

THE UNIVERSITY OF CALGARY

The Wrinkling of Anisotropic Elastic Membranes

by

Chao Sun

A THESIS

SUBJECTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF MECHANICAL & MANUFACTURING
ENGINEERING

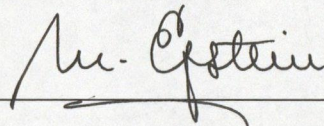
CALGARY, ALBERTA

JUNE, 2002

© Chao Sun 2002

THE UNIVERSITY OF CALGARY
FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommended to the Faculty of Graduate Studies for acceptance, a thesis entitled "The wrinkling of Anisotropic Elastic Membranes" submitted by Chao Sun in partial fulfillment of the requirements for the degree of Master of Science.



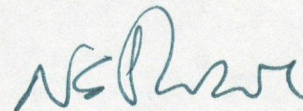
Supervisor, Dr. M. Epstein

Mechanical Engineering



Dr. S. Lukasiewicz

Mechanical Engineering



Dr. N. Shrive

Civil Engineering

August 2002

Abstract

A new membrane wrinkling theory based on the concept of a relaxed strain energy function is introduced. The theory is derived as a specific case of the saturated elasticity theory. The anisotropy is imported by means of a material metric tensor. The corresponding formulations for Neo-Hookean material, including the impending wrinkling surface are presented. The theory is then implemented into a finite element code, and several examples are studied by using the finite element code. Results of those examples are presented to show the effectiveness of the new theory for predicting wrinkling in anisotropic membranes. A membrane wrinkle pattern test is also carried out to verify the new theory.

Acknowledgements

I would like to thank Dr. Marcelo Epstein for his trust, guidance and patience put in this work, as well as his friendly financial help that make our life in Calgary much easier and enjoyable.

Many thanks to Mr. Jim McNeely for his kindly help with designing and manufacturing the membrane wrinkle pattern test device.

A special thanks to my loving wife Shirley and the whole family for their understanding, help and constant support.

to *Xiaohong and Gourui*

Table of Contents

| | |
|------------------------|------|
| Approval Page..... | ii |
| Abstract..... | iii |
| Acknowledgement..... | iv |
| Dedication..... | v |
| Table of Contents..... | vi |
| List of Figures..... | viii |

Chapter ONE

| | |
|---|---|
| Introduction..... | 1 |
| 1.1 The Concept of A Membrane..... | 1 |
| 1.2 The Wrinkling Phenomenon of Membranes..... | 1 |
| 1.3 The Importance and Necessity of The Research on The Wrinkling Phenomenon of Membranes..... | 4 |

Chapter TWO

| | |
|--|---|
| Review of Nonlinear Membrane Theory..... | 8 |
| 2.1 Model I..... | 8 |

| | |
|-------------------|----|
| 2.2 Model II..... | 12 |
|-------------------|----|

Chapter THREE

| | |
|---|----|
| Wrinkling Theory of Anisotropic Membrane..... | 16 |
| 3.1 The Saturated Elasticity Theory..... | 16 |
| 3.1.1 The Concept of Saturated Elasticity..... | 17 |
| 3.1.2 The Relaxed Energy Function..... | 19 |
| 3.2 The Wrinkling Theory of Anisotropic Membrane..... | 23 |
| 3.2.1 The Wrinkling Condition..... | 23 |
| 3.2.2 The Stress and Strain Spaces..... | 24 |
| 3.2.3 The Wrinkling Strain..... | 27 |
| 3.2.4 Orthotropization Technique..... | 29 |
| 3.3 Application to The Neo-Hookean Material | 31 |

Chapter FOUR

| | |
|---|----|
| Finite Element Implementation | 33 |
| 4.1 The Strain Tensor In Curvilinear Coordinate System..... | 33 |
| 4.2 Geometry of Membrane Element..... | 35 |
| 4.3 Tangent Stiffness Matrix of Membrane Element..... | 39 |
| 4.4 The Scheme For Determining The Relaxed Energy Function...41 | |
| 4.5 The Finite Element Program..... | 42 |

Chapter FIVE

| | |
|---|----|
| Examples | 43 |
| 5.1 Example 1: A Square Membrane | 43 |
| 5.2 Example 2: An Annular Concentric Membrane | 49 |
| 5.3 Example 3: An Annular Eccentric Membrane..... | 54 |
| 5.4 Example 4: An Annular Square Membrane..... | 57 |

Chapter SIX

| | |
|---|----|
| Membrane Wrinkling Pattern Test | 61 |
| 6.1 Testing Device..... | 61 |
| 6.2 Membrane Wrinkling Pattern Test | 63 |

Chapter SEVEN

| | |
|---------------------------------------|----|
| Conclusions and Recommendations | 69 |
| Bibliography..... | 71 |
| Appendix..... | 75 |

List of Figures

| | |
|---|----|
| Figure 1.1 Three kinds of stability of equilibrium condition..... | 2 |
| Figure 2.1 Wrinkled, pseudo and lengthened surface..... | 9 |
| Figure 3.1 Impending-wrinkling surfaces in the stress and strain space..... | 25 |
| Figure 4.1 The convected curvilinear coordinate system | 34 |
| Figure 4.2 Four-node space bilinear isoparametric element | 37 |
| Figure 4.3 Co-variant base vectors and unit base vectors of orthotropy..... | 39 |
| Figure 5.1 The meshed reference configuration of a square sheet | 45 |
| Figure 5.2 The deformed mesh for displacements $u = 0.1m$ and $v = 0.2m$ | 45 |
| Figure 5.3 The deformed mesh for displacements $u = 0.2m$ and $v = 0.2m$ | 46 |
| Figure 5.4 The deformed mesh for displacements $u = 0.5m$ and $v = 0.5m$ | 46 |
| Figure 5.5 The deformed mesh for displacements $u = 0.2m$ and $v = 1.0m$ | 47 |
| Figure 5.6 The deformed 5×5 mesh for 0.1m simple stretch..... | 48 |
| Figure 5.7 The deformed 10×10 mesh for 0.1m simple stretch..... | 48 |
| Figure 5.8 The meshed reference configuration of an annular membrane | 51 |
| Figure 5.9 The deformed configuration for 45° counterclockwise rigid rotation of the inner boundary for isotropic case | 51 |
| Figure 5.10 The deformed configuration for 0.3m lateral deflection and counterclockwise | |

| | |
|---|----|
| rigid rotation 45° of the inner boundary | 52 |
| Figure 5.11 The single wrinkling zone of Figure 5.2c in a plane figure..... | 52 |
| Figure 5.12 The deformed configuration for 45° counterclockwise rigid rotation of the inner boundary for anisotropic case $\beta_1 = 1.5, \beta_2 = 0.75$ | 53 |
| Figure 5.13 The deformed configuration for 45° counterclockwise rigid rotation of the inner boundary for anisotropic case $\beta_1 = 0.75, \beta_2 = 1.5$ | 53 |
| Figure 5.14 The meshed reference configuration of an annular eccentric membrane..... | 55 |
| Figure 5.15 The deformed configuration for isotropic case | 55 |
| Figure 5.16 The deformed configuration for $\beta_1 = 1.5, \beta_2 = 0.75$ | 56 |
| Figure 5.17 The deformed configuration for $\beta_1 = 0.75, \beta_2 = 1.5$ | 56 |
| Figure 5.18 The meshed reference configuration of an annular square membrane..... | 58 |
| Figure 5.19 The deformed configuration for isotropic case $\beta_1 = 1.0, \beta_2 = 1.0$ | 58 |
| Figure 5.20 The deformed configuration for $\beta_1 = 0.8, \beta_2 = 1.0$ | 59 |
| Figure 5.21 The deformed configuration for $\beta_1 = 1.2, \beta_2 = 1.0$ | 59 |
| Figure 5.22 The deformed configuration for $\beta_1 = 1.0, \beta_2 = 0.8$ | 60 |
| Figure 5.23 The deformed configuration for $\beta_1 = 1.0, \beta_2 = 1.2$ | 60 |
| Figure 6.1 Testing device of membrane wrinkling pattern test..... | 62 |
| Figure 6.2 The wrinkling pattern of an isotropic membrane with concentric boundary... | 65 |
| Figure 6.3 The wrinkling pattern of an isotropic membrane with eccentric boundary..... | 66 |

Figure 6.4 The wrinkling pattern of an orthotropic membrane with concentric boundary.....67

Figure 6.5 The wrinkling pattern of an orthotropic membrane with eccentric boundary...68

Chapter ONE

Introduction

1.1 The Concept of Membrane

A membrane here is an ideal mechanical model. One dimension of the membrane is far less than the other two dimensions, thus is negligible. So a membrane is a plane model. Besides, a membrane has no bending stiffness and thus can not withstand any compressive stresses. However, a real membrane has always some bending stiffness, although its bending stiffness may be so small that it has little effect on the stresses.

The membrane model can be applied to all thin structures bearing tension forces. Such examples include parachutes, fabric structures, light aircraft, automobile airbags, space-based radar, skin, etc.

1.2 The Wrinkling Phenomenon of Membranes

When a membrane is in a uniaxial tension, there will appear some wrinkles on the membrane and those wrinkles will be aligned with the tension direction. Compressive stresses are usually involved in the development of off-plane deformation. However, in the

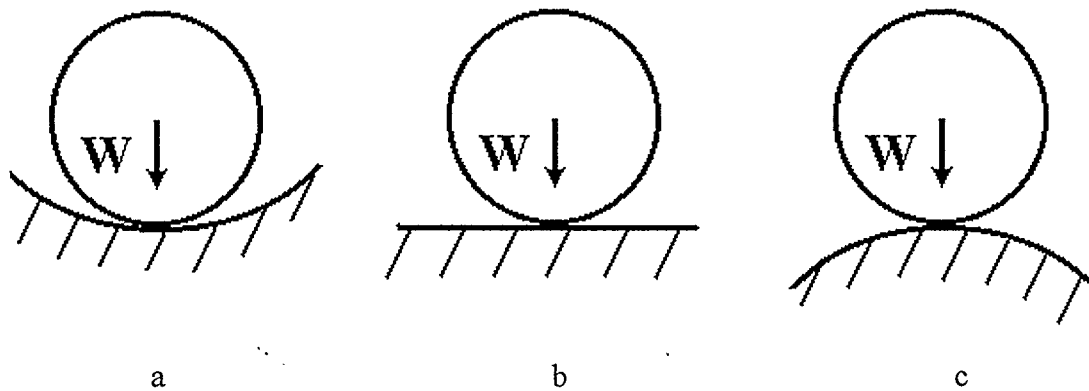


Figure 1.1 Three kinds of stability of equilibrium condition

case of membranes, no compressive stress can exist since a membrane has no bending stiffness and can not withstand any compressive stress. In fact, the wrinkling of membranes is essentially due to the fact that the membrane loses the stability of the equilibrium condition.

There are three kinds of possible stability of the equilibrium condition—the stable equilibrium, the unstable equilibrium and the indifferent equilibrium, as shown in Figure 1.1. The ball in Figure 1.1a is in the equilibrium condition at the lowest point on a concave surface, and it will return to its original equilibrium position due to its weight after being taken away a small distance from its original equilibrium position. This kind of equilibrium condition is called the stable equilibrium. The ball in Figure 1.1b is in the equilibrium condition on a horizontal plane, and it can keep balance at any point near its original equilibrium position on the plane. This kind of equilibrium condition is called the indifferent equilibrium. The ball in Figure 1.1c is in the equilibrium condition at the top point of a convex surface, it will roll away under its weight after being given a small

disturbance and never return to its original equilibrium position. This kind of equilibrium condition is called the unstable equilibrium.

After being disturbed, the ball moves to a new position from its equilibrium position. When the potential energy of the ball at any possible new position is higher than that at its equilibrium position, the potential energy at the equilibrium position is a minimum, and the ball will return to a position of lower potential energy from a position of higher potential energy. On the contrary, if the potential energy at the equilibrium position is a maximum, the ball will not return to a position of higher potential energy from a position of lower potential energy, unless an external force is involved. When the potential energy of the ball at any possible new position is always the same as that at its equilibrium position, any possible new position could be its equilibrium position.

The stability of equilibrium condition of a structure or a structural element is usually more complicated than the case of the ball mentioned above. However, no matter how many variables are involved, the concept of stability of equilibrium condition is same, there are always three kinds of possible stability of the equilibrium condition—the stable equilibrium, the unstable equilibrium and the indifferent equilibrium.

The buckling of a shell is an example of losing stability of equilibrium. At the impending buckling point, a shell is in an unstable equilibrium. But before buckling, a shell with low bending stiffness can support a small amount of compressive stress. If such a shell is subjected to compression in one principal direction and tension in the other principal direction, it will buckle and many narrow wrinkles will form with crests and troughs roughly parallel to the tension direction. As the bending stiffness decreases, so do the

critical buckling stress and the distance between the crests. When the bending stiffness vanishes, so does the critical buckling stress and there would be an infinite number of infinitely narrow wrinkles exactly parallel to the tension direction. This is the case of the wrinkling of membrane. So a membrane is essentially a thin shell with no bending stiffness.

In fact, there always exists a small amount of bending stiffness in a real membrane, although it may be so small that it is usually neglected in a stress analysis. It is the small amount of bending stiffness that determines the real configuration of a wrinkled membrane as a finite number of finitely narrow wrinkles. Pogorelov (1967) mentioned an approach to membrane wrinkling by viewing the membrane as a thin elastic shell.

At any point on its surface, a membrane must be in one of three states. In a “double wrinkling” state, the membrane is not stretched in any direction. In a “no wrinkling” state, the membrane is in tension in all directions. In a “single wrinkling” state, the membrane is in a uniaxial tension, which is what we are most interested in.

1.3 The Importance and Necessity of The Research on The Wrinkling Phenomenon of Membranes

Membranes are widely used as structural elements in many engineering fields. Cohen and Wang (1984) studied the response of elastic and hyperelastic membrane points. Contri and Schrefler (1988) carried out a geometrical nonlinear analysis of wrinkled membranes using a no-compression material model. Lukasiewicz et al. (1983,1985,1990) studied the stability of air-supported cylindrical membranes and the collapse mode and loads of an inflatable freestanding membrane. Miller et al. (1985) analyzed partly wrinkled membranes

using finite element method. Stanuszek and Glockner (1995) studied key response features of spherical inflatables under axisymmetric liquid and hydrostatic loads. Jenkins (1996) presented a comprehensive description of the state of the art up to 1996 for all aspects of the theory and applications of membranes. Such applications include civil engineering and architecture, aeronautics, mechanical engineering, astronautics, biomechanics, etc. However, the conventional membrane theory cannot explain the wrinkling phenomenon of membranes. In conventional membrane theory, the membrane can resist compression without wrinkling although its bending stiffness vanishes. Therefore, the phenomenon of wrinkling of membranes is of both practical and theoretical importance.

In solid mechanics, there are usually three aspects that need to be considered to establish the relation between external forces and displacements. They are equilibrium equations—the relation between external forces and stresses, the constitutive law—the relation between stresses and strains, and geometric equations—the relation between strains and displacements. To describe the phenomenon of wrinkling of membranes, some modifications must be made to the conventional membrane theory. Among the three aspects above, it seems nothing can be done with equilibrium equations. Thus, there are only two possible ways to model the phenomenon of wrinkling of membranes, one is to modify the constitutive law and the other is to modify the geometric equations.

In fact, all existing models of wrinkling of membranes fall into two categories. One is to modify the deformation gradient tensor to eliminate compressive stresses occurring in the course of the solution procedure. Wu and Canfield (1981) introduced an extra parameter to modify the deformation tensor. Roddeman et al. (1987,1991) generalized their approach

to include anisotropic materials and developed the criterion to judge the state of the membrane at a point. Their theory was then implemented into a finite element program to simulate wrinkled membranes. Muttin (1996) generalized the wrinkling theory of Roddeman et al. for curved membranes using curvilinear coordinates. Kang and Im (1997) adopted Muller's method to search for multiple roots of non-linear equations to find the wrinkling direction. Lu et al. (2001) derived explicit formulas for the internal forces and the tangent stiffness matrix using curvilinear coordinates and presented a new scheme to find the wrinkling direction.

The other is to modify the constitutive relationship of the membrane to simulate wrinkling. This method was first proposed by Wagner (1929) and subsequently developed by other researchers, such as Reissner (1939), Kondo et al. (1955), Mansfield (1970), Wu (1978), Pipkin (1986,1993,1994), Steigmann (1986,1990), Li (1993), Haseganu (1994) and Epstein (1999,2001). Pipkin (1986) showed that the wrinkling of membranes could be obtained as minimum-energy states by means of a relaxed strain energy function. Steigmann (1990) further developed the concept of the relaxed strain energy function in the context of tension-field theory, and Haseganu (1994) developed the numerical implementation of their theory. Pipkin (1993) also presented an algorithm for obtaining the relaxed energy for anisotropic membranes. Epstein (1999) presented a relaxed energy function applicable to arbitrary anisotropic elastic membrane that can be obtained by regarding membrane wrinkling as a specific case of saturated elasticity.

Both models mentioned above can describe the wrinkling phenomenon of membranes, but they interpret it differently. More and more investigations on the wrinkling

phenomenon of membranes will increase our understanding on the mechanism of the wrinkling of membranes. The work we have done is one of those trials.

The work presented here is based on the wrinkling theory of Epstein, which is introduced in Chapter three. The wrinkling theory of Epstein is then formulated using curvilinear coordinates and implemented into a finite element program to simulate the wrinkling of membranes in Chapter four. Several numerical examples and the membrane wrinkle pattern test are presented in Chapter five and Chapter six to demonstrate the method.

Chapter TWO

Review of Non-linear Membrane Wrinkling Theory

At any point on its surface, a membrane must take one of three states: double wrinkling—being not stretched in any direction, no wrinkling—being stretched in all directions, and single wrinkling—being in a uniaxial tension. Thus a membrane theory must answer two questions: how to determine the state of a point on the membrane and how to calculate the strain and stress in the three different states.

There are two different models to simulate the wrinkling of membranes, model I by modifying the deformation gradient tensor and model II by modifying the constitutive relationship of the membrane.

2.1 Model I

Consider a membrane represented by its mid-surface in a plane stress state. A curvilinear coordinate system (ξ^1, ξ^2) is attached to the mid-surface. The covariant base vectors of the coordinate system on the reference and deformed membrane are defined, respectively, as

$$\mathbf{G}_\alpha = \frac{\partial \mathbf{X}}{\partial \xi^\alpha}, \quad \mathbf{g}_\alpha = \frac{\partial \mathbf{x}}{\partial \xi^\alpha} \quad (2.1)$$

where \mathbf{X} and \mathbf{x} are position vectors of a particle referred to origins on the reference and deformed membrane, respectively. Greek indices take values in $\{1,2\}$. The covariant components of the metric tensor on the reference and deformed membrane are defined, respectively, as

$$G_{\alpha\beta} = \mathbf{G}_\alpha \cdot \mathbf{G}_\beta, \quad g_{\alpha\beta} = \mathbf{g}_\alpha \cdot \mathbf{g}_\beta \quad (2.2)$$

The contravariant components of metric tensor and the contra base vectors on the reference membrane are obtained from

$$[G^{\alpha\beta}] = [G_{\alpha\beta}]^{-1}, \quad \mathbf{G}^\alpha = G^{\alpha\beta} \mathbf{G}_\beta \quad (2.3)$$

Now, let's introduce a smoothed pseudo-surface—plane ABCD to replace the real wrinkled region, as shown in figure 2.1. Particles on the real wrinkled surface ABCD are projected

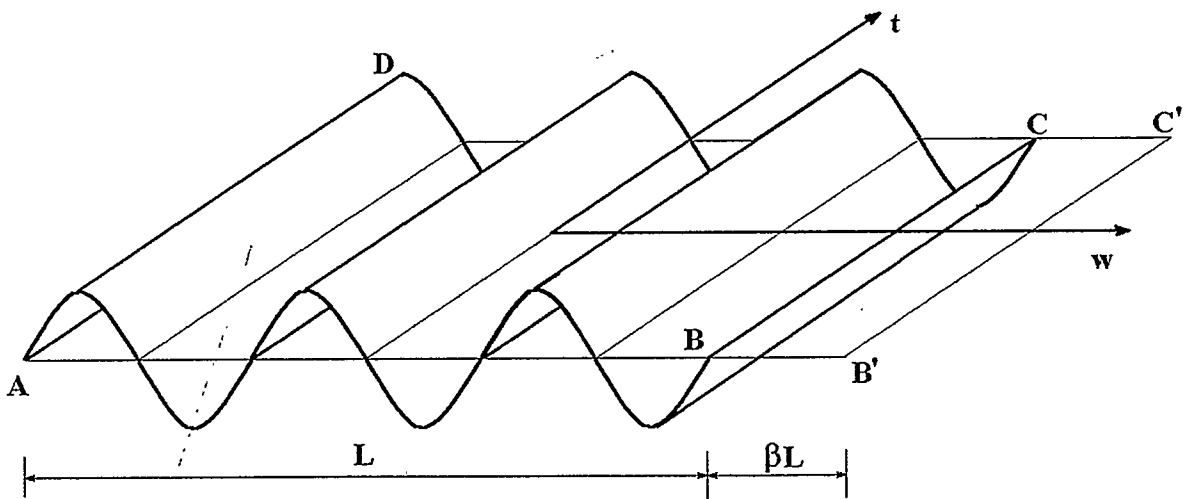


Figure 2.1 Wrinkled, pseudo and lengthened surface

onto the pseudo-surface. The quantities measured on the pseudo-surface are called “nominal” ones while the quantities measured on the real surface are called “real” ones. The nominal deformation gradient tensor is

$$\mathbf{F} = \mathbf{g}_\alpha \otimes \mathbf{G}^\alpha \quad (2.4)$$

The nominal Green strain tensor on the reference membrane is then obtained

$$\mathbf{E} = \frac{1}{2}(\mathbf{F}^T \mathbf{F} - \mathbf{I}) = E_{\alpha\beta} \mathbf{G}^\alpha \otimes \mathbf{G}^\beta, \quad E_{\alpha\beta} = \frac{1}{2}(g_{\alpha\beta} - G_{\alpha\beta}) \quad (2.5)$$

where \mathbf{I} is the unit tensor. The nominal second Piola-Kirchhoff stress is determined through the constitutive relation

$$\mathbf{S} = S^{\alpha\beta} \mathbf{G}_\alpha \otimes \mathbf{G}_\beta = \mathbf{H}(\mathbf{E}) \quad (2.6)$$

where \mathbf{H} is a tensor function of \mathbf{E} .

In a double wrinkling state, the membrane is not stretched in any direction, so the Green strain must satisfy

$$E_{11} + E_{22} \leq 0, \quad E_{11}E_{22} - E_{12}E_{21} \geq 0 \quad (2.7)$$

The real stress and strain both vanish.

In a no wrinkling state, the membrane is stretched in all directions, so the second Piola-Kirchhoff stress must satisfy

$$S^{11} + S^{22} \geq 0, \quad S^{11}S^{22} - S^{12}S^{21} \geq 0 \quad (2.8)$$

The real strain and stress are the same as the nominal ones.

In a single wrinkling state, the membrane is in a uniaxial tension state. Define $\mathbf{t} = t_\alpha \mathbf{g}^\alpha$ as a unit vector in the tensile principal direction and $\mathbf{w} = w_\alpha \mathbf{g}^\alpha$ as a unit vector

in the other principal direction, as shown in figure 2.1. Then the following conditions hold:

$$\boldsymbol{w} \cdot \boldsymbol{w} = 1, \boldsymbol{t} \cdot \boldsymbol{t} = 1, \boldsymbol{t} \cdot \boldsymbol{w} = t^\alpha w_\alpha = 0 \quad (2.9)$$

When the membrane in single wrinkling state is lengthened transverse to the wrinkles, each element of area will undergo rigid body movement until the wrinkles vanish. Both the Green strain and Cauchy stress do not change during the rigid body movement. When the wrinkles just vanish, the lengthened membrane is still in the same uniaxial tension, then define the deformation gradient tensor on the lengthened membrane as

$$\tilde{\boldsymbol{F}} = (\boldsymbol{I} + \beta \boldsymbol{w} \otimes \boldsymbol{w}) \boldsymbol{F} \quad (2.10)$$

where $\beta > 0$ is the ratio of lengthening. The Green strain on the wrinkled membrane, which is equal to the Green strain on the lengthened membrane, is obtained as

$$\tilde{\boldsymbol{E}} = \frac{1}{2} (\tilde{\boldsymbol{F}}^T \tilde{\boldsymbol{F}} - \boldsymbol{I}) = \boldsymbol{E} + \frac{1}{2} \beta (2 + \beta) \hat{\boldsymbol{w}} \otimes \hat{\boldsymbol{w}} \quad (2.11)$$

where

$$\tilde{\boldsymbol{E}} = \tilde{E}_{\alpha\beta} \boldsymbol{g}^\alpha \otimes \boldsymbol{g}^\beta, \tilde{E}_{\alpha\beta} = E_{\alpha\beta} + \frac{1}{2} \beta (2 + \beta) w_\alpha w_\beta, \hat{\boldsymbol{w}} = \boldsymbol{w} \boldsymbol{F} = w_\alpha \boldsymbol{G}^\alpha \quad (2.12)$$

The second Piola-Kirchhoff stress and the Cauchy stress on the lengthened membrane are then given by

$$\tilde{\boldsymbol{S}} = \boldsymbol{H}(\tilde{\boldsymbol{E}}) = \tilde{S}^{\alpha\beta} \boldsymbol{G}_\alpha \otimes \boldsymbol{G}_\beta \quad (2.13)$$

$$\tilde{\boldsymbol{\sigma}} = \tilde{J}^{-1} \tilde{\boldsymbol{F}} \tilde{\boldsymbol{S}} \tilde{\boldsymbol{F}}^T \quad (2.14)$$

where $\tilde{J} = \det(\tilde{\boldsymbol{F}}) > 0$. The Cauchy stress thus satisfies the following conditions:

$$\tilde{\boldsymbol{\sigma}} \boldsymbol{w} = 0 \quad (2.15)$$

$$t \tilde{\sigma} t > 0 \quad (2.16)$$

From these two conditions, one can obtain the following equivalent conditions:

$$\hat{w} \tilde{S} \hat{w} = 0 \quad (2.17)$$

$$\nu \tilde{S} \hat{w} = 0 \quad (2.18)$$

$$n S n > 0 \quad (2.19)$$

where \hat{w} is the unit vector along the principal direction of \tilde{S} where the corresponding principal stress vanishes, ν is an arbitrary vector tangent to the reference membrane which is linearly independent of \hat{w} , and n is the unit vector along the other principal direction of \tilde{S} .

2.2 Model II

Consider a flat reference membrane Ω with coordinates (x_1, x_2) . A point in the membrane can then be expressed as $x = x_\alpha e^\alpha$, where Greek indices run from 1 to 2 and $\{e_1, e_2\}$ is a fixed orthonormal basis. The deformed membrane is of a three dimensional configuration, the material particle at x on the reference membrane is displaced to the point $r(x) = r_i(x) e_i$, where Latin indices run from 1 to 3 and $e_3 = e_1 \times e_2$ is the unit normal to the reference membrane. Thus $r_i(x)$ are the Cartesian coordinates of material points on the deformed membrane.

Define a deformation gradient in Cartesian coordinate system as

$$F = F_{i\alpha}(x) e_i \otimes e_\alpha, \quad F_{i\alpha} = r_{i,\alpha} \quad (2.20)$$

The associated Cauchy-Green strain tensor is:

$$\mathbf{C} = \mathbf{F}^T \mathbf{F} = C_{\alpha\beta} \mathbf{e}_\alpha \otimes \mathbf{e}_\beta, \quad C_{\alpha\beta} = F_{i\alpha} F_{i\beta} \quad (2.21)$$

Since $\mathbf{C}(\mathbf{x})$ is symmetric, positive semi-definite for every $\mathbf{x} \in \Omega$, it possesses an orthonormal pair $\{\mathbf{L}(\mathbf{x}) = L_\alpha(\mathbf{x})\mathbf{e}_\alpha, \mathbf{M}(\mathbf{x}) = M_\alpha(\mathbf{x})\mathbf{e}_\alpha\}$ of eigenvectors. Define non-negative scalars

$$\lambda(\mathbf{x}) = |\mathbf{FL}| = \sqrt{\mathbf{L} \cdot \mathbf{CL}}, \quad \mu(\mathbf{x}) = |\mathbf{FM}| = \sqrt{\mathbf{M} \cdot \mathbf{CM}} \quad (2.22)$$

and unit vectors

$$\mathbf{l}(\mathbf{x}) = \lambda^{-1} \mathbf{FL}, \quad \mathbf{m}(\mathbf{x}) = \mu^{-1} \mathbf{FM} \quad (2.23)$$

By using these scalars and unit vectors, the deformation gradient and strain can then be expressed as.

$$\mathbf{F} = \mathbf{FL} \otimes \mathbf{L} + \mathbf{FM} \otimes \mathbf{M} = \lambda \mathbf{l} \otimes \mathbf{L} + \mu \mathbf{m} \otimes \mathbf{M} \quad (2.24)$$

$$\mathbf{C} = \lambda^2 \mathbf{L} \otimes \mathbf{L} + \mu^2 \mathbf{M} \otimes \mathbf{M} + \lambda\mu \mathbf{l} \cdot \mathbf{m} (\mathbf{L} \otimes \mathbf{M} + \mathbf{M} \otimes \mathbf{L}) \quad (2.25)$$

According to the definition of \mathbf{L} and \mathbf{M} , $\mathbf{l} \cdot \mathbf{m} = 0$ unless $\lambda = 0$ or $\mu = 0$. So the pair $\{\mathbf{l}(\mathbf{x}), \mathbf{m}(\mathbf{x})\}$ is an orthonormal basis for the plane tangent to the deformed membrane at the point \mathbf{x} . Then λ and μ are the principal stretches, the square roots of the eigenvalues of \mathbf{C} .

