then, a feasible region is defined,

$$\tilde{H} = \left\{ \delta \in R^n \middle| \begin{matrix} d_i(\delta) = 0, & i = 1, \dots, r \\ d_i(\delta) \geq i, & i = r+1, \dots, m \end{matrix} \right\}$$

Corresponding to a linear model, the constraints:

$$c(x + \delta) \approx d(\delta) = c(x) + J_c(x)\delta.$$

First the step parameter α and matrix $W(x)$ need to be calculated.

Next, the step length alpha must be calculated. If h turns out to be too large, the quadratic model may be a poor approximation of the true variation of the cost function. Therefore, a soft line search is made (a so-called *exact penalty function*) described in Fradsen. For $\mu_i \geq |\lambda_i|$ is:

$$\pi(y,\mu) = f(y) + \sum_{i=1}^{r} \mu_i |c_i(x)| + \sum_{i=r+1}^{m} \mu_i |min\{0, c_i(y)\}| \tag{3.30}$$

The choice of the penalty factor is divided into two parts:

(i) the first iteration step $\mu \geq |\lambda|$, and

(ii) the later iteration steps:

$$\mu_i = max\{|\lambda_i|, 0.5(\mu_i + |\lambda_i|)\} \tag{3.31}$$

The linear approximation for $c_i(y)=c_i(x + \alpha h)$ is:

$$\pi(\alpha) \approx \psi(\alpha) =$$
$$f(x) + \alpha h^T f'(x) + \sum_{i=1}^{r} \mu_i |c_i(x) + \alpha h^T c_i'(x)| + \sum_{i=r+1}^{m} \mu_i |min\{0, c_i(x) + \alpha h^T c_i'(x)\}|.$$

The change in the gradient of Lagrange's function is:

$$y = L_x'(x_{new}, \lambda) - L_x'(x, \lambda) = f'(x_{new}) - f'(x) - (J_c(x_{new}) - J_c(x))^T \lambda,$$

where $x_{new} = x + \alpha h$.

In each iteration, the curvature condition, $y^T (x_{new}-x) > 0$ must be checked If the condition does not satisfy $W_{new} = W$, then for $u = Wh$:

$$W_{new} = W + \frac{1}{\alpha h^T y} y y^T - \frac{1}{h^T u} u u^T$$

The whole procedure of the Lagrange Newton method is repeated until the stop criterion is satisfied. For $x = x_{prev} + \alpha h$. the stop criterion is:

$$\eta(x,\lambda) = |q(\alpha h) - f(x)| + \left| \sum_{i \in \tau} \lambda_i |c_i(x)| \right| + \sum_{i \in M} |min\{0, c_i(x)\}|,$$

where τ is a set of active inequality and equality constraints, and M is a set of inactive inequality constraints.

3.6 COST FUNCTIONS DEFINITION

First, [KKD2010] introduces the definition of two optimization cost functions with the constraints: (i) local and (ii) global is given. The cost functions will be formed from mesh vertices. In our case, the functions will ensure similarity of mesh elements (equal triangles, quads, etc. in one 2D mesh structure) on the (i) local and (ii) global level. The most important constraint is that the vertices are moved with a fixed connectivity. That means the vertices must be connected with the same edges even after optimization.

3.6.1 Local Cost Function

In the first step the equation of the local average length is defined:

$$l_{(av,i)} = \frac{\sum_{j=0}^{p} \| v_{(i,j+1)} - v_{(i,j)} \|}{p}, for\ i = 1\ to\ n. \tag{3.33}$$

[KKD2010] begins with the vertex of the mesh structure $v_{i,j}$. Operator || || is the Euclidian norm operator. Indices i and j define vertex position in the mesh structure. Index i represents the number in the mesh structure vertices (i = 1 to n) and index j represents any mesh element of the mesh structure (j = 1 to m). $x_{i,k}$ (triangle, quad, etc.) are the edge lengths. The mesh elements in Fig. 2.3 are represented by indices i and k. Index k represents the k^{th} edge of the local mesh element (k = 1 to p). There are three constant values for the input mesh structure: (i) number of all vertices n), (ii) number of all mesh elements m), and (iii) number of edges in one mesh element p).

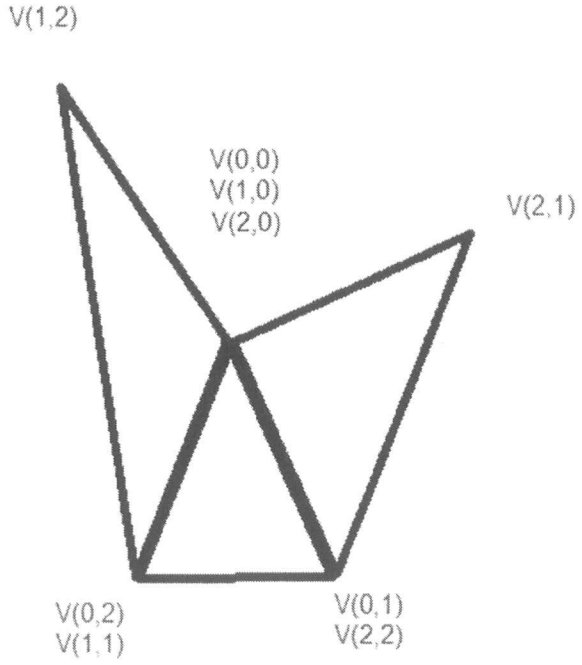


Fig. 3.8 Triangular mesh elements presentation

In the second step the local cost function is defined:

$$f_{local} = \sum_{i=0}^{m-1} \sum_{j=0}^{p-1} \left(\| v_{i,j+1} - v_{i,j} \| - l_{av,i} \right)^2$$

Eq. (3.33) presents the average local length of an input mesh element (triangle, quad, etc.).

3.6.2 Global Cost Function

In this cost function pre-calculated average edge lengths are used. The edge lengths are constant during iterations and can thus be considered as global. The global cost function is defined as:

$$f_{global} = \sum_{i=0}^{m-1} \sum_{j=0}^{p-1} \left(\left\| v_{i,j+1} - v_{i,j} \right\| - l_{av,gl} \right)^2$$

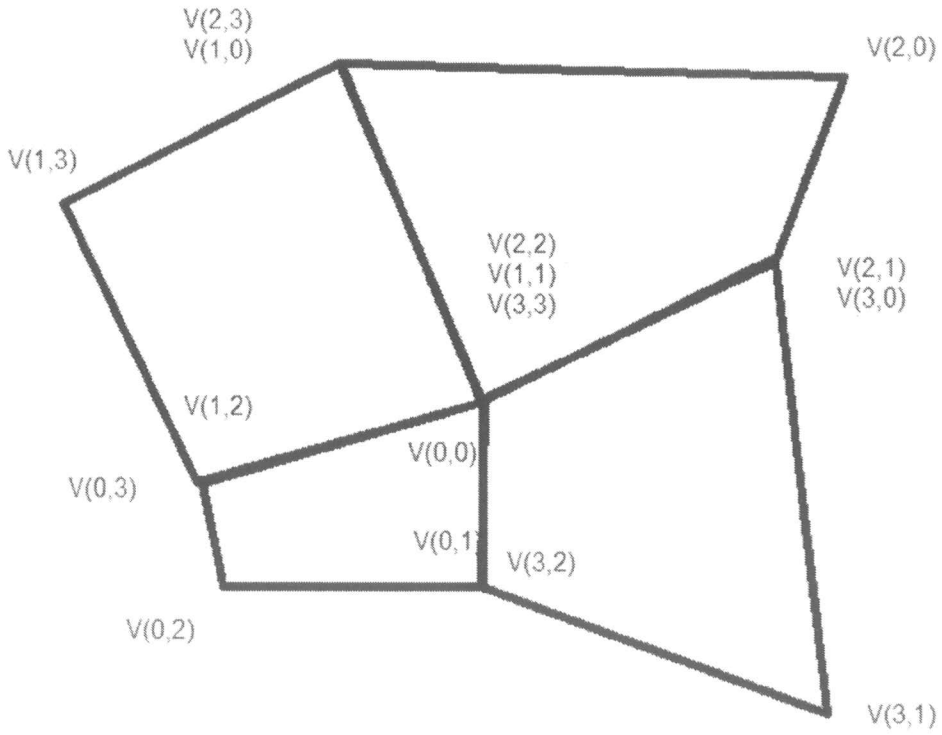


Fig. 3.9 Quadrilateral mesh elements presentation

with the global average length (see Fig. 3.4):

$$f_{av,gl} = \frac{\sum_{i=0}^{m-1} \sum_{j=0}^{p-1} \left\| v_{i+1,0} - v_{i,0} \right\|}{m + p}$$

The set of nonlinear equations is solved using the Lagrange-Newton optimization method. The LNO method searches the design vector solution $\mathbf{X} = (x_1, \ldots, x_n)^T$ which minimizes $f(x)$.

The design vector x_1 for $i = 1$ to n is a set of optimized vertex coordinates.

The most common smoothing method is Laplacian, where each vertex is moved to the centroid of its neighbors. Laplacian smoothing defines the number of adjacent vertices to vertex i with $N_i = \{ j \mid j$ shares an edge with $i\}$ averaged over number of vertices $\alpha_i = 1/|N_i|$, and the force between vertices $f(d) = 1$. Vertices thus always attract each other regardless of the distance.

The Lennard-Jones potential from chemistry describes attraction/repulsion behavior.

Its smoothing function is $f(d) = d^{-13} - d^{-4}$. This model suffers from numerical instabilities. The Pliant method with re-triangulation uses $f(d) = (1-d^4) \cdot \exp(-d^4)$. The goal of the method is to create meshes with normalized edge length of 1. Reasoning behind this smoothing function is that if two vertices are too close to each other ($d < 1$), they *repel*, and if they are too distant ($d > 1$), they *attract* each other.

