Mechanical response of nets of graphical objects

Rakesh Kumar Bajaj

MECHANICAL RESPONSE OF NETS OF GRAPHICAL OBJECTS

by

RAKESH KUMAR BAJAJ

A THESIS

Presented to the Faculty of the Graduate School of the

UNIVERSITY OF MISSOURI–ROLLA

in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE IN COMPUTER SCIENCE

2002

Approved by

[illegible]
Michael G. Hilgers, Advisor

[illegible]
Bruce M. McMillin

[illegible]
Raymond Kluczny

ABSTRACT

This research produces networks of graphical objects that respond to human manipulation in a fashion that would be deemed physically reasonable. That is, a graphical network is given a *mechanics-like behavior* in the presence of the human interaction. In this approach, nonlinear models replace commonly used linear models to mimic physical reality better. Software solutions are provided via a class library useful for such interactive visual modeling.

ACKNOWLEDGMENT

I would like to thank Dr. Hilgers for all the help, encouragement and support that he gave me as my advisor. This work would not have been possible without his guidance and the support of the Computer Science department at the University of Missouri - Rolla.

I would also like to thank Dr. Bruce M. McMillin and Dr. Raymond Kluczny for serving on my thesis committee and for taking time to review and critique this work.

I am most grateful to my parents and sister who gave me support and encouragement in all aspects, helped me achieve my goals, and allowed me to pursue my graduate study at UMR.