The strain energy per unit area of the reference configuration for isotropic materials can be expressed as a symmetric function of λ and μ ,

$$W(\mathbf{F}) = W(\lambda, \mu) = W(\mu, \lambda) \quad (2.26)$$

the second Piola-Kirchhoff stress tensor is then obtained as

$$\mathbf{S} = \frac{\partial W}{\partial \lambda} \mathbf{l} \otimes \mathbf{L} + \frac{\partial W}{\partial \mu} \mathbf{m} \otimes \mathbf{M} \quad (2.27)$$

For incompressible isotropic materials, the principal stretches λ and μ in a minimum-energy configuration must satisfy the following condition:

$$\lambda \geq \mu^{-1/2} \quad \text{and} \quad \mu \geq \lambda^{-1/2} \quad (2.28)$$

To describe the wrinkling of membrane, a relaxed energy $W_R(\mathbf{F})$ is introduced to replace the original energy. The relaxed energy is defined as

$$W_R(\mathbf{F}) = W_R(\lambda, \mu) = \begin{cases} W(\lambda, \mu) & \lambda > \mu^{-1/2}, \mu > \lambda^{-1/2} \\ \hat{W}(\lambda) & \lambda > 1, \mu \leq \lambda^{-1/2} \\ \hat{W}(\mu) & \mu > 1, \lambda \leq \mu^{-1/2} \\ 0 & \lambda \leq 1, \mu \leq 1 \end{cases} \quad (2.29)$$

where $W(\lambda, \mu)$ is the original strain energy, and

$$\hat{W}(x) = W(x, x^{-1/2}) = W(x^{-1/2}, x) \quad (2.30)$$

is the strain energy in uniaxial tension.

Therefore, the values of λ and μ are used to determine the state of a point on the membrane. In equation (2.29), the first row accounts for no wrinkling state; the last row is double wrinkling state, while the other two rows represent single wrinkling state. For all these three states, the Cauchy-Green strain are given by equation (2.21), and the second Piola-Kirchhoff stress can then be obtained through the constitutive law by substituting the corresponding relaxed energy $W_R(\mathbf{F})$.

It should be noted that the strain and stress both vanish for double wrinkling case because of its zero relaxed energy while they are the same as those of conventional

membrane theory for no wrinkling case because the relaxed energy here is the strain energy in conventional membrane theory. This is also in accordance with model I.

Chapter THREE

Wrinkling Theory of Anisotropic Membranes

To model the wrinkling of membranes, two questions must be answered. One is what the wrinkling condition is, while the other is how to calculate the stress and strain in a wrinkling configuration. The wrinkling phenomenon of membranes is viewed as a particular case of saturated elasticity in this chapter. Thus the saturated elasticity theory is introduced first. Then the wrinkling condition of membranes is derived based on the saturated elasticity theory, and the wrinkling strain and the constitutive law effective in the wrinkling configuration are obtained as well. Next an orthotropization technique is presented to introduce the anisotropy into the wrinkling theory of membranes. Finally, specific formulations for Neo-Hookean material are obtained due to its wide engineering application, although the wrinkling theory can be applied to a wide variety of materials.

3.1 The Saturated Elasticity Theory

3.1.1 The concept of saturated elasticity

Consider a hyperelastic material characterized by a strain energy density

$$W = W(\mathbf{C}) \quad (3.1)$$

per unit volume of a given reference configuration, where $\mathbf{C} = \mathbf{F}^T \mathbf{F}$ is the right Cauchy-Green strain tensor (with components C_{ij}), and \mathbf{F} is the deformation gradient (with components F_i^j). Cartesian coordinates are used throughout. The second Piola-Kirchhoff stress tensor \mathbf{S} (with components S_{ij}) is given by:

$$\mathbf{S} = 2 \frac{\partial W}{\partial \mathbf{C}} \quad (3.2)$$

When the stresses satisfy a certain condition called the saturation condition, the material enters a state of saturated elasticity. That is just as what happens in plasticity case, the material yields when the stresses satisfy the yield condition, which is usually expressed as the equivalent stress in terms of the yield limit of the material. But all the strains are elastic strains in the saturation elasticity case and thus are reversible. Assume that the saturation condition can be expressed mathematically by a homogeneous function f of degree m , in the form:

$$f(S_{ij}) \leq k^2 \quad (3.3)$$

where k^2 is a fixed material constant (similar to a uniaxial “yield stress”). The assumed homogeneity of the function f naturally leads to the following identity:

$$mf = S_{ij} \frac{\partial f}{\partial S_{ij}} \quad (3.4)$$

where the summation convention for diagonally repeated indices has been adopted.

The concept of saturation can then be interpreted as follows:

- (i) The saturation condition equation (3.3) must always be observed.
- (ii) If the strict inequality given in the saturation condition holds, the material is in the “unsaturated domain” and the stresses are given by the unsaturated constitutive law — equations (3.1) and (3.2), using the originally given strain energy function.
- (iii) If the stresses obtained from the unsaturated constitutive law would not satisfy the saturation condition, then the stresses are no longer derivable from the unsaturated constitutive law, but satisfy the strict equality given in the saturation condition and the material is in the “post-saturated domain”.
- (iv) the stresses in the post-saturated domain are derivable from the post-saturated constitutive law using a relaxed energy function which is a C^1 -continuation of the given strain energy function from the unsaturated domain.

The equality in saturation condition can be interpreted as defining a surface Σ in the stress space as well as its counterpart a surface Σ' in strain space through the constitutive law. Both stress and strain spaces are 9-dimensional, and Cartesian dot product of vectors therein corresponds to the usual inner product of tensors. By taking advantage of the symmetry of the stress and strain, the dimensions of these two space can then be reduced to just 6, and the off-diagonal components of tensors should be understood as multiplied by

the square root of 2, in order to recover the usual inner product.

3.1.2 The relaxed energy function

To determine the relaxed energy function, let us first recall a result from the theory of first-order partial differential equations (see Courant (1962)).

Given a first-order partial differential equation for a function u of n independent variables (x^1, \dots, x^n)

$$G(x^1, \dots, x^n, p_1, \dots, p_n, u) = 0 \quad (3.5)$$

where $p_i = \frac{\partial u}{\partial x^i}$, the so-called characteristic strips are given by solutions of the following characteristic system of ordinary differential equations for the $2n+1$ functions x^i, p_i, u of a parameter s :

$$\frac{dx^i}{ds} = \frac{\partial G}{\partial p_i} \quad (3.6)$$

$$\frac{dp_i}{ds} = -\left(p_i \frac{\partial G}{\partial u} + \frac{\partial G}{\partial x^i} \right) \quad (3.7)$$

$$\frac{du}{ds} = p_i \frac{\partial G}{\partial p_i} \quad (3.8)$$

Besides satisfying the characteristic system, a characteristic strip must also satisfy the first-order partial differential equation in that the function G should vanish along it. In the post-saturated domain, the equality given in the saturation condition holds and can be regarded as a partial differential equation for the relaxed energy function, where the strain

components C_{ij} , the stress components $\frac{1}{2}S_{ij} = \frac{\partial W}{\partial C_{ij}}$ and the relaxed energy function W correspond to x^i, p_i , and u respectively. Then the corresponding characteristic strips of the equality given in the saturation condition can be obtained from the following characteristic system of ordinary differential equations for the $2n+1$ functions C_{ij}, S_{ij}, W of a parameter s :

$$\frac{dC_{ij}}{ds} = 2 \frac{\partial f}{\partial S_{ij}} \quad (3.9)$$

$$\frac{dS_{ij}}{ds} = 0 \quad (3.10)$$

$$\frac{dW}{ds} = S_{ij} \frac{\partial f}{\partial S_{ij}} = mk^2 \quad (3.11)$$

The characteristics of the relaxed energy function are disclosed by these three ordinary differential equations. First, equation (3.10) indicates that the second Piola-Kirchhoff stress keeps constant along characteristic strips. Based on this fact, equation (3.9) means that the first order derivatives of the strain with respect to the only parameter s are all constants since the derivatives in the right-hand side of equation (3.9) depend on the second Piola-Kirchhoff stress only. Thus the characteristic lines in the strain space are straight lines.

Secondly, equation (3.9) and (3.11) imply that the directional derivative in the direction of the characteristic is matched as well. This can be shown by calculating the derivative

$\left(\frac{dW}{ds}\right)_u$ in the direction of $\frac{dC_{ij}}{ds}$ approaching the boundary Σ' from the unsaturated domain.

$$\left(\frac{dW}{ds}\right)_u = \frac{\partial W}{\partial C_{ij}} \frac{dC_{ij}}{ds} = \frac{1}{2} S_{ij}^2 \frac{\partial f}{\partial S_{ij}} = S_{ij} \frac{\partial f}{\partial S_{ij}} = \frac{dW}{ds} \quad (3.12)$$

This ensures that the transition between the given strain energy in the unsaturated domain and the obtained relaxed energy in the post-saturated domain is of class C^1 .

Thirdly, equation (3.11) shows that the relaxed energy function grows linearly with s along the characteristic lines. Therefore, the initial conditions for the relaxed energy, stress and strain along any specific characteristic can be obtained from the known values of the given strain energy, stress and strain at the boundary Σ' between the unsaturated and post-saturated domains.

Fourthly, equation (3.9) shows that the characteristic lines in the strain space are perpendicular to the surface Σ in stress space at the impending saturation point. This is called the “normality rule”, it is a consequence of the fact that the saturation function contains stresses only. The characteristics, in general, are not perpendicular to the surface Σ' , but have the direction of the normal to Σ at the corresponding point.

Based on the normality rule, the value of the relaxed energy function for a given strain in the post-saturated domain can be determined as follows. Shoot straight lines from the given point to the surface Σ' until, for some straight line, its direction coincides with the normal at the point in Σ that corresponds to the intersection of the line with Σ' . Then

determine the value of the parameter s by calculating the distance between the given point and the intersection point of the characteristic with Σ' and the length of the gradient vector of Σ at the corresponding point. Indeed, equation (3.9) implied that:

$$ds = \frac{dC}{2\sqrt{\frac{\partial f}{\partial S} \cdot \frac{\partial f}{\partial S}}} \quad (3.13)$$

where dC is the ordinary Cartesian length element in the strain space.

Finally, by integrating equation (3.9), the total strain can be decomposed additively into two parts—the saturated strain and the post-saturated strain, as following:

$$C_{ij} = C_{ij}^0 + 2s \left(\frac{\partial f}{\partial S_{ij}} \right)_{C_{ij}^0} = C_{ij}^0 + \Delta C_{ij}^p \quad (3.14)$$

where C_{ij}^0 is the integration constant, a function of the second Piola-Kirchhoff stresses and represents the saturated strain component. To see that this is the case, let $s = 0$, then the total strain is just the saturated strain. As long as the second Piola-Kirchhoff stress keeps unchanged, the saturated component remains constant. ΔC_{ij}^p is the additive post-saturated strain component, which abides by the normality rule.

The additivity of the Cauchy-Green tensor does not contradict the multiplicative decomposition of the deformation gradient, but imposes a restriction upon its post-saturated component via the condition:

$$\mathbf{C} = \mathbf{F}^T \mathbf{F} = \mathbf{F}^{0T} \mathbf{F}^{pT} \mathbf{F}^p \mathbf{F}^0 = \mathbf{C}^0 + 2s \left(\frac{\partial f}{\partial \mathbf{S}} \right) \quad (3.15)$$

where the post-saturated strain component can be expressed as:

$$\mathbf{F}^{P^T} \mathbf{F}^P = \mathbf{I} + 2s \mathbf{F}^{0-T} \left(\frac{\partial f}{\partial \mathbf{S}} \right) \mathbf{F}^{0-1} \quad (3.16)$$

\mathbf{F}^P can also be explicitly solved from equation (3.15) for the particular case of membrane wrinkling. It should be pointed out that the two post-saturated strain tensors, $\Delta \mathbf{C}^P$ obtained from the additive decomposition of the Cauchy-Green strain tensor and $\mathbf{F}^{P^T} \mathbf{F}^P$ obtained from the multiplicative decomposition of the deformation gradient, are always different, their relation is defined by equation (3.16).

3.2 The wrinkling theory of anisotropic membranes

3.2.1 The wrinkling condition

When the saturated elasticity theory is applied to the wrinkling of membranes, the wrinkling phenomenon can be viewed as a particular case of the saturated elasticity and the analysis can thus be carried out in a purely two-dimensional context, whereby the spaces of symmetric second order tensors become three-dimensional. The function f in the saturation condition now becomes the determinant function of the stress tensor and the constant k is equal to zero. The inequality in the saturation condition is reversed. The saturation condition is replaced in this case by the wrinkling condition:

$$S_{11}S_{22} - S_{12}S_{21} \geq 0 \quad (3.17)$$

By substituting equation (3.2) into the wrinkling condition, one obtains:

$$\frac{\partial W}{\partial C_{11}} \frac{\partial W}{\partial C_{22}} - \frac{\partial W}{\partial C_{12}} \frac{\partial W}{\partial C_{21}} \geq 0 \quad (3.18)$$

where the assumption that the dependence of W on C_{12} and C_{21} has been symmetrized has been adopted, so that $\frac{\partial W}{\partial C_{12}} = \frac{\partial W}{\partial C_{21}}$. Since the membrane has no bending stiffness and cannot withstand any in-plane compressive stresses, the product and sum of the principal stresses, which are equivalent to the determinant and trace of the stress tensor, must be non-negative. The first condition is implemented by the wrinkling condition, while the second condition can be handled by directly excluding a “forbidden zone”.

3.2.2 The stress and strain spaces

The equality in the wrinkling condition defines a surface Σ in the stress space and a surface Σ' in the strain space through the constitutive law. The surface Σ in the three-dimensional stress space with coordinates S_{11} , S_{22} , $\sqrt{2}S_{12} = \sqrt{2}S_{21}$ is a right-angled cone with its axis being the bisector of S_{11} and S_{22} . The cone has two sheets, which divide the stress space into three zones. The one, whose intersection with the plane $S_{12} = 0$ falls in the third quadrant, encloses the forbidden zone of double wrinkling; the other, whose intersection with the plane $S_{12} = 0$ falls in the first quadrant, encloses the no wrinkling zone; between these two sheets is the single wrinkling zone. The shape of surface Σ' depends on the particular constitutive equation of the membrane, and is obtained by the

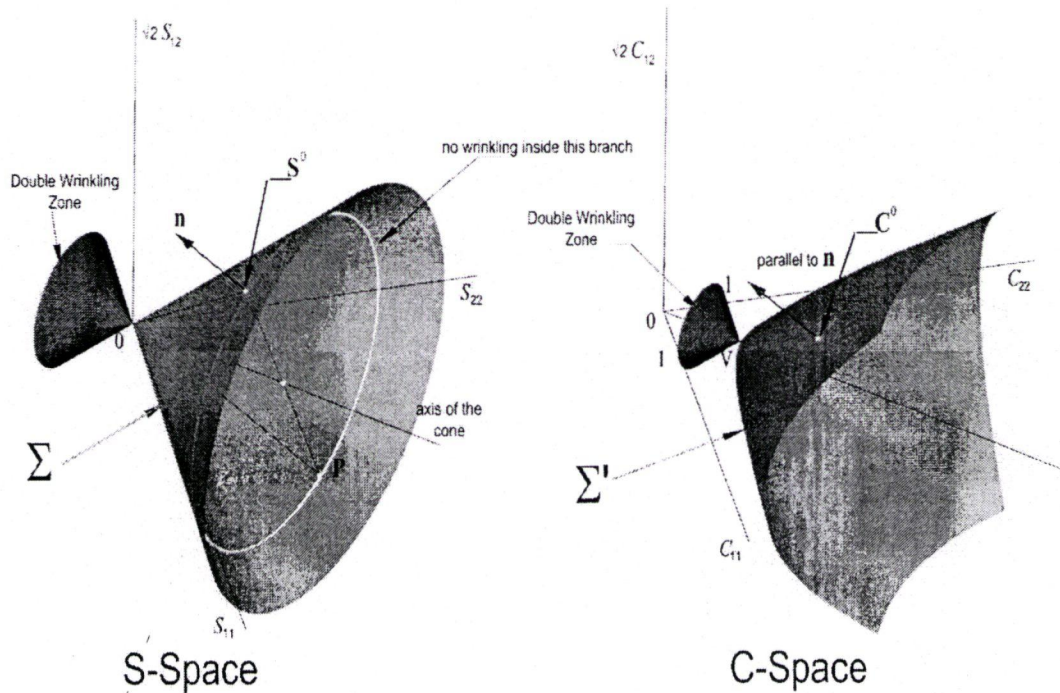


Figure 3.1 Impending-wrinkling surfaces in the stress and strain space

strict equality in wrinkling condition. Assuming the reference configuration to be in a natural state, the vertex O of the cone Σ is mapped to the point V in Σ' with coordinates $(1, 1, 0)$. At this point the normality rule implies the emergence of a forbidden cone which is identical with its counterpart in the stress space as shown in Figure 3.1.

Within the double wrinkling zone both principal stresses are negative, the membrane is not stretched in any direction at all, and the configuration can not be determined through equilibrium. So the double wrinkling zone should be excluded as a forbidden zone and the strain energy density is set zero. Within the no wrinkling zone both principal stresses are positive, the membrane is stretched in all direction, and the strain energy density is exactly the originally given strain energy density. In single wrinkling zone, one of the principal

stresses is zero while the other is positive, the membrane is in a uniaxial tension. The original constitutive relationship is no longer effective in this zone, but the strain energy function is replaced by a relaxed energy function. The relaxed energy function is a C^1 continuation of the original strain energy function, and keeps constant along the characteristic lines. Thus the value of the relaxed energy function at a given point in the single wrinkling zone, is just the value of the original strain energy function at the intersection point C^0 of the characteristic line passing through the given point and the impending wrinkling surface Σ' . The characteristic line is parallel to the normal n at the corresponding point S^0 of the intersection point C^0 on impending wrinkling surface Σ in stress space.

A plane perpendicular to the axis of the cone will intersect with the cone on an intersection circle, which represents a plane stress state. Given that the square root of 2 scaling is removed, the projection of such a circle on the S_{11}, S_{12} plane coincides with Mohr's circle. Thus, different points on the circle represent states of stress rotated rigidly with respect to each other. On the other hand, two different points could be obtained through rotating the circle around the axis of the cone a certain angle θ , and the connection between the coordinates of these two points in stress space is given by following matrix:

$$\begin{bmatrix} \cos^2 \theta & \sin^2 \theta & \sqrt{2} \sin \theta \cos \theta \\ \sin^2 \theta & \cos^2 \theta & -\sqrt{2} \sin \theta \cos \theta \\ \sqrt{2} \sin \theta \cos \theta & -\sqrt{2} \sin \theta \cos \theta & \cos^2 \theta - \sin^2 \theta \end{bmatrix} \quad (3.19)$$

Therefore, different points on the circle can also represent the same state of stress in different orthogonal coordinate systems. Since the cone is the impending wrinkling surface, any point on the cone in the first quadrant represents a uniaxial state of stress. This is clear on a Mohr's circle. The projection of such a circle on the S_{11}, S_{12} plane must always pass through the origin because the cone is right angled, so the second principal stress is zero, and the circle represents a uniaxial state of stress.

3.2.3 The wrinkling strain

Since the cone Σ is right angled, a vector normal to the cone at a point S^0 coincides with the direction of a uniaxial state of stress represented by P , which is the symmetrical point of S^0 about the axis of the cone as shown in Figure 3.1. Point P can be obtained by rotating the circle by 180 degrees from S^0 , so the first principal direction of the uniaxial state of stress at P coincides with the second principal direction of the uniaxial state of stress at S^0 . Combining this fact with the normality rule, it turns out that the wrinkling (post-saturated) additive component of the strain should be of the form:

$$\Delta C^W = -\gamma^2 \mathbf{u} \otimes \mathbf{u} \quad (3.20)$$

where \mathbf{u} is the unit eigenvector of S^0 corresponding to the zero principal stress, and γ^2 is a scalar parameter, whose range of values is determined by the fact that the total strain tensor must keep always positive definite. Next, as has been pointed out previously, equation (3.15) can be solved explicitly for F^P in the case of membrane wrinkling. To stress that the results are only valid for the wrinkling case, we now denote F^P by F^W

and define

$$\mathbf{F}^W = \mathbf{Q}(e^{-\alpha^2} \mathbf{e} \otimes \mathbf{e} + \mathbf{g} \otimes \mathbf{g}) \quad (3.21)$$

where \mathbf{Q} is an arbitrary rotation, and \mathbf{e} and \mathbf{g} are unit eigenvectors of the Cauchy stress tensor $\boldsymbol{\sigma}$ corresponding, respectively, to the zero and the positive principal stresses at $\mathbf{C} = \mathbf{C}^0$. The parameter α^2 is a measure of wrinkling while $\alpha = 0$ corresponds to impending wrinkling and $\alpha \rightarrow \infty$ indicates dimensional collapse. Notice that, the connection between the Cauchy stress tensor $\boldsymbol{\sigma}$ and the second Piola-Kirchhoff stress tensor \mathbf{S} can be expressed as:

$$\boldsymbol{\sigma} = J^{-1} \mathbf{F} \mathbf{S} \mathbf{F}^T \quad (3.22)$$

where J is the determinant of \mathbf{F} , so it turns out that the vector

$$\mathbf{u}' = \mathbf{F}^{0T} \mathbf{e} \quad (3.23)$$

is collinear with \mathbf{u} . In fact,

$$\mathbf{S} \mathbf{u}' = J \mathbf{F}^{0-1} \boldsymbol{\sigma} \mathbf{F}^{0-T} \mathbf{u}' = J \mathbf{F}^{0-1} \boldsymbol{\sigma} \mathbf{e} = 0 \quad (3.24)$$

so \mathbf{u}' is an eigenvector corresponding to the zero eigenvalue of \mathbf{S}^0 . Then we have:

$$\begin{aligned} \mathbf{C} &= \mathbf{F}^{0T} \mathbf{F}^{W T} \mathbf{F}^W \mathbf{F}^0 \\ &= \mathbf{F}^{0T} (e^{-2\alpha^2} \mathbf{e} \otimes \mathbf{e} + \mathbf{g} \otimes \mathbf{g}) \mathbf{F}^0 \\ &= \mathbf{F}^{0T} (\mathbf{I} - (1 - e^{-2\alpha^2}) \mathbf{e} \otimes \mathbf{e}) \mathbf{F}^0 \\ &= \mathbf{C}^0 - (1 - e^{-2\alpha^2}) \mathbf{u}' \otimes \mathbf{u}' \\ &= \mathbf{C}^0 - \gamma^2 \mathbf{u} \otimes \mathbf{u} \end{aligned} \quad (3.25)$$

where

$$\gamma^2 = (1 - e^{-2\alpha^2}) \sqrt{\mathbf{u}' \cdot \mathbf{u}'} \quad (3.26)$$

This shows that a deformation gradient of the form (3.21) is a solution of equation (3.15) via equation (3.20). In other words, if any other tensor $F^{W'}$ happens to be a solution of equation (3.15), it must be able to be resolved uniquely into an orthogonal and a positive definite symmetric part through polar decomposition. However, it follows immediately that the symmetric positive definite parts of $F^{W'}$ and F^W must be equal, and only the orthogonal parts can be different. This is exactly what is expressed by equation (3.21). Thus, equation (3.21) is the general solution of equation (3.15) for the case of membrane wrinkling. This also confirms the physically intuitive idea that, at constant saturated uniaxial stress, the wrinkles remain aligned with the principal tension. Moreover, since $k = 0$ in equation (3.11), the elastic energy remains constant throughout the wrinkling process at constant uniaxial stress.

3.2.4 Orthotropization technique

To introduce the anisotropy into the membrane wrinkling theory, let us consider a general isotropic hyperelastic constitutive law first,

$$W = W(\lambda_1, \lambda_2) \quad (3.27)$$

where λ_1 and λ_2 are the eigenvalues of the right Cauchy-Green strain tensor C . The relation between the Lagrangian strain tensor L and the right Cauchy-Green strain tensor C is

$$L = \frac{1}{2}(C - I) \quad (3.28)$$

where I is the identity tensor. Then the characteristic equation satisfied by those eigenvalues λ can be written as:

$$\det[2L - (\lambda - 1)I] = 0 \quad (3.29)$$

Now we introduce a material metric through a symmetric positive definite tensor M and define the modified eigenvalues μ satisfying the following characteristic equation,

$$\det[2L - (\mu - 1)MI] = 0 \quad (3.30)$$

Then define the orthotropized constitutive equation associated with the general isotropic constitutive law (3.27) via the material metric M as:

$$W' = W(\mu_1, \mu_2) \quad (3.31)$$

where the functional dependence W is the same as before, but where the eigenvalues λ have been replaced by the weighted eigenvalues μ . M can be written in its eigenbasis in the diagonal component form as:

$$M = \begin{pmatrix} \beta_1 & 0 \\ 0 & \beta_2 \end{pmatrix} \quad (3.32)$$

where $\beta_1 > 0$ and $\beta_2 > 0$ can be called the orthotropy parameters. The value $\beta_1 = 1$ and $\beta_2 = 1$ correspond to isotropy. By solving the modified characteristic equation (3.30), the weighted eigenvalues μ corresponding to the orthotropized strain space can be

obtained as:

$$\begin{aligned} \mu_{1,2} - 1 &= \frac{E_{11}}{\beta_1} + \frac{E_{22}}{\beta_2} \pm \sqrt{\left(\frac{E_{11}}{\beta_1} + \frac{E_{22}}{\beta_2}\right)^2 - 4 \frac{E_{11}E_{22} - E_{12}E_{21}}{\beta_1\beta_2}} \\ &= \frac{C_{11} - 1}{2\beta_1} + \frac{C_{22} - 1}{2\beta_2} \pm \sqrt{\left(\frac{C_{11} - 1}{2\beta_1} + \frac{C_{22} - 1}{2\beta_2}\right)^2 - \frac{(C_{11} - 1)(C_{22} - 1) - C_{12}C_{21}}{\beta_1\beta_2}} \end{aligned} \quad (3.33)$$

3.3 Application to the Neo-Hookean material

The Neo-Hookean material is taken as an example here to investigate its impending wrinkling surface.

The strain energy potential of the Neo-Hookean material in 2 dimension can be written as:

$$W = \frac{1}{2}G(\lambda_1 + \lambda_2 + \lambda_1^{-1}\lambda_2^{-1} - 3) \quad (3.34)$$

where G is a positive constant with dimensions of force/area (the shear modulus for infinitesimal strain). Through replacing the eigenvalues λ with the weighted eigenvalues μ , the strain energy function can be orthotropized as:

$$W = \frac{1}{2}G\left(\frac{C_{11} - 1}{\beta_1} + \frac{C_{22} - 1}{\beta_2} - 1 + \frac{\beta_1\beta_2}{\Delta}\right) \quad (3.35)$$

where $\Delta = (C_{11} - 1)(C_{22} - 1) - C_{12}C_{21} + \beta_1\beta_2 + \beta_2(C_{11} - 1) + \beta_1(C_{22} - 1)$. The derivatives can be derived as:

$$\frac{1}{2}S_{11} = \frac{\partial W}{\partial C_{11}} = \frac{1}{2}G \left(\frac{1}{\beta_1} - \frac{\beta_1\beta_2(C_{22}-1+\beta_2)}{\Delta^2} \right) \quad (3.36)$$

$$\frac{1}{2}S_{22} = \frac{\partial W}{\partial C_{22}} = \frac{1}{2}G \left(\frac{1}{\beta_2} - \frac{\beta_1\beta_2(C_{11}-1+\beta_1)}{\Delta^2} \right) \quad (3.37)$$

$$\frac{1}{2}S_{12} = \frac{\partial W}{\partial C_{12}} = \frac{1}{2}G \frac{\beta_1\beta_2 C_{21}}{\Delta^2} \quad (3.38)$$

Thus the impending wrinkling surface Σ' for Neo-Hookean material can be given by:

$$\Delta^3 - \beta_1\beta_2(\beta_1(C_{22}-1+\beta_2) + \beta_2(C_{11}-1+\beta_1))\Delta + \beta_1^3\beta_2^3 = 0 \quad (3.39)$$

Chapter FOUR

Finite Element Implementation

The wrinkling theory of anisotropic membrane is formulated using curvilinear coordinates for the general configuration, and implemented into a finite element code using the four-node bilinear isoparametric element. A scheme for determining the relaxed energy function is also presented.