In [KKD2010] smoothing optimization local (LCF) and global (GCF) cost function (Sec. 3.5 and 3.6) optimization is proposed. The main advantage of that optimization method is global mesh structure optimization. That means that all the vertices in mesh structure are repositioned in one iterations step. Other methods use just one vertex (diffusion) repositioning per iteration step.

With the local cost function vertices of mesh structure are moved to ensure equal edge lengths of faces. In contrast to other smoothing methods LCF method calculates average edge lengths in every iteration step for all faces. This ensures flexible vertices in the mesh structure.

For GCF [KKD2010] proposes mesh smoothing with constant average edges lengths which are calculated before smoothing begins. That optimization algorithm can ensure approximately equal external form as the initial mesh structure even after optimization without boundary constraints.

3.7 Results of applying global and local optimization using Matlab tools

The results of the optimization method with global cost function are applied to our data sets in Fig 3.5. The upper two shapes represent un-optimized meshes, while the lower two data sets show the final vertices positions after being subject to global optimization.

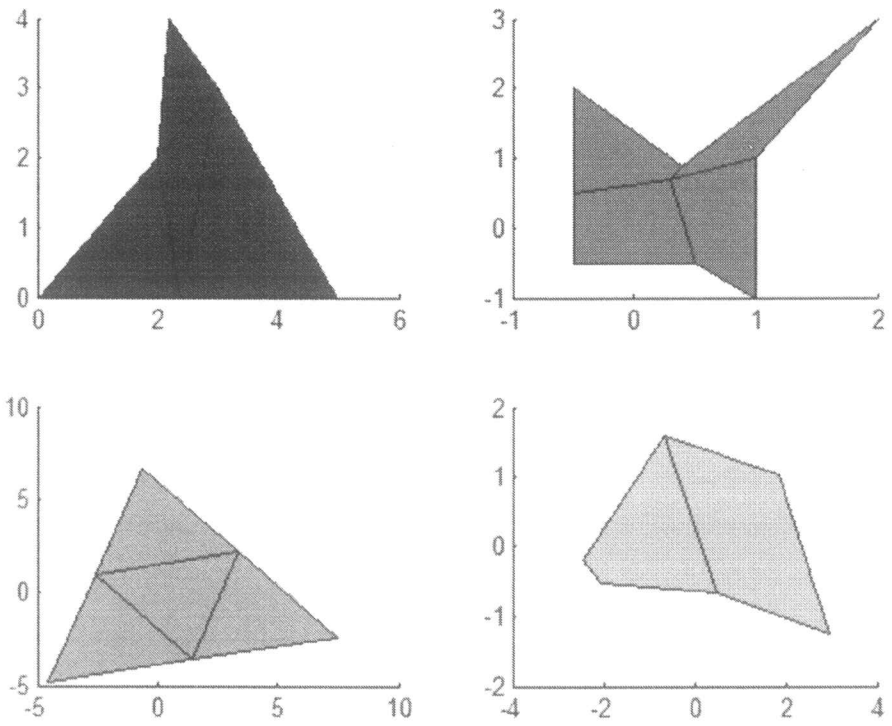


Figure 3.10 At the top is the original meshes, on the bottom we can find the optimized meshes or the lower section of the mesh in case the quad mesh.

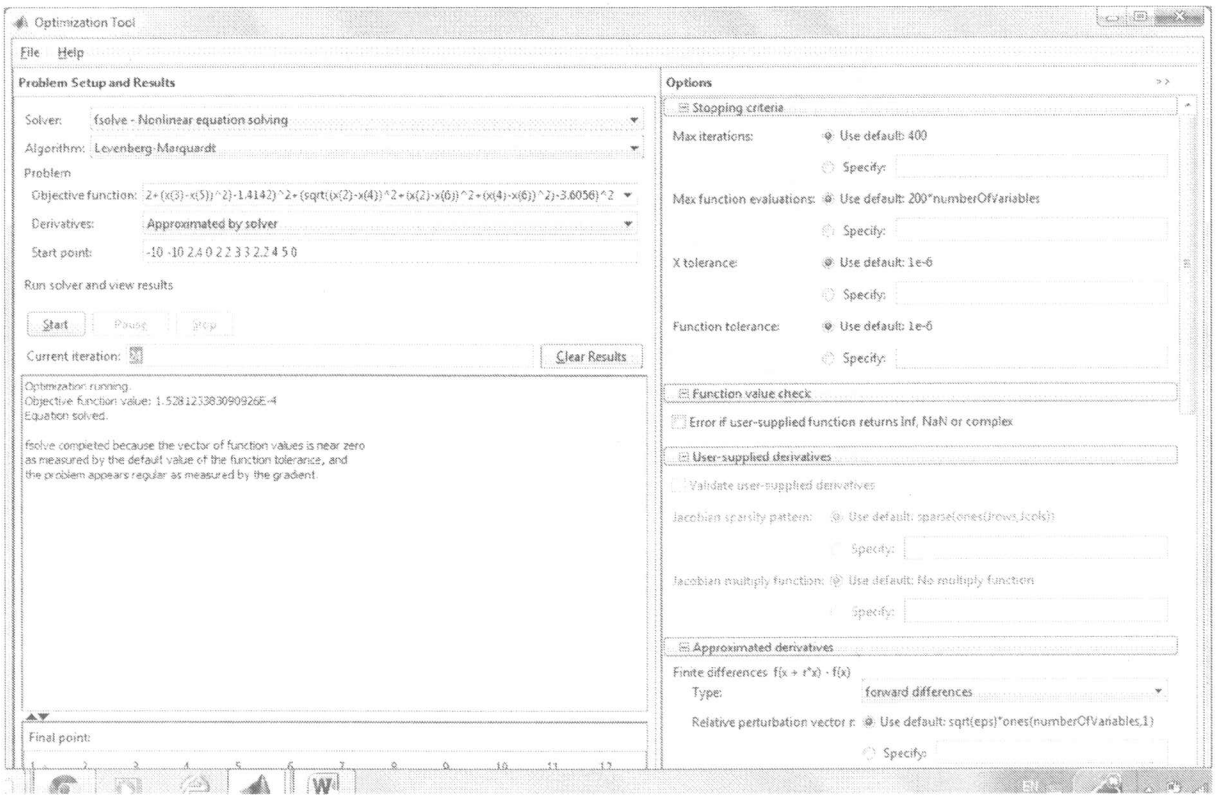


Fig. 3.12 Using Matlab optimization tool to achieve global optimization

The Levenberg-Marquardt algorithm was applied. We used Matlab Optimization tool. In case of the triangular mesh the resultant mesh structure shows four triangles that are equilateral. In case of the quadrilateral mesh, the choice of the start point affected greatly the resultant mesh after conversion. This is reasonable since the objective function is a function in 18 variables representing the x and the y positions of the vertices points which means that the start point if selected near minima that cause the optimizations function to drop below the minimum tolerance value, then global minima will not be reached. That is why the bottom right of figure 3.10 shows only the bottom section of the quadrilateral mesh. As can be seen the quadrilateral on the right is equidistant while the one to the left isn't which indicates that a global minima hasn't been reached.

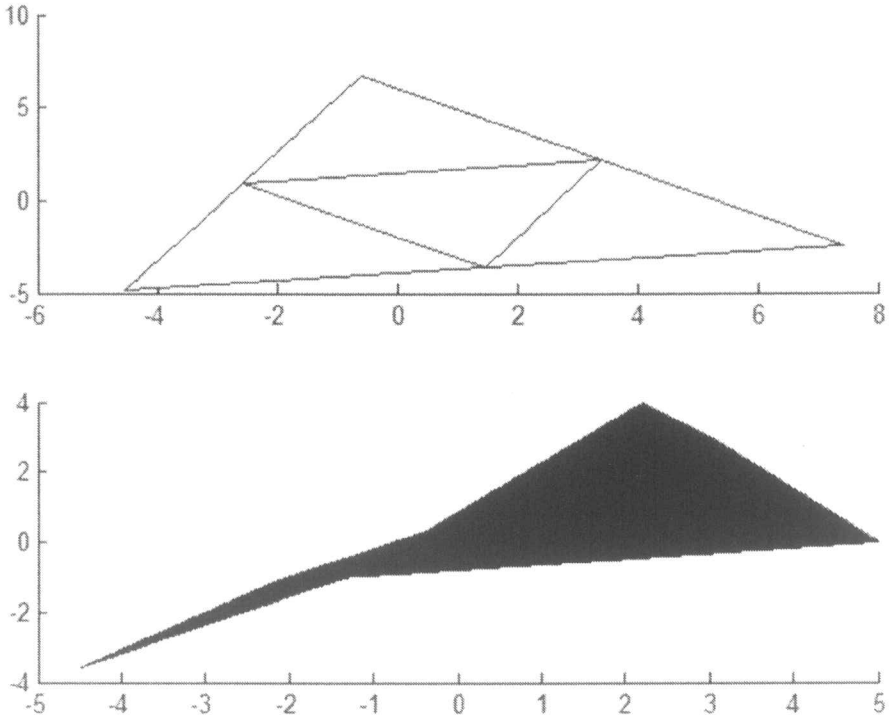


Fig. 3.13. Comparison between global optimization in the top of the figure and local optimization in the bottom of the figure.