TABLE OF CONTENTS

Page

ABSTRACT iii

ACKNOWLEDGMENT iv

LIST OF ILLUSTRATIONS vii

1 INTRODUCTION 1

1.1 MOTIVATION 1

1.2 OVERVIEW 2

2 PHYSICAL REALISM, COMPUTER GRAPHICS AND GRAPHICAL INTERFACES 5

2.1 PHYSICAL MODELS AND COMPUTER GRAPHICS 5

2.1.1 Physics based explosion modeling 5

2.1.2 Graphical modeling and animation of brittle fracture . . . 5

2.1.3 Computer graphics techniques for modeling cloth 6

2.2 PHYSICAL MODELS AND GRAPHICAL INTERFACES 6

2.2.1 Window layout manager 7

2.2.2 Graceful interaction with graphical constraints 7

3 MATHEMATICAL MODEL OF NETS OF GRAPHICAL OBJECTS . 8

4 ELEMENTAL SUBNET 12

5 SQUARE NETS OF ORDER 3 22

6 SQUARE NETS OF ORDER N 26

7 OBJECT-ORIENTED MODEL 28

7.1 CLASS DESIGN 28

7.2 CLASS DIAGRAM 30

7.3 SEQUENCE DIAGRAMS 31

7.3.1 Start-up operation 31

7.3.2 Mouse is clicked to select a vertex 31

7.3.3 Mouse is dragged after selecting a vertex 32

7.3.4 Mouse is released after selecting and moving a vertex . . . 33

8 IMPLEMENTATION OF THE OBJECT MODEL 36

8.1 ELEMENTAL SUBNETS 36

8.1.1 Vertex v0 fixed and vertex v3 moved 37

8.1.2 Vertices v0 and v2 fixed and vertex v1 moved 37

8.1.3 Vertices v0 and v1 fixed and vertex v2 moved 37

8.2 SQUARE NETS OF ORDER N 39

8.2.1 All the vertices in square net of order N-1 fixed and vertex $v_{(N^2-1)}$ moved 40

8.2.2 All the vertices in N-1 columns are fixed and a vertex in N^{th} column is moved 42

8.2.3 All the vertices in N-1 rows are fixed and a vertex in N^{th} row is moved 44

9 CONCLUSIONS 46

9.1 SUMMARY 46

9.2 FUTURE WORK 46

BIBLIOGRAPHY 47

VITA 49

LIST OF ILLUSTRATIONS

Figure Page

1.1 Product Design Web-Site Sponsored by Honda Motorcycles 2

1.2 Motorcycle Image Composed of Interconnected Movable Graphical Objects 3

1.3 Front Wheel has been moved 3

4.1 Elemental Subnet 12

4.2 Elemental Subnet with vertex v0 fixed 13

4.3 Elemental Subnet with vertex v0 fixed and vertex v3 moved 13

4.4 Elemental Subnet with vertex v0 fixed and vertex v3 moved in the 1^{st} quadrant 15

4.5 Elemental Subnet with vertex v0 fixed and vertex v3 moved in the 2^{nd} quadrant 16

4.6 Elemental Subnet with vertex v0 fixed and vertex v1 moved 19

4.7 Elemental Subnet with vertices v0 and v2 fixed and vertex v1 moved . . 20

5.1 Square net of order 3 22

5.2 Square net of order 3 having Elemental Subnet with vertices v0, v1, v3 and v4 fixed and vertex v8 moved 23

5.3 Square net of order 3 with all the vertices of first two columns fixed . . 25

5.4 Square net of order 3 with all the vertices of first two rows fixed 25

6.1 Square net of order N Locked 26

7.1 Class diagram showing relationship of class GUI with the other Java classes 29

7.2 Class diagram showing various classes and their relationships 30

7.3 Sequence diagram showing start-up operation 32

7.4 Sequence diagram showing mouse is clicked to select a vertex 32

7.5 Sequence diagram showing mouse is dragged after selecting a vertex . . 33

7.6 Sequence diagram showing mouse is released after selecting and moving a vertex 34

8.1 Elemental Subnet with vertex v0 fixed 36

8.2 Elemental subnet with vertex v0 fixed and vertex v3 moved 37

8.3 Elemental subnet with vertices v0 and v2 fixed and vertex v1 moved . . 38

8.4 Elemental subnet with vertices v0 and v1 fixed and vertex v2 moved . . 38

8.5 Square net of order N 39

8.6 Square net of order N before vertex $v_{(N^2-1)}$ is moved 40

8.7 Square net of order N after vertex $v_{(N^2-1)}$ is moved 41

8.8 Square net of order N with all the vertices in N -1 columns fixed 42

8.9 Square net of order N after a vertex in N^{th} column is moved 43

8.10 Square net of order N with all the vertices in N -1 rows fixed 44

8.11 Square net of order N after a vertex in N^{th} row is moved 45

1 INTRODUCTION

Constrained graphical user interfaces have been an active area of research for many years, with the primary focus being the human manipulation of window objects overlying each other on a restricted desktop [1] [2]. Mathematical models of this window manager problem take the form of constrained linear systems of equations open to analysis via linear programming methods [1]. Most activity on this problem concerns the timely solution needed to provide comfortable computer-human interaction.

This research explores human manipulation of graphical objects subject to mechanical constraints. Linear programming techniques are no longer applicable as the governing equations are nonlinear in nature. The goal of the added complexity is to produce networks of graphical objects that respond to human manipulation in a fashion that would be deemed physically reasonable. That is, the graphical network is given a *mechanics-like behavior* in the presence of human interaction.

To this end, a foundational theory is developed to provide a basis for analysis and implementation. Several theorems are proven, establishing the solvability of the interaction problem subject to a variety of constraints. Furthermore, software solutions are provided via a class library useful for such interactive visual modeling. Examples of the nature of the computer-human interactions are provided.

1.1 MOTIVATION

The motivation to consider interconnected networks of graphical objects subject to mechanical constraints arose from research into the feasibility of allowing people to participate in product design via the web. In a project [3] funded by Honda Motorcycles, a web-site was developed that supported the image of a motorcycle (see Figure 1.1). By using the mouse, a person viewing the site may move various parts of the motorcycle, thereby altering its design. For instance, by comparing Figure 1.2 and Figure 1.3, it is seen that the front wheel has been moved and the front forked dialated and rotated.