4.1 The Strain Tensor in Curvilinear Coordinate System

Consider a membrane in a plane stress state. Let Ω_0 denote the reference configuration, which is assumed unstressed. Let Ω denote the deformed configuration. Assume the membrane moves in a three-dimensional Euclidean space. A convected curvilinear coordinate system ξ^1, ξ^2 is attached to the membrane surface.

The covariant base vectors of the convected coordinate system in reference and deformed configuration are defined, respectively, as

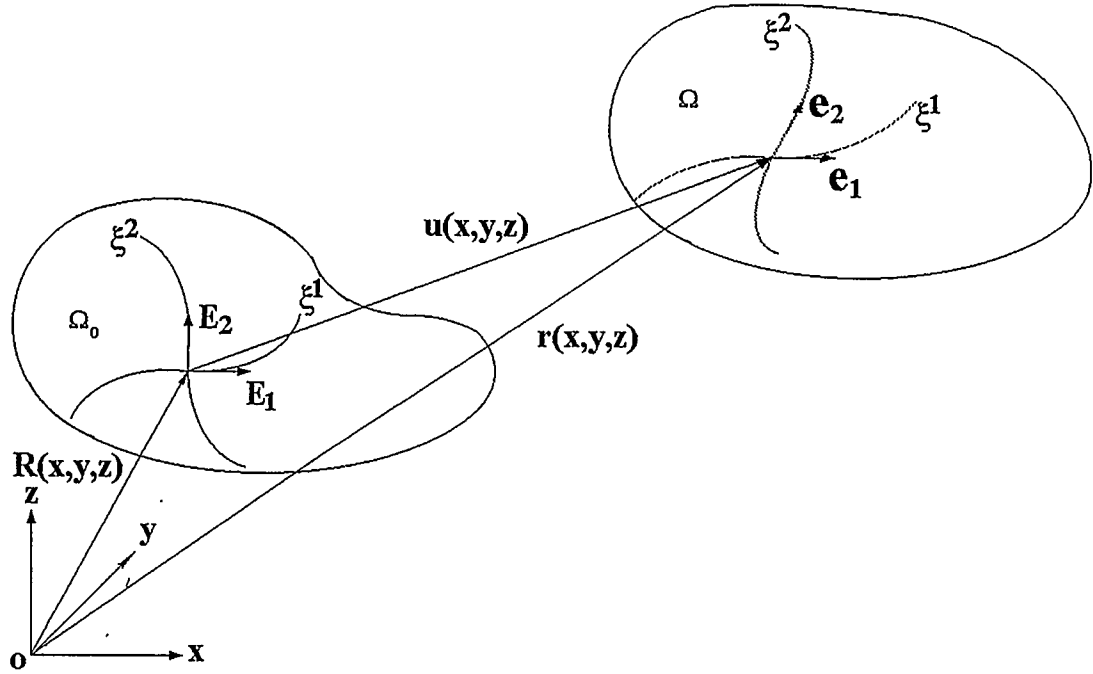


Figure 4.1 The convected curvilinear coordinate system

$$E_\alpha = \frac{\partial \mathbf{R}}{\partial \xi^\alpha}, \quad e_\alpha = \frac{\partial \mathbf{r}}{\partial \xi^\alpha} \quad (4.1)$$

where \mathbf{R} and \mathbf{r} are position vectors of a particle referred to origins in reference and deformed configuration, respectively. The covariant components of the metric tensor in reference and deformed configuration are defined, respectively, as

$$G_{\alpha\beta} = \mathbf{E}_\alpha \cdot \mathbf{E}_\beta, \quad g_{\alpha\beta} = \mathbf{e}_\alpha \cdot \mathbf{e}_\beta \quad (4.2)$$

with their determinants as

$$G = |G_{\alpha\beta}|, \quad g = |g_{\alpha\beta}| \quad (4.3)$$

The construction of the convected coordinate system assumes that G does not vanish

on Ω_0 . The contravariant components of metric tensors in reference configuration are obtained from

$$[G^{\alpha\beta}] = [G_{\alpha\beta}]^{-1} \quad (4.4)$$

and the contravariant base vectors are obtained from

$$E^\alpha = G^{\alpha\beta} E_\beta \quad (4.5)$$

The deformation gradient tensor F on Ω is

$$F = e_\alpha \otimes E^\alpha \quad (4.6)$$

The right Cauchy-Green strain tensor C is obtained from

$$C = F^T \cdot F = g_{\alpha\beta} E^\alpha \otimes E^\beta = g_{\alpha\beta} G^{\alpha\rho} G^{\beta\sigma} E_\rho \otimes E_\sigma \quad (4.7)$$

For an orthotropic material, the covariant base vectors E_α can be expressed in the unit base vectors M_ξ of orthotropy as

$$E_\alpha = M_\alpha^\xi M_\xi \quad (4.8)$$

So the right Cauchy-Green strain tensor C becomes

$$C = g_{\alpha\beta} G^{\alpha\rho} G^{\beta\sigma} E_\rho \otimes E_\sigma = g_{\alpha\beta} G^{\alpha\rho} G^{\beta\sigma} M_\rho^\xi M_\sigma^\eta M_\xi \otimes M_\eta = C^{\xi\eta} M_\xi \otimes M_\eta \quad (4.9)$$

4.2 Geometry of Membrane Element

A total Lagrangian formulation and a displacement-based isoparametric finite element formulation are adopted here. Both the fixed Cartesian coordinate system x, y, z in the Euclidean space and the natural coordinate system ξ^1, ξ^2 attached to the membrane element are the same as before (see Figure 4.1). The coordinate interpolations on Ω_0 and Ω and the displacement interpolation are:

$$\mathbf{R} = \sum_{I=1}^q N_I \mathbf{R}_I \quad (4.10)$$

$$\mathbf{r} = \sum_{I=1}^q N_I \mathbf{r}_I \quad (4.11)$$

$$\mathbf{u} = \sum_{I=1}^q N_I \mathbf{u}_I \quad (4.12)$$

where N_I is the interpolation function for the I th node and q is the total number of nodes of the element. The covariant base vectors of the natural coordinate system on Ω_0 and Ω are:

$$\mathbf{E}_\alpha = \sum_{I=1}^q N_{I,\alpha} \mathbf{R}_I \quad (4.13)$$

$$\mathbf{e}_\alpha = \sum_{I=1}^q N_{I,\alpha} \mathbf{r}_I \quad (4.14)$$

where

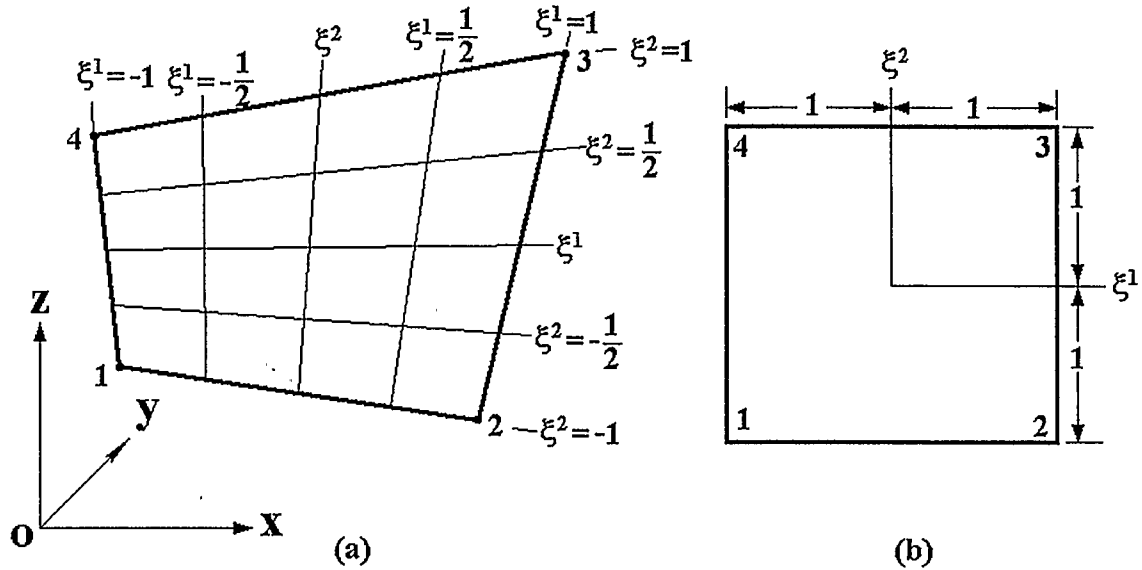


Figure 4.2 Four-node space bilinear isoparametric element

(a) in xyz space (b) in $\xi^1 \xi^2$ space

$$N_{,\alpha}^I = \frac{\partial N^I}{\partial \xi^\alpha} \quad (4.15)$$

Here the four-node space bilinear isoparametric element is chosen. The isoparametric coordinates in the x, y, z space are shown in Figure 4.2a. For a four-node isoparametric element, axes ξ^1 and ξ^2 pass through midpoints of opposite sides. Axes ξ^1 and ξ^2 are arbitrary curvilinear axes. Sides of the element are at $\xi^1 = \pm 1$ and at $\xi^2 = \pm 1$. Coordinates x, y and z within the element are defined by

$$x = \sum_{I=1}^4 x_I N_I \quad (4.16)$$

$$y = \sum_{I=1}^4 y_I N_I \quad (4.17)$$

$$z = \sum_{I=1}^4 z_I N_I \quad (4.18)$$

The individual shape functions are

$$N_1 = \frac{1}{4}(1-\xi^1)(1-\xi^2) \quad (4.19)$$

$$N_2 = \frac{1}{4}(1+\xi^1)(1-\xi^2) \quad (4.20)$$

$$N_3 = \frac{1}{4}(1+\xi^1)(1+\xi^2) \quad (4.21)$$

$$N_4 = \frac{1}{4}(1-\xi^1)(1+\xi^2) \quad (4.22)$$

Let θ denote the angle between the unit base vector M_1 of orthotropy and the covariant base vector E_1 in a particular element (see Figure 4.3). Then according to their geometric relation, the covariant base vectors can be expressed in the unit base vectors of orthotropy as:

$$E_1 = |E_1| \cos \theta M_1 + |E_1| \sin \theta M_2 \quad (4.23)$$

$$E_2 = |E_2| \cos(\gamma + \theta) M_1 + |E_2| \sin(\gamma + \theta) M_2 \quad (4.23)$$

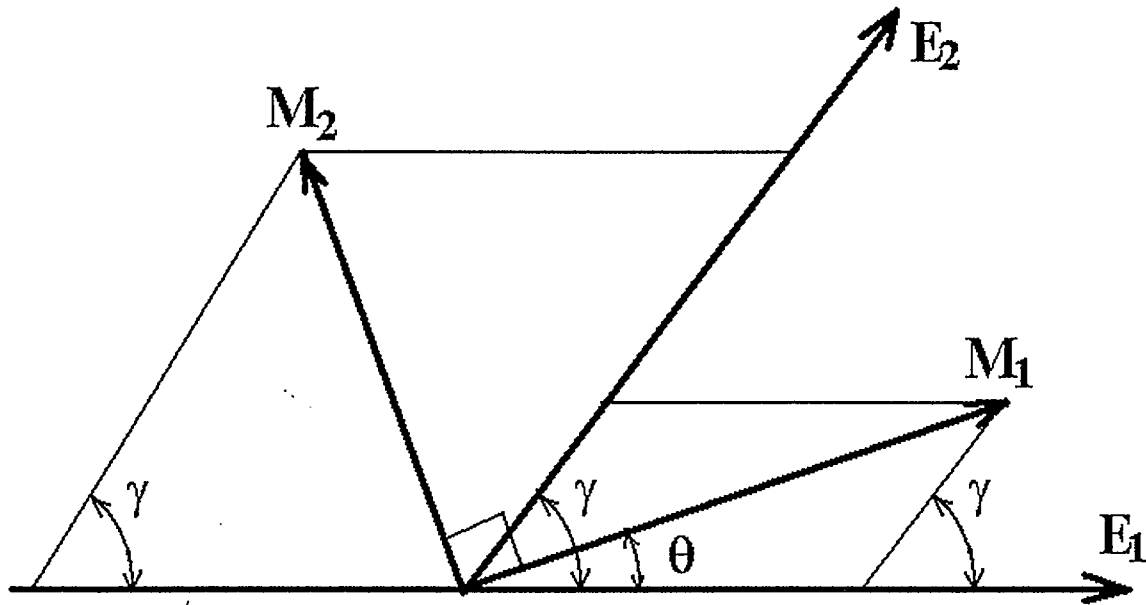


Figure 4.3 Co-variant base vectors and unit base vectors of orthotropy

where γ is the angle between the two covariant base vectors E_1 and E_2 . The positive direction of γ and θ is the counterclockwise direction.

4.3 Tangent Stiffness Matrix of Membrane Element

The total potential energy Π_p of a membrane is composed of the strain energy W and the potential V of external forces.

$$\Pi_p = W + V \quad (4.24)$$

According to the principle of stationary potential energy, the equilibrium configuration

can be found from the stationary value of Π_p :

$$\frac{\partial \Pi_p}{\partial \delta} = \frac{\partial W}{\partial \delta} + \frac{\partial V}{\partial \delta} = \frac{\partial W}{\partial C} \frac{\partial C}{\partial \delta} - \mathbf{R} = \mathbf{K}(\delta)\delta - \mathbf{R} = 0 \quad (4.25)$$

where δ is the displacement vector of the membrane, $\mathbf{K}(\delta)$ is the general stiffness matrix, also a function of the displacement state, and \mathbf{R} is the external force vector. The equilibrium equation (4.18) cannot be solved directly because of its double non-linearity. The double non-linearity lies on the geometric non-linearity that the general stiffness matrix depends on the displacement state and the material non-linearity that even within the wrinkling, a neo-Hookean material is nonlinear. Thus, the Newton-Raphson method is adopted to solve the equilibrium equation (4.18). The tangent stiffness matrix \mathbf{K}_T is obtained as:

$$\mathbf{K}_T = \frac{\partial^2 W}{\partial \delta^2} \quad (4.26)$$

with the coefficient $k_{T\delta_i^I \delta_j^J}$ corresponds to displacement components δ_i^I and δ_j^J being determined from

$$k_{T\delta_i^I \delta_j^J} = \int_A \frac{\partial^2 W}{\partial \delta_i^I \partial \delta_j^J} t dA = \int_A \left(\frac{\partial^2 W}{\partial C_{\alpha\beta}^2} \frac{\partial C_{\alpha\beta}}{\partial \delta_j^J} \frac{\partial C_{\rho\sigma}}{\partial \delta_i^I} + \frac{\partial W}{\partial C_{\rho\sigma}} \frac{\partial^2 C_{\rho\sigma}}{\partial \delta_i^I \partial \delta_j^J} \right) t dA \quad (4.27)$$

where A is the integration area of the membrane element, and t is the thickness of the membrane. For a given point in double wrinkling zone in strain space, its contribution to the coefficient $k_{T\delta_i^I \delta_j^J}$ is zero because of its zero strain energy density. For single wrinkling

case and no wrinkling case, the derivatives appeared in equation (4.20) can be evaluated numerically by means of the relaxed energy function and the original strain energy function, respectively.

4.4 The Scheme for Determining the Relaxed Energy Function

The value of the relaxed energy function at a given point C^G in single wrinkling zone equals to the value of the original strain energy function at the intersection point C^0 of the characteristic line passing through C^G and the impending wrinkling surface Σ' . The characteristic line is parallel to the normal n of Σ at the corresponding point S^0 . Based on these two facts, the iterative scheme for the determination of the relaxed energy function follows:

- (1) For a given point C^G in single wrinkling zone, set its initial direction n^0 which forms 45° with the bisector of C_{11} and C_{22} .
- (2) Shoot a straight line along n^0 from the given point C^G to intersect the bisector of C_{11} and C_{22} at C^I .
- (3) Find the intersection point C^0 of $C^G C^I$ and the surface Σ' through dichotomy. Construct the vector $C^0 - C^G$.
- (4) Calculate the normal n at the corresponding point S^0 of C^0 on Σ .
- (5) Calculate the cross product between the two vectors n and $C^0 - C^G$.

- (6) If the length of the product is less than a given tolerance, stop and calculate the strain energy density at C^0 .
- (7) Otherwise, change the direction n^0 to $\frac{1}{2}(n + C^0 - C^G)$ and repeat steps (2)~(7).

4.5 The Finite Element Program For the Wrinkling Analysis of Anisotropic Membranes

By using the finite element formulations obtained above, a finite element program for the wrinkling analysis of anisotropic membranes is developed from scratch with the Microsoft Visual C++ on the Windows' 98 platform. The finite element program consists of three parts—preprocessor, main analyzer and postprocessor. The preprocessor generates node coordinates data, finite element grid data, material property data and boundary data in a proper data format for the main analyzer. The main analyzer then completes the nonlinear wrinkling analysis and gives out the numerical results of stresses, strains and displacements. The postprocessor displays the wrinkling analysis results visually according to the numerical results. The postprocessor can also be used to check node coordinates data, finite element grid data and boundary data visually.

The code of these three parts is listed in the appendix.

Chapter FIVE

Examples

To verify the validity of the wrinkling theory of anisotropic membrane, several numerical examples are carried out in this chapter. Some of examples are taken from reference papers and the results obtained here agree well with those previously published results.

5.1 Example 1: A Square Membrane with Traction-free Boundaries

Consider a 1×1 m square membrane with traction-free horizontal boundaries. The thickness of the membrane is 0.001m. The shear modulus is 400MPa. The meshed reference configuration is shown in Figure 5.1. The membrane is first deformed by holding the left vertical boundary fixed, stretching horizontally and translating upward the right vertical boundary 0.1m and 0.2m respectively. The deformed mesh is shown in Figure 5.2.

The short lines in the elements in single wrinkling zone represent the magnitude of principal stress. The single wrinkling zone nearly covers the entire membrane. The computed maximum stress is 181MPa and occurred at the point of the lower left and upper right corner elements.

Next the right vertical boundary is stretched more with the total horizontal stretch being 0.2m while keeping its transverse translation unchanged. The result is the wrinkling is suppressed and the single wrinkling zone is limited to two free edges and the middle of the membrane as shown in Figure 5.3. The computed maximum stress is 235M Pa at the same points as before.

Further, the right vertical boundary is stretched and translated more and both of the stretch and translation are 0.5m now. The single wrinkling zones are further suppressed and limited only to two free edges as shown in Figure 5.4. The computed maximum stress is 341MPa.

Finally, the right vertical boundary is translated more to a total translation of 1.0m while its stretch is held unchanged. As expected, this increases the single wrinkling zones as shown in Figure 5.5. The maximum stress is 363M Pa in this final configuration. These figures agree well with previously published result by Haseganu and Steigmann(1994).

To test the sensitivity with the mesh, the same square membrane is meshed with a 5×5 grid and a 10×10 grid separately, and subjected to a simple stretch of 0.1m. The deformed meshes are shown in Figure 5.6 and 5.7. The maximum stresses are 114MPa and 125MPa respectively, and the difference is about 10%.

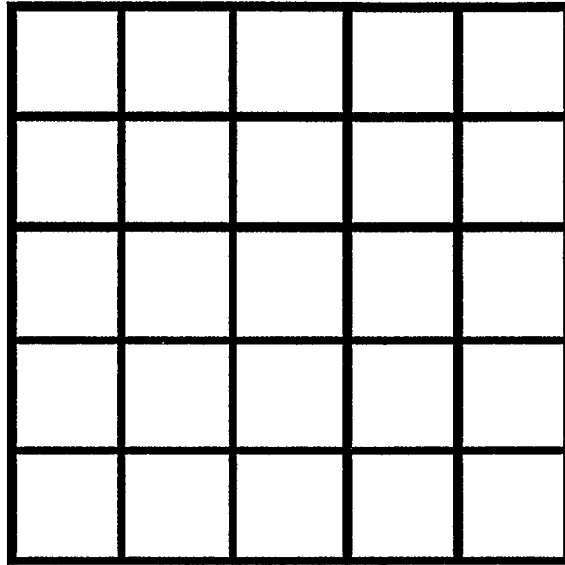


Figure 5.1 The meshed reference configuration of a square sheet with traction-free horizontal boundaries

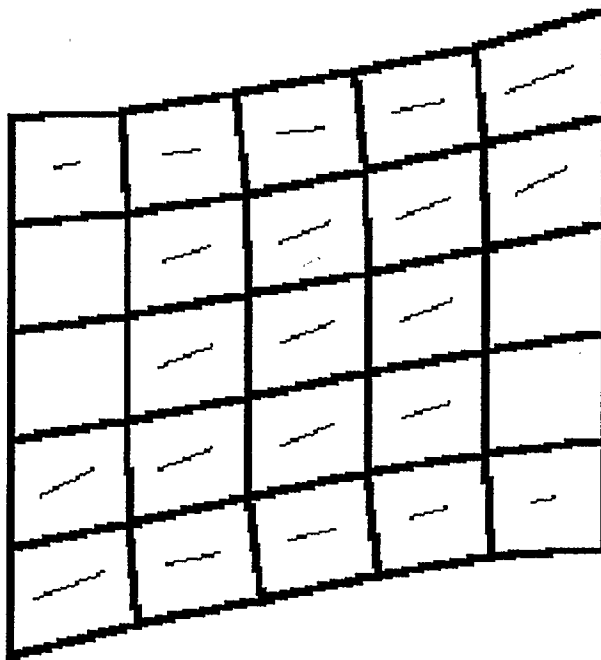


Figure 5.2 The deformed mesh for displacements $u = 0.1m$ and $v = 0.2m$

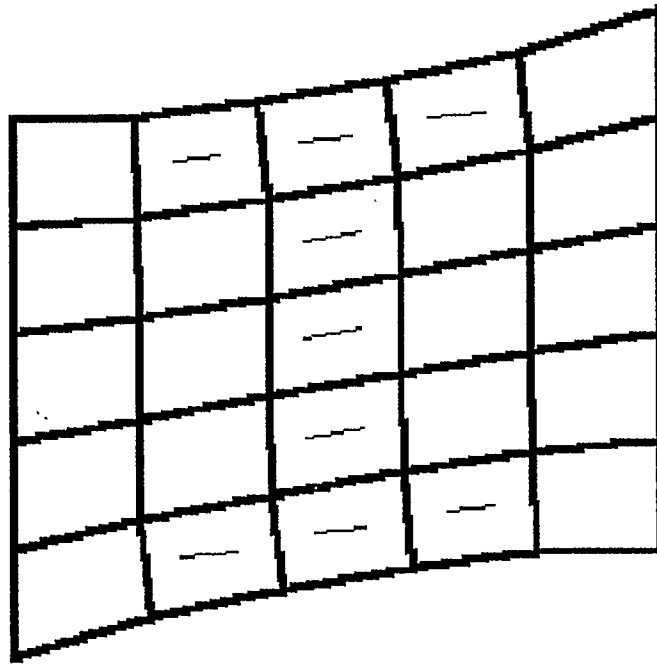


Figure 5.3 The deformed mesh for displacements $u = 0.2m$ and $v = 0.2m$

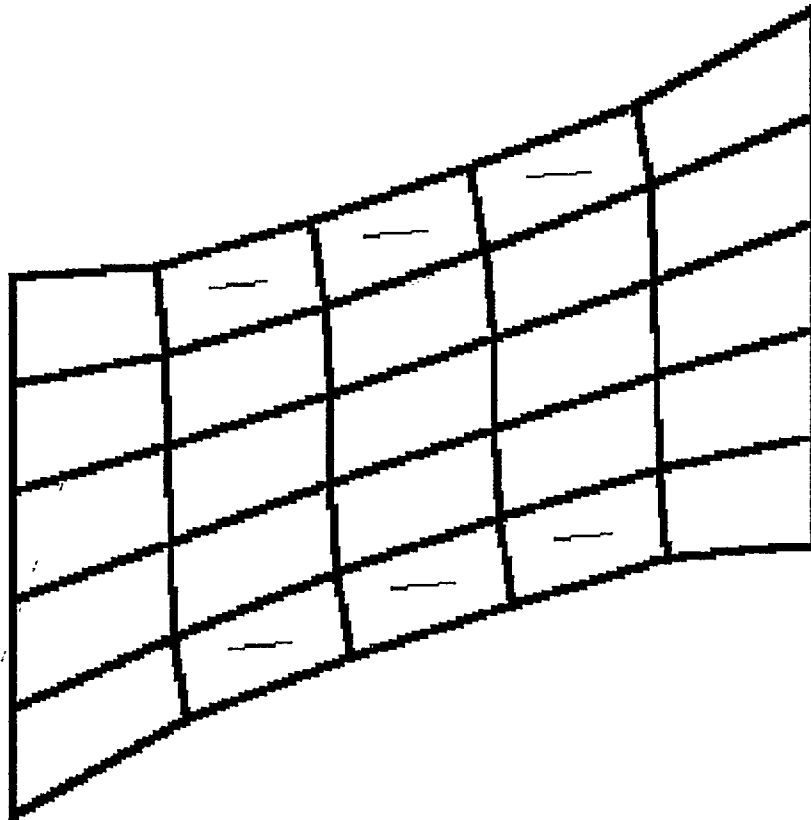


Figure 5.4 The deformed mesh for displacements $u = 0.5m$ and $v = 0.5m$

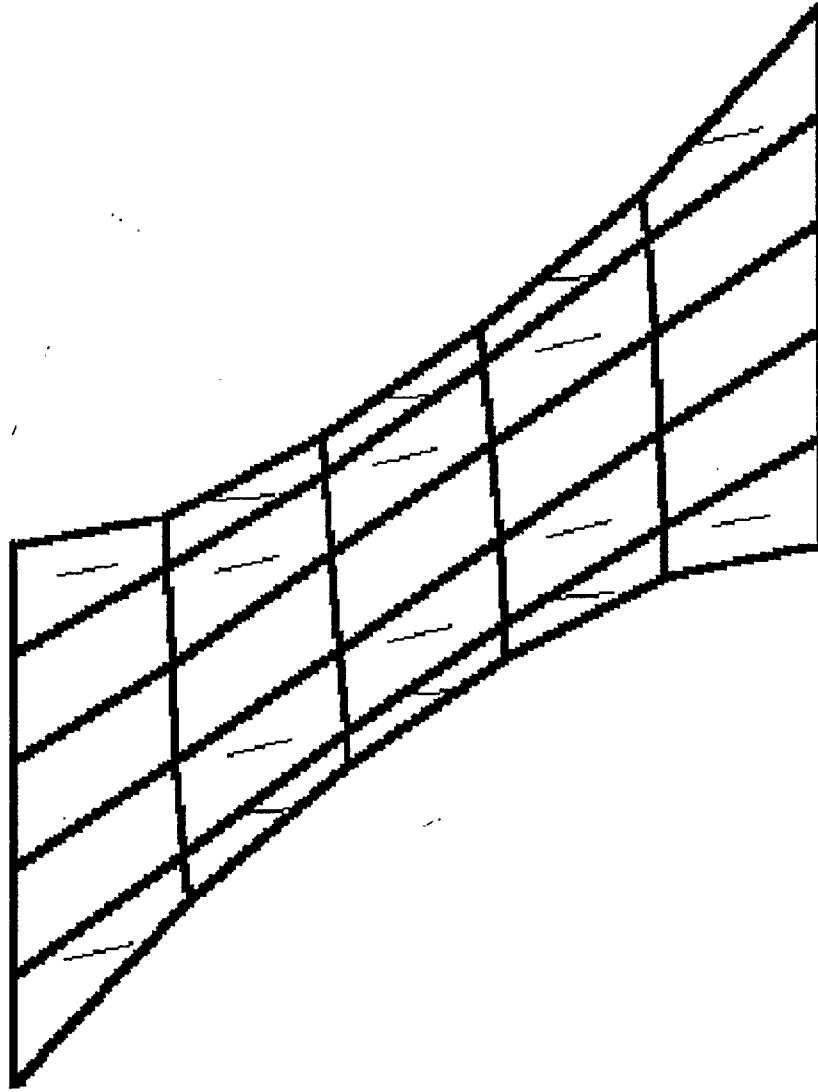


Figure 5.5 The deformed mesh for displacements $u = 0.5m$ and $v = 1.0m$

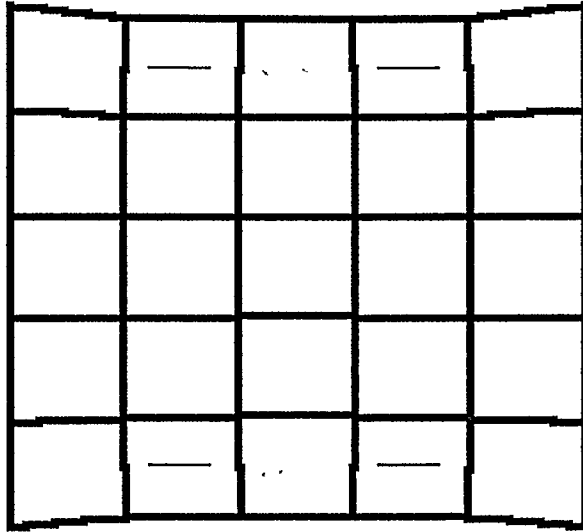


Figure 5.6 The deformed 5×5 mesh for 0.1m simple stretch

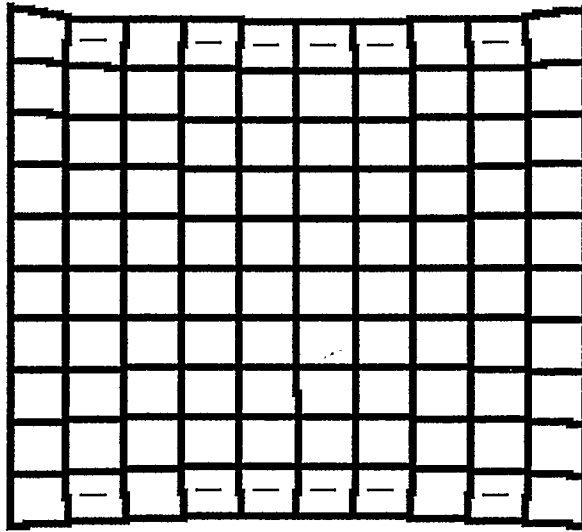


Figure 5.7 The deformed 10×10 mesh for 0.1m simple stretch

5.2 Example 2: An Annular Membrane with Concentric Circular Boundaries

Consider first an annular membrane bounded by concentric circles. The outer radius is 0.5m and the inner radius is 0.125m. The thickness is 0.001m. The shear modulus of the membrane is 40MPa. The meshed reference configuration is shown in Figure 5.8. The isotropic annular membrane is first deformed by holding the outer boundary fixed and rotating the inner boundary rigidly 45° counterclockwise. The deformed configuration is shown in Figure 5.9. The short lines indicate the first principal direction in the single wrinkling zone and the length of these lines is proportional to the first principal stress at each point. The computed maximum first principal stress is 23MPa and occurred at the center points of elements along the inner boundaries.