Table 3.1: Initial Value for X and Y and their values after both local and global optimization

Initial X	X After Global optimization	X After local Optimization	Initial Y	Y After Global Optimization	Y After Local Optimization
-10	**-4.528**	**-4.476**	-10	**-4.812**	**-3.586**
2.4	**1.425**	**-2.184**	0	**-3.62**	**-1.103**
2	**-2.569**	**-1.286**	2	**0.964**	**-.966**
3	**3.411**	**3**	3	**2.53**	**3**
2.2	**-.601**	**2.2**	4	**6.739**	**4**
5	**7.434**	**5**	0	**-2424**	**0**

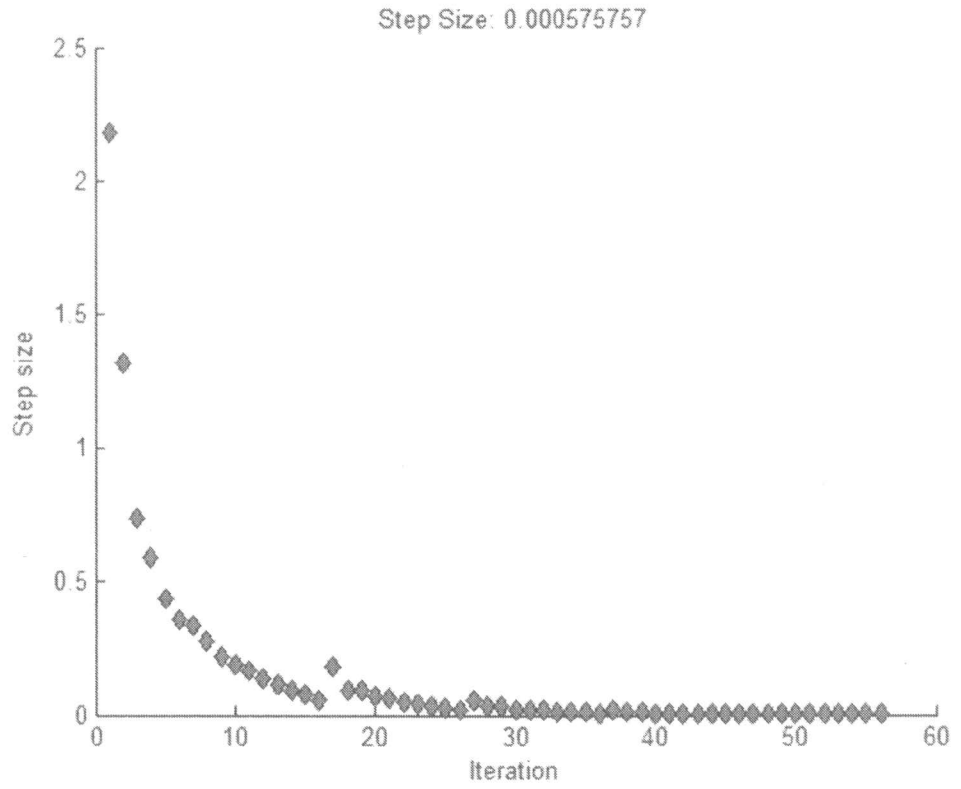


Fig. 3.13. Convergence of global cost function for quadrilateral mesh.

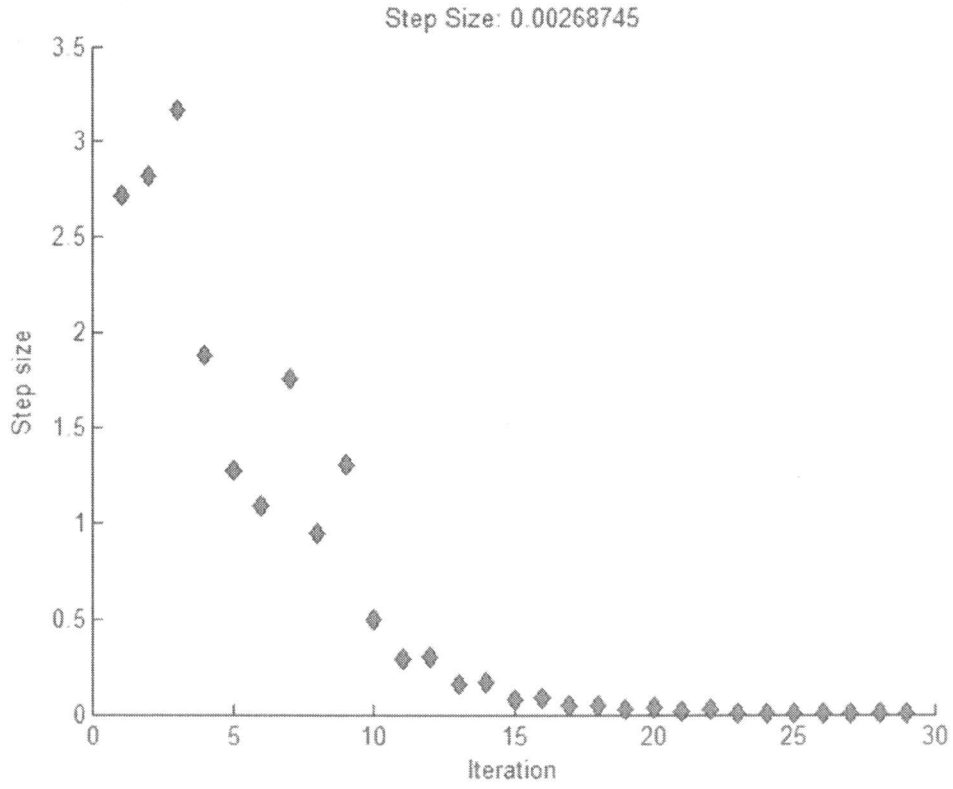


Fig. 3.14. Convergence global cost function of the triangular mesh.

As can be seen from figure Fig 3.13, and Figure 3.14 it took around 30 iterations for the triangular mesh while it took double that number for the quadrature mesh to converge; this is because the size of the variables that were involved in the quadrature mesh was larger: 18 variables in comparison to 12 variables in case of the triangular mesh.

C H A P T E R 4 Non Superfluous Time Solutions Of The Laplacian Framework

Non Superfluous Time Solutions Of The Laplacian Framework

In this chapter some techniques are going to be presented to solve the optimization and smoothing problem of the Laplacian framework in time saving methods.

We begin by some definitions of what is Geometric Processing is: Geometric processing can be defined as the field which is concerned with how geometric objects are worked upon with a computer [BGAA12].The word processing indicates that there is an algorithm involved in the processing action. The data part that the algorithm works on is the Geometry. On the other hand Geometry processing is mostly about applying algorithms to geometric models as stated by [BKPAL 10].

Surface representation and processing is one key topic in computer aided design. The surface representation of a 3D object may affect the information that we can perceive about that object. For example the triangular mesh representation can be used to: display the surface, deduce some topological information about the object, in addition to knowing the differential properties of the object that the model represents.

In this introduction the Laplacian Framework is presented by [Sor05]. This framework produces smoothed 3D models by optimization. This section is concluded by an example on how to build up the required matrices to get the solution.

In section 4.1 the time improvement results of our work to divide the Laplacian Framework into several smaller divided problems is shown. And we then solve the smaller problems and combine their solution to produce the complete solution in less time than solving the complete solution in one step.

In section 4.2 it is shown that if the model under investigation is perturbed we can use the solution of the original problem and the delta changes to get the solution of the perturbed problem.

In this chapter we build on work done on mesh processing and modeling that is based on the Laplacian framework and differential representations pointed out by [Sor05]

and [NISA06]. As opposing to dealing with Cartesian coordinates, the differential representation utilized in the Laplacian framework results in detail-preserving operations.

Laplacian operator and differential surface representation and surface reconstruction:

To understand the work that this chapter presents we will first talk about the Laplacian differential surface representation presented by [Sor05].

If " M " is a mesh representing an object with " V " vertices and "E " edges and " F " faces. For each vertex " $\mathbf{v_i}$" we have three Cartesian coordinated associated with each vertex: x_i, y_i , and z_i.

The differential or δ-coordinates of $\mathbf{v_i}$ is defined to be the difference between the absolute coordinates of $\mathbf{v_i}$ and the center of mass of its immediate neighbors in the mesh

$$\delta_i = \left(\delta_i^{(x)} + \delta_i^{(y)} + \delta_i^{(z)} \right) = \mathbf{v_i} - \frac{1}{d_i} \sum_{j \in N(i)} \mathbf{v_i} \qquad (4.1)$$

Where N(i) = { j |(i, j) ∈ E} and di = |N(i)| is the number of immediate neighbors of i (1-ring of the vertex). The transformation of the vector of absolute Cartesian coordinates to the vector of δ -coordinates can be represented in matrix form. Starting from the adjacency matrix A which is a square matrix that has 1 in the cell if both the row "i" and column "j" of the cell in the matrix represent an edge between the two vertices i, and j.