As the graphical objects move, the state of the motorcycle is recorded. When satisfied with a design, the user clicks the submit button and the new motorcycle

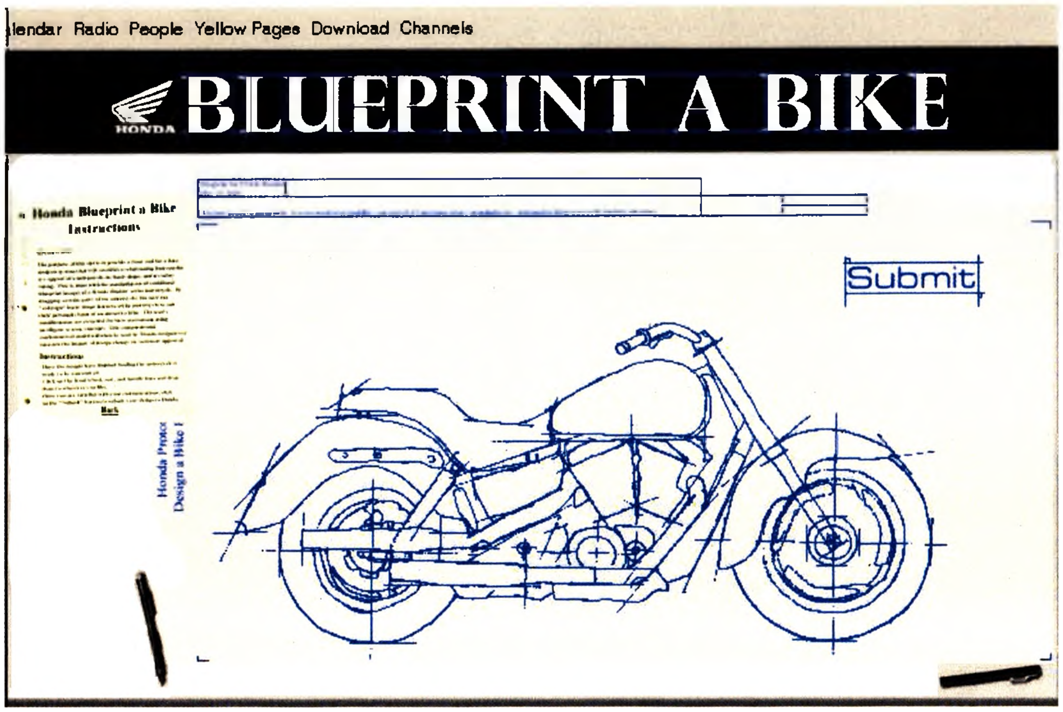

lendar Radio People Yellow Pages Download Channels
BLUEPRINT A BIKE
HONDA
Honda Blueprint a Bike Instructions
Submit
Honda Protot
Design a Bike

Figure 1.1 Product Design Web-Site Sponsored by Honda Motorcycles

design is sent to a cgi program that enters the design into the database. These numbers provide an indication of the design preferences of the viewing public.

In order for this to be a realistic measurement, it is of paramount importance that the virtual motorcycle responds to the mouse in a fashion that seemed reasonable. This meant that some parts had fixed path of translation, others could not change shape and some of the distances between the parts needed to be fixed. In short, the interconnected network of graphical objects needed to satisfy nonlinear constraints imposed by physical laws.

1.2 OVERVIEW

In the next section, the influence of the physical sciences on computer graphics and visualization is explored. While most prior works do not directly address physical constraints in graphical user interfaces, they do prove the point that the human eye is a very critical judge of computer-generated behavior.