Then the inner boundary is lifted up 0.3m. The corresponding deform configuration is shown in Figure 5.10 while the single wrinkling zone is shown in a plane figure in Figure 5.11. Due to the radial stretch, the single wrinkling zone is suppressed. That is similar to the stretching and shearing of the square membrane. The maximum stress is 29.7MPa and occurred at the center points of elements along the outer boundaries.

Next consider an anisotropic annular membrane with orthotropy parameters $\beta_1 = 1.5$ and $\beta_2 = 0.75$. The inner boundary is also rotated 45° counterclockwise while the outer boundary is holding fixed. The deformed configuration is shown in Figure 5.12. The single wrinkling zone did not change in this case, but the stress field changed a lot. The

maximum stress is 31MPa and occurred at the center points of two elements by the outer boundaries along the weak direction of the orthotropy \mathbf{e}_2 .

Then switch these two orthotropy parameters, that is $\beta_1 = 0.75$ and $\beta_2 = 1.5$. This is just the case of turning the reference configuration 90° and the resulting single wrinkling zone reflected this kind of geometric relation as shown in Figure 5.13. The maximum stress is the same as before but occurred at the center points of two elements by the outer boundaries along the weak direction of the orthotropy \mathbf{e}_1 . These figures conform to the results of Lu et al(2001).

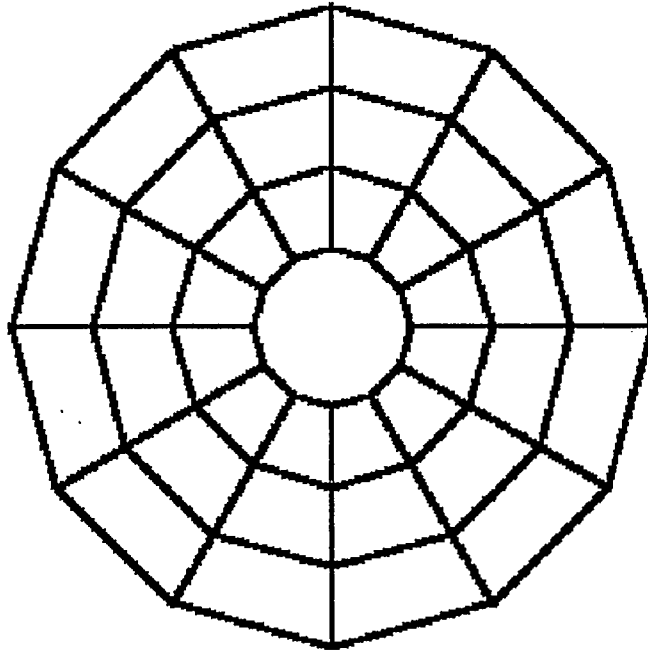


Figure 5.8 The meshed reference configuration of an annular membrane

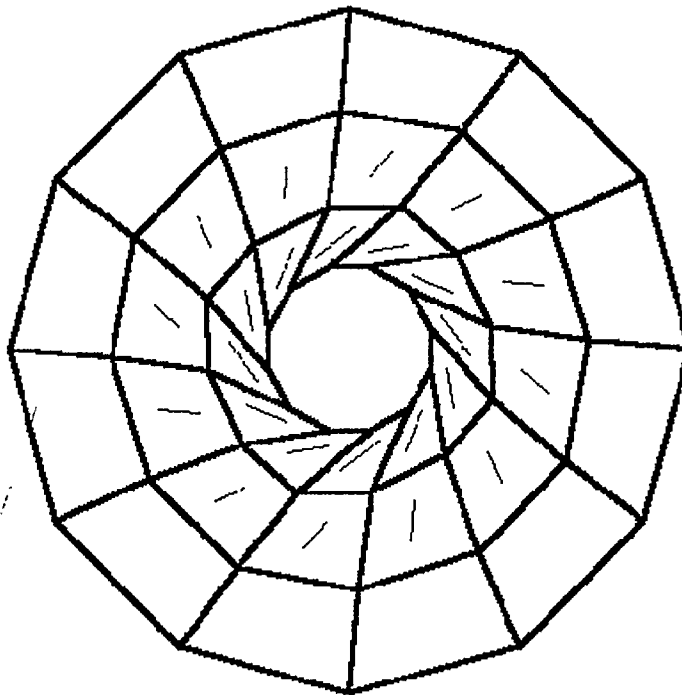


Figure 5.9 The deformed configuration for 45° counterclockwise rigid rotation of the inner boundary for isotropic case

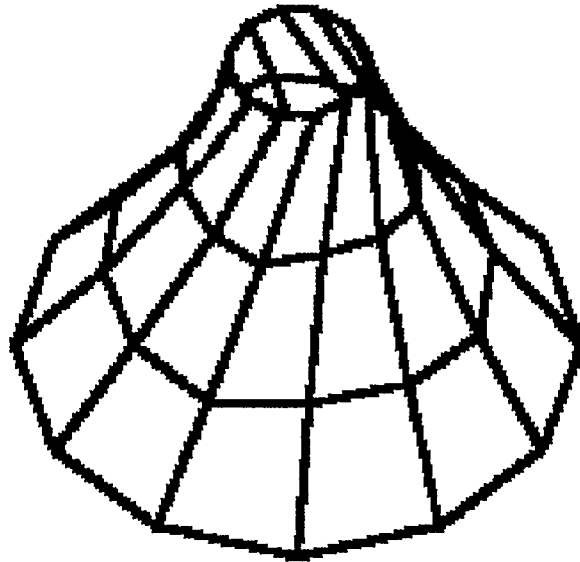


Figure 5.10 The deformed configuration for 0.3m lateral deflection and 45° counterclockwise rigid rotation of the inner boundary

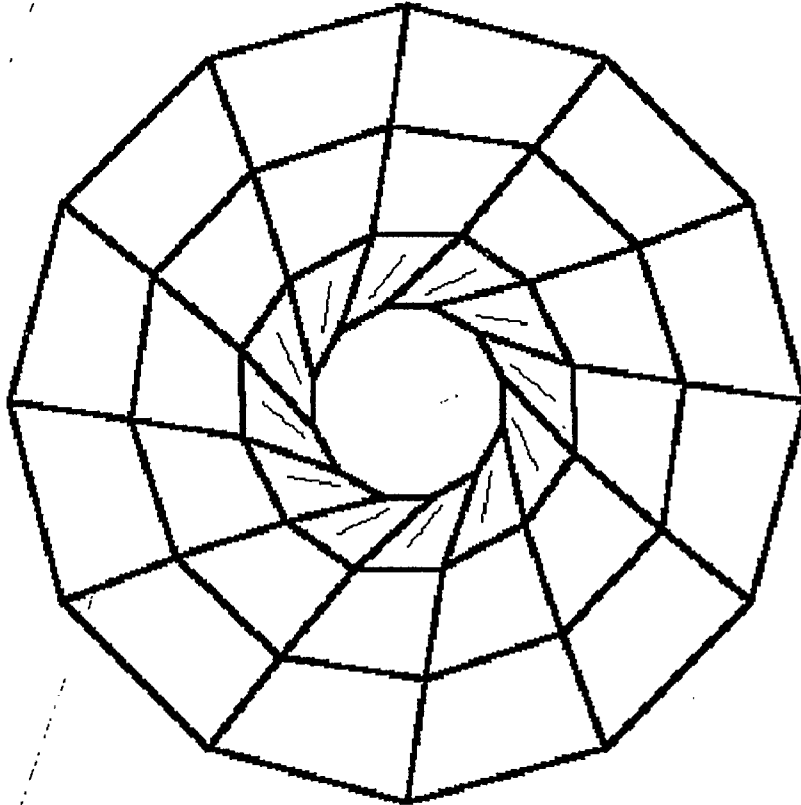


Figure 5.11 The single wringing zone of Figure 5.8 in a plane figure

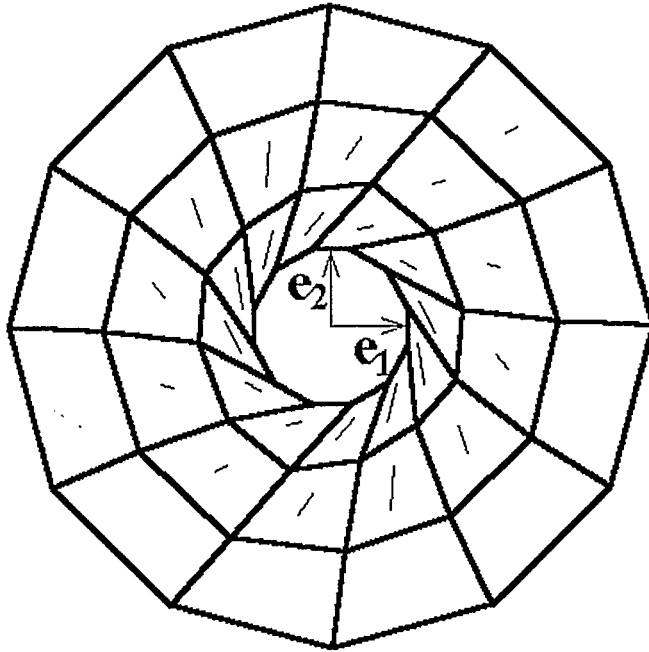


Figure 5.12 The deformed configuration for 45° counterclockwise rigid rotation of the inner boundary for anisotropic case $\beta_1 = 1.5, \beta_2 = 0.75$

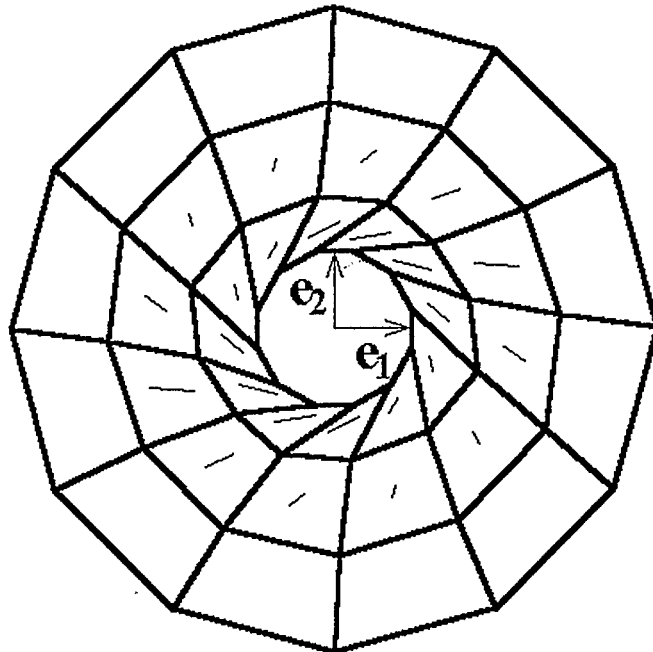


Figure 5.13 The deformed configuration for 45° counterclockwise rigid rotation of the inner boundary for anisotropic case $\beta_1 = 0.75, \beta_2 = 1.5$

5.3 Example 3: An Annular Membrane with Eccentric Circular Boundaries

Another example is the twisting of an annular membrane with eccentric circular boundaries. The outer radius is 0.5m and the inner radius is 0.125m. The center of the inner boundary is moved to (0.2, 0.0). The membrane is deformed by rotating and lifting the inner boundary rigidly 30° counterclockwise and 0.1m upward respectively while keeping the outer boundary fixed. Figure 5.14 gives the meshed reference configuration while the other three figures 5.15, 5.16 and 5.17 shows the deformed configuration for $\beta_1 = 1, \beta_2 = 1$, $\beta_1 = 1.5, \beta_2 = 0.75$ and $\beta_1 = 0.75, \beta_2 = 1.5$ respectively. The maximum stresses are 30.4M Pa, 38.2M Pa and 45M Pa respectively.

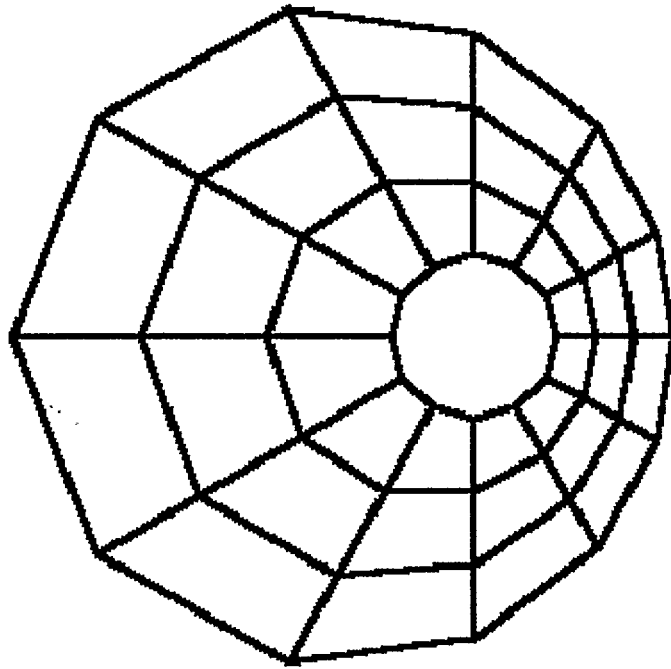


Figure 5.14 The meshed reference configuration of an annular membrane with eccentric circular boundaries

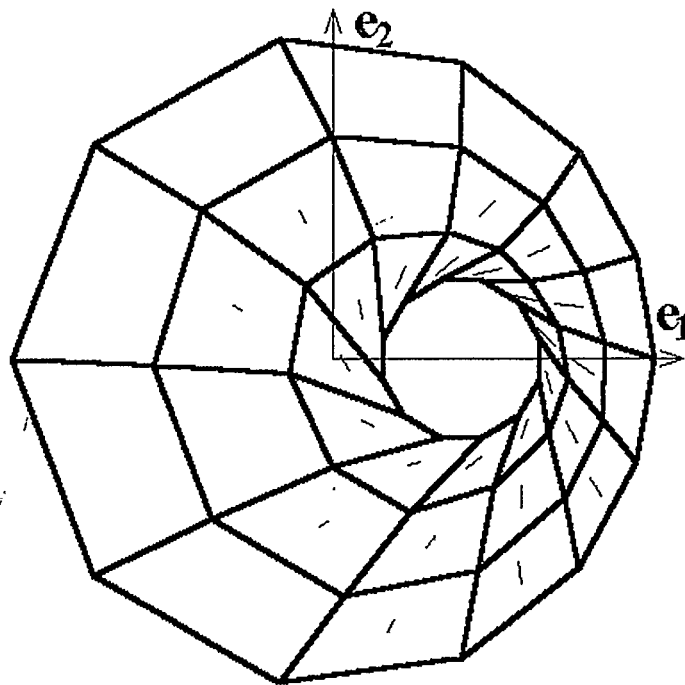


Figure 5.15 The deformed configuration for isotropic case

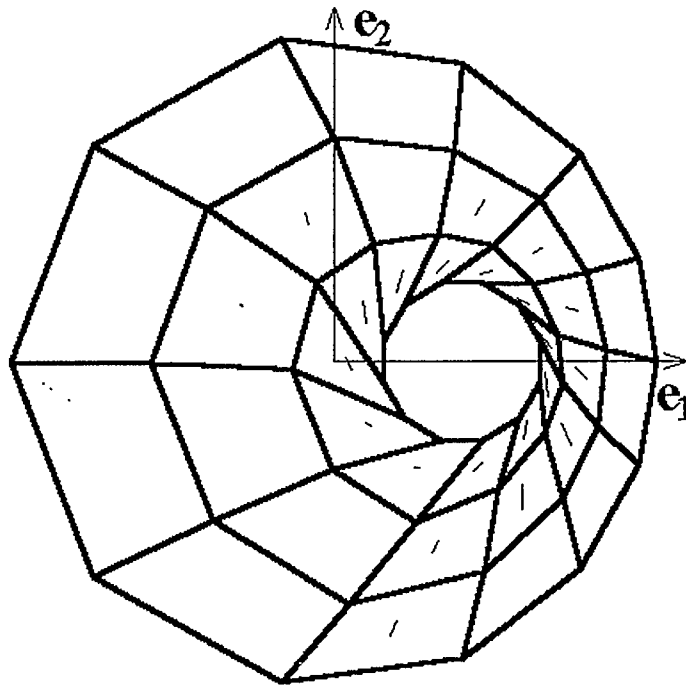


Figure 5.16 The deformed configuration for $\beta_1 = 1.5, \beta_2 = 0.75$

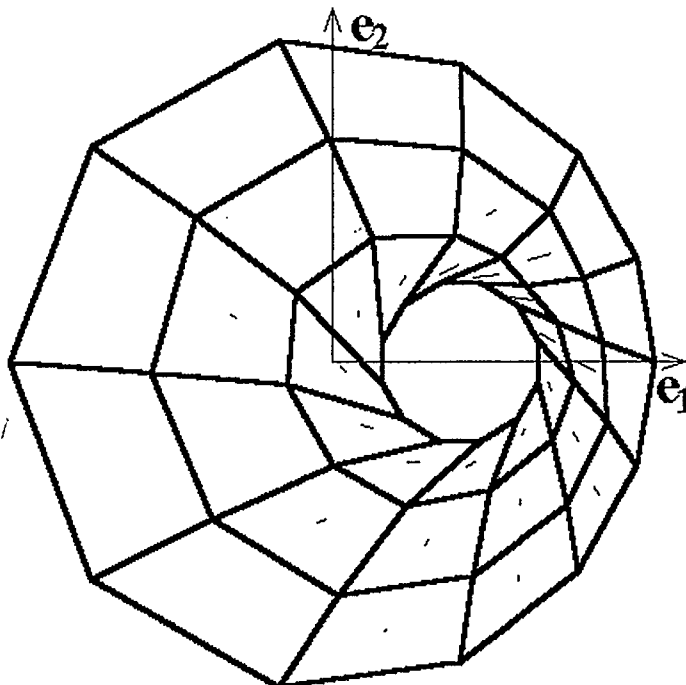


Figure 5.17 The deformed configuration for $\beta_1 = 0.75, \beta_2 = 1.5$

5.4 Example 4: An Annular Square Membrane

Consider a 2.0×2.0 m square membrane with a 0.7×0.7 m central square hole. The thickness is 0.001m and the shear modulus of the membrane is 400MPa. The meshed reference configuration is shown in Figure 5.18. The double arrow marked with β_1 and β_2 indicates the orthotropy direction of the material on the quarter of the square membrane. The annular square membrane is deformed by holding the outer boundary fixed, rotating the inner boundary rigidly 45° counterclockwise and lifting the inner boundary upward 0.5m. The deformed configuration for isotropic case, $\beta_1 = \beta_2 = 1.0$, is shown in Figure 5.19. The short lines indicate the first principal direction in the single wrinkling zone and the length of these lines is proportional to the first principal stress at each point. The computed maximum first principal stress is 345MPa and occurred at the center points of elements on the center of the outer boundaries. The deformed configuration for orthotropic cases, $\beta_1 = 0.8, \beta_2 = 1.0$, $\beta_1 = 1.2, \beta_2 = 1.0$, $\beta_1 = 1.0, \beta_2 = 0.8$ and $\beta_1 = 1.0, \beta_2 = 1.2$, are shown in Figure 5.20, 5.21, 5.22 and 5.23 respectively.

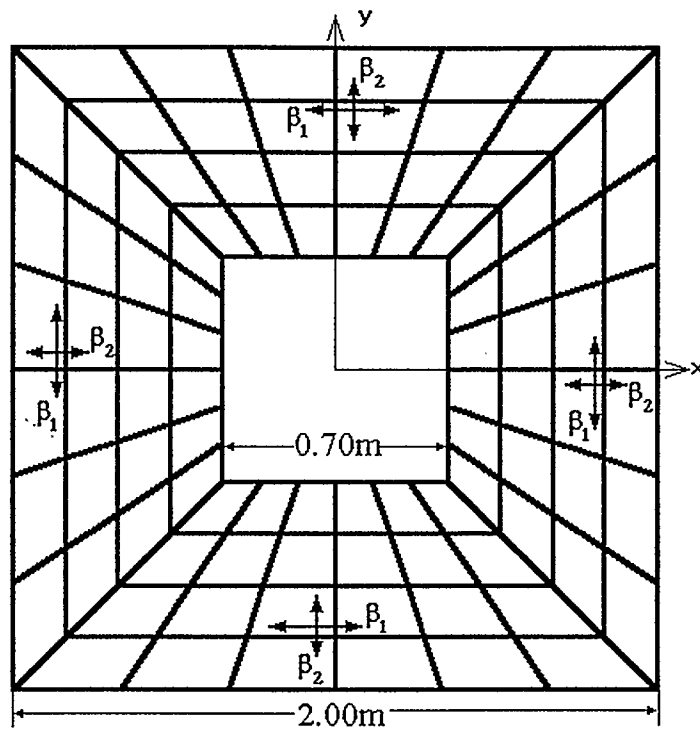


Figure 5.18 The meshed reference configuration of an annular square membrane

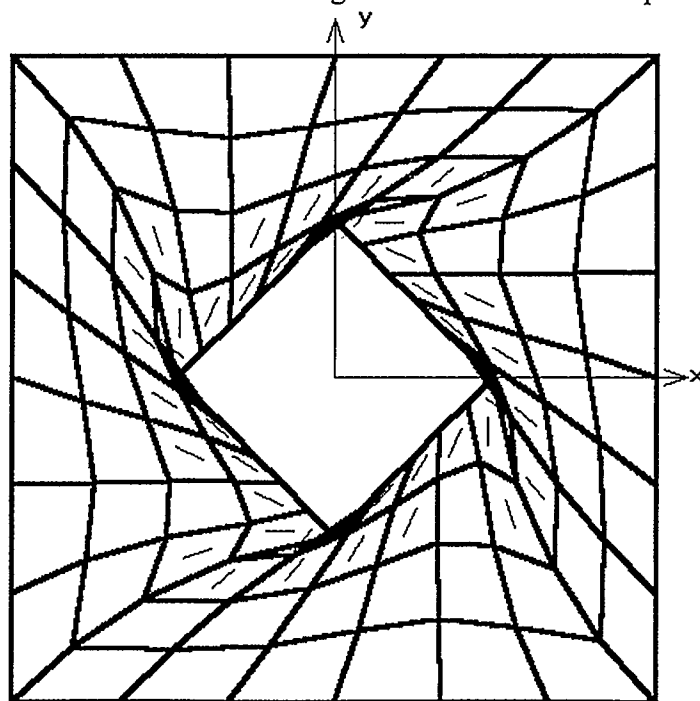


Figure 5.19 The deformed configuration for isotropic case, $\beta_1 = \beta_2 = 1.0$

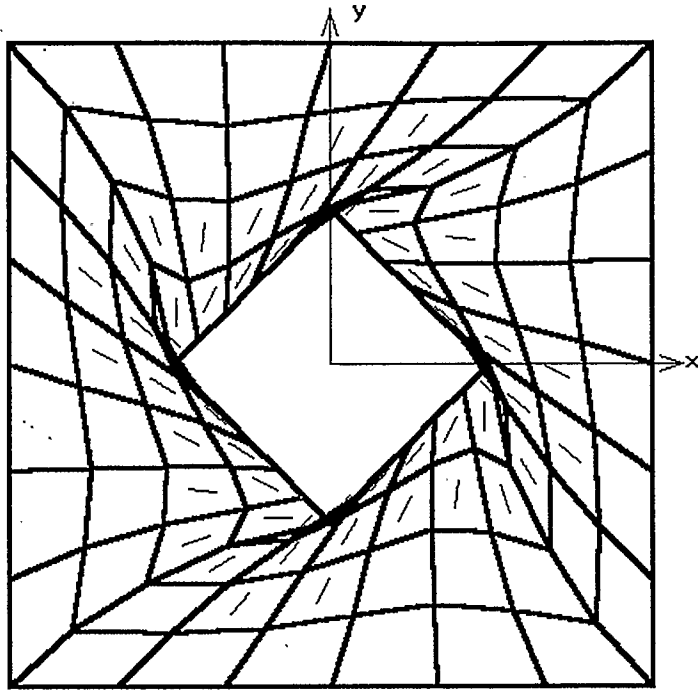


Figure 5.20 The deformed configuration for anisotropic case, $\beta_1 = 0.8, \beta_2 = 1.0$

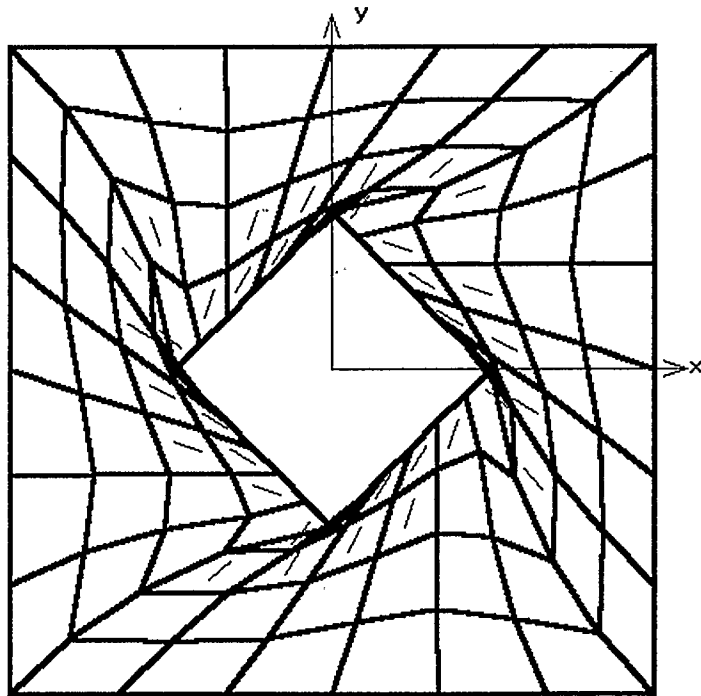


Figure 5.21 The deformed configuration for anisotropic case, $\beta_1 = 1.2, \beta_2 = 1.0$

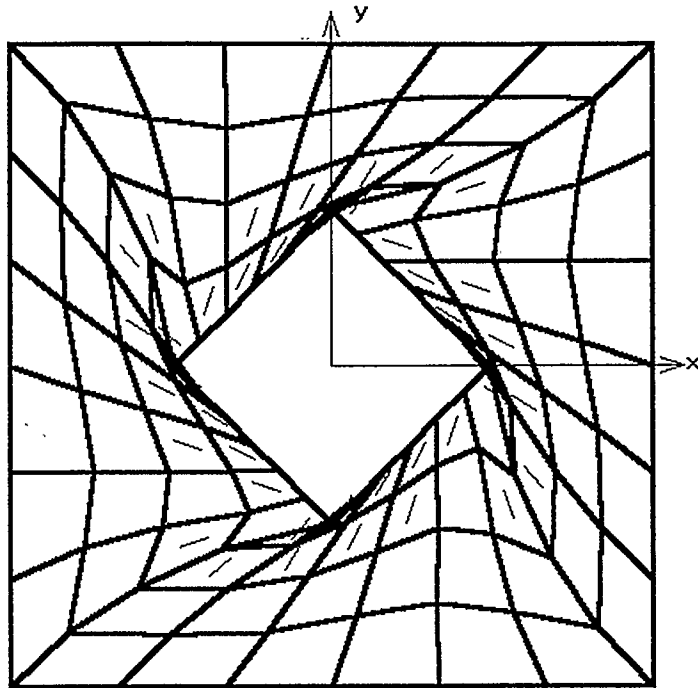


Figure 5.22 The deformed configuration for anisotropic case, $\beta_1 = 1.0, \beta_2 = 0.8$

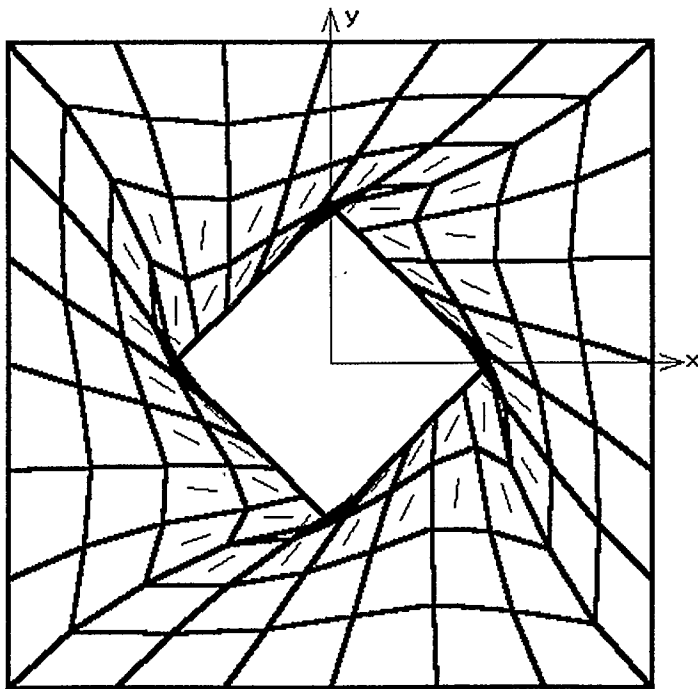


Figure 5.23 The deformed configuration for anisotropic case, $\beta_1 = 1.0, \beta_2 = 1.2$

Chapter SIX

Membrane Wrinkle Pattern Test

To demonstrate the wrinkling phenomenon and wrinkling patterns of membranes, the membrane wrinkle pattern test is carried out and test results are presented.