Also we have the diagonal matrix D that has $D_{ii} = d_i$, hence we can write the L matrix

$$L = I - D^{-1}A \qquad (4.2)$$

And the symmetric version L_s

$$L_s = DL = D - A \qquad (4.3)$$

Then we can write:

$$L\,x = \delta^x \qquad (4.4)$$

$$L\,y = \delta^y \qquad (4.5)$$

$$L\,z = \delta^z \qquad (4.6)$$

We cannot restore the Cartesian coordinates starting from δ –coordinates; because L is singular. In order to restore the Cartesian coordinates we need to specify the

Cartesian coordinates of one vertex to resolve the translational degree of freedom. Substituting the coordinates of vertex i is equivalent to dropping both the *i*th row and column from L, which makes the matrix invertible [Sor05]. Usually how this is done is by placing more than one special constraint of the mesh vertices. We have therefore |C| additional constraints (called the positional constraints) of the form:

$$\mathbf{v_j} = \mathbf{c_j}\ , \mathbf{j} \in C \tag{4.7}$$

If the vertices are ordered from 1 to m, then the linear system looks like:

$$\left(\frac{L}{\omega I_{m*m}|\mathbf{0}}\right)x = \begin{pmatrix}\delta^x \\ \omega c_{1:m}\end{pmatrix} \tag{4.8}$$

The additional constraints make the linear system over-determined (more equations than unknowns) and in general no exact solution may exist. However, the system is full-rank and thus has a unique solution in the least-squares sense [FMS07] and [Mey00]:

$$\check{x} = argmin_x\left(||Lx - \delta^x||^2 + \sum_{j\in C}\omega^2|x_j - c_j|\right) \tag{4.9}$$

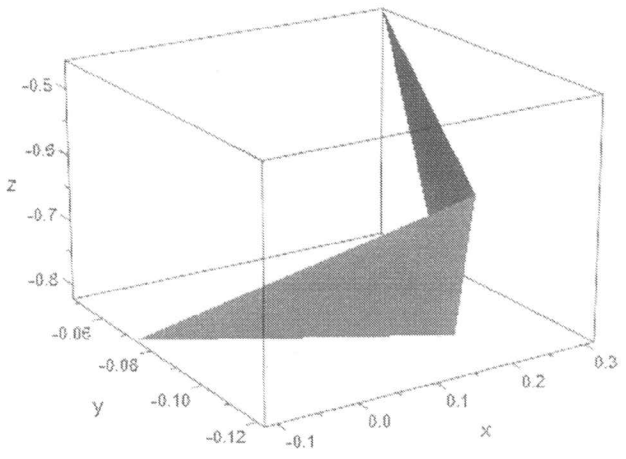
Equation 4.8 can be written on the following form

$$Ax = b \tag{4.10}$$

The algorithm to finding a solution can be written as:

Starting from Vertices and Faces
Calculate the valence ring for each vertex using the information stored in the faces
Calculate the adjacency matrix A which is a square matrix that has 1 in the cell if both the row "i" and column "j" of the cell in the matrix represent an edge between the two vertices i, and j
Calculate diagonal matrix D that has Dii =di
Calculate the diagonal matrix L and the Symmetrical matrix Ls using 3.2, and 3.3
Calculate the δ-coordinates of v_i using 3.1
Augment to L and δ the positional Constraints using 3.8 where m=n
Find the solution using least square method 3.9
The previous should be repeated for both y and z coordinates

In the following it is demonstrated how to get the equation to get the x coordinates of a mesh of two triangles, it also has four points and two faces



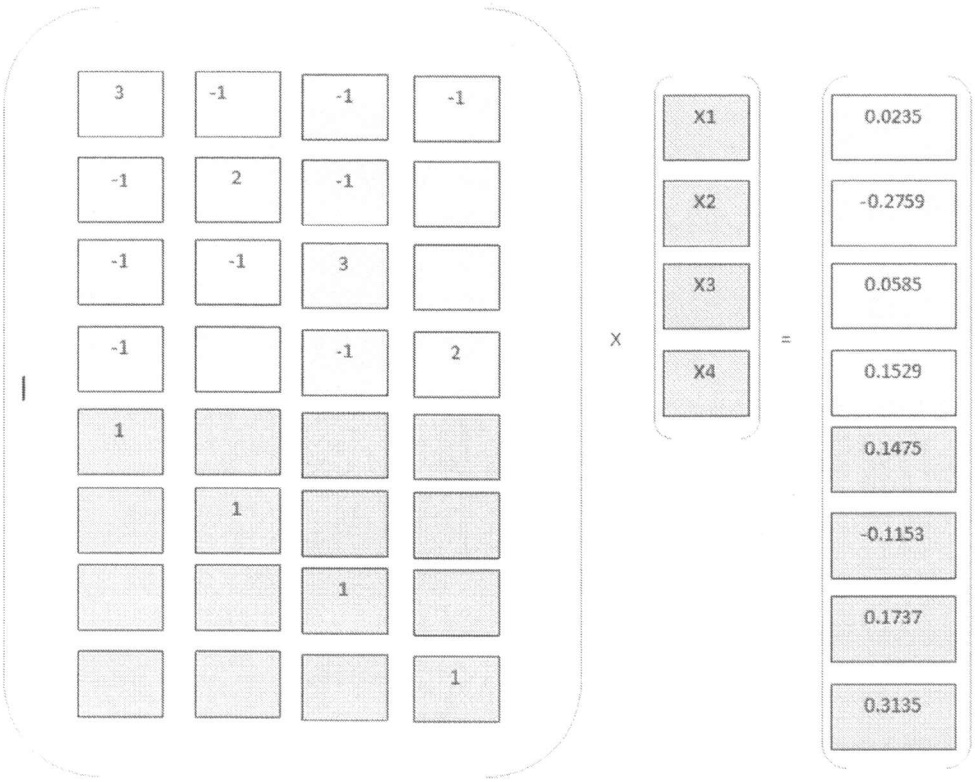


Figure 4.1: shows in the upper half a mesh composed of two faces and four points

In the lower part we see the Lapalacian Framework of such mesh

As can be seen from Figure 4.1 and with reference to equation 4.8

$$\left(\frac{L}{\omega I_{m*m}|\mathbf{0}}\right)x = \begin{pmatrix}\delta^x \\ \omega c_{1:m}\end{pmatrix}$$

Here we find that is L is a 4x4 matrix, $\omega = 1$, δ^x is a 4x1 matrix, m=4 .

This system will find the x axis coordinated of the smoothed system, we should repeat again for the y and the z coordinates.

4.1 Laplacian Mesh Processing Division

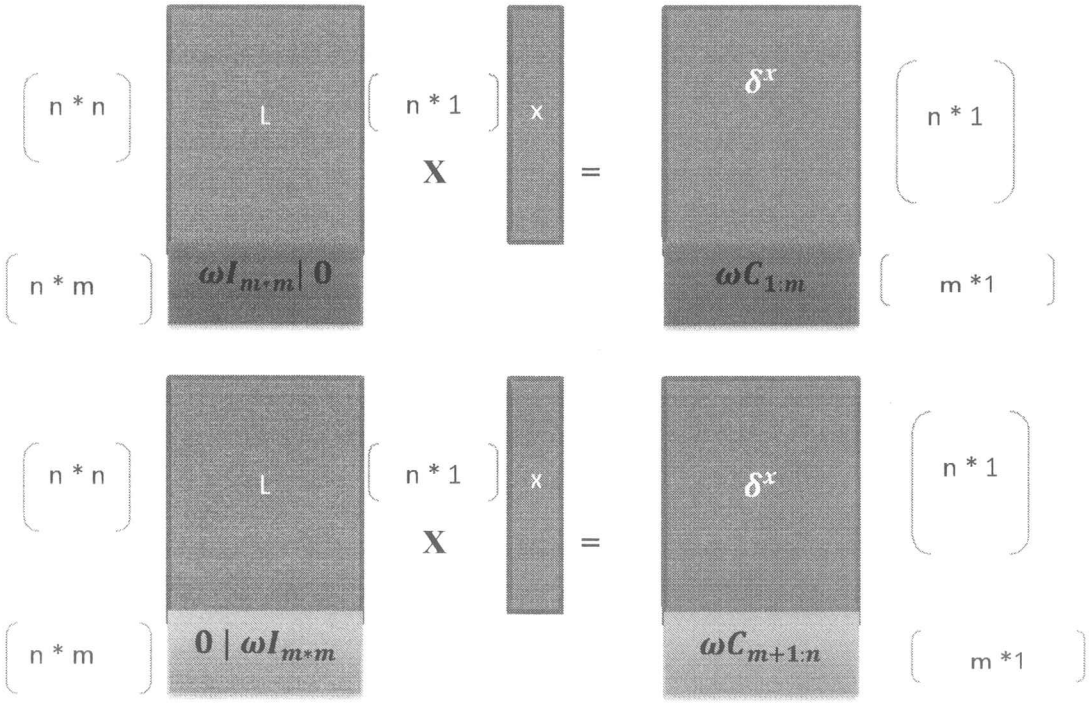


Figure 4.2: shows in the upper half solving the Laplacian framework with the first section of the positional constraints while the lower half is solving the Laplacian framework with the last section of positional constraints

4.1.1 Dividing the Linear problem Ax=b

Instead of solving the least squares problem $Ax = b$ we suggest to divide it into two steps as follows (we refer to Figure 4.2 with m=n/2):

1. We will first solve the upper linear system with, this will give us a solution x, which is composed of two parts, the upper half is a direct result from solving both the non-positional constraints and the positional constraints, the lower half comes only from non-positional constraints.
2. Next we solve the lower linear system, again the solution x is composed of two parts, the lower is a direct result from solving both the non-positional

constraints and the positional constraints, the upper half comes only from non-positional constraints.

3. Then, the final solution will be composed of the upper half from step one combined with the lower half from step two.

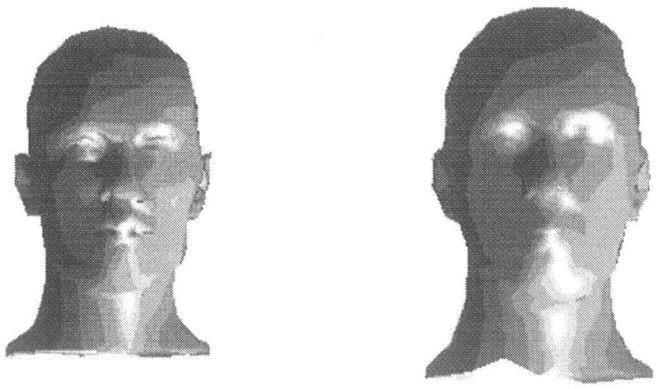


Figure 4.3: to the left the original object, to the right the resulting object after combining the two half problem solutions

Figure 4.3 shows the combined solution object. Of course each divided half takes less time to calculate than the complete problem in which we solve all the positional and non-positional constraints as once. In deed this is evident from [Hea97] who shows that the solution is proportional to size of the problem.