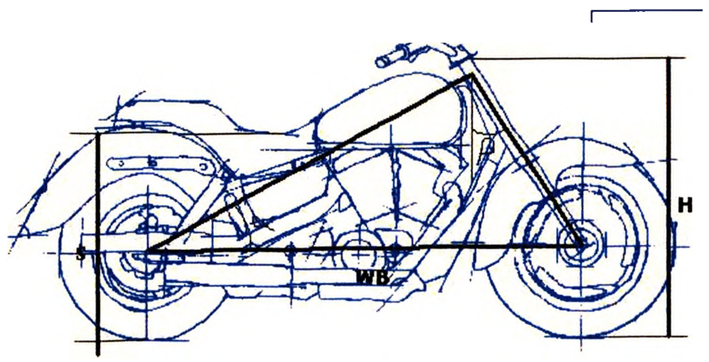


Figure 1.2 Motorcycle Image Composed of Interconnected Movable Graphical Objects

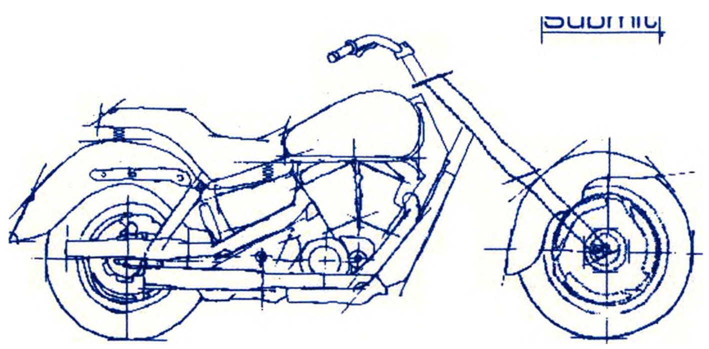


Figure 1.3 Front Wheel has been moved

In Section 3, the foundation of the theory is established. In the model proposed, the graphical network is viewed as a net of vertices connected to each other by means of rigid connections. Also, some vertices are fixed to restrict the motion of the net. The model allows a user to move a vertex and it is the goal of the analysis to predict the changes in the shape of the net. From a certain perspective, this is a classic mechanics problem.

Section 4 focuses on the so-called *elemental subnet* consisting of four vertices arranged as a square and its potential manipulations with a mouse. In this simplest case, several scenarios are seen, with the behavior ranging from interlocked nets incapable of motion to highly ambiguous configurations. Results from this analysis are extended to the case of a square network consisting of nine vertices. The summary of this analysis is given in Section 5. The case of a general $N \times N$ net is examined in Section 6.

With the theory established, a software solution is proposed. In this research, the object-oriented methodology is used in designing the class library. Section 7 contains Unified Modeling Language (UML) diagrams explaining the object-oriented model developed. While the design is not dependent on any particular language, a Java implementation is examined in Section 8. Several examples of computer-human interaction are studied. Conclusions and opportunities for future work are discussed in Section 9.

2 PHYSICAL REALISM, COMPUTER GRAPHICS AND GRAPHICAL INTERFACES

As the graphical capabilities of computer systems have matured, so has the sophistication of their human users. No longer satisfied with monochrome displays or cartoon-like animations, computer artists find that their work must imitate their physical surroundings. Hence, the world of computer graphics increasingly involves models based upon mathematical or physical laws. In this section, a survey is presented of various examples of the impact of physical models on computer graphics and graphical interfaces.

2.1 PHYSICAL MODELS AND COMPUTER GRAPHICS

The need to incorporate physical realism in computer graphics and animation has been driven by the movie industry. With Hollywood turning to the computer industry to supply special effects, computer scientists find themselves writing software to mimic everything from a single explosion to intergalactic war. While the survey in this section is far from comprehensive, it does provide examples from recent literature of the impact of mechanical models on graphics.

2.1.1 Physics based explosion modeling Producing real explosions is costly and potentially dangerous. Special effects technicians have turned to software to generate explosion sequences. Older software tools allowed artists to re-create the appearance of an explosion ignoring the physical processes, which looked artificial. Recent research [4] produces physically correct explosion incurring cost. The authors observe that substantial computational resources are spent in providing realism which pleases the human eye.

2.1.2 Graphical modeling and animation of brittle fracture Modeling explosions necessitates the need to simulate the consequences. Hence, a computer graphics artist must produce scenes with breaking windows, collapsing walls and debris flying in all directions. Again, physical laws must be used to provide the realism that the audience demand.

O' Brien and Hodgins [5] suggested the use of mechanical stress tensors, produced by finite element methods, to determine where cracks should initiate and in what directions they should propagate. By varying the shape of the objects, the material properties, and the initial conditions of the simulations, strikingly different effects were created. These range from a wall that shatters when it is hit by a wrecking ball to a bowl that breaks in to two when it is dropped on an edge.

2.1.3 Computer graphics techniques for modeling cloth In computer graphics or animation, appearance is generally more important than physical accuracy, so the emphasis has been more on visual realism than on physical accuracy [6], this distinction, though, has blurred as audience mature.

Geometrical models do not consider the physical properties of cloth. Rather, they focus on appearance, particularly on folds and creases, which they represent by geometrical equations [6]. Geometrical techniques require a considerable degree of user intervention. They can be regarded typically as little more than a form of an advanced drawing tool.