6.1 Testing Device

The testing device of membrane wrinkle pattern test is consisted of a main frame, a guard ring, a rigid disk, a walking frame and three adjustable bolts, as shown in Figure 6.1.

The main frame is a two-layer rigid space frame. It offers necessary support to other components. The guard ring is a rigid ring attached to the top layer of the main frame, and is used to fix the outer boundary of a sample membrane by means of a fastening belt. The rigid disk is supported by a small frame attached to the walking frame through an adjustable bolt, and is used to fix the inner boundary of a sample membrane through a fastening belt. Through the adjustable bolt connected to the rigid disk, the inner boundary of a sample membrane can be displaced laterally, so the bolt is called the lateral adjustive bolt. The walking frame is installed on the lower layer of the main frame, and can be driven horizontally by another

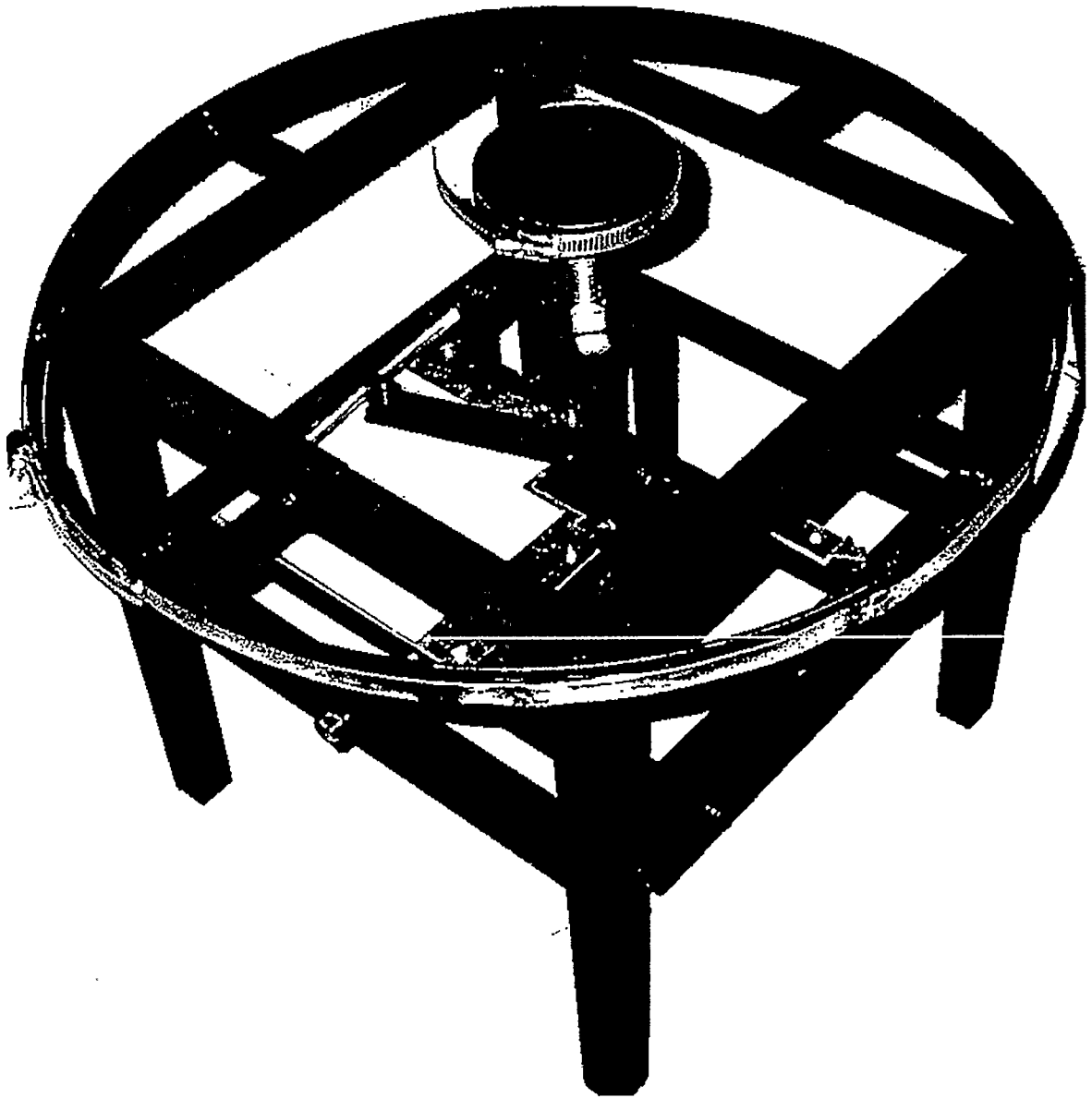


Figure 6.1 Testing device of membrane wrinkle pattern test

adjustable bolt. The function of the walking frame is to achieve an eccentric inner boundary, so the driving bolt is called the eccentric adjustive bolt. The last adjustable bolt is mounted on the walking frame, and is connected to a rocker arm through turnbuckle, which is assembled

on the lateral adjustive bolt by keyway. Through the third adjustable bolt, the rigid disk can be rotated up to 45 degree with respect to the guard ring. Thus the third adjustable bolt is called the rotating adjustive bolt.

With this testing device, samples with both concentric and eccentric circular boundaries can be twisted and displaced laterally.

6.2 Membrane Wrinkle Pattern Test

The objective of the membrane wrinkling pattern test is to demonstrate and obtain the wrinkling pattern of an annular membrane with concentric and eccentric circular boundaries for both isotropic and anisotropic cases. The sample membranes used here are two sheets of thin polyethylene membrane. One of the sample membranes is scotch-taped parallelly to simulate an orthotropic membrane.

In the test, the walking frame is first driven to one end where the rigid disk for fixing the inner boundary of sample membrane is concentric with the guard ring and the rigid disk is lifted about 5 centimeters. Then the sample membrane is mounted upon the testing device and both inner and outer boundaries are fixed by using the fastening belts. Finally, the inner boundary is rotated counterclockwise 15 degree and the wrinkling pattern obtained is shown in figure 6.2.

After unloading the sample membrane, the walking frame is driven to the other end, which is about 8 centimeters from its original position, and the eccentric boundaries are set up. Then the wrinkling pattern test is repeated with the same isotropic sample membrane. The corresponding wrinkling pattern obtained is shown in figure 6.3.

At last, the test was repeated using the orthotropic sample membrane with concentric and eccentric boundaries, and test results are shown in figure 6.4 and 6.5, respectively.

Comparing the numerical analysis results of annular membranes with concentric and eccentric boundaries in chapter 5 with the membrane wrinkle pattern test results, we can see that they agree well each other although there exist some effect factors, such as different material property and boundary conditions. The difference of wrinkle patterns of orthotropic and isotropic samples in our test is not as apparent as that in chapter 5. There are three possible reasons. The first reason could be that what is plotted out in Chapter 5 is the first principal stress instead of the deformed configuration. The second could be that the difference of the properties along two orthotropic axes in our test is not big enough, and the deformed configuration is so far away from the impending wrinkling state that the orthotropy has little effect on the wrinkle pattern. The third could be that our orthotropic sample membrane did not successfully reflect the orthotropic property due to the difficulty of modeling the orthotropic material.

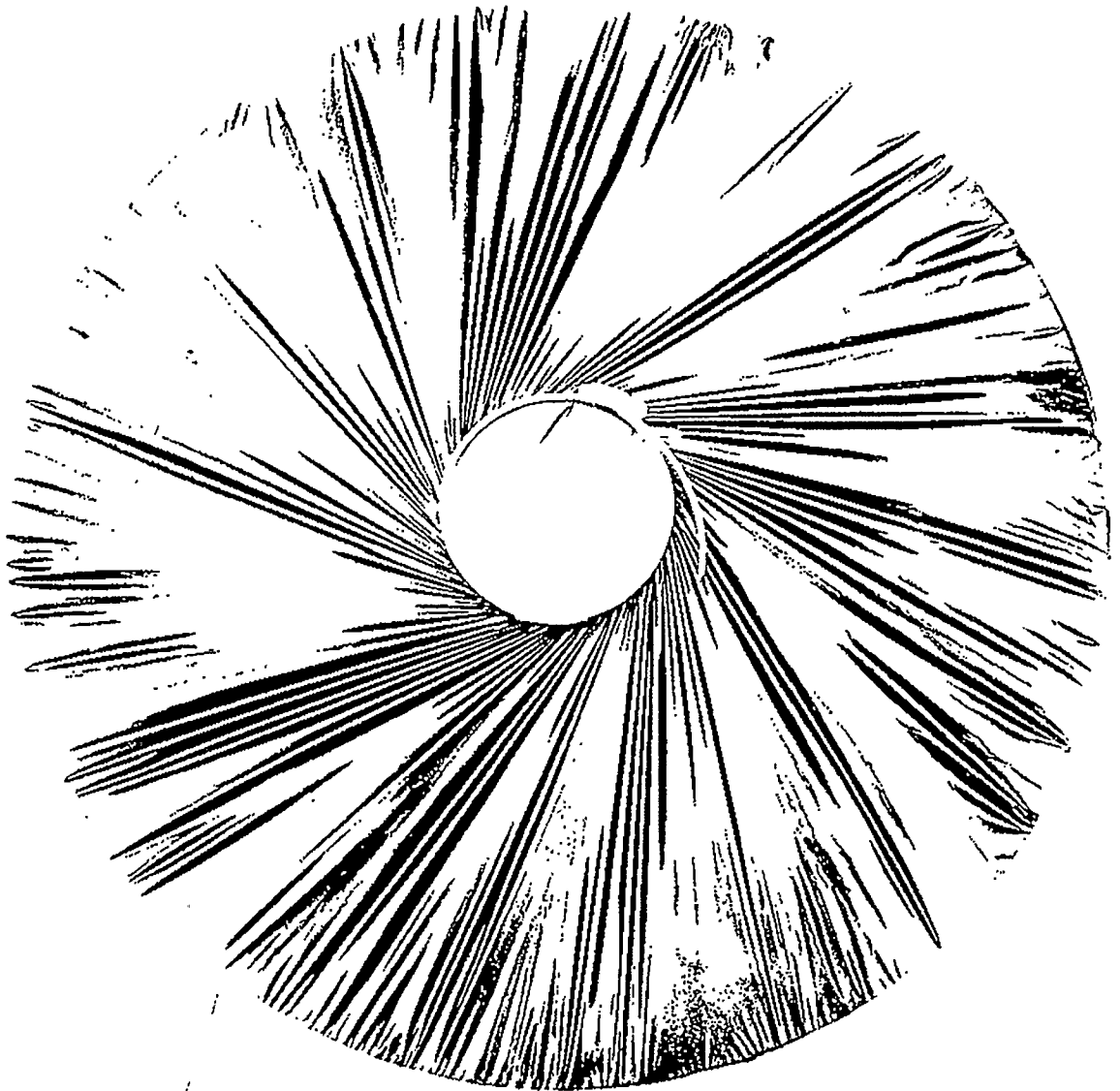


Figure 6.2 The wrinkling pattern of an isotropic membrane with concentric boundaries



Figure 6.3 The wrinkling pattern of an isotropic membrane with eccentric boundaries

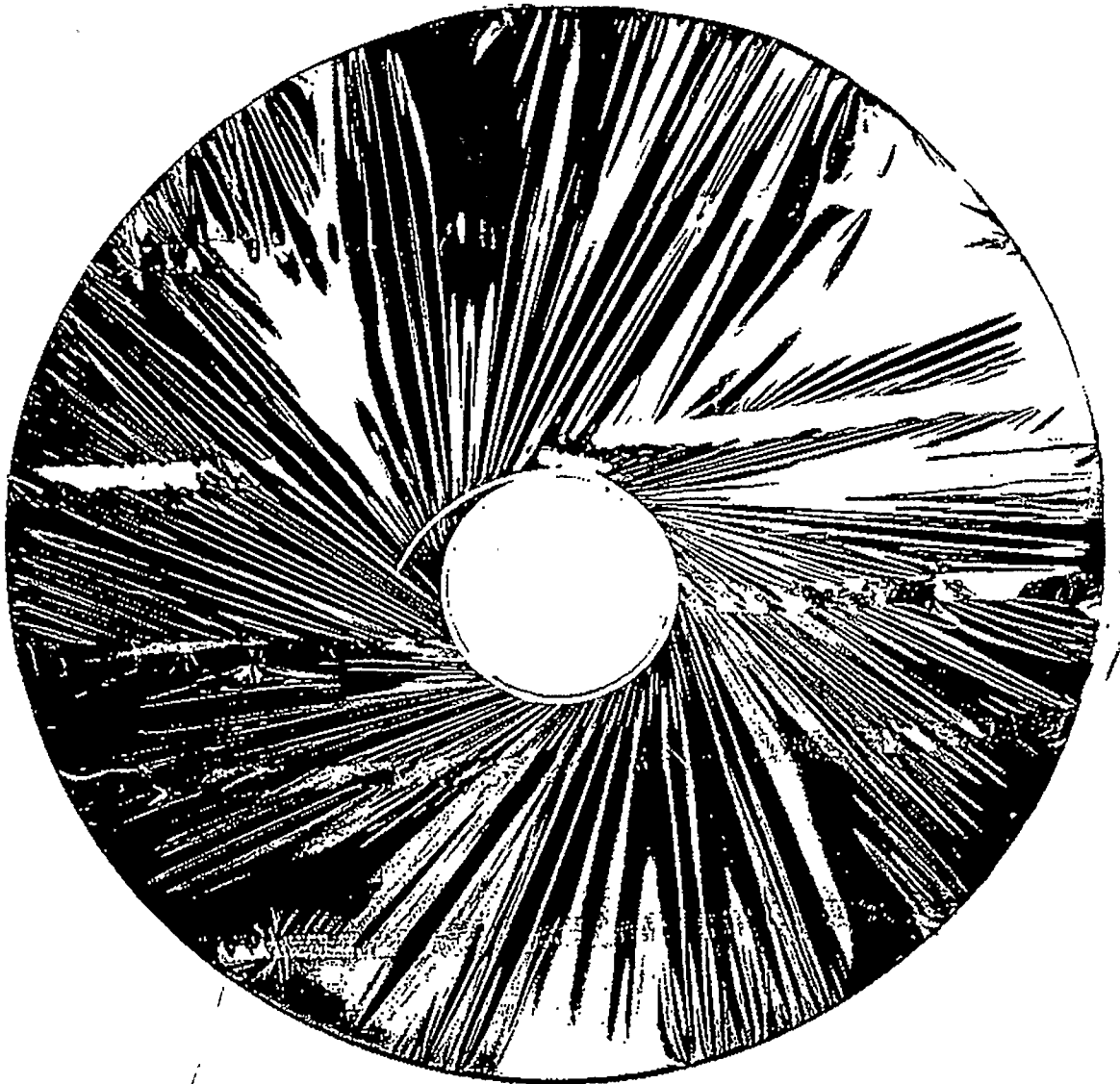


Figure 6.4 The wrinkling pattern of an orthotropic membrane with concentric boundaries

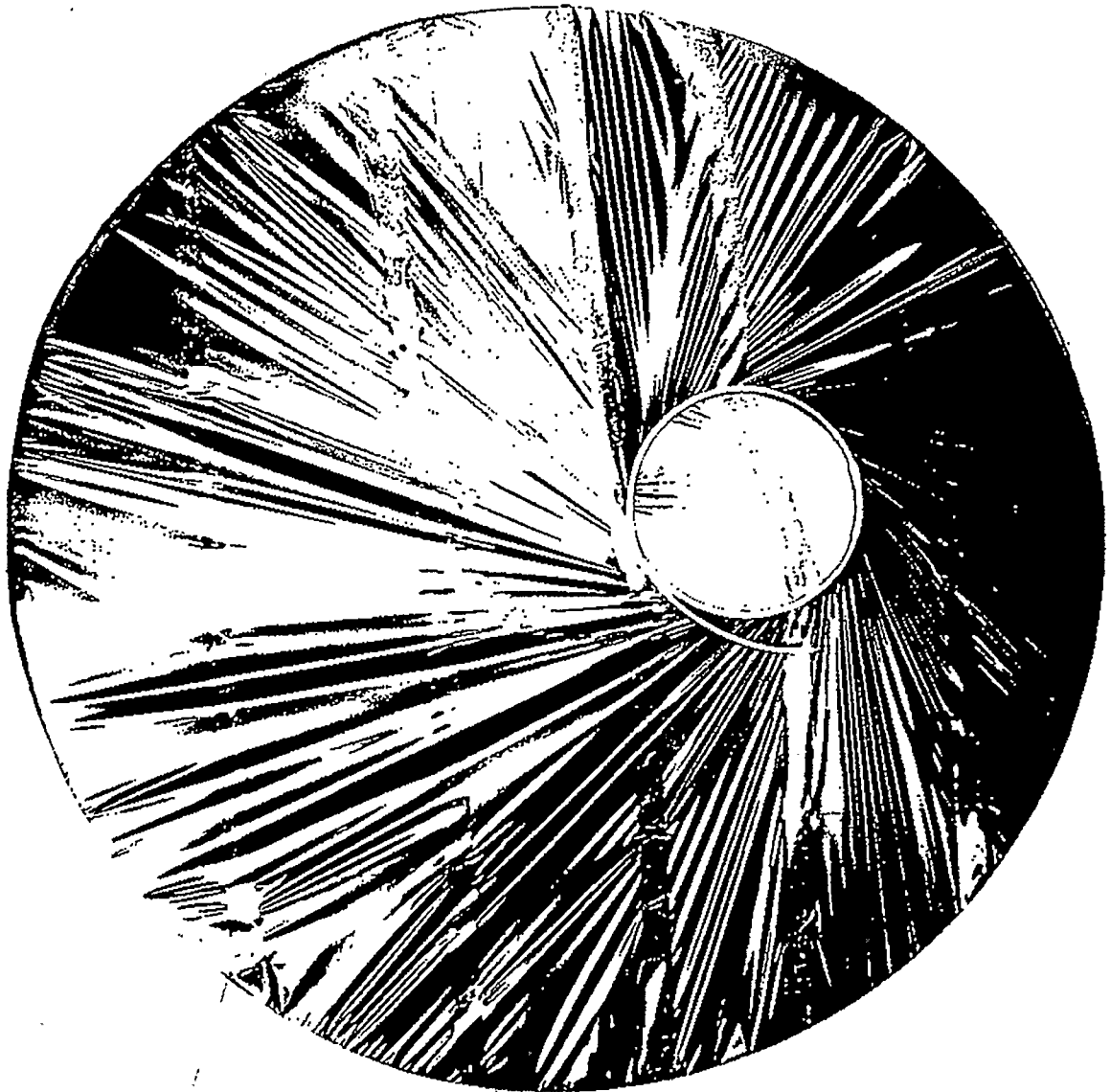


Figure 6.5 The wrinkling pattern of an orthotropic membrane with eccentric boundaries

Chapter SEVEN

Conclusions and Recommendations

The validity of a new wrinkling theory of anisotropic elastic membrane is tested by implementing the theory into a finite element code and analyzing a series of problems involving membrane wrinkling by using the finite element code. The results obtained are in accordance with both the previously published results by other investigators and the testing results obtained from a membrane wrinkle pattern test.

The wrinkling theory of anisotropic elastic membrane is based on a somewhat different way to look at problems. By viewing the wrinkling of membrane as a particular case of the saturated elasticity, the wrinkling condition is naturally obtained from the theory of saturated elasticity and the wrinkling theory of elastic membrane is established accordingly. The orthotropization technique is then adopted to introduce the anisotropy into the wrinkling theory.

Although numerical examples studied are not real engineering problems, they successfully show the ability of the new wrinkling theory in predicting wrinkling phenomenon in anisotropic elastic membrane. Results of the membrane wrinkle pattern test

agree well with that of numerical examples, except that wrinkle patterns of orthotropic and isotropic samples in our test are not so different from each other.

Since being viewed as a particular case of the saturated elasticity, the phenomenon of membrane wrinkling is no longer an isolated specific case of solid mechanics, it has been integrated into the whole picture of deformation in solid mechanics. Thus, the further investigation on the phenomenon of membrane wrinkling will also increase our understanding on the deformation mechanism from elastic deformation to saturated elastic deformation, and to plastic deformation.

To describe the wrinkled configuration exactly, the bending stiffness of a real membrane must be taken into consideration. This is not included in our present work, but it should be done in the nearest future.

Bibliography

- Cohen, H., Wang, C.C. (1984) On the response and symmetry of elastic and hyperelastic membrane points, *Arch. Rat. Mech. Anal.*, 85:343-379.
- Contri, P., Schrefler, B.A. (1988) A geometrically nonlinear finite element analysis of wrinkled membrane surfaces by a no-compression material model, *Communication in Applied Numerical Methods*, 4:5-15.
- Courant, R., Hilbert, D. (1962) *Methods of Mathematical Physics*, John Wiley, New York.
- Epstein, M. (1999) On the Wrinkling of anisotropic elastic membranes, *Journal of Elasticity*, 55:99-109.
- Epstein, M., Forcinito, M.A. (2001) Anisotropic membrane wrinkling: theory and analysis, *International Journal of Solids and Structures*, 38: 5253-5272.
- Haseganu, E. (1994) Analytical Investigation of Tension Fields in Lightweight Membrane Structures, Phd thesis, University of Alberta, Edmonton.
- Haseganu, E., Steigmann, D.J. (1994) Analysis of partly wrinkled membranes by the method of dynamic relaxation, *Computational Mechanics*, 14:596-614.
- Haseganu, E., Steigmann, D.J. (1994) Theoretical flexural response of a pressurized

- cylindrical membrane, *International Journal of Solids Structures*, 31:27-50.
- Jenkins, C.H. (1996) Nonlinear dynamic response of membranes: state of the art-update, *Applied Mechanics Review*, 49:S41-S48
- Kang, S., Im, S. (1997) Finite element analysis of wrinkling membranes, *ASME Journal of Applied Mechanics*, 64:263-269.
- Kondo, K., Iai, T., Moriguti, S., Murasaki, T. (1955) Tension field theory, *Memoirs of the unifying study of the basic problems in engineering science by means of geometry*, vol. 1, c.-v.:61-85.
- Li, X., Steigmann, D.J. (1993) Finite plane twist of an annular membrane, *Q. J. Mech. Appl. Math*, 46:601-625.
- Lu, K., Accorsi, M., Leonard, J. (2001) Finite element analysis of membrane wrinkling, *International Journal for Numerical Methods in Engineering*, 50:1017-1038.
- Lukasiewicz, S., Glockner, P.G. (1983) Ponding instability of cylindrical air-supported membranes under nonsymmetrical loadings, *Journal of Structural Mechanics*, 10:419-435.
- Lukasiewicz, S., Glockner, P.G. (1985) Stability of lofty air-supported cylindrical membranes, *Journal of Structural Mechanics*, 12:543-555.
- Lukasiewicz, S., Balas, L. (1990) Collapse mode of an inflatable freestanding membrane, *Mechanical Structural Machines*, 18:483-497.
- Lukasiewicz, S., Balas, L. (1990) Collapse loads of a cylindrical or toroidal freestanding inflatable membrane, *Mechanical Structural Machines*, 18:499-513.
- Mansfield, E.H. (1970) Load transfer via a wrinkled membrane, *Proceedings of Royal*

- Society of London, A316:269.
- Miller, R.K., Hedgepath, J.M., Weingarten, V.I., Das, P., Kahyai, S. (1985) Finite element analysis of partly wrinkled membranes, *Computers and Structures*, 20:631-639
- Muttin, F. (1996) A finite element for wrinkled curved elastic membranes and its application to sails, *Communications in Numerical Methods in Engineering*, 12(11): 775-786.
- Pipkin, A.C. (1986) Relaxed energy densities for isotropic elastic membranes, *IMA Journal of Applied Math*, 36:85-99.
- Pipkin, A.C. (1993) Relaxed energy densities for small deformations of membranes, *IMA Journal of Applied Math*, 50:225-237.
- Pipkin, A.C. (1994) Relaxed energy densities for large deformations of membranes, *IMA Journal of Applied Math*, 52:297.
- Pogorelov, A.V. (1967) Geometrical Methods in Nonlinear Theory of Elastic Shell, Moscow.
- Reissner, E. (1939) Tension-field theory, Proceedings of 5th International Congress on Applied Mechanics, 88-92.
- Roddeman, D.G., Drukker, J., Oomens, D.W.J., Janssen, J.D. (1987) The wrinkling of thin membranes: part I—theory, *ASME Journal of Applied Mechanics*, 54:884-887.
- Roddeman, D.G., Drukker, J., Oomens, D.W.J., Janssen, J.D. (1987) The wrinkling of thin membranes: part II—numerical analysis, *ASME Journal of Applied Mechanics*, 54:888-892.
- Roddeman, D.G. (1991) Finite element analysis of wrinkling membranes, *Communication*

in Applied Numerical Methods, 7:299-307.

Stanuszek, M., Glockner, P.G. (1995) Key response features of spherical inflatables under axisymmetric liquid loading, *Applied Mechanics Review*, 48:S36-S43

Stanuszek, M., Glockner, P.G. (1995) Further results on the response of spherical inflatables under axisymmetric hydrostatic loads, *Computation Structure*, 57:33-45

Steigmann, D.J. (1990) Tension-field theory, *Proceedings of Royal Society of London*, A429:141-173.

Wagner, H. (1929) Ebene blechwandträger mit sehr dünnem stegblech, *Z. Flugtechnik u. Motorluftschiffahrt*, 20:200-207, 227-233, 236-262, 279-284, 306-314.

Wu, C.H. (1978) Nonlinear wrinkling of nonlinear membranes of revolution, *ASME Journal of Applied Mechanics*, 45:533-538.

Wu, C.H., Canfield T.R. (1981) Wrinkling in finite plane-stress theory, *Quarterly of Applied Mathematics*, 39:179-199.