Table 4.1: Initial Value for X and Y and Z and their values after dividing the problem into two halves and applying Laplacian Framework

Original X	X after laplacian framework	Delta X	Original Y	Y after laplacian framework	Delta Y	OriginalZ	Z after laplacian framework	Delta Z
0.0919	0.0632	-0.0287	0.0384	0.0817	0.0433	0.0972	0.0259	-0.0713
-0.3246	-0.3485	-0.0239	0.1882	0.1972	0.0090	-0.1355	-0.1445	-0.0090
-0.4345	-0.4611	-0.0266	0.1602	0.1203	-0.0399	-0.6445	-0.5788	0.0657
-0.1054	-0.1206	-0.0152	0.0473	0.0964	0.0491	-0.1265	-0.1452	-0.0187
-0.2645	-0.3212	-0.0567	0.0253	0.0392	0.0139	-0.8881	-0.8569	0.0312
0.3254	0.3470	0.0216	-0.0421	-0.0066	0.0355	-1.0001	-1.0080	-0.0079
-0.1080	-0.0460	0.0620	0.0743	0.0740	-0.0003	0.2217	0.3272	0.1055

4.1.2 Parallel processing and Performance Improvements:

In the previous section we divided the problem into two halves; we could have assigned each half to a different processor and solved them in parallel.

We could extend this idea to N processors, were in each processor we solve a linear system composed of positional constraints in addition to non-positional constraints.

In this case we will have N-2 linear systems like Figure 4.3, in addition to the two linear systems in Figure 4.2. We will take from each X solution of these systems the part that corresponds to the "m" positional constraints used in it.

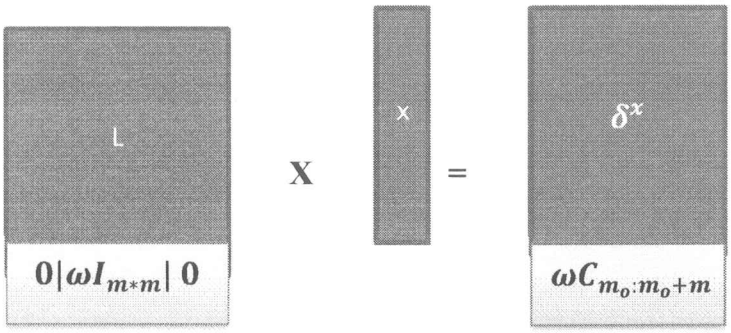


Figure 4.4: solving the Laplacian framework with a middle section of the positional constraints

Then we will construct the final solution as the upper part from the upper linear system in Figure 4.2, followed by N-2 parts from the linear systems mentioned in Figure 4.4, and finally the lower part from the lower linear system from Figure 4.2.

In case of using heterogeneous processors with different powers the divisions that we make need not be equal, and we can assign the more powerful processor a bigger size problem.

In Figure 4.5, we can see that to solve the problem into four parts, each time we solve the Laplacian framework with the non-positional constraints and one quarter of the positional constraints. As can be seen when we divide the problem on four processors the time is almost halved (max time=57% of complete solution).

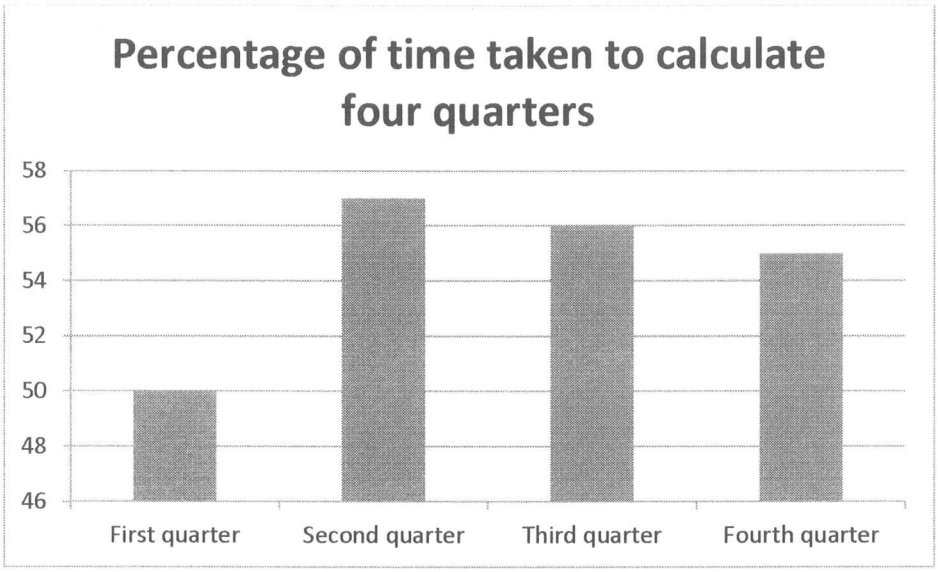


Figure 4.5: time taken after dividing the positional constraints into four parts

4.2 Least squares Perturbation and Laplacain Mesh Processing(The Effect of delta Changes on Relative Error)

4.1.2 The Effect of Changes in b on Relative Error

In this section we consider what if scenarios applied to equation (4.10). For example what if want to make changes on the model under investigation, the question that arises do we need to solve the new problem from the beginning. [Fas13] states that the liner system $Ax = b$ may be perturbed as $A(x + \delta x) = (b + \delta b)$ this implies that$A\delta x = \delta b$, and hence we can solve this linear equation to get δx and add to solution of the original problem and get $x + \delta x$, i.e the solution of the perturbed problem.

4.2.2 The Effect of Changes in A, and b on Relative Error

If we solve (4.8) without including all the positional constraints we will have a deformed object of the original object. So what if we want to add more constraints to our problem latter. Again the question arises: Do we need to solve a new problem from the beginning? The answer is no, we can work on the perturbed system $(A + \delta A)(x + \delta x) = (b + \delta b)$ to find:

$$\delta x = (A + \delta A)^{-1}(\delta b - \delta A x)$$

4.2.3 Implementation details and Results

We have used Matlab, and Graph tool box created by "Gabriel Peyre" [Pey04]. To calculate the least squares solution "mldivide" command was used. Due to the nature of the problem the solver used is Sparse QR Factorization via "SuiteSparseQR".

1. The Effect of Changes in b on Relative Error
   - The changes in b can result from multiple factors: aging that occurred on an object, or simply because we want to make a change on the object replacing a piece by another piece.
   - In figure 4.6 we see the effect of modifying part of the model
   - The Implementation details go as follows:
     (1) Solve the original problem $Ax = b\ to\ get\ x$
     (2) Introduce a delta change on the b part in a limited number of vertices (about 10% of the total number)
     (3) Calculate δb
     (4) Calculate δx from $\delta x = A^{-1}\delta b$
     (5) Add $x + \delta x$

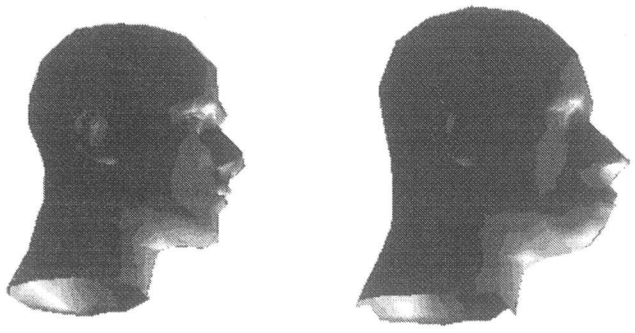


Figure 4.6: on the left the original object, on the right the perturbed object

Table 4.2: Initial Values for Z component and the values after introducing changes in b and applying Laplacian Framework

Original Z Component value	Z with 10% increase and laplacian framework	Delta Z
0.0244	0.0122	0.0122
0.3155	0.3433	-0.0278
0.0310	-0.0091	0.0402
0.0642	0.0831	-0.0189
0.0711	0.0533	0.0178
-0.0535	0.0321	-0.0856
0.3543	0.3924	-0.0381

2. The Effect of Changes in A, and b on Relative Error

- If we don't include all the positional constraints the solution will be deformed as can be seen in Figure 4.7. In this figure only half of the positional constraints were used.
- If we now create δA that has the remaining positional constraints and also δb.
- Note by that the size of δA is different from A, this also applies for b and δb, and we need to modify both the size of A and b.
- [HL95] and [HJ08] suggests a method to modify $A, and\ b$. What we need to do is to add rows to both $A, and\ b$ to represent the remaining positional constraints, so we can pretend that theses rows were actually in the original model and that they are filled with zeros.
- The Implementation details go as follows:
  (1) Solve the original problem $Ax = b$ to get x. Note by that this x is a result of only including half of the positional constraint
  (2) Create δb and δA that account for the remaining positional constraints

(3) Modify b and A of the original problem to have the perturbed problem size

(4) Calculate δx from $\delta x = (A + \delta A)^{-1}(\delta b - \delta A x)$

(5) Add $x + \delta x$ to get the solution

Figure 4.7: on the left the original object, on the right only half of the positional constraints were used

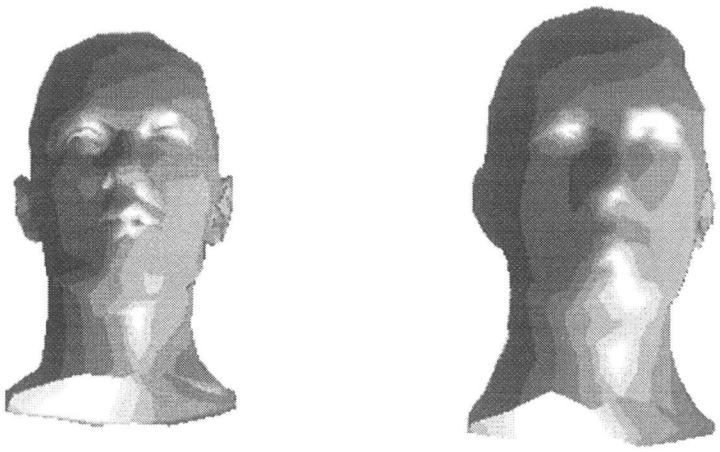


Figure 4.8: on the left the original object, the right image represents the solution after adding all the positional constraints and solving using the perturbation method

As can be seen from Figure 4.9 that there is always time saving –the execution time varies from run to sun but always less from solving the complete problem from scratch- if we calculate the solution using the previous method due to changes in (δA and δb) rather than solving a new problem with new A and b.