Weil [7] was probably first to apply geometrical techniques to cloth visualization in computer graphics. He represented hanging cloth as a grid of points and simulated its shape by fitting catenary curves between the hanging or constraint points. Catenary curves were appropriate because of inflexible fiber models which have origins within the mechanics community traceable to Tchebychev [8] in 1878. Hence, physics influences even the so-called geometrical techniques.

Agui et al. [9] presented a geometrical method for modeling a sleeve on a bending arm. They represented the cloth as a hollow cylinder consisting of a series of circular rings and observed folds form as a consequence of the differences in curvature between the inner and outer part of the bent sleeve. Again the importance of curvature in the model is explained by mechanics. Resistance to changes of curvature in elastic sheets is called bending stiffness and is foundational to fold modeling [10].

2.2 PHYSICAL MODELS AND GRAPHICAL INTERFACES

In the sub-section that follows, it is seen that graphical user interfaces have always included some elements of physical modeling. For instance the traditional window manager problem treats window components like pieces of paper overlying each other on a desktop [1]. The problem, then becomes finding what is observable. Besides layout managing, physics has appeared in other interactive applications and interfaces.

2.2.1 Window layout manager Badros et al. have a series of papers exploring the window layout manager problem [1] [2]. Their model reduces the desktop to a series of overlying rectangles with various sub portions visible, all constrained to be within a larger rectangle. Since a rectangle's boundaries are lines, determining visible regions forms a constrained linear problem [1].

Human manipulation of window objects creates the need to solve such problems repeatedly. To achieve interactive response times, Badros et al. [1] have suggested fast incremental algorithms that exploit prior computations. They show that the Cassowary algorithm [1] is one such efficient constraint solver for interactive user interface applications.

2.2.2 Graceful interaction with graphical constraints An early effort to incorporate non-linear graphical constraints into a graphical user interface was a project named GRACE [11]. GRACE, the Graphical Constraint Editor, lets users define constraints among graphical objects. Inextensibility was one such constraint in GRACE, as it is in this thesis. However, GRACE offers no theoretical foundation for the interaction of the network.

3 MATHEMATICAL MODEL OF NETS OF GRAPHICAL OBJECTS

This section is concerned with developing a mathematical model of nets of graphical objects that respond in a familiar mechanical way when interacting with a mouse. To this end, terms are defined that are needed to build and analyze such an interface.

3.1 Definition. An undirected graph G consists of two sets: $V(G)$ the set of vertices of G and $E(G)$ the set of edges of G, together with with a function $\gamma : E(G) \to \{(u, v)|u, v \in V(G)\}$.

3.2 Definition. The *picture* of a graph is a diagram consisting of P points corresponding to the vertices of G and the lines corresponding to edges.

Both the graph and its picture will be referred with the name G unless the context requires further distinction. For visualization, the picture requires a co-ordinate system. Imposing a cartesian system yields a vertex $u \in V(G)$ with the co-ordinates (X_u, Y_u) in the picture of G.

3.3 Definition. A graph N is said to be a *rectangular net* if its picture satisfies the following characteristics:

1. The vertices are arranged in a rectangular matrix consisting of n rows and m columns.

2. For each row of m vertices, there are $m - 1$ edges arranged horizontally to connect adjacent vertices.

3. For each column of n vertices, there are $n - 1$ edges arranged vertically to connect adjacent vertices.

A *square net of order* N is a rectangular net with N rows and N columns. A *net* is more general and relaxes the requirement of edges forming an orthogonal grid.

Though the concepts examined in this thesis could be applied to a wide variety of graphs, the graphs of particular interest are those whose picture looks like a *net.* The fundamental problem of concern in this thesis supposes that an interactive medium, such as a Java APPLET on a web page, displays a net and allows the user to click and drag a vertex on the net. The goal is for the transformed picture to have a *mechanics-like* response. To this end, reasonable physical properties must be assigned to the net.

3.4 Definition. A *graphical transformation* of the picture of a graph G is a mapping T taking vertex u with the initial co-ordinates (X_u, Y_u) to vertex $T(u)$ with location (x_u, y_u) for all vertices u in the graph.

The graphical transformation of fundamental interest in this thesis is associated with the so-called *click and drag* event.