Appendix

A listing of the relevant three parts—the preprocessor, the main analyzer and the postprocessor of the visual C++ procedure is given here for future reference.

```
//-----The Preprocessor-----  
// T3.cpp version 1.1 March 28, 2002  
  
#include <stdio.h>  
#include <math.h>  
#include <iostream.h>  
#include <fstream.h>  
  
void main()  
{double b1,b2,x,y,z,s,t,ex,ey,w,x1,x2,y1,y2,z1,z2,r1,r2,rr,th,th1,jd=180.0/3.1415926,  
G,L,L1,L2;  
int n,i,j,k,i1,j1,s1,t1,t2,ii,jj;  
//main menu  
cout<<" Data file type:\n";  
cout<<" 1 square sheet\n";  
cout<<" 2 square sheet with a center square hole\n";  
cout<<" 3 circular sheet with a center circular hole\n";  
cout<<" 4 circular sheet with an eccentric circular hole\n";  
cout<<" Please select:"; cin>>s1;  
ofstream fout("sl.dt");  
switch (s1) {  
case 1: //square sheet  
cout<<"\n Square sheet:\n";  
cout<<" Length of edge:"; cin>>L;  
cout<<"\n Length of element:"; cin>>s;
```

```

z=0.0;    n=int(L/s+0.1);    i1=n+1;
fout<<" 1 "<<i1*i1<<"\n";    //nodal coordinates
for (i=0;i<i1;i++)    for (j=0;j<i1;j++)
    {x=j*s;    y=i*s;
    if (j==0) k=0;    else k=1;
    fout<<x<<" "<<y<<" "<<z<<" "<<k<<" "<<k<<" "<<k<<"\n";    };
cout<<"\n Number of different material property:";    cin>>i;
fout<<i<<"\n";    //Material property
cout<<"\n Shear Modulus:";    cin>>G;    fout<<G<<" ";
cout<<"\n Thickness of membrane:";    cin>>t;    fout<<t<<" ";
cout<<"\n Orthotropy parameter B1:";    cin>>b1;    fout<<b1<<" ";
cout<<"\n Orthotropy parameter B2:";    cin>>b2;    fout<<b2<<"\n";
fout<<n*n<<"\n";    //element data
for (i=0;i<i1-1;i++)    for (j=1;j<i1;j++)
    fout<<i*i1+j<<" "<<i*i1+j+1<<" "<<(i+1)*i1+j+1<<" "<<(i+1)*i1+j<<" 1 0.0\n";
fout<<3*i1<<"\n"; z=0.0;    //given displacement
cout<<"\n X displacement:";    cin>>x;
cout<<"\n Y displacement:";    cin>>y;
for (i=1;i<i1+1;i++)
    {j=i*i1;
    fout<<j<<" 1 "<<x<<"\n";
    fout<<j<<" 2 "<<y<<"\n";
    fout<<j<<" 3 "<<z<<"\n";    };
fout<<"0\n";    //load data
fout.close();
cout<<"\n The data file is in sl.dt now!";
break;

```

```

case 2:    //square sheet with a center square hole
cout<<"\n square sheet with a center square hole\n";
cout<<" Length of outer edge:";    cin>>L2;
cout<<"\n Length of inner edge:";    cin>>L1;
cout<<"\n Layer of element along circular direction in a quarter area:";    cin>>t1;
cout<<"\n Layer of element along radius direction:";    cin>>t2;
cout<<"\n Z coordinate of inner boundary:";    cin>>z;
n=4*t1*(t2+1);    z=0.0;    i1=t1+1;    j1=t2+1;
fout<<" 2 "<<n<<"\n";    //nodal coordinates
for (i=0;i<t1;i++)
    {y1=-L1*0.5;    x1=i*L1/t1+y1;    y2=-L2*0.5;    x2=i*L2/t1+y2;
    s=(x2-x1)/t2;    t=(y2-y1)/t2;
    for (j=0;j<j1;j++)
        {x=x1+j*s;    y=y1+j*t;    if ((j==0)||j==j1) k=0;    else k=1;
        fout<<x<<" "<<y<<" "<<z<<" "<<k<<" "<<k<<" "<<k<<"\n";    };

```

```

};
for (i=1;i<i1;i++)
{ x1=-L1*0.5; y1=i*L1/t1+x1; x2=-L2*0.5; y2=i*L2/t1+x2;
s=(x2-x1)/t2; t=(y2-y1)/t2;
for (j=0;j<j1;j++)
{x=x1+j*s; y=y1+j*t; if ((j==0)||(j==j1)) k=0; else k=1;
fout<<x<<" "<<y<<" "<<z<<" "<<k<<" "<<k<<" "<<k<<"\n"; };
};
for (i=0;i<t1;i++)
{ x1=L1*0.5; y1=i*L1/t1-x1; x2=L2*0.5; y2=i*L2/t1-x2;
s=(x2-x1)/t2; t=(y2-y1)/t2;
for (j=0;j<j1;j++)
{x=x1+j*s; y=y1+j*t; if ((j==0)||(j==j1)) k=0; else k=1;
fout<<x<<" "<<y<<" "<<z<<" "<<k<<" "<<k<<" "<<k<<"\n"; };
};
for (i=1;i<i1;i++)
{ y1=L1*0.5; x1=i*L1/t1-y1; y2=L2*0.5; x2=i*L2/t1-y2;
s=(x2-x1)/t2; t=(y2-y1)/t2;
for (j=0;j<j1;j++)
{x=x1+j*s; y=y1+j*t; if ((j==0)||(j==j1)) k=0; else k=1;
fout<<x<<" "<<y<<" "<<z<<" "<<k<<" "<<k<<" "<<k<<"\n"; };
};
cout<<"\n Number of different material property:"; cin>>i;
fout<<i<<"\n"; //Material property
cout<<"\n Shear Modulus:"; cin>>G; fout<<G<<" ";
cout<<"\n Thickness of membrane:"; cin>>t; fout<<t<<" ";
cout<<"\n Orthotropy parameter B1:"; cin>>b1; fout<<b1<<" ";
cout<<"\n Orthotropy parameter B2:"; cin>>b2; fout<<b2<<"\n";
fout<<t1*t2*4<<"\n"; //element data
for (i=0;i<t1-1;i++) for (j=0;j<t2;j++)
fout<<i*j1+j+1<<" "<<(i+1)*j1+j+1<<" "<<(i+1)*j1+j+1+1<<" "<<i*j1+j+1+1<<
" 1 0.0\n";
for (j=0;j<t2;j++)
fout<<(t1-1)*j1+j+1<<" "<<2*t1*j1+j+1<<" "<<2*t1*j1+j+1+1<<" "<<
(t1-1)*j1+j+1+1<<" 1 0.0\n";
for (j=0;j<t2;j++)
fout<<t1*j1+j+1<<" "<<j+1<<" "<<j+1+1<<" "<<t1*j1+j+1+1<<" 1 0.0\n";
for (i=1;i<t1;i++) for (j=0;j<t2;j++)
fout<<(i+t1)*j1+j+1<<" "<<(i+t1-1)*j1+j+1<<" "<<(i+t1-1)*j1+j+1+1<<
" "<<(i+t1)*j1+j+1+1<<" 1 0.0\n";
for (i=0;i<t1-1;i++) for (j=0;j<t2;j++)
fout<<(i+2*t1)*j1+j+1<<" "<<(i+2*t1+1)*j1+j+1<<" "<<(i+2*t1+1)*j1+j+1+1<<
" "<<(i+2*t1)*j1+j+1+1<<" 1 0.0\n";

```



```

i=3*t1-1;
for (j=0;j<t2;j++)
fout<<i*j1+j+1<<" "<<(i+t1)*j1+j+1<<" "<<(i+t1)*j1+j+1+1<<" "<<i*j1+j+1+1<<
" 1 0.0\n";
i=2*t1-1;
for (j=0;j<t2;j++)
fout<<(i+t1+1)*j1+j+1<<" "<<i*j1+j+1<<" "<<i*j1+j+1+1<<
" "<<(i+t1+1)*j1+j+1+1<<" 1 0.0\n";
for (i=1;i<t1;i++) for (j=0;j<t2;j++)
fout<<(i+3*t1)*j1+j+1<<" "<<(i+3*t1-1)*j1+j+1<<" "<<(i+3*t1-1)*j1+j+1+1<<
" "<<(i+3*t1)*j1+j+1+1<<" 1 0.0\n";
fout<<3*t1*4<<"\n"; // given displacement
cout<<"\n Z displacement of inner boundary:"; cin>>z;
cout<<"\n Rotating angle of inner boundary:"; cin>>rr;
rr=rr/jd; r1=cos(rr); r2=sin(rr); s=45.0/jd; z2=sin(s); z1=cos(s);
x1=L2/z2*(z2-sin(s-rr)); x2=L2/z1*(cos(s-rr)-z1); y1=-x2; y2=x1;
s=(x2-x1)/t1; t=(y2-y1)/t1;
for (i=0;i<t1;i++)
{ x=i*s+x1; y=i*t+y1; j=(i+1)*j1;
fout<<j<<" 1 "<<x<<"\n";
fout<<j<<" 2 "<<y<<"\n";
fout<<j<<" 3 "<<z<<"\n"; };
x2=-L2*0.5; x1=0.207*L2; y1=-x2; y2=x1; s=(x2-x1)/t1; t=(y2-y1)/t1;
for (i=0;i<t1;i++)
{ x=i*s+x1; y=i*t+y1; j=(i+t1+t1+1)*j1;
fout<<j<<" 1 "<<x<<"\n";
fout<<j<<" 2 "<<y<<"\n";
fout<<j<<" 3 "<<z<<"\n"; };
x1=L2*0.5; x2=-0.207*L2; y1=x2; y2=-x1; s=(x2-x1)/t1; t=(y2-y1)/t1;
for (i=1;i<i1;i++)
{ x=i*s+x1; y=i*t+y1; j=(i+t1)*j1;
fout<<j<<" 1 "<<x<<"\n";
fout<<j<<" 2 "<<y<<"\n";
fout<<j<<"3 "<<z<<"\n"; };
x2=-L2*0.5; x1=-0.207*L2; y1=x2; y2=-x1; s=(x2-x1)/t1; t=(y2-y1)/t1;
for (i=1;i<i1;i++)
{ x=i*s+x1; y=i*t+y1; j=(i+3*t1)*j1;
fout<<j<<" 1 "<<x<<"\n";
fout<<j<<" 2 "<<y<<"\n";
fout<<j<<" 3 "<<z<<"\n"; };
fout<<"0\n"; //load data
fout.close();
cout<<"\n The data file is in sl.dt now!";

```

break;

```

case 3:           //circular sheet with a center circular hole
  cout<<"\n Circular sheet with a center circular hole\n";
  cout<<"\n Outer radius:";   cin>>r2;
  cout<<"\n Inner radius:";   cin>>r1;
  cout<<"\n Layer of element along circular direction in a quarter area:";   cin>>t1;
  cout<<"\n Layer of element along radius direction:";   cin>>t2;
  cout<<"\n Z coordinate of inner boundary:";   cin>>z1;
  cout<<"\n Z displacement of inner boundary:";   cin>>w;
  cout<<"\n Rotating angle of inner boundary:";   cin>>rr;   rr=rr/jd;
  cout<<"\n The angle between x axis and orthotropy direction:";   cin>>th1;
  n=4*t1*(t2+1);   i1=4*t1;   j1=t2+1;   y1=1.5707963/t1;   z2=z1/t2;
  fout<<" 3 "<<n<<"\n";           //nodal coordinates
  for (j=0;j<j1;j++)
  { x1=r1+(r2-r1)/t2*j;   z=z1-j*z2;
    for (i=0;i<i1;i++)
    { y2=y1*i;   x=x1*cos(y2);   y=x1*sin(y2);
      if (j==t2) k=0;   else k=1;
      fout<<x<<" "<<y<<" "<<z<<" "<<k<<" "<<k<<" "<<k<<"\n";   };
    };
  cout<<"\n Number of different material property:";   cin>>i;
  fout<<i<<"\n";           //Material property
  cout<<"\n Shear Modulus:";   cin>>G;   fout<<G<<" ";
  cout<<"\n Thickness of membrane:";   cin>>t;   fout<<t<<" ";
  cout<<"\n Orthotropy parameter B1:";   cin>>b1;   fout<<b1<<" ";
  cout<<"\n Orthotropy parameter B2:";   cin>>b2;   fout<<b2<<"\n";
  fout<<t1*t2*4<<"\n";   y1=y1/jd;           //element data
  for (i=0;i<i1;i++)   for (j=0;j<t2;j++)
  { ii=i1*(j+1)+i+1+1;   jj=i+1*j+1+1;   if (jj==(1+j)*i1+1) jj=j*i1+1;
    if (ii==(2+j)*i1+1) ii=(j+1)*i1+1;   th=(i+0.5)*y1-th1;
    fout<<i+j*i1+1<<" "<<(1+j)*i1+i+1<<" "<<ii<<" "<<jj<<" 1 "<<th<<"\n";   };
  fout<<3*i1<<"\n";   y1=y1/jd;           //given displacement
  for (i=0;i<i1;i++)
  { x2=y1*i;   x1=rr+x2;   x=r1*(cos(x1)-cos(x2));   y=r1*(sin(x1)-sin(x2));
    fout<<i+1<<" 1 "<<x<<"\n";
    fout<<i+1<<" 2 "<<y<<"\n";
    fout<<i+1<<" 3 "<<w<<"\n";   };
  fout<<"0\n";           //load data
  fout.close();
  cout<<"\n The data file is in sl.dt now!";
  break;

```

```

case 4:      //circular sheet with an eccentric circular hole
cout<<"\n Circular sheet with an eccentric circular hole\n";
cout<<"\n Outer radius:";   cin>>r2;
cout<<"\n Inner radius:";   cin>>r1;
cout<<"\n Layer of element along circular direction in a quarter area:";   cin>>t1;
cout<<"\n Layer of element along radius direction:";   cin>>t2;
cout<<"\n Z coordinate of inner boundary:";   cin>>z1;
cout<<"\n Z displacement of inner boundary:";   cin>>w;
cout<<"\n Rotating angle of inner boundary:";   cin>>rr;   rr=rr/jd;
cout<<"\n The angle between x axis and orthotropy direction:";   cin>>th1;
cout<<"\n X coordinate of the center of inner circle:";   cin>>ex;
cout<<"\n Y coordinate of the center of inner circle:";   cin>>ey;
n=4*t1*(t2+1);   i1=4*t1;   j1=t2+1;   y1=1.5707963/t1;   z2=z1/t2;
fout<<" 4 "<<n<<"\n";           //nodal coordinates
for (j=0;j<j1;j++)
{ z=z1-j*z2;
  for (i=0;i<i1;i++)
  { y2=y1*i;
    L=(-r1-ey*sin(y2)-ex*cos(y2)+sqrt(r2*r2-(ex*sin(y2)-
      ey*cos(y2))*(ex*sin(y2)-ey*cos(y2))))/t2;
    x1=r1+j*L; x=x1*cos(y2)+ex; y=x1*sin(y2)+ey;
    if (j==t2) k=0;   else k=1;
    fout<<x<<" "<<y<<" "<<z<<" "<<k<<" "<<k<<" "<<k<<"\n"; };
  };
cout<<"\n Number of different material property:";   cin>>i;
fout<<i<<"\n";   //Material property
cout<<"\n Shear Modulus:";   cin>>G;   fout<<G<<" ";
cout<<"\n Thickness of membrane:";   cin>>t;   fout<<t<<" ";
cout<<"\n Orthotropy parameter B1:";   cin>>b1;   fout<<b1<<" ";
cout<<"\n Orthotropy parameter B2:";   cin>>b2;   fout<<b2<<"\n";
y1=y1*jd;   fout<<t1*t2*4<<"\n";           //element data
for (i=0;i<i1;i++)   for (j=0;j<t2;j++)
{ ii=i1*(j+1)+i+1+1;   jj=i+1*j+1+1;   if (jj==(1+j)*i1+1) jj=j*i1+1;
  if (ii==(2+j)*i1+1) ii=(j+1)*i1+1;   th=(i+0.5)*y1-th1;
  fout<<i+j*i1+1<<" "<<(1+j)*i1+i+1<<" "<<ii<<" "<<jj<<" 1 "<<th<<"\n"; };
fout<<3*i1<<"\n";   y1=y1/jd;           //given displacement
for (i=0;i<i1;i++)
{ x2=y1*i;   x1=rr+x2;   x=r1*(cos(x1)-cos(x2));   y=r1*(sin(x1)-sin(x2));
  fout<<i+1<<" 1 "<<x<<"\n";
  fout<<i+1<<" 2 "<<y<<"\n";
  fout<<i+1<<" 3 "<<w<<"\n"; };
fout<<"0\n";           //load data
fout.close();

```

```

    cout<<"\n The data file is in sl.dt now!";
    break;
    default: break;  };
}

//-----The Main Analyzer-----
//Sheet2.h version 1.1 March 28,2002
//Definition of element and structure classes

#include <Math.h>

const dofn=3L,maxnode=200L,maxele=200L,maxk=25000L,maxv=1000L,namel=25L;
const NNo=4L;
const double ksi[4]={-1.0,1.0,1.0,-1.0},eta[4]={-1.0,-1.0,1.0,1.0};
const double r=sqrt(1.2),ki[4]={sqrt((3.0+r+r)/7.0),-sqrt((3.0+r+r)/7.0),
    sqrt((3.0-r-r)/7.0),-sqrt((3.0-r-r)/7.0)};
const double H[4]={0.5-1/6.0/r,0.5-1/6.0/r,0.5+1/6.0/r,0.5+1/6.0/r};

class GivenDp
{int no,dof,tdof; double v,rk;
public:
    int Getno();
    int Getdof();
    int Gettdof();
    void Settdof(int i);
    double Getv();
    double Getrk();
    void Setv(double s);
    void Setrk(double s);
    GivenDp(int i, int j, double s);
    GivenDp();
    ~GivenDp();
};

class EM4Load
{int type,dof,node; double v;
public:
    int Gettype();
    int Getdof();
    int Getnodeno();
    double Getv();
    EM4Load(int atype, int adof, int ano, double s);
    EM4Load();
};

```

```

    ~EM4Load();
};

class MPoint
{
    double x,y,z, dp[dofn],ddp[dofn]; //dp:displacement,ddp:delta displacement
    int No, dof[dofn];
public:
    double GetX(){return x;};
    double GetY(){return y;};
    double GetZ(){return z;};
    void SetPoint(double px,double py,double pz){x=px;y=py;z=pz;};
    void Setdof(int i,int j);
    int GetNo(){return No; };
    void SetNo(int i){No=i; };
    int Getdof(int i);
    void Setdp(int i);
    double Getdp(int i);
    void Setddp(int i,double dp1);
    double Getddp(int i);
    void totaldp();
    MPoint(int aNo,double px,double py,double pz, int adof[dofn]);
    void SetMPoint(int aNo,double px,double py,double pz, int adof[dofn]);
    MPoint({});
    ~MPoint({});
};

class EM4
{
    int No,LN,EPN,ln;
    double t,b1,b2,g,th1;
    MPoint pt[NNo];
    EM4Load ld[dofn];
    double dp[12];
    double N(int i, double ks, double et); //Shape Function
    double NKsi(int i, double et); //N,ksi
    double NEta(int i, double ks); //N,eta
    double DCDU(int h, int i, int j, int k, double ks, double et, double dp1[12]);
    //h=1 dc/du 2 d^2c/du^2;i=1 c11, 2 c22, 3 c12; j,k=0~3 u1~u4 4~7 v1~v4 8~11 w1~w4
    void findit(int sv,double xb[7],double &x,double &y,double &z);
    double jjsp(double x1,double y1,double z1,double x2,double y2,double z2,double
        &x,double &y,double &z);
    double jjcs(double x1,double y1,double z1,double x2,double y2,double z2);
    void direction(double x1,double y1,double z1,double x2,double y2,double z2,double
        &x,double &y,double &z);
};

```

```

void normals(double xb[7],double &x,double &y,double &z,double &s1,double
            &s2,double &s12);
void intersectpoint(int sv,double xb[7],double dx,double dy,double dz,double
                  &x,double &y,double &z,int &i);
double svalues(double x[7]);
double csurface(double x[7]);
public:
void STIF0(double c[12][12]);
void STIFT(double c[12][12]);
void StrainC(double ss[3],double ks, double et, double dp1[12]);
void Setdof(int i,int j,int k);
void Setdp(int m, int NDISP, double DP1[maxv], MPoint pa[maxnode]);
MPoint GetPoint(int i);
void Sett(double s);
double GetB(int i);
void SetB(double s1,double s2);
double GetG();
void SetG(double s);
double GetTh1();
void SetTh1(double s);
double Gett();
double Getdp(int i);
void SetLoad(EM4Load s);
void FORMII(int i,int II[12]);
double w(double x[6]);
double spk(int m, double x[7]);
double nd1(int n, int sub[2], double x[7]);
double nd2(int n, int sub[2], double x[7]);
double nd(int n, int sub[2], double x[7]);
double nda(int n, int sub[2], double x[7]);
double w1(double x[7], int &i);
void Formdp(int NDISP, double dp[maxv]);
void Formdp2(int NDISP, double dp1[maxv]);
void SetEM4(int aNo,int p[NNo],MPoint pp[],double aG,double at,double
            ab1,double ab2,double ath);
EM4(int aNo,int p[NNo],MPoint pp[],double aG,double at,double ab1,double
    ab2,double ath);
EM4();
~EM4();
};

class Sheet2
{ int nodenumber,elementnumber,gdnumber,loadnumber,NDISP,type;

```

```

MPoint pa[maxnode]; //mpointarray
EM4 ea[maxele]; //EM4array
GivenDp gd[maxele]; //GivenDisplacementarray
EM4Load ld[maxnode]; //Loadarray
void ASSEMB2(int IB,int II[12],double C[12][12],double R[maxk],int NDISP,int
            BV,int V[maxv],int PHASE);
void DIAGADR(int NDISP,int BV,int IV,int V[maxv]);
void LDLT1(int NDISP,int BV,int V[maxv],double R[maxk],double T[maxv]);
void SLVEQ1(int NDISP,int BV, int V[maxv], double R[maxk], double B[maxv], int
            BB);
int Setdof(int p);
void SetGDK(int NDISP,int BV, int V[maxv], double R[maxk]);
void SetGDR(int NDISP, double B[maxk]);
public:
int Getnodenumber(){return nodenumber;};
int Getelementnumber(){return elementnumber;};
MPoint GetMPoint(int i);
EM4 GetEM4(int i);
EM4Load GetEM4Load(int i);
void SetSheet2(CString fn);
void Analyse();
void Outresult(int ii);
Sheet2();
~Sheet2();
};

// Sheet2.cpp version 1.1 April 3,2002
//Implimentation of definition of element and structure classes

#include "stdafx.h"
#include "Sheet2.h"
#include <Math.h>
#include <iostream.h>
#include <fstream.h>

//***** GivenDp*****
int GivenDp::Getno(){return no;};
int GivenDp::Getdof(){return dof;};
int GivenDp::Gettdof(){return tdof;};
void GivenDp::Settdof(int i){tdof=i;};
double GivenDp::Getv(){return v;};
double GivenDp::Getrk(){return rk;};
void GivenDp::Setrk(double s){rk=s;};

```

```

void GivenDp::Setv(double s){v=s;};
GivenDp::GivenDp(int i, int j, double s){no=i; dof=j; v=s; rk=0.0;};
GivenDp::GivenDp(){};
GivenDp::~~GivenDp(){};
//***** EM4Load *****
int EM4Load::GetType(){return type;};
//type: 1-concentrate force; 2-surface traction; 3-body force
int EM4Load::Getdof(){return dof;};
int EM4Load::Getnodeno(){ return node;};
double EM4Load::Getv(){ return v;};

EM4Load::EM4Load(int atype, int adof, int ano,double s)
    {type=atype; dof=adof; node=ano; v=s;};

EM4Load::EM4Load(){v=0.0;};
EM4Load::~~EM4Load(){};
//***** MPoint *****
int MPoint::Getdof(int i){ return dof[i];};
void MPoint::Setdof(int i,int j) {dof[i]=j; };

MPoint::MPoint(int aNo,double px,double py,double pz, int adof[dofn])
{
    int i;
    x=px; y=py; z=pz; No=aNo;
    for (i=0;i<dofn;i++) {dof[i]=adof[i]; dp[i]=0.0; ddp[i]=0.0;};
};

void MPoint::SetMPoint(int aNo,double px,double py,double pz, int adof[dofn])
{
    int i;
    x=px; y=py; z=pz; No=aNo;
    for (i=0;i<dofn;i++) {dof[i]=adof[i]; dp[i]=0.0; ddp[i]=0.0;};
};

void MPoint::Setdp(int i) { dp[i]=dp[i]+ddp[i]; };
double MPoint::Getdp(int i) { return dp[i]; };
void MPoint::Setddp(int i,double dp1) { ddp[i]=dp1; };
double MPoint::Getddp(int i) { return ddp[i]; };
//***** EM4--4 nodes membrane element *****
double EM4::N(int i, double ks, double et) //i=0,1,2,3->node 1,2,3,4
{return (1.0+ksi[i]*ks)*(1.0+eta[i]*et)*0.25;};

double EM4::NKsi(int i, double et) {return ksi[i]*(1.0+eta[i]*et)*0.25;};
//i=0,1,2,3->node 1,2,3,4
double EM4::NEta(int i, double ks) {return eta[i]*(1.0+ksi[i]*ks)*0.25;};

```


//i=0,1,2,3->node 1,2,3,4

```
void EM4::STIF0(double c[12][12])
{ int i,j,k,ii,jj,sub[3][2]; double x[7],s,cc[4][4][3];
  for (i=0;i<12;i++) for (j=0;j<12;j++) c[i][j]=0.0; x[4]=b1; x[5]=b2; x[6]=g;
  for (i=0;i<12;i++) dp[i]=0.0;
  for (j=0;j<3;j++) {sub[j][1]=0; sub[j][0]=j+1; };
  for (i=0;i<12;i++)
  { dp[i]=10.0;
    for (ii=0;ii<4;ii++) for (jj=0;jj<4;jj++) StrainC(cc[ii][jj],ki[ii],ki[jj],dp);
    for (j=0;j<12;j++) for (ii=0;ii<4;ii++) for (jj=0;jj<4;jj++)
    { for (k=0;k<3;k++) x[k]=cc[ii][jj][k]; x[3]=x[2]; s=0.0;
      for (k=0;k<3;k++) s=s+nd2(1,sub[k],x)*DCDU(1,k,j,j,ki[ii],ki[jj],dp);
      c[i][j]=c[i][j]+H[ii]*H[jj]*s*t;
    }; dp[i]=0.0;
  };
};
```

```
void EM4::STIFT(double c[12][12])
{ int i,j,k,m,ii,jj,sub[3][3][2];
  double x[7],ks,et,s,s1,cc[4][4][3],cdu1[3][12][4][4],cdu2[3][12][12][4][4];
  for (i=0;i<12;i++) for (j=0;j<12;j++) c[i][j]=0.0; x[4]=b1; x[5]=b2; x[6]=g;
  for (ii=0;ii<4;ii++) for (jj=0;jj<4;jj++)
  { ks=ki[ii]; et=ki[jj]; StrainC(cc[ii][jj],ks,et,dp);
    for (k=0;k<3;k++) for (i=0;i<12;i++)
    { cdu1[k][i][ii][jj]=DCDU(1,k,i,j,ks,et,dp);
      for (j=0;j<12;j++) cdu2[k][i][j][ii][jj]=DCDU(2,k,i,j,ks,et,dp);
    };
  };
  for (i=0;i<3;i++) for (j=0;j<3;j++) {sub[i][j][0]=i+1; sub[i][j][1]=j+1; };
  for (i=0;i<12;i++) for (j=0;j<12;j++) for (ii=0;ii<4;ii++) for (jj=0;jj<4;jj++)
  { for (k=0;k<3;k++) x[k]=cc[ii][jj][k]; x[3]=x[2]; s1=0.0;
    for (m=0;m<3;m++)
    { s=0.0; for (k=0;k<3;k++) s=s+nd2(2,sub[m][k],x)*cdu1[k][j][ii][jj];
      s1=s1+s*cdu1[m][i][ii][jj]+nd2(1,sub[m][0],x)*cdu2[m][i][j][ii][jj]; };
    c[i][j]=c[i][j]+H[ii]*H[jj]*s1*t;
  };
};
```

```
double EM4::csurface(double x[7])
{double d,b1,b2;
  b1=x[4]; b2=x[5]; d=(x[1]-1.0+b2)*(x[0]-1.0+b1)-x[2]*x[3];
  return d*d*d-b1*b2*d*(b1*(x[1]-1.0+b2)+b2*(x[0]-1.0+b1))+b1*b1*b1*b2*b2*b2; };
```

```

double EM4::svalues(double x[7]) //Return s11+s22 for a point (c11,c22,c12)
{ double d,b1,b2;
  b1=x[4]; b2=x[5]; d=(x[1]-1.0+b2)*(x[0]-1.0+b1)-x[2]*x[3];
  return (b1+b2)*d*d-b1*b1*b2*b2*(x[0]+x[1]-2.0+b1+b2); };

void EM4::intersectpoint(int sv,double xb[7],double dx,double dy,double dz,double
    &x,double &y,double &z,int &i)
{ double x1,y1,z1,x2,y2,z2,p,v,t2,x0,y0,z0,tp[7];
  x0=xb[0]; y0=xb[1]; z0=xb[2]; for (i=0;i<7;i++) tp[i]=xb[i];
  //Get the intersection point (x2,y2,z2) of the straight line and the plane z=0
  //If z0=0 then get the intersection point of the straight line and line x=y,z=0
  if (fabs(z0*1.0e6)<1.0) {x2=dx-dy; y2=(y0*dx-x0*dy)/x2; x2=y2; }
  else {x2=x0-z0*dx/dz; y2=y0-z0*dy/dz; };
  z2=0; p=0.5; x1=x0; y1=y0; z1=z0;
  //find the mid point of (x1,y1,z1) and (x2,y2,z2)
  x=(x2+x1)*p; y=(y2+y1)*p; z=(z2+z1)*p;
  i=0; t2=1.0e-6;
  //check if point(x,y,z) is on the surface, otherwise improve it
  tp[0]=x; tp[1]=y; tp[2]=z; tp[3]=tp[2];
  if (sv==1) v=svalues(tp); else if (sv==0) v=csurface(tp);
  while
    ((fabs(v)>t2)&&((fabs(x2-x1)>1.0e-8)||((fabs(y2-y1)>1.0e-8)||((fabs(z2-z1)>1.0
      e-8))))
  { if (v>0.0) {x2=x; y2=y; z2=z; } else {x1=x; y1=y; z1=z; };
    x=(x2+x1)*p; y=(y2+y1)*p; z=(z2+z1)*p; i=i+1;
    tp[0]=x; tp[1]=y; tp[2]=z; tp[3]=tp[2];
    if (sv==1) v=svalues(tp); else if (sv==0) v=csurface(tp);
  };
  if (sv==1) {i=0; intersectpoint(0,tp,dx,dy,dz,x,y,z,i);};
};

void EM4::normals(double xb[7],double &x,double &y,double &z,double &s1,double
    &s2,double &s12)
{ double d,b1,b2,g;
  b1=xb[4];b2=xb[5]; g=xb[6]; d=(xb[1]-1.0+b2)*(xb[0]-1.0+b1)-xb[2]*xb[3]; d=1.0/d/d;
  //normal (s22,s11,-s12)
  s2=g*(1.0/b2-b1*b2*(xb[0]-1.0+b1)*d); s1=g*(1.0/b1-b1*b2*(xb[1]-1.0+b2)*d);
  s12=g*b1*b2*xb[2]*d; d=sqrt(s1*s1+s2*s2+s12*s12);
  if (fabs(d*1.0e6)>1.0) {x=s2/d; y=s1/d; z=-s12/d;};
};

void EM4::direction(double x1,double y1,double z1,double x2,double y2,double z2,
double &x,double &y,double &z)