This sounds reasonable, since for small δA and δb the linear system is sparser, this means that having a smaller change leads to a faster solution.

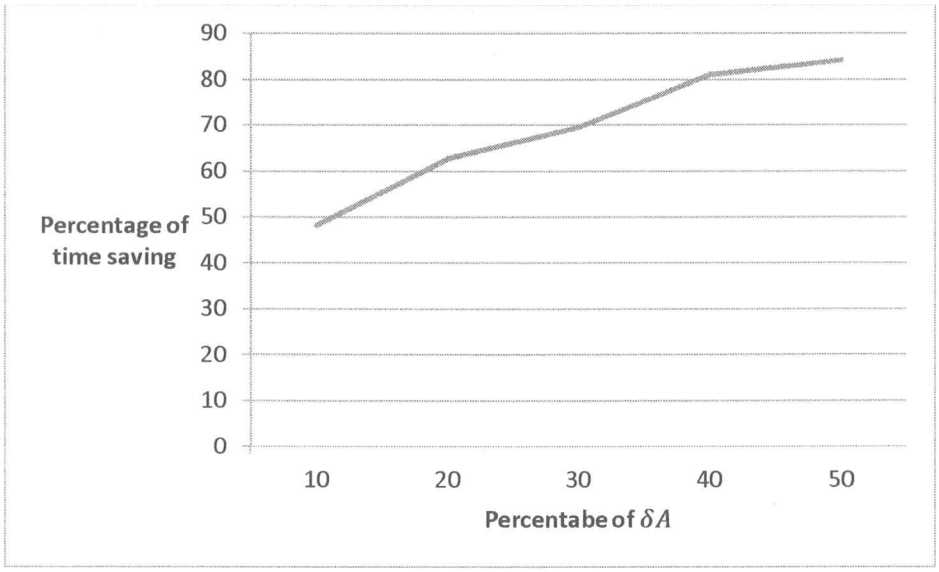


Figure 4.9: Percentage Time saving in calculating the Laplacian Framework solution when introducing additional delta changes as percentage of calculated solution

Summary:

We have presented two time saving techniques to find the solution of the Linear equation resulting from the Laplacian Framework. It was shown that time execution drops dramatically when dividing the problem to multiple processors. Also we can find solutions to delta changes of the linear system in less time required to solve a new problem.

CHAPTER 5 Conclusion and Future Work

Conclusion and Future Work

5.1 Conclusion

We can use 3D Imaging to protect breast Cancer Patients. We have tampered with female bust models to simulate breast cancer. We have using smoothing by optimization to make the models as realistic as possible. A neural network with a hidden layer has been trained to identify a new patient if ill or healthy. This thesis was a simulation effort not realistic data.

5.2 Future Work

We used generated models to represent the different models that were used in training of the neural network. Real case 3D models can be used to train the Neural Network. It then can be tested if our neural network will be able to discriminate between malignant breast cancer due to the growing habits and different growing rate and shape of the two tumors. We can test our neural network with real scanned models of breast cancer and healthy patients or patients having benign breast tumors instead of the simulation that we have proposed.

We achieved smoothing using the Laplacian Framework local optimization. Future work can compare the usage of global optimization techniques to achieve smoothing of 3D models to remove the noise and generate better 3D models.

Bibliography

[1] [BGAA12] Baerentzen J., Gravesen J., Anton F., Aanaes H.: Guide to Computational Geometry Processing. Springer (2012)

[2] [BKPAL 10] Botsch M., Kobbelt L., Pauly M., Alliez P., Levy B.: Polygon Mesh Processing. AK Peters,Wellesley (2010)

[3] [DBJH 14]Demuth H, Beale M, Jesús O., Hagan M.: Neural Network design

[4] [DMSB99] Desbrun, M., Meyer, M., Schröder, P., Barr, A.H.: Implicit fairing of irregular meshes using diffusion and curvature flow. In: SIGGRAPH'99: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, pp. 317–324. ACM Press, New York(1999).

[6] [Fas13] Fasshauer G.: online notes for course MATH 477/577 at IIT

[7] [FMS07] Ferris M., Mangasarian O., Wright S.: Linear Programming With MATLAB. Society for Industrial and Applied Mathematics (SIAM)

[8] [GKTTLMC06] Gervasi, O., Kumar, V., Tan, C.J.K., Taniar, D., Laganà, A., Mun, Y., Choo, H.: Computational Science and Its Applications - ICCSA 2006, International Conference, Glasgow, UK, May 8-11, 2006, Proceedings, Part II

[9] [HDBD14] Hagan M, Demuth H., Beale M., De Jesús O.: Neural Network Design(2nd Edition), hagan.okstate.edu/NNDesign.pdf

[10] [Hea97] Heath M.: Scientific Computing An Introductory Survey. McGraw-Hill (1997)

[11] [HJ08] Heaton J.: Introduction to Neural Network for Java. Heaton Research (2008)

[12] [HL95] Hillier F., Lieberman G.: Introduction to Operations Research. McGraw hill (1995)

[13] [HM01] Herniter M.: Programming in MATLAB. BROOKs/COLE

[14] [KCVS98] Kobbelt, L., Campagna, S., Vorsatz, J., Seidel, H.-P.: Interactive multi-resolution modeling on arbitrary meshes. In: ACM SIGGRAPH'98 Proceedings, pp. 105–114 (1998)

[15] [KKD2010] Simon Kulovec – Leon Kos – Jožef Duhovnik, Mesh Smoothing with Global Optimization under Constraints. Journal of Mechanical Engineering 57(2011)7-8, 555-567

[16] [Kob00] Kobbelt, L.P.: Discrete fairing and variational subdivision for freeform surface design. Vis. Comput. 16, 142–158 (2000)

[17] [LCJF12] Jing Li, Ji-hang Cheng, Jing-yuan Shi, Fei Huang: Brief Introduction of Back Propagation (BP) Neural Network Algorithm and Its Improvement. Advances in Intelligent and Soft Computing Volume 169, 2012, pp 553-558

[18] [LZ10] Levy, B., Zhang, R.H.: Spectral geometry processing. In: ACM SIGGRAPH Course Notes (2010)

[19] [MDSB03] Meyer, M., Desbrun, M., Schröder, P., Barr, A.H.: Discrete differential-geometry operators for triangulated 2-manifolds. In: Hege, H.-C., Polthier, K. (eds.) Visualization and Mathematics III, pp. 35–57. Springer, Heidelberg (2003)

[20] [Mey00] Meyer C. Matrix Analysis and Applied Linear Algebra. SIAM (2000)

[21] [Mol93] Moller M.: A scaled conjugate gradient algorithm for fast supervised learning. Neural Networks Volume 6, Issue 4, 1993, Pages 525-533

[22] [NISA06] Nealen A., Igarashi T., Sorkine O., Alexa M.: Laplacian mesh optimization. Proceedings of ACM GRAPHITE (2006) , Pages 381-389

[23] [Pey04] Peyre G.: Toolbox Graph Mathworks (2004)

[24] [R02] Raida Z. : Modeling EM structures in the neural network toolbox of MATLAB, 2002, Antennas and Propagation Magazine, IEEE (Volume:44 , Issue: 6) Pages 46-67

[25] [SB04] Shen, Y., Barner, K.E.: Fuzzy vector median-based surface smoothing. IEEE Trans. Vis. Comput. Graph. 10(3), 252–265 (2004)

[26] [She05] Shewchuk: An Introduction to the Conjugate Gradient Method without the Agonizing Pain. Online Tutorial

[27] [Sor05] Sorkine O.: Laplacian Mesh Processing. Proceedings of EUROGRAPHICS 2005, STAR Volume. The annual conference of the European Association for Computer Graphics

[28] [Tau95] Gabriel Taubin, G.: A signal processing approach to fair surface design. In: ACM SIGGRAPH'95 Proceedings (1995)

[29] [TK09] Theodoridis S., Koutroumbas K. : Pattern Recognition. Elsevier Inc. 2009

[30] [VL08] Vallet, B., Lévy, B.: Spectral geometry processing with manifold harmonics. Comput. Graph. Forum 27(2), 251–260 (2008)

[31] [SE03] Yuzhong Shen Barner, K.E.: Surface denoising with directional fuzzy vector median filtering Multimedia and Expo, 2003. ICME '03. Proceedings. 2003 International Conference on

[32] [Zhe11] Zheng, Y., Fu, H., Au, O.K.-C., Tai, C.-L.: Bilateral normal filtering for mesh denoising. IEEE Trans. Vis. Comput. Graph. 17(10), 1521–1530 (2011).

[33] [ZSS96] Denis Zorin, Peter Schröder and Wim Sweldens, ``Interpolating subdivision for meshes with arbitrary topology," in Proceedings of SIGGRAPH 1996, ACM SIGGRAPH, 1996, pp. 189-192.