3.5 Definition. A *click and drag* event selects a vertex $\hat{u}$ initially at $(X_{\hat{u}}, Y_{\hat{u}})$ in the picture and attempts to assign it to the transformed vertex $T(\hat{u})$ with location $(x^*_{\hat{u}}, y^*_{\hat{u}})$.

The problem at hand is to uniquely determine a graphical transformation T which satisfies the specified *click and drag* event and transforms the net in a manner that the eye would deem as mechanically reasonable. This will require mechanics-like properties on the edges themselves.

3.6 Definition. An edge e in a graph G connecting vertices $\{u, v\}$ is called a *rigid-link* if the corresponding line segment in the picture is *inextensible.* That is, if the picture of the graph G undergoes a graphical transformation T, then

$$\text{length}\, e = \text{length}\, T(e)$$

or

$$\sqrt{(X_u - X_v)^2 + (Y_u - Y_v)^2} = \sqrt{(x_u - x_v)^2 + (y_u - y_v)^2}$$

In order to develop a reasonable mechanics for a net of graphical objects, attention must be given to the possibility of resistance to *shearing.* That is, suppose an edge e connects $\{u, v\}$ and edge f connects $\{v, w\}$. Denote $\theta(e, f)$ as the angle between e and f in the picture of G. A graphical transformation on T gives $T(\theta(e, f)) = \theta(T(e), T(f))$. Changes in the angle θ are generically described with the term *shearing.* A graph is said to *resist shearing* if there is a penalty associated with the angle change. In this thesis, there is no resistance to shearing so that the angle between edges can freely change.

3.7 Definition. A *simple mechanical net* is a net with all the edges as rigid-links and offers no resistance to shearing. A graphical transformation is said to have *simple mechanics-like* behavior if it transforms the picture of the graph under the constraint of rigid-links with no shearing resistance.

Almost all mathematical models of a mechanical system require a description of the initial state of the system, a specification of what is happening to the system along its boundary, and the inclusion of global body forces. In this research, the fundamental boundary or body interaction is to *fix* a vertex.

3.8 Definition. If the picture of the graph G undergoes a graphical transformation T taking vertex u with the initial co-ordinates (X_u, Y_u) to vertex $T(u)$ with location (x_u, y_u), then a vertex u in a graph G is said to be *fixed* if

$$X_u = x_u \quad \text{and} \quad Y_u = y_u.$$

It should stand to reason that at least one vertex of a simple net must be fixed in order to have any possibility of mechanically reasonable behavior under the influence of a click and drag event. With only one vertex location known, any shearing transformation of the angles in a subnet would satisfy the click and drag event, yet would yield highly non-unique configurations with most lacking any mechanically realistic behavior. Hence, it is assumed that at least one vertex is fixed. For simplicity, also assume that all rigid-links have length l.

There are now two constraints on a graphical transformation of simple mechanical nets: it must not change the length of an edge (inextensibility) and it cannot move fixed vertices.

3.9 Definition. Suppose N is a simple mechanical net. Let F be a non empty subset of vertices of N that are fixed. A graphical transformation of N is *mechanically admissible* if it satisfies the rigid-link and fixed vertices constraint.

Note that the set of mechanically admissible graphical transformations is not empty. It always contains the identity mapping. Of special interest, however, is the situation when the identity mapping is the *only* mechanically admissible graphical transformation.

3.10 Definition. A simple mechanical net with fixed vertices F is said to be *locked* if the identity mapping is the only mechanically admissible graphical transformation.

In many graphical user interfaces, the mouse may attempt to drag an object to a location $(x^*_{\hat{u}}, y^*_{\hat{u}})$ not permitted by the mechanical constraints. Hence, a properly formulated theory should address several issues. The first is mechanical admissibility.

3.11 Definition. A click and drag event of a vertex $\hat{u}$ is *mechanically admissible* if there exists a graphical transformation T such that $T(\hat{u})$ has the property

$$x_{\hat{u}} = x^*_{\hat{u}}$$

$$y_{\hat{u}} = y^*_{\hat{u}}.$$

The existence of such a graphical transformation T means that $(x^*_{\hat{u}}, y^*_{\hat{u}})$ can be connected to the remaining vertices while maintaining the rigid-link constraint. Conversely, mechanical inadmissibility implies that rigid-link constraint forbids the desired location from being visualized. With the notion of admissible solutions defined, a second concern is finding *a priori* conditions-determining whether a click and drag event is mechanically admissible. This is a challenging problem, some results will be provided in Sections 4, 5 and 6.