```

```

{ double d;
  x=x2-x1; y=y2-y1; z=z2-z1; d=sqrt(x*x+y*y+z*z);
  if (fabs(d*1.0e6)>1.0) {x=x/d; y=y/d; z=z/d;} else {x=0.0; y=0.0; z=0.0;};
};

double EM4::jjcs(double x1,double y1,double z1,double x2,double y2,double z2)
{ return x1*x2+y1*y2+z1*z2; };

double EM4::jjsn(double x1,double y1,double z1,double x2,double y2,double
                 z2,double &x,double &y,double &z)
{ double d;
  x=y1*z2-z1*y2; y=z1*x2-x1*z2; z=x1*y2-y1*x2; d=sqrt(x*x+y*y+z*z);
  return d; };

void EM4::findit(int sv,double xb[7],double &x,double &y,double &z)
{int i,j; double x1,y1,z1,x2,y2,z2,d,alfa,alfa1,p,s1,s2,s12,x0,y0,z0,tp[7],b1,b2,g,sg;
 double hd=3.1415926585/180.0,jd=1/hd,x3,y3,z3; p=0.5;
 //Set initial direction: form 45 degree angle with line c12=0,c11=c22;
 x0=xb[0]; y0=xb[1]; z0=xb[2]; b1=xb[4]; b2=xb[5]; g=xb[6];
 for (i=0;i<7;i++) tp[i]=xb[i];
 x2=0.5*(x0+y0+sqrt((x0-y0)*(x0-y0)+2.0*z0*z0)); y2=x2; z2=0.0;
 d=sqrt((x2-x0)*(x2-x0)+(y2-y0)*(y2-y0)+z0*z0);
 x2=(x2-x0)/d; y2=(y2-y0)/d; z2=-z0/d;
 i=0; intersectpoint(sv,xb,x2,y2,z2,x,y,z,i); tp[0]=x; tp[1]=y; tp[2]=z; tp[3]=tp[2];
 normals(tp,x1,y1,z1,s1,s2,s12); j=0;
 alfa=jjsn(x1,y1,z1,x2,y2,z2,x3,y3,z3); alfa1=jjcs(x1,y1,z1,x2,y2,z2); sg=1.0;
 while (fabs(alfa)>=5.0e-3)
 { if (alfa1<-1.0e-8) sg=-1.0; else sg=1.0;
  x2=(x2+sg*x1)*p; y2=(y2+sg*y1)*p; z2=(z2+sg*z1)*p;
  d=sqrt(x2*x2+y2*y2+z2*z2);
  if (fabs(d)>1.0e-6) {x2=x2/d; y2=y2/d; z2=z2/d;};
  i=0; intersectpoint(sv,xb,x2,y2,z2,x,y,z,i); tp[0]=x; tp[1]=y; tp[2]=z; tp[3]=tp[2];
  normals(tp,x1,y1,z1,s1,s2,s12);
  alfa=jjsn(x1,y1,z1,x2,y2,z2,x3,y3,z3); alfa1=jjcs(x1,y1,z1,x2,y2,z2);
  j=j+1;
 };
};

double EM4::nd1(int n, int sub[2], double x[7])
{ double v,s,x1[7],rv; int i;
  for (i=4;i<7;i++) x1[i]=x[i]; for (i=0;i<4;i++) x1[i]=0.0;
  v=csurface(x); s=svalues(x);
  if ((x[0]<0.0)||(x[1]<0.0)) rv=0.0;
};

```

```

else if ((x[0]<1.0)&&(x[1]<1.0)) rv=0.0;
else if ((v>=1.0e-8)&&(s>=0.0)) rv=nd(n,sub,x);
else if (s<0.0) {findit(1,x,x1[0],x1[1],x1[2]); x1[3]=x1[2]; rv=nd(n,sub,x1);}
else if ((v<-1.0e-8)&&(s>=0.0))
    {findit(0,x,x1[0],x1[1],x1[2]);x1[3]=x1[2]; rv=nd(n,sub,x1);}
return rv;
};

double EM4::nd2(int n, int sub[2], double x[7])
{ double v,s,x1[7],rv; int i;
  for (i=4;i<7;i++) x1[i]=x[i]; for (i=0;i<4;i++) x1[i]=0.0;
  v=csurface(x); s=svalues(x);
  if ((x[0]<0.0)||x[1]<0.0)) rv=0.0;
  else if ((x[0]<1.0)&&(x[1]<1.0)) rv=0.0;
  else if ((v>=1.0e-8)&&(s>=0.0)) rv=nda(n,sub,x);
  else if (s<0.0) {findit(1,x,x1[0],x1[1],x1[2]); x1[3]=x1[2]; rv=nda(n,sub,x1);}
  else if ((v<-1.0e-8)&&(s>=0.0))
    {findit(0,x,x1[0],x1[1],x1[2]); x1[3]=x1[2]; rv=nda(n,sub,x1);}
  return rv;
};

double EM4::w1(double x[7], int &i)
{ double v,s,x1[7];
  for (i=4;i<7;i++) x1[i]=x[i]; for (i=0;i<4;i++) x1[i]=0.0;
  v=csurface(x); s=svalues(x);
  if ((x[0]<0.0)||x[1]<0.0)) {i=2; return 0.0;}
  else if ((x[0]<0.997)&&(x[1]<0.997)) {i=2; return 0.0;}
  else if ((v>=1.0e-8)&&(s>=0.0)) {i=0; return w(x);}
  else if (s<0.0) {i=1; findit(1,x,x1[0],x1[1],x1[2]); x1[3]=x1[2]; return w(x1);}
  else if ((v<-1.0e-8)&&(s>=0.0))
    {i=1; findit(0,x,x1[0],x1[1],x1[2]); x1[3]=x1[2]; return w(x1);}
  else { i=3; return w(x);};
};

double EM4::w(double x[7])
{ double c11,c22,c12,c21,bt1,bt2,gg;
  c11=x[0]-1.0; c22=x[1]-1.0; c12=x[2]; c21=x[3];
  gg=x[6]; bt1=x[4]; bt2=x[5];
  return 0.5*gg*(c11/bt1+c22/bt2-1.0+bt1*bt2/((c22+bt2)*(c11+bt1)-c12*c21));
};

double EM4::spk(int m, double x[7])
{int sub[2]; double s;

```

```

sub[0]=m+1; sub[2]=0;
if (m<2) s=nd(1,sub,x); else if (m==2)
    {s=nd(1,sub,x); sub[0]=m+2; s=0.5*(s+nd(1,sub,x)); };
return s;
};

double EM4::nda(int n, int sub[2], double x[7])
{double s,d,gg,bt1,bt2,c11,c22,c12,c21,c3,gb;
  c11=x[0]-1.0; c22=x[1]-1.0; c12=x[2]; c21=x[3]; bt1=x[4]; bt2=x[5]; gg=x[6];
  d=(c22+bt2)*(c11+bt1)-c12*c21;
  if (n==1)
    { gb=0.5*gg*bt1*bt2/d/d;
      switch (sub[0]) {
        case 1: s=0.5*gg/bt1-(c22+bt2)*gb; break;
        case 2: s=0.5*gg/bt2-(c11+bt1)*gb; break;
        case 3: s=gb*c21; break;
        default:break; } ;
    }
  else if (n==2)
    { c3=0.5*(c12+c21); gb=gg*bt1*bt2/d/d/d;
      switch (sub[0]) {
        case 1: switch (sub[1])
          {case 1:s=gb*(c22+bt2)*(c22+bt2); break;
            case 2:s=0.5*gb*(2.0*(c22+bt2)*(c11+bt1)-d); break;
            case 3:s=-gb*(c22+bt2)*c3; break;
            default:break; }; break;
          case 2: switch (sub[1])
            {case 1:s=0.5*gb*(2.0*(c22+bt2)*(c11+bt1)-d); break;
              case 2:s=gb*(c11+bt1)*(c11+bt1); break;
              case 3:s=-gb*(c11+bt1)*c3; break;
              default:break; }; break;
            case 3: switch (sub[1])
              {case 1:s=-gb*(c22+bt2)*c3; break;
                case 2:s=-gb*(c11+bt1)*c3; break;
                case 3:s=0.25*gb*(d+c3+c3); break;
                default:break; }; break;
              default:break; };
        } ;
    }
  return s;
};

double EM4::nd(int n, int sub[2], double x[7])
{int i,m=7; double x1[7],s,h;

```

```

for (i=0;i<m;i++) x1[i]=x[i]; i=sub[n-1]-1; h=fabs(x1[i])*1.0e-8;
x1[i]=x1[i]+h;
if (n>1) s=(nd(n-1, sub, x1)-nd(n-1, sub, x))/h; else s=(w1(x1,i)-w1(x,i))/h;
return s;
};

void EM4::StrainC(double ss[3],double ks, double et, double dp1[12])
{double g[2][2],cg[2][2],s,s1,s2,d,vb[6],M[2][2],c[2][2]; int i,j,k1,k2,ii,jj;
for (i=0;i<6;i++) vb[i]=0.0;
for (i=0;i<4;i++)
{ s1=NKsi(i,et); s2=NEta(i,ks);
vb[0]=vb[0]+(pt[i].GetX()+dp1[i*3])*s1; vb[3]=vb[3]+(pt[i].GetX()+dp1[i*3])*s2;
vb[1]=vb[1]+(pt[i].GetY()+dp1[i*3+1])*s1;vb[4]=vb[4]+(pt[i].GetY()+dp1[i*3+1])*s2;
vb[2]=vb[2]+(pt[i].GetZ()+dp1[i*3+2])*s1; vb[5]=vb[5]+(pt[i].GetZ()+dp1[i*3+2])*s2;
};
for (i=0;i<2;i++) for (j=0;j<2;j++) g[i][j]=0.0;
for (i=0;i<3;i++)
{ g[0][0]=g[0][0]+vb[i]*vb[i]; g[1][1]=g[1][1]+vb[i+3]*vb[i+3];
g[0][1]=g[0][1]+vb[i]*vb[i+3];
};
g[1][0]=g[0][1];
for (i=0;i<6;i++) vb[i]=0.0;
for (i=0;i<4;i++)
{ s1=NKsi(i,et); s2=NEta(i,ks);
vb[0]=vb[0]+pt[i].GetX()*s1;vb[1]=vb[1]+pt[i].GetY()*s1;vb[2]=vb[2]+pt[i].GetZ()*s1;
vb[3]=vb[3]+pt[i].GetX()*s2;vb[4]=vb[4]+pt[i].GetY()*s2;vb[5]=vb[5]+pt[i].GetZ()*s2;
};
for (i=0;i<2;i++) for (j=0;j<2;j++) cg[i][j]=0.0;
for (i=0;i<3;i++)
{ cg[0][0]=cg[0][0]+vb[i]*vb[i]; cg[1][1]=cg[1][1]+vb[i+3]*vb[i+3];
cg[0][1]=cg[0][1]+vb[i]*vb[i+3];
};
s1=sqrt(cg[0][0]); s2=sqrt(cg[1][1]); d=th1*3.1415926/180.0;
s=acos(cg[0][1]/s1/s2)*180.0/3.1415926; d=(s+th1-90.0)*3.1415926/180.0;
s=(270.0-s-s+th1)*3.1415926/180.0;
M[0][0]=cos(d)/s1; M[0][1]=sin(d)/s1; M[1][0]=cos(s)/s2; M[1][1]=sin(s)/s2;
for (i=0;i<2;i++) for (j=0;j<2;j++)
{c[i][j]=0.0;
for (ii=0;ii<2;ii++) for (jj=0;jj<2;jj++) c[i][j]=c[i][j]+g[ii][jj]*M[ii][i]*M[jj][j];
};
ss[0]=c[0][0]; ss[1]=c[1][1]; ss[2]=c[0][1];
};

```

```

double EM4::DCDU(int h, int i, int j, int k, double ks, double et, double dp1[12])
//h=1 dc/du 2 d^2c/du^2;i=0 c11, 1 c22, 2 c12; j,k=0~11 ul,vl,w1...w4
{ double s,s1,s2,s3,r[4],sk,se; int i1,j1,i2,j2,kk;
  double g[2][2],cg[2][2],vb[6],M[2][2]; int k1,k2,ii,jj;
  j1=j%3; i1=j/3; sk=NKsi(i1,et); se=NEta(i1,ks);
  for (ii=0;ii<6;ii++) vb[ii]=0.0;
  for (ii=0;ii<4;ii++)
  { s1=NKsi(ii,et); s2=NEta(ii,ks);
    vb[0]=vb[0]+pt[ii].GetX()*s1; vb[3]=vb[3]+pt[ii].GetX()*s2;
    vb[1]=vb[1]+pt[ii].GetY()*s1; vb[4]=vb[4]+pt[ii].GetY()*s2;
    vb[2]=vb[2]+pt[ii].GetZ()*s1; vb[5]=vb[5]+pt[ii].GetZ()*s2;
  };
  for (ii=0;ii<2;ii++) for (jj=0;jj<2;jj++) cg[ii][jj]=0.0;
  for (ii=0;ii<3;ii++)
  {cg[0][0]=cg[0][0]+vb[ii]*vb[ii]; cg[1][1]=cg[1][1]+vb[ii+3]*vb[ii+3];
    cg[0][1]=cg[0][1]+vb[ii]*vb[ii+3];
  };
  s1=sqrt(cg[0][0]); s2=sqrt(cg[1][1]); s=th1*3.1415926/180.0;
  s3=acos(cg[0][1]/s1/s2)*180.0/3.1415926; s=(s3+th1-90.0)*3.1415926/180.0;
  s3=(270.0-s3-s3+th1)*3.1415926/180.0;
  M[0][0]=cos(s)/s1; M[0][1]=sin(s)/s1; M[1][0]=cos(s3)/s2; M[1][1]=sin(s3)/s2;
  switch (h)
  { case 1: switch (j1)
    { case 0: for (kk=0;kk<4;kk++) r[kk]=dp1[kk*3]+pt[kk].GetX(); break;
      case 1: for (kk=0;kk<4;kk++) r[kk]=dp1[kk*3+1]+pt[kk].GetY(); break;
      case 2: for (kk=0;kk<4;kk++) r[kk]=dp1[kk*3+2]+pt[kk].GetZ(); break;
      default: break;
    };
    s1=0.0; s2=0.0; s3=0.0;
    for (kk=0;kk<4;kk++) {s1=s1+r[kk]*NKsi(kk,et); s2=s2+r[kk]*NEta(kk,ks);};
    s3=sk*s2+se*s1; s1=s1*2.0*sk; s2=s2*2.0*se;
    g[0][0]=s1; g[1][1]=s2; g[0][1]=s3; g[1][0]=s3; s=0.0;
    switch (i)
    {case 0:for (ii=0;ii<2;ii++) for (jj=0;jj<2;jj++) s=s+g[ii][jj]*M[ii][0]*M[jj][0];break;
      case 1:for (ii=0;ii<2;ii++) for (jj=0;jj<2;jj++) s=s+g[ii][jj]*M[ii][1]*M[jj][1];break;
      case 2:for (ii=0;ii<2;ii++) for (jj=0;jj<2;jj++)s=s+g[ii][jj]*M[ii][0]*M[jj][1]; break;
      default: break;
    }; break;
  }
case 2: j2=k%3; i2=k/3;
  if (j1==j2)
  {r[0]=NKsi(i2,et); r[1]=NEta(i2,ks);
    s1=2.0*sk*r[0]; s2=2.0*se*r[1]; s3=sk*r[1]+se*r[0];
    g[0][0]=s1; g[1][1]=s2; g[0][1]=s3; g[1][0]=s3; s=0.0;
  }
}

```

```

        switch (i)
        {case 0: for (ii=0;ii<2;ii++) for (jj=0;jj<2;jj++) s=s+g[ii][jj]*M[ii][0]*M[jj][0]; break;
        case 1: for (ii=0;ii<2;ii++) for (jj=0;jj<2;jj++) s=s+g[ii][jj]*M[ii][1]*M[jj][1]; break;
        case 2: for (ii=0;ii<2;ii++) for (jj=0;jj<2;jj++) s=s+g[ii][jj]*M[ii][0]*M[jj][1]; break;
        default: break; };
    } else s=0.0; break;
default: break;
};
return s;
};

void EM4::Formdp2(int NDISP, double dp1[maxv])
{int i,j,k,m,ii,jj,sub[3][2]; double x[7],ks,et,s,s1[12],cc[3];
  for (j=0;j<3;j++) {sub[j][0]=j+1; sub[j][1]=0;};
  x[4]=b1; x[5]=b2; x[6]=g;
  for (j=0;j<12;j++)
  { s1[j]=0.0; for (ii=0;ii<4;ii++) for (jj=0;jj<4;jj++)
    { ks=ki[ii]; et=ki[jj]; StrainC(cc,ks,et,dp); for (k=0;k<3;k++) x[k]=cc[k];
      x[3]=x[2]; s=0.0; for (k=0;k<3;k++) s=s+nd2(1,sub[k],x)*DCDU(1,k,j,j,ks,et,dp);
      s1[j]=s1[j]+H[ii]*H[jj]*s;
    };
  }; k=3;
  for (i=0;i<NNo;i++) for (j=0;j<3;j++)
  {m=pt[i].Getdof(j); if ((m>0)&&(m<=NDISP)) dp1[m]=dp1[m]-s1[i*k+j]*t;};
};

MPoint EM4::GetPoint(int i) {return pt[i];};

void EM4::Setdof(int i,int j,int k){pt[i].Setdof(j,k);};

void EM4::Setdp(int m, int NDISP, double DP1[maxv], MPoint pa[maxnode])
{int i,j,k;
  for (i=0;i<NNo;i++) for (j=0;j<m;j++)
  {k=pt[i].Getdof(j);
  if ((k>0)&&(k<=NDISP))
  {pt[i].Setddp(j,DP1[k]); pt[i].Setdp(j); dp[i*dofn+j]=pt[i].Getdp(j);
  pa[pt[i].GetNo()].Setddp(j,DP1[k]);
  };
};
};

double EM4::Getdp(int i){return dp[i];};
void EM4::SetB(double s1,double s2){b1=s1;b2=s2;};

```



```

double EM4::GetB(int i){ if (i==1) return b1; else return b2; };
void EM4::SetG(double s){g=s;};
double EM4::GetG(){return g;};
void EM4::SetTh1(double s){th1=s;};
double EM4::GetTh1(){return th1;};
void EM4::Sett(double s){t=s;};
double EM4::Gettt(){return t;};

void EM4::SetLoad(EM4Load s)
{ld[ln]=EM4Load(s.Gettype(),s.Getdof(),s.Getnodeno(),s.Getv()); ln=ln+1;};

void EM4::FORMII(int i, int II[12])
{int j,k;
 for (j=0;j<12;j++) II[j]=0;
 for (j=0;j<NNo;j++) for (k=0;k<i;k++) II[k+j*i]=pt[j].Getdof(k);
};

void EM4::Fofmdp(int NDISP, double dp[maxv])
{int i,j,k,m; double s;
 if (ln>0) for (i=0;i<ln;i++) switch (ld[i].Gettype())
 {case 1: k=ld[i].Getnodeno();
   for (j=0;j<NNo;j++) if (pt[j].GetNo()==k) m=pt[j].Getdof(ld[i].Getdof()-1);
   if ((m>0)&&(m<=NDISP)) dp[m]=dp[m]+ld[i].Getv(); break;
 case 2: k=ld[i].Getnodeno();
   for (j=0;j<NNo;j++) if (pt[j].GetNo()==k) m=pt[j].Getdof(ld[i].Getdof()-1);
   if ((m>0)&&(m<=NDISP)) dp[m]=dp[m]+ld[i].Getv(); break;
 case 3: for (m=0;m<4;m++)
   {s=0.0; for (j=0;j<4;j++) for (k=0;k<4;k++) s=s+H[j]*H[k]*N(m,ki[j],ki[k]);
   j=pt[m].Getdof(ld[i].Getdof()-1);
   if ((j>0)&&(j<=NDISP)) dp[j]=dp[j]+ld[i].Getv()*s; }; break;
 default: break;
 };
};

void EM4::SetEM4(int aNo,int p[NNo],MPoint pp[],double aG,double at,double
ab1,double ab2,double ath)
{int i,j,s[3];
 No=aNo; g=aG; t=at; b1=ab1; b2=ab2; th1=ath;
 for (i=0;i<NNo;i++)
 {for (j=0;j<dofn;j++) s[j]=pp[p[i]].Getdof(j);
 pt[i].SetMPoint(pp[p[i]].GetNo(),pp[p[i]].GetX(),pp[p[i]].GetY(),pp[p[i]].GetZ(),s);};
 for (i=0;i<12;i++) dp[i]=0.0;
};

```

```
EM4::EM4(int aNo,int p[NNo],MPoint pp[],double aG,double at,double ab1,double
        ab2,double ath)
```

```
{int i,j,s[3];
  No=aNo; ln=0; g=aG; t=at; b1=ab1; b2=ab2; th1=ath;
  for (i=0;i<NNo;i++)
    {for (j=0;j<dofn;j++) s[i]=pp[p[i]].Getdof(j);
      pt[i]=MPoint(pp[p[i]].GetNo(),pp[p[i]].GetX(),pp[p[i]].GetY(),pp[p[i]].GetZ(),s); };
    for (i=0;i<12;i++) dp[i]=0.0;
  };
```

```
EM4::EM4()
```

```
{int i,s[3]={0,0,0};
  for (i=0;i<NNo;i++) pt[i]=MPoint(0,0.0,0.0,0.0,s); ln=0;
  for (i=0;i<12;i++) dp[i]=0.0;
};
```

```
EM4::~~EM4()
```

```
{int i;
  for (i=0;i<NNo;i++) pt[i].~MPoint(); if (ln>0) for (i=0;i<ln;i++) ld[i].~EM4Load();
};
```

```
//***** Sheet2 *****
```

```
void Sheet2::ASSEMB2(int IB,int II[12],double C[12][12],double R[maxk],int NDISP,int
                    BV,int V[maxv],int PHASE)
```

```
{ int I,J,K,IA,JA;
  for (I=1;I<=IB;I++)
    { for (J=1;J<=I;J++)
      { IA=II[I-1]; JA=II[J-1]; if (IA<JA) { K=IA; IA=JA; JA=K; };
        if ((JA>0)&&(IA<=NDISP))
          {if (PHASE==2) {K=V[BV+IA]-IA+JA; R[K]=R[K]+C[I-1][J-1];}
            else if (V[BV+IA]<IA-JA) V[BV+IA]=IA-JA;};
      };
    };
};
```

```
void Sheet2::DIAGADR(int NDISP,int BV,int IV, int V[maxv])
```

```
{ int I;
  V[BV]=IV; for (I=1;I<=NDISP;I++) V[BV+I]=V[BV+I]+V[BV+I-1]+1;
};
```

```
void Sheet2::LDLT1(int NDISP,int BV,int V[maxv],double R[maxk],double T[maxv])
```

```
{ int H,I,J,K,L,G,P,VI1,VJ1,VI,VJ; double S,SK;
```

```

VI=V[BV];
for (I=1;I<=NDISP;I++)
{ VI1=VI; VI=V[BV+I]; H=I+1+VI1-VI; VJ=V[BV+H-1];
  for (J=H;J<=I;J++)
  { S=0.0; VJ1=VJ; VJ=V[J+BV]; G=VJ-J; P=J-1; L=VJ1+1-G; if (L<H) L=H;
    for (K=L;K<=P;K++) S=S+T[I-K]*R[G+K];
    if (I==J) { SK=R[VI]; S=R[VI]-S; R[VI]=S; }
    else { G=VI-I+J; S=R[G]-S; T[I-J]=S; R[G]=S/R[VJ]; };
  };
};

void Sheet2::SLVEQ1(int NDISP,int BV,int V[maxv],double R[maxk],double
                  B[maxv],int BB)
{ int I,J,K,P,VI,VI1,Q; double S;
  VI=V[BV];
  for (I=1;I<=NDISP;I++)
  { VI1=VI; VI=V[BV+I]; P=VI-I; Q=I-1; S=0.0;
    for (J=VI1+1-P;J<=Q;J++) S=S+R[P+J]*B[J+BB]; B[BB+I]=B[BB+I]-S; }
  for (I=1;I<=NDISP;I++) B[BB+I]=B[BB+I]/R[V[BV+I]];
  for (I=NDISP;I>1;I--)
  { VI=V[BV+I]; J=I-1; VI1=V[BV+J]; P=VI-I; S=B[BB+I];
    for (K=VI1+1-P;K<=J;K++) B[BB+K]=B[BB+K]-R[P+K]*S;
  };
};

void Sheet2::SetGDK(int NDISP,int BV, int V[maxv], double R[maxk])
{int i,j,k;
  if (gdnumber>0) for (i=0;i<gdnumber;i++)
  {j=gd[i].Gettdof();
   if ((j>0)&&(j<=NDISP)) {k=V[BV+j]; R[k]=R[k]*1.0e6; gd[i].Setrk(R[k]);};
  };
};

void Sheet2::SetGDR(int NDISP, double B[maxk])
{int i,j;
  if (gdnumber>0) for (i=0;i<gdnumber;i++)
  {j=gd[i].Gettdof(); if ((j>0)&&(j<=NDISP)) {B[j]=gd[i].Getrk()*gd[i].Getv();};};
};

MPoint Sheet2::GetMPoint(int i) {return pa[i-1];};
EM4 Sheet2::GetEM4(int i){return ea[i];};
EM4Load Sheet2::GetEM4Load(int i){return ld[i];};

```

```

int Sheet2::Setdof(int p) //p=2 linear case, p=3 nonlinear case
{ int I,J,k,I1,J1;
  k=0;
  for (I=0;I<nodenumber;I++) for (J=0;J<p;J++) if (pa[I].Getdof(J)==1)
  { k=k+1; for (I1=0;I1<elementnumber;I1++) for (J1=0;J1<NNo;J1++)
  if (ea[I1].GetPoint(J1).GetNo()==I) {ea[I1].Setdof(J1,J,k); break;};
  if (gdnumber>0) for (I1=0;I1<gdnumber;I1++)
  if ((I==gd[I1].Getno())&&(J==gd[I1].Getdof())){gd[I1].Setdof(k); break;};}
  return k;
};

void Sheet2::Analyse()
{ int m,i,k,I,J,K,JJ,BV,kk,dofm,sub[3][2],II[12],V[maxv];
  double mdp,mrv,mrv0,mdp0,dpe[12],x[7],cc[3],ss[3],s,t1,s1,s2,ZK[maxk],ZK1[maxk];
  double C[12][12],DP[maxv],jd=180.0/3.1415926;
  MPoint p;

  ofstream fout("middle.r");
  dof=2;
  if (loadnumber>0) for (I=0;I<loadnumber;I++) if (ld[I].Getdof()==2) dof=3;
  if (gdnumber>0) for (I=0;I<gdnumber;I++) if (gd[I].Getdof()==2) dof=3;
  NDISP=Setdof(dof); BV=1;
  for (I=0;I<maxv;I++) V[I]=0;
  for (I=0;I<elementnumber;I++)
  {ea[I].FORMII(dof,II); ASSEMB2(dof*4,II,C,ZK,NDISP,BV,V,0); };
  DIAGADR(NDISP,BV,0,V);
  fout<<"NDISP="<<NDISP<<" V[NDISP]="<<V[NDISP+BV]<<"\n";
  for (I=0;I<maxk;I++) ZK[I]=0.0;
  for (I=0;I<elementnumber;I++)
  {ea[I].FORMII(dof,II);
  ea[I].STIF0(C); ASSEMB2(12,II,C,ZK,NDISP,BV,V,2);
  };
  SetGDK(NDISP;BV,V,ZK);
  LDLT1(NDISP,BV,V,ZK,DP);
  for (I=0;I<maxv;I++) DP[I]=0.0;
  for (I=0;I<elementnumber;I++) ea[I].Formdp(NDISP,DP);
  SetGDR(NDISP,DP);
  mrv0=0.0; for (I=0;I<maxv;I++) if (fabs(DP[I])>mrv0) mrv0=fabs(DP[I]);
  SLVEQ1(NDISP,BV,V,ZK,DP,0);
  fout<<"-----Linear Displacement-----\n";
  for (I=1;I<=NDISP;I++) fout<<I<<" "<<DP[I]<<"\n";
  fout<<"*****\n\n\n";
}