List Of Publications

1- [WAE14] Wahed M., Abdallah M., Elkomy M. : Non Superfluous Time Solutions of the Laplacian Framework. ijcsns 2014 Vol. 14 No. 5 pp. 8-13

2- [WAE14] Wahed M., Abdallah M., Elkomy M. : Simulation of the detection of noxious breast cancer. ijcsns 2014 Vol. 14 No. 9 pp. 1-4

APPENDIX

Code to Calculate the Laplacian Framework:

```
tic
name = 'C:\... \venus';
%mannequin';
%iphi_bad10k';
%beetle';
%nefertiti';
%iphi_fine100k';
options.name = name;
[vertex,face] = read_off([name '.off']);

alpha=.3;
nvert = size(vertex,2);
nface = size(face,2);

vring = compute_vertex_ring(face);
B= zeros(3,size(vertex,2));
Bd= zeros(3,size(vertex,2));
for i=1:nvert
    rings=size(vring{i}(:),1);
    ivertex=vertex(1:3,i);
    sum=[0.0;0.0;0.0];
    for j=1:rings
        sum=sum+vertex(1:3,vring{i}(j));
    end
    sum= sum./rings;

    vrtx=vertex(1:3,i);
    Bd(1:3,i)=vrtx;
    B(1:3,i)=vrtx-sum(1:3);
end
laplacian_type = 'conformal';
%laplacian_type = 'combinatorial';
%laplacian_type = 'distance';

options.symmetrize = 1;
options.normalize = 0;
L = compute_mesh_laplacian(vertex,face,laplacian_type,options);
L= vertcat(L,alpha.*eye([size(vertex,2) size(vertex,2)]));

B1= B(1:1,:);
B2= vertcat(B1',alpha.*Bd(1:1,:)');
dd=mldivide(L,B2);
```

```
E= B(2:2,:);
E2= vertcat(E',alpha.*Bd(2:2,:)');
ee=mldivide(L,E2);

F= B(3:3,:);
F2= vertcat(F',alpha.*Bd(3:3,:)');
ff=mldivide(L,F2);


r= zeros(3,nvert);
for i =1: nvert
r(1:1,i)=dd(i,1);
r(2:2,i)=ee(i,1);
r(3:3,i)=ff(i,1);
end

%laplacian_type = 'conformal';
laplacian_type = 'combinatorial';
%laplacian_type = 'distance';

options.symmetrize = 1;
options.normalize = 0;
L = compute_mesh_laplacian(vertex,face,laplacian_type,options);
L= vertcat(L,alpha.*eye([size(vertex,2) size(vertex,2)]));

B1= B(1:1,:);
B2= vertcat(B1',alpha.*Bd(1:1,:)');
dd=mldivide(L,B2);


E= B(2:2,:);
E2= vertcat(E',alpha.*Bd(2:2,:)');
ee=mldivide(L,E2);

F= B(3:3,:);
F2= vertcat(F',alpha.*Bd(3:3,:)');
ff=mldivide(L,F2);


r1= zeros(3,nvert);
for i =1: nvert
r1(1:1,i)=dd(i,1);
r1(2:2,i)=ee(i,1);
```

```
r1(3:3,i)=ff(i,1);
end



subplot(1,2,1);
options2.edge_color= .5;
plot_mesh(r1,face,options2);
shading interp; camlight; axis tight;
subplot(1,2,2);
options2.edge_color= .5;
plot_mesh(vertex,face,options2);
shading interp; camlight; axis tight;

toc
```

Code for Constructing the Beast Cancer models

```
tic
name =
'C:\Users\Nhassan\Documents\MATLAB\toolbox_graph\off\venus';

%venus';
%mannequin';
%iphi_bad10k';
%beetle';
%nefertiti';
%iphi_fine100k';
options.name = name;
[vertex,face] = read_off([name '.off']);
%[vertex2,face2] =
perform_mesh_simplification(vertex,face,size(vertex,2)/10);
%vertex= vertx2;
%face= face2;
alpha=5;
nvert = size(vertex,2);
nface = size(face,2);

vring = compute_vertex_ring(face);
B= zeros(3,size(vertex,2));
Bd= zeros(3,size(vertex,2));
for i=1:nvert
    rings=size(vring{i}(:),1);
    ivertex=vertex(1:3,i);
    sum=[0.0;0.0;0.0];
    for j=1:rings
        sum=sum+vertex(1:3,vring{i}(j));
    end
    sum= sum./rings;

    vrtx=vertex(1:3,i);
    Bd(1:3,i)=vrtx;
    %B(1:3,i)=vrtx-sum(1:3);
end


%laplacian_type = 'conformal';
laplacian_type = 'combinatorial';
%laplacian_type = 'distance';

options.symmetrize = 1;
```

```
options.normalize = 0;
L = compute_mesh_laplacian(vertex,face,laplacian_type,options);
L= vertcat(L,alpha.*eye([size(vertex,2) size(vertex,2)]));



%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%


zlocation=0;
maxvertex=zeros(3,1);
location =1;
for i=1:nface
    for j=1:3
        fa=face(j:j,i);
        pointofface=vertex(:,fa);
        if(pointofface(2)>0)
            if(pointofface(3)>zlocation)
                zlocation =pointofface(3);
                maxvertex=pointofface;
                location =fa;
            end
        end
    end

end



ptr=1;

fv=location-25;
lv=location+25;
alpha=[2 1.5 1 .7];



all=zeros(1,161);
er=1;
for i=1:nvert
    vrtx=vertex(1:3,i);
    if i<=location+100 && i>=location-60
        % brozat
```

```
%              vertex(3:3,i)=vertex(3:3,i)+.5;
%              vertex(2:2,i)=vertex(2:2,i)+.5;
%              vertex(1:1,i)=vertex(1:1,i)+.5;
        all(er)=i;
        er=er+1;
    end
end



co=4;   % cancer =4
foof{co*4}=[];
regions{co}=[];
k=0;
highpoint =0;
for i=1:co
     tr= [];
     for j=1:40
         if all(j+k)==location
             highpoint=i;
         end
         tr=[tr,all(j+k)];
     end
     k=k+40;
     regions{i}=tr;
end

B= zeros(3,size(vertex,2));
Bd= zeros(3,size(vertex,2));
for i=1:nvert
    rings=size(vring{i}(:),1);
    ivertex=vertex(1:3,i);
    sum=[0.0;0.0;0.0];
    for j=1:rings
        sum=sum+vertex(1:3,vring{i}(j));
    end
    sum= sum./rings;

    vrtx=vertex(1:3,i);

    Bd(1:3,i)=vrtx;
   % B(1:3,i)=vrtx-sum(1:3);

end
we=0;
```

```
for i=1:nvert
    vrtx=vertex(1:3,i);
    if distance(vrtx,maxvertex)<.7
        we=we+1;
    end
end

m=1;
idex=zeros(1,161);
Y={16};%%%%%%%%%%
OX={16};%%%%%%%%%%%%
u=1;
k=4;%%%%%%%%%%%%%%
for m=1:4
    for k=1:4  % cancer =4
        kolo=[];
        for i=1:nface
            if(any(regions{k}(:)==i))
                if any(kolo==face(1:1,i))
                else
                    kolo=[kolo,face(1:1,i)];
                end
                if any(kolo==face(2:2,i))
                else
                    kolo=[kolo,face(2:2,i)];
                end
                if any(kolo==face(3:3,i))
                else
                    kolo=[kolo,face(3:3,i)];
                end
            end
        end



        laplacian_type = 'combinatorial';

        options.symmetrize = 1;
        options.normalize = 0;
        L =
compute_mesh_laplacian(vertex,face,laplacian_type,options);
        L= vertcat(L,alpha(m).*eye([size(vertex,2)
size(vertex,2)]));

        B1= B(1:1,:);
        B2= vertcat(B1',alpha(m).*Bd(1:1,:)');
```

```
dd3=mldivide(L,B2);

db=zeros(1,nvert);
Bdd=zeros(1,nvert);

for i=1:nvert
    if(any(kolo==i))
         db(i)=B2(i)*.3;
         Bdd(i)=vertex(1:1,i)*.3;
    end
end

db=vertcat(db',alpha(m).*Bdd(1:1,:)');
dx=mldivide(L,db);
dd=dd3 ;%-dx;



E= B(2:2,:);
E2= vertcat(E',alpha(m).*Bd(2:2,:)');
ee=mldivide(L,E2);

de=zeros(1,nvert);
Bdd=zeros(1,nvert);
for i=1:nvert
    if(any(kolo==i))
         de(i)=E2(i)*.3;
         Bdd(i)=vertex(2:2,i)*.3;
    end
end

de=vertcat(de',alpha(m).*Bdd(1:1,:)');
dy=mldivide(L,de);

ee=ee ;%-dy;



F= B(3:3,:);
F2= vertcat(F',alpha(m).*Bd(3:3,:)');
ff=mldivide(L,F2);

db=zeros(1,nvert);
```

```
        Bdd=zeros(1,nvert);
        for i=1:nvert
            if(any(regions{k}(:)==i))
             db(i)=F2(i)*1;%%%%%%%%%%%%%%%%%%%%%%
             Bdd(i)=vertex(3:3,face(2:2,i))*1;
            end
        end

        db=vertcat(db',alpha(m).*Bdd(1:1,:)');
        dz=mldivide(L,db);
        ss=ff;
        ff=ff-dz;


      %%%%%%%%%%
%%%%%%%%%%%%%
        r1= zeros(3,nvert);
for i =1: nvert
r1(1:1,i)=dd(i,1);
r1(2:2,i)=ee(i,1);
r1(3:3,i)=ff(i,1);
end
options2.edge_color= .5;


%set(plot_mesh(vertex,face,options2),'FaceColor','blue','EdgeCol
or','black');


%%%%%%%%%%%%
      %%%%%%%%%%


        nvert = size(vertex,2);
        r= zeros(3,nvert);
        r2=zeros(1,nvert);
        for i =1: nvert
        r(1:1,i)=dd(i,1);
        r(2:2,i)=ee(i,1);
        r(3:3,i)=ff(i,1);
        end
        for i =1: nvert
        r2(1,i)=ss(i,1);
        end
        m1=1;
        TX=zeros(1,161);
        TX2=zeros(1,161);
```

```
        for i=1:nvert
            if(any(all==i))
                TX(1,m1)=r(3:3,i);
                TX2(1,m1)=r2(1:1,i);
                idex(1,m1)=i;
                m1=m1+1;
            end
        end
        ll=zeros(1,161);
        for i=1:161
            if k==1
              if i>=1 && i<=40
                  ll(1,i)=1;
              end
            end
            if k==2
              if i>=41 && i<=80
                  ll(1,i)=1;
              end
            end
            if k==3
              if i>=81 && i<=120
                  ll(1,i)=1;
              end
            end
            if k==4
              if i>=121
                  ll(1,i)=1;
              end
            end
        end
        Y{u}=TX;  %%%%%%%%%%%%
        OX{u}=ll;
        u=u+1;



        for i=1:nvert
            if(any(all==i))
                %foof{k+mod((m-1),4)}=[foof{k+mod((m-
1),4)},r(3:3,i)];
                 %foof{k+4*mod((m-1),4)}=[foof{k+4*mod((m-
1),4)},r(3:3,i)];
            end
        end
    end  %%%%
```

```
end
```