```

```

for (I=0;I<elementnumber;I++) ea[I].Setdp(dofn,NDISP,DP,pa);
for (I=0;I<nodenumbr;I++) for (K=0;K<dofn;K++) pa[I].Setdp(K);
mdp0=0.0;
for (I=0;I<nodenumbr;I++) for (J=0;J<dofn;J++) if (pa[I].Getdof(J)>0)
  if (fabs(pa[I].Getdp(J))>mdp0) mdp0=fabs(pa[I].Getdp(J));
fout<<"\n\nNonlinear Analysis...\n";
NDISP=Setdof(dofn); BV=1;
for (I=0;I<maxv;I++) V[I]=0;
for (I=0;I<elementnumber;I++)
  {ea[I].FORMII(dofn,II); ASSEMB2(12,II,C,ZK,NDISP,BV,V,0); };
DIAGADR(NDISP,BV,0,V); JJ=0;
fout<<"NDISP="<<NDISP<<"  V[NDISP]="<<V[NDISP+BV]<<"\n";
do
{ if ((JJ==0)||((JJ%5==0)))
  { for (I=0;I<maxk;I++) ZK1[I]=0.0;
    for (I=0;I<elementnumber;I++)
      { ea[I].FORMII(dofn,II);
        ea[I].STIFT(C); ASSEMB2(12,II,C,ZK1,NDISP,BV,V,2);
      };
    };
  for (I=0;I<maxk;I++) ZK[I]=ZK1[I];
  if (gdnumber>0) for (I=0;I<gdnumber;I++)
    {J=gd[I].Getno(); K=gd[I].Getdof(); gd[I].Setv(gd[I].Getv()-pa[J].Getddp(K)); };
  SetGDK(NDISP,BV,V,ZK);
  LDLT1(NDISP,BV,V,ZK,DP);
  for (I=0;I<maxv;I++) DP[I]=0.0;
  for (I=0;I<elementnumber;I++) ea[I].Formdp(NDISP,DP);
  for (I=0;I<elementnumber;I++) ea[I].Formdp2(NDISP,DP);
  SetGDR(NDISP,DP);
  mrv=0.0; for (I=0;I<maxv;I++) if (fabs(DP[I])>mrv) mrv=fabs(DP[I]);
  mrv=mrv/mrv0;
  SLVEQ1(NDISP,BV,V,ZK,DP,0);
  fout<<"-----Displacement-----\n";
  for (I=1;I<=NDISP;I++) fout<<I<<" "<<DP[I]<<"\n";
  fout<<"*****\n";
mdp=0.0;
  for (I=1;I<=NDISP;I++) if (fabs(DP[I])/mdp0>mdp)
    {mdp=fabs(DP[I])/mdp0; kk=I;};
  for (I=0;I<elementnumber;I++) ea[I].Setdp(dofn,NDISP,DP,pa);
  for (I=0;I<nodenumbr;I++) for (K=0;K<dofn;K++) pa[I].Setdp(K);
  mdp0=0.0;
  for (I=0;I<nodenumbr;I++) for (J=0;J<dofn;J++) if (pa[I].Getdof(J)>0)
    if (fabs(pa[I].Getdp(J))>mdp0) mdp0=fabs(pa[I].Getdp(J));

```

```

JJ=JJ+1;
fout<<"Loop:"<<JJ<<" Max Ddp/Dp="<<mdp<<" Max RightVector="<<mrsv<<"
      Mdp0="<<mdp0<<" I="<<kk<<"\n";
} while ((JJ==1)||((mdp>5.0e-2)||((mrsv>1.0e-2)));
fout<<"*****\n";
fout.close(); Outresult(1);
};

```

```

void Sheet2::SetSheet2(CString fn)

```

```

{int i,j,k,m,ii,ke,ij[4],adof[3]; double x,y,z,emt[6][5];

```

```

    ifstream fin(fn); fin>>type>>nodenumber; for (i=0; i<nodenumber; i++)
    {fin>>x>>y>>z; for (j=0;j<3;j++) fin>>adof[j];
    pa[i].SetMPoint(i,x,y,z,adof); };
    fin>>ke; for (i=0;i<ke;i++) for (j=0;j<4;j++) fin>>emt[i][j];
    fin>>elementnumber; for (i=0;i<elementnumber;i++)
    {fin>>ij[0]>>ij[1]>>ij[2]>>ij[3]>>m>>x; m=m-1; for (j=0;j<NNo;j++) ij[j]=ij[j]-1;
    ea[i].SetEM4(i,ij,pa,emt[m][0],emt[m][1],emt[m][2],emt[m][3],x);
    };
    fin>>gdnumber; if (gdnumber>0) for (i=0;i<gdnumber;i++)
    {fin>>j>>k>>x; gd[i]=GivenDp(j-1,k-1,x); };
    fin>>loadnumber; if (loadnumber>0) for (i=0;i<loadnumber;i++)
    {fin>>ii>>j>>k>>m>>x; m=m-1; ld[i]=EM4Load(j,k,m,x); ii=ii-1;
    for (k=0;k<NNo;k++) if (ea[ii].GetPoint(k).GetNo()==m) ea[ii].SetLoad(ld[i]);};
    fin.close(); NDISP=0;
    Outresult(0);
};

```

```

void Sheet2::Outresult(int ii)

```

```

{int i,j,k,m,sub[3][2]; double s,s1,s2,t1,ss[3],x[7],dpe[12],cc[3],jd=180.0/3.1415926;

```

```

    if (ii==0)

```

```

    { ofstream fout("slr.r0");
    fout<<nodenumber<<"\n";
    for (i=0; i<nodenumber; i++)
    { fout<<i+1<<" "<<pa[i].GetX()<<" "<<pa[i].GetY()<<" "<<pa[i].GetZ()<<" ";
    for (j=0;j<3;j++) fout<<pa[i].Getdof(j); fout<<"\n"; };
    fout<<elementnumber<<"\n";
    for (i=0;i<elementnumber;i++)
    {fout<<i+1; for (j=0;j<4;j++) fout<<" "<<ea[i].GetPoint(j).GetNo()+1;
    fout<<" "<<ea[i].GetG()<<" "<<ea[i].Gett()<<" "<<ea[i].GetTh1()<<
    " "<<ea[i].GetB(1)<<" "<<ea[i].GetB(2)<<"\n"; };
    fout<<gdnumber<<"\n"; if (gdnumber>0)
    for (i=0;i<gdnumber;i++)

```

```

        fout<<i+1<<" "<<gd[i].Getno()+1<<" "<<gd[i].Getdof()<<" "<<gd[i].Getv()<<"\n";
        fout<<loadnumber<<"\n";
        if (loadnumber>0) for (i=0;i<loadnumber;i++)
            fout<<i+1<<" "<<ld[i].Gettype()<<" "<<ld[i].Getnodeno()+1<<" "<<ld[i].Getdof()<<
                " "<<ld[i].Getv()<<"\n";
        fout.close();}
else if (ii==1)
{ofstream fout1("slr.r1");
  ofstream fout2("slr.r2");  fout2<<type<<" "<<nodenumber<<"\n";
  for (i=0;i<nodenumber;i++)
  {fout2<<pa[i].GetX()<<" "<<pa[i].GetY()<<" "<<pa[i].GetZ()<<" ";
    for (k=0;k<dofn;k++) fout2<<pa[i].Getdp(k)<<" "; fout2<<"\n";}
  fout2<<elementnumber<<"\n";
  for (i=0;i<nodenumber;i++) {fout1<<"Node"<<i+1<<": ";
    for (k=0;k<dofn;k++) fout1<<pa[i].Getdp(k)<<" "; fout1<<"\n";}
  fout1<<"
  _____\n";
  for (k=0;k<dofn;k++) {sub[k][0]=k+1; sub[k][1]=0;};
  for (i=0;i<elementnumber;i++)
  { fout1<<"Element No:"<<i+1<<" "; for (k=0;k<12;k++) dpe[k]=ea[i].Getdp(k);
    ea[i].StrainC(cc,0.0,0.0,dpe);x[4]=ea[i].GetB(1);x[5]=ea[i].GetB(2);x[6]=ea[i].GetG();
    for (k=0;k<NNNo;k++) fout2<<ea[i].GetPoint(k).GetNo()+1<<" ";
    for (k=0;k<3;k++) {x[k]=cc[k]; fout1<<x[k]<<" "; }; x[3]=x[2];
    s=ea[i].w1(x,m); fout2<<m<<" ";
    for (k=0;k<dofn;k++)
      {ss[k]=2.0*ea[i].nd2(1,sub[k],x); fout1<<ss[k]<<","; fout2<<ss[k]<<" "};
    t1=0.5*atan2(ss[2]+ss[2],ss[0]-ss[1]);
    s=sqrt(0.25*(ss[0]-ss[1])*(ss[0]-ss[1])+ss[2]*ss[2]);
    s2=0.5*(ss[0]+ss[1]); s1=s2+s; s2=s2-s;
    fout1<<"s1="<<s1<<" s2="<<s2<<" t1="<<t1*jd<<" "<<m<<"\n";
    fout2<<s1<<" "<<s2<<" "<<t1<<"\n";
  };
  fout1<<"
  _____\n";
  fout1.close(); fout2.close();};
};

Sheet2::Sheet2(){};

Sheet2::~~Sheet2()
{int i;
  if (nodenumber>0) for (i=0;i<nodenumber;i++) pa[i].~MPoint();
  if (elementnumber>0) for (i=0;i<elementnumber;i++) ea[i].~EM4();
  if (gdnumber>0) for (i=0;i<gdnumber;i++) gd[i].~GivenDp();
  if (loadnumber>0) for (i=0;i<loadnumber;i++) ld[i].~EM4Load();
}

```

```

};

//-----The Postprocessor-----
// s1View.cpp : implementation of the CS1View class
// CS1View drawing

void CS1View::OnDraw(CDC* pDC)
{ CPen aPen1,aPen2,aPen3;   CBrush aBrush1,aBrush2;
  int i,j,k,nn,ne,me[200][5],nd,ty,xl=200,yl=300,zl=100,z0=250,y0=350;
  double ncd[200][7],xmin,xmax,ymin,ymax,zmin,zmax,xd,th1[200],th;
  CString s,s1;

  CS1Doc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);
  if (drawmode==0)
  {ifstream fin0(dfn);  fin0>>ty>>nn;
   for (i=0; i<nn; i++) for (j=0;j<6;j++) fin0>>ncd[i][j];
   fin0>>k; for (i=0; i<k; i++) fin0>>xd>>xd>>xd>>xd;
   fin0>>ne; for (i=0;i<ne;i++) { for (j=0;j<5;j++) fin0>>me[i][j]; fin0>>xd;};
   for (i=0; i<nn; i++) for (j=3;j<6;j++) ncd[i][j]=0.0;
   fin0>>nd; if (nd>0) for (i=0; i<nd; i++)
   {fin0>>j>>k>>xd; ncd[j-1][k+2]=xd; };
   fin0.close();
  }
  else
  {ifstream fin1("slr.r2"); fin1>>ty; fin1>>nn; th=0.0;
   for (i=0; i<nn; i++) for (j=0;j<6;j++) fin1>>ncd[i][j];
   fin1>>ne;
   for (i=0;i<ne;i++)
   {for (j=0;j<5;j++) fin1>>me[i][j];
    fin1>>xmin>>ymin>>zmin>>ncd[i][6]>>ymax>>th1[i];
    if (ncd[i][6]>th) th=ncd[i][6];
   };
   fin1.close();
  };
  switch (drawmode) {
  case 0:
    for (i=0; i<nn; i++) for (j=0;j<3;j++) ncd[i][j+3]=ncd[i][j]+ncd[i][j+3];
    for (i=0;i<ne;i++) for (j=0;j<4;j++) me[i][j]=me[i][j]-1;
    xmin=100.0; xmax=-100.0; ymin=100.0; ymax=-100.0; zmin=100.0; zmax=-100.0;
    for (i=0;i<nn;i++)
    { xd=ncd[i][3]; if (xmin>xd) xmin=xd; if (xmax<xd) xmax=xd;
      xd=ncd[i][4]; if (ymin>xd) ymin=xd; if (ymax<xd) ymax=xd;
    }
  }
}

```



```

        xd=ncd[i][5]; if (zmin>xd) zmin=xd; if (zmax<xd) zmax=xd; };
xmin=xmax-xmin; ymin=ymax-ymin; zmin=zmax-zmin;
if (xmin>0.001) xmax=0.707*xl/xmin; else xmax=1.0;
if (ymin>0.001) ymax=yl/ymin; else ymax=1.0;
if (zmin>0.001) zmax=zl/zmin; else zmax=1.0;
aPen2.CreatePen(PS_SOLID,1,RGB(255,0,0));
pDC->SelectStockObject(NULL_BRUSH);
/*
pDC->SelectObject(&aPen2);
pDC->MoveTo(y0,z0); k=20;
pDC->LineTo(y0+yl+k,z0);
pDC->LineTo(y0+yl+k-15,z0-5);
pDC->MoveTo(y0+yl+k,z0);
pDC->LineTo(y0+yl+k-15,z0+5);
pDC->TextOut(y0+yl+k+2,z0-10,"y");
pDC->MoveTo(y0,z0);
pDC->LineTo(y0,z0-zl-k);
pDC->LineTo(y0+5,z0-zl-k+15);
pDC->MoveTo(y0,z0-zl-k);
pDC->LineTo(y0-5,z0-zl-k+15);
pDC->TextOut(y0+15,z0-zl-k-10,"z");
pDC->MoveTo(y0,z0);
y1=y0+int(-(xl+k+k)*0.707);
z1=z0+int((xl+k+k)*0.707);
pDC->LineTo(y1,z1);
pDC->LineTo(y1+15,z1-6);
pDC->MoveTo(y1,z1);
pDC->LineTo(y1+6,z1-15);
pDC->TextOut(y1-15,z1-20,"x");
*/
aPen1.CreatePen(PS_SOLID,3,RGB(0,0,0));
aPen3.CreatePen(PS_SOLID,3,RGB(0,0,255));
pDC->SelectStockObject(NULL_BRUSH);
pDC->SelectObject(&aPen1);
for (i=0;i<nn;i++)
/* { yi[i]=y0+int(ncd[i][4]*ymax-ncd[i][3]*xmax);
    zi[i]=z0-int(ncd[i][5]*zmax-ncd[i][3]*xmax);};
*/ { yi[i]=y0+int(ncd[i][0]*ymax); //-ncd[i][0]*xmax);
    zi[i]=z0-int(ncd[i][1]*ymax); } //-ncd[i][0]*xmax);};
for (i=0;i<ne;i++)
{pDC->SelectObject(&aPen3); for (j=0;j<3;j++)
  {y1=yi[me[i][j]]; z1=zi[me[i][j]];
   y2=yi[me[i][j+1]]; z2=zi[me[i][j+1]];
   pDC->MoveTo(y1,z1); pDC->LineTo(y2,z2); }
  y1=yi[me[i][3]]; z1=zi[me[i][3]];

```

```

        y2=yi[me[i][0]]; z2=zi[me[i][0]];
        pDC->MoveTo(y1,z1); pDC->LineTo(y2,z2);
/*      if (me[i][4]==2) pDC->SelectObject(&aPen2); else pDC->SelectObject(&aPen3);
        y1=yi[me[i][3]]; z1=zi[me[i][3]];
        y2=yi[me[i][1]]; z2=zi[me[i][1]];
        pDC->MoveTo(y1,z1); pDC->LineTo(y2,z2);
        y1=yi[me[i][2]]; z1=zi[me[i][2]];
        y2=yi[me[i][0]]; z2=zi[me[i][0]];
        pDC->MoveTo(y1,z1); pDC->LineTo(y2,z2);
*/    };
    break;
    case 1:
        for (i=0; i<nn; i++) for (j=0; j<3; j++) ncd[i][j+3]=ncd[i][j+3]+ncd[i][j];
        for (i=0; i<ne; i++) for (j=0; j<4; j++) me[i][j]=me[i][j]-1;
        xmin=100.0; xmax=-100.0; ymin=100.0; ymax=-100.0; zmin=100.0; zmax=-100.0;
        for (i=0; i<nn; i++)
        { xd=ncd[i][0]; if (xmin>xd) xmin=xd; if (xmax<xd) xmax=xd;
          xd=ncd[i][1]; if (ymin>xd) ymin=xd; if (ymax<xd) ymax=xd;
          xd=ncd[i][5]; if (zmin>xd) zmin=xd; if (zmax<xd) zmax=xd; };
        xmin=xmax-xmin; ymin=ymax-ymin; zmin=zmax-zmin;
        if (xmin>0.001) xmax=xl/xmin; else xmax=1.0;
        if (ymin>0.001) ymax=0.5*yl/ymin; else ymax=1.0;
        if (zmin>0.001) zmax=zl/zmin; else zmax=1.0;
        //if (zmax>ymax) zmax=ymax; else ymax=zmax;
        aPen2.CreatePen(PS_SOLID,1,RGB(255,0,0));
        pDC->SelectStockObject(NULL_BRUSH);
        pDC->SelectObject(&aPen2);
/*      pDC->MoveTo(y0,z0); k=20;
        pDC->LineTo(y0+y1+k,z0);
        pDC->LineTo(y0+y1+k-15,z0-5);
        pDC->MoveTo(y0+y1+k,z0);
        pDC->LineTo(y0+y1+k-15,z0+5);
        pDC->TextOut(y0+y1+k+2,z0-10,"y");
        pDC->MoveTo(y0,z0);
        pDC->LineTo(y0,z0-zl-k);
        pDC->LineTo(y0+5,z0-zl-k+15);
        pDC->MoveTo(y0,z0-zl-k);
        pDC->LineTo(y0-5,z0-zl-k+15);
        pDC->TextOut(y0+15,z0-zl-k-10,"z");
        pDC->MoveTo(y0,z0);
        y1=y0+int(-(xl+k+k)*0.707);
        z1=z0+int((xl+k+k)*0.707);
        pDC->LineTo(y1,z1);

```

```

pDC->LineTo(y1+15,z1-6);
pDC->MoveTo(y1,z1);
pDC->LineTo(y1+6,z1-15);
pDC->TextOut(y1-15,z1-20,"x");
*/
aPen1.CreatePen(PS_SOLID,3,RGB(0,0,0));
aPen3.CreatePen(PS_SOLID,3,RGB(0,0,255));
pDC->SelectStockObject(NULL_BRUSH);
pDC->SelectObject(&aPen1);
/*
for (i=0;i<nn;i++)
{ yi[i]=y0+ int (ncd[i][1]*ymax-ncd[i][0]*xmax);
  zi[i]=z0- int (ncd[i][2]*zmax-ncd[i][0]*xmax);};
for (i=0;i<ne;i++)
{pDC->SelectObject(&aPen1); for (j=0;j<3;j++)
  {y1=yi[me[i][j]];   z1=zi[me[i][j]];
   y2=yi[me[i][j+1]]; z2=zi[me[i][j+1]];
   pDC->MoveTo(y1,z1); pDC->LineTo(y2,z2); }
  y1=yi[me[i][3]];   z1=zi[me[i][3]];
  y2=yi[me[i][0]];   z2=zi[me[i][0]];
  pDC->MoveTo(y1,z1); pDC->LineTo(y2,z2);
  y1=int (0.25*(yi[me[i][0]]+yi[me[i][2]]+yi[me[i][1]]+yi[me[i][3]]));
  z1=int (0.25*(zi[me[i][0]]+zi[me[i][2]]+zi[me[i][1]]+zi[me[i][3]]));
  pDC->SelectObject(&aPen2);
  if (me[i][4]==1)
  {j=me[i][0]; k=me[i][1]; xd=(ncd[k][0]-ncd[j][0])*0.5; xd=0.1; th=th1[i];
   j=xd*cos(th)*xmax; k=xd*sin(th)*ymax;
   pDC->MoveTo(y1-k+j,z1-j);pDC->LineTo(y1+k-j,z1+j);}
  else if (me[i][4]==2) pDC->TextOut(y1-5,z1-5,"+");
};
*/
for (i=0;i<nn;i++)
{ yi[i]=y0+ int (ncd[i][3]*xmax);
  zi[i]=z0- int (ncd[i][4]*ymax+ncd[i][5]*zmax);};
for (i=0;i<ne;i++)
{pDC->SelectObject(&aPen3);
  for (j=0;j<3;j++)
  {y1=yi[me[i][j]];   z1=zi[me[i][j]];
   y2=yi[me[i][j+1]]; z2=zi[me[i][j+1]];
   pDC->MoveTo(y1,z1); pDC->LineTo(y2,z2); }
  y1=yi[me[i][3]];   z1=zi[me[i][3]];
  y2=yi[me[i][0]];   z2=zi[me[i][0]];
  pDC->MoveTo(y1,z1); pDC->LineTo(y2,z2);
/*
  y1=int (0.25*(yi[me[i][0]]+yi[me[i][2]]+yi[me[i][1]]+yi[me[i][3]]));
  z1=int (0.25*(zi[me[i][0]]+zi[me[i][2]]+zi[me[i][1]]+zi[me[i][3]]));

```

```

    pDC->SelectObject(&aPen2);
    if (me[i][4]==1)
    {j=me[i][0]; k=me[i][1]; xd=(ncd[k][0]-ncd[j][0])*0.5;
    j=xd*cos(th1[i])*xmax; k=xd*sin(th1[i])*ymax;
    pDC->MoveTo(y1-k+j,z1-j);pDC->LineTo(y1+k-j,z1+j);}
    else if (me[i][4]==2) pDC->TextOut(y1-5,z1-5,"+");
*/   };
    break;
case 2:
    for (i=0; i<nn; i++) for (j=0;j<3;j++) ncd[i][j+3]=ncd[i][j+3]+ncd[i][j];
    for (i=0;i<ne;i++) for (j=0;j<4;j++) me[i][j]=me[i][j]-1;
    xmin=100.0; xmax=-100.0; ymin=100.0; ymax=-100.0; zmin=100.0; zmax=-100.0;
    for (i=0;i<nn;i++)
    { xd=ncd[i][0]; if (xmin>xd) xmin=xd; if (xmax<xd) xmax=xd;
      xd=ncd[i][1]; if (ymin>xd) ymin=xd; if (ymax<xd) ymax=xd;
      xd=ncd[i][2]; if (zmin>xd) zmin=xd; if (zmax<xd) zmax=xd; };
    xmin=xmax-xmin; ymin=ymax-ymin; zmin=zmax-zmin;
    xl=400; yl=400; if (ty==1) {xl=200; yl=200; y0=y0-200; z0=z0+200; };
    if (xmin>0.001) xmax=xl/xmin; else xmax=1.0;
    if (ymin>0.001) ymax=yl/ymin; else ymax=1.0;
    if (zmin>0.001) zmax=zl/zmin; else zmax=1.0;
    //if (zmax>ymax) zmax=ymax; else ymax=zmax;
// aPen2.CreatePen(PS_SOLID,1,RGB(150,100,0));
aPen2.CreatePen(PS_SOLID,1,RGB(250,0,0));
pDC->SelectStockObject(NULL_BRUSH);
pDC->SelectObject(&aPen2);
pDC->MoveTo(y0,z0); k=20; xl=xl/2;
pDC->LineTo(y0+xl+k,z0);
pDC->LineTo(y0+xl+k-15,z0-5);
pDC->MoveTo(y0+xl+k,z0);
pDC->LineTo(y0+xl+k-15,z0+5);
pDC->TextOut(y0+xl+k+2,z0-10,"x");
pDC->MoveTo(y0,z0); yl=z0-k-30;
pDC->LineTo(y0,z0-yl-k);
pDC->LineTo(y0+5,z0-yl-k+15);
pDC->MoveTo(y0,z0-yl-k);
pDC->LineTo(y0-5,z0-yl-k+15);
pDC->TextOut(y0+15,z0-yl-k-10,"y");
aPen1.CreatePen(PS_SOLID,3,RGB(0,0,0));
aPen3.CreatePen(PS_SOLID,3,RGB(0,0,255));
pDC->SelectStockObject(NULL_BRUSH);
/* pDC->SelectObject(&aPen1);
    for (i=0;i<nn;i++)

```

```

{ yi[i]=y0+ int (ncd[i][0]*xmax);
  zi[i]=z0- int (ncd[i][1]*ymax);};
for (i=0;i<ne;i++)
{pDC->SelectObject(&aPen1); for (j=0;j<3;j++)
  {y1=yi[me[i][j]];    z1=zi[me[i][j]];
   y2=yi[me[i][j+1]]; z2=zi[me[i][j+1]];
   pDC->MoveTo(y1,z1); pDC->LineTo(y2,z2); }
  y1=yi[me[i][3]];    z1=zi[me[i][3]];
  y2=yi[me[i][0]];    z2=zi[me[i][0]];
  pDC->MoveTo(y1,z1); pDC->LineTo(y2,z2);
  y1=int (0.25*(yi[me[i][0]]+yi[me[i][2]]+yi[me[i][1]]+yi[me[i][3]]));
  z1=int (0.25*(zi[me[i][0]]+zi[me[i][2]]+zi[me[i][1]]+zi[me[i][3]]));
  pDC->SelectObject(&aPen2);
  if (me[i][4]==1)
  { th=0.0; xd=0.05; th=th+th1[i]; if ((i>=ne/4)&&(i<ne/4*3)) th=th+1.5707963;
    j=xd*cos(th)*xmax; k=xd*sin(th)*ymax;
    pDC->MoveTo(y1-j,z1+k);pDC->LineTo(y1+j,z1-k);}
  else if (me[i][4]==2) pDC->TextOut(y1-5,z1-5,"+");
};
*/ for (i=0;i<nn;i++)
{ yi[i]=y0+ int (ncd[i][3]*xmax);
  zi[i]=z0- int (ncd[i][4]*ymax);};
pDC->SelectObject(&aPen3);
for (i=0;i<ne;i++)
{ for (j=0;j<3;j++)
  {y1=yi[me[i][j]];    z1=zi[me[i][j]];
   y2=yi[me[i][j+1]]; z2=zi[me[i][j+1]];
   pDC->MoveTo(y1,z1); pDC->LineTo(y2,z2); }
  y1=yi[me[i][3]];    z1=zi[me[i][3]];
  y2=yi[me[i][0]];    z2=zi[me[i][0]];
  pDC->MoveTo(y1,z1); pDC->LineTo(y2,z2);
};
pDC->SelectObject(&aPen2);
xmin=ncd[me[0][1]][0]-ncd[me[0][3]][0];
ymin=ncd[me[0][1]][1]-ncd[me[0][3]][1];
xmin=sqrt(xmin*xmin+ymin*ymin)/th*0.25;
for (i=0;i<ne;i++)
{ y1=int (0.25*(yi[me[i][0]]+yi[me[i][2]]+yi[me[i][1]]+yi[me[i][3]]));
  z1=int (0.25*(zi[me[i][0]]+zi[me[i][2]]+zi[me[i][1]]+zi[me[i][3]]));
  if (me[i][4]==1)
  { th=0.0; xd=xmin*ncd[i][6]; th=th+th1[i];
    if ((ty==2)&&(i>=ne/4)&&(i<ne/4*3)) th=th+1.5707963;
    j=xd*cos(th)*xmax; k=xd*sin(th)*ymax; pDC->SelectObject(&aPen2);

```

```
        pDC->MoveTo(y1-j,z1+k);pDC->LineTo(y1+j,z1-k);}
    else if (me[i][4]==2) pDC->TextOut(y1-5,z1-5,"");
/*
th=0.0; xd=xmin*ncd[i][6]; th=th+th1[i];
if ((ty==2)&&(i>=ne/4)&&(i<ne/4*3)) th=th+1.5707963;
j=xd*cos(th)*xmax; k=xd*sin(th)*ymax;
if (me[i][4]==1) pDC->SelectObject(&aPen2);
else if (me[i][4]==0) pDC->SelectObject(&aPen1);
pDC->MoveTo(y1-j,z1+k);pDC->LineTo(y1+j,z1-k);
*/
};
break;
default :break;
};

if (m_pSelection == NULL)
{ POSITION pos = pDoc->GetStartPosition();
  m_pSelection = (CS1CntrItem*)pDoc->GetNextClientItem(pos);
}
if (m_pSelection != NULL)
  m_pSelection->Draw(pDC, CRect(10, 10, 210, 210));
}
```