Code For Global optimization

```
function [x,resnorm,residual,exitflag,output,lambda,jacobian] =
test_tri(x0)
%% This is an auto generated MATLAB file from Optimization Tool.

%% Start with the default options
options = optimoptions('lsqnonlin');
%% Modify options setting
options = optimoptions(options,'Display', 'off');
options = optimoptions(options,'Algorithm', 'levenberg-
marquardt');
[x,resnorm,residual,exitflag,output,lambda,jacobian] = ...
lsqnonlin(@(x)(sqrt((x(1)-x(3))^2+(x(2)-x(4))^2)-
6.0974)^2+(sqrt((x(1)-x(5))^2+(x(2)-x(6))^2)-
6.0974)^2+(sqrt((x(3)-x(5))^2+(x(4)-x(6))^2)-
6.0974)^2+(sqrt((x(3)-x(5))^2+(x(4)-x(6))^2)-
6.0974)^2+(sqrt((x(3)-x(7))^2+(x(4)-x(8))^2)-
6.0974)^2+(sqrt((x(5)-x(7))^2+(x(6)-x(8))^2)-
6.0974)^2+(sqrt((x(5)-x(7))^2+(x(6)-x(8))^2)-
6.0974)^2+(sqrt((x(5)-x(9))^2+(x(6)-x(10))^2)-
6.0974)^2+(sqrt((x(7)-x(9))^2+(x(8)-x(10))^2)-
6.0974)^2+(sqrt((x(3)-x(11))^2+(x(4)-x(12))^2)-
6.0974)^2+(sqrt((x(3)-x(7))^2+(x(4)-x(8))^2)-
6.0974)^2+(sqrt((x(7)-x(11))^2+(x(8)-x(12))^2)-
6.0974)^2,x0,[],[],options);
```

Code For Neural Network

```
% Solve a Pattern Recognition Problem with a Neural Network
% Script generated by Neural Pattern Recognition app
% Created Wed Oct 15 19:38:36 AST 2014
%
% This script assumes these variables are defined:
%
%   IN - input data.
%   OUT - target data.

x = IN';
t = OUT';

% Create a Pattern Recognition Network
hiddenLayerSize = 161;
net = patternnet(hiddenLayerSize);


% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;


% Train the Network
[net,tr] = train(net,x,t);

% Test the Network
y = net(x);
e = gsubtract(t,y);
tind = vec2ind(t);
yind = vec2ind(y);
percentErrors = sum(tind ~= yind)/numel(tind);
performance = perform(net,t,y)

% View the Network
view(net)

% Plots
% Uncomment these lines to enable various plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, plotconfusion(t,y)
%figure, plotroc(t,y)
%figure, ploterrhist(e)
```

جامعة قناة السويس
كلية الحاسبات والمعلومات
الدراسات العليا

بعض الحلول المثلي لنماذج لابلاس

رسالة مقدمة إلى قسم علوم الحاسب - كلية الحاسبات والمعلومات-جامعة قناة السويس
كجزء من متطلبات الحصول على درجة الماجستير في علوم الحاسب.

مقدم من

محمد سليمان راضي الكومي

بكالوريوس الهندسة الكهربي- كلية الهندسة - جامعة عين شمس 2001

تحت أشراف

أ.د / محمد السيد وحيد

استاذ علوم الحاسب

كلية الحاسبات والمعلومات - جامعة قناة السويس

د / محمد عبدالله عبدالغفار

مدرس نظم معلومات

كلية الحاسبات والمعلومات - جامعة قناة السويس

جامعة قناة السويس
كلية الحاسبات والمعلومات
2016

جامعة قناة السويس

كلية الحسابات و المعلومات

بعض الحلول المثلي لنماذج لابلاس

رسالة مقدمة إلى قسم علوم الحاسب – كلية الحاسبات والمعلومات–جامعة قناة السويس

كجزء من متطلبات الحصول على درجة الماجستير في علوم الحاسب.

مقدم من

محمد سليمان راضي الكومي

بكالوريوس الهندسة الكهربي– كلية الهندسة – جامعة عين شمس 2001

لجنة المناقشة

ا.د/ السيد محمد السيد الهربيطي أستاذ بكلية الحسابات و المعلومات جامعة عين شمس.	Prof. Dr. / Alsayed Mohamed Alsayed Elherbaty Professor of Computer Science Faculty of Computers and Informatics, Ain Shams University	
ا.د / محمد امين عبد الواحد أستاذ بكلية العلوم جامعة المنوفية.	Prof. Dr. / Mohamed Amin Abdelwahed Professor of Computer Science, Mathematics and Computer Science department Monofia University	
أ.د / محمد السيد وحيد أستاذ بكلية الحاسبات جامعة قناة السويس.	Prof. Dr. / Mohamed Alsayed Wahed Professor of Computer Science Faculty of Computers and Informatics, Suez Canal university	

ملخص الرسالة

تهدف الرسالة الي الوصول الي وسيلة نستطيع بها الحكم عن طريق نموذج ثلاثي الابعاد لجسم الانسان اذا كان مصابا بسرطان الثدي .

العمل المقدم في هذه الرسالة هو محاكاة لاستخدام الشبكة العصبية الاصطناعية للحكم علي نموذج ثلاثي الابعاد لجسم الانسان اذا كان مصابا بسرطان الثدي ام لا.

وقد تم استخدام اطار العمل الابلاسي لتكوين نماذج مختلفة لتدريب و اختبار الشبكة العصبية الاصطناعية .

وتتكون الرسالة من خمسة أبواب وقائمة من المراجع التي تم الاستعانة بها وهي كما يلي:

الباب الأول:

في هذا الباب نناقش اهداف الرسالة و توصيف المشكلة و المنهجية المستخدمة لاتمام البحث كما يعرض مكونات البحث

الباب الثاني:

تم في هذا الباب عرض كيفية استخدام الشبكة العصبية الاصطناعية كوسيلة لتصنيف النماذج ثلاثية الابعاد لجسم الانسان و التي تم انشائها باستخدام اطار العمل الابلاسي لاستخدامها لمحاكاة جسم الانسان و الي تم تقسيمها الي فئتين مختلفتين : فئة تمثل المصابين بسرطان الثدي و اخري تمثل الاجسام الصحيحة .

لباب الثالث:

في هذا الباب نتعرض للتمثيلات المختلفة للاجسام ثلاثية الابعاد . و يتم الحديث عن المعالجة الهندسية الرقمية للاجسام في الابعاد الثلاثية . و يتم استنتاج بعض العلاقات الرياضية عن المعاملات التفاضلية الرقمية التي سيتم استخدامها مع اطار العمل الابلاسي .ونقدم مقارنة بين الحلول المثلي المحلية و الحلول المثلي الشاملة

الباب الرابع:

تم في هذا الباب عرض عدة طرق لحل اطار العمل الابلاسي بطرق سريعة عن طريق تقسيم المسألة الي عدة مسائل اصغر في الحجم . كما يتم عرض طريقة لاحداث تغيير طفيف في النموذج قيد الدراسة و كيفية حل اطار العمل الابلاسي بعد احداث التغيير عن طريق عمل الحسابات الازمة للجزء المتغيير و عدم الحاجة لحل المسألة ككل من جديد .

الباب الخامس :

لخص ما تم الوصول اليه من النتائج مع اقتراح استخدام الحلول المثلي الشاملة بدلا عن اطار العمل الابلاسي و التوصية لاستخدام نماذج حقيقية للاجسام المصابة بسرطان الثدي

تم نشر بحثين من الرسالة في مجلة دولية متخصصة في هذا المجال :

1- [WAE14] Wahed M., Abdallah M., Elkomy M. : Non Superfluous Time Solutions of the Laplacian Framework. The International Journal Of Computer Science and Network Security 2014 Vol. 14 No. 5 pp. 8-13

2- [WAE14] Wahed M., Abdallah M., Elkomy M. : Simulation of the detection of noxious breast cancer. The International Journal Of Computer Science and Network Security 2014 Vol. 14 No. 9 pp. 1-4