Finally a mechanics problem has the liberty to conclude that there is no physically admissible solution due to the constraints in the problem. A computer-human interaction should result in some reasonable picture of the net displayed. Hence the click and drag event location $(x^*_{\hat{u}}, y^*_{\hat{u}})$ must be projected into the set of mechanically admissible events so that the corresponding graphical transformation T can be used for the display. This is the final major issue of concern in the following sections.

4 ELEMENTAL SUBNET

In this section, the focus is on the simplest possible net consisting of only four vertices. However, its behavior provides a foundation for all that happens in the larger nets.

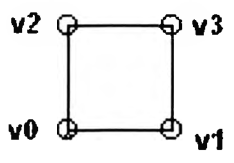


Figure 4.1 Elemental Subnet

4.1 Definition. An *elemental subnet* is a net consisting of 2 rows and 2 columns of vertices. All the edges are rigid-links and there is no resistance to shearing (see Figure 4.1).

The properties needed for the picture of an elemental subnet to behave in a mechanical fashion when interacting with a mouse are considered herein. If the coordinates of all the vertices remain the same after a graphical transformation, there is no change in the shape and position of the net. *A priori* conditions specifying whether an elemental subnet is locked can be readily offered.

4.1 Lemma. *For an elemental subnet, if the diagonally opposite vertices are fixed, then the subnet is locked.*

Proof. Let G be an elemental subnet as denoted in Figure 4.1. Without loss of generality suppose vertices $v0$ and $v3$ are *fixed* (see Definition 3.8). It is also known that all the edges in the subnet are rigid-links (see Definition 3.6). Due to the edges being rigid-links, the identity mapping is the only mechanically admissible graphical transformation (see Definition 3.10). This graphical transformation does not bring any change either to the shape or to the position of the subnet.

4.2 Lemma. *For an elemental subnet, if every row and every column contains a fixed vertex, then the subnet is locked.*

Proof. Note: This has been proved for an elemental subnet (see Lemma 4.1).

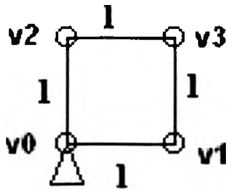


Figure 4.2 Elemental Subnet with vertex v0 fixed

A priori conditions determining whether a click and drag event is mechanically admissible are relatively straightforward in the case of an elemental subnet. In Figure 4.2, suppose vertex $v0$ is fixed. Two cases exist. If the clicked vertex $u \in \{v1, v2\}$, then mechanically admissible click and drag events must satisfy

$$(x_u^* - x_0)^2 + (y_u^* - y_0)^2 = l^2. \tag{4.1}$$

Since the connecting links are inextensible, mechanical admissibility requires

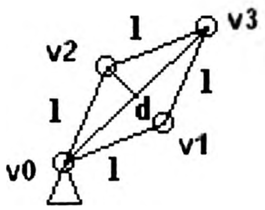


Figure 4.3 Elemental Subnet with vertex v0 fixed and vertex v3 moved

$$\sqrt{(x_3^* - x_0)^2 + (y_3^* - y_0)^2} \leq 2l, \tag{4.2}$$

if the selected vertex is $v3$ (see Figure 4.3). This is because the length of any side of a triangle is less than the sum of the other two. That these conditions yield mechanically admissible click and drag events is the subject of Lemma 4.3.

If the click and drag event of vertex u is not mechanically admissible, then (x_u^*, y_u^*) must be projected into the set of admissible solutions, for purpose of display. If $u \in \{v1, v2\}$, then Equation 4.1 leaves with little flexibility, in that (x_u^*, y_u^*) must be projected onto the circle of radius l along the ray with $v0$ as the center.

If $u = v3$, then the condition in Equation 4.2 is more permissive. Projecting (x_3^*, y_3^*) to any point inside the circle of radius $2l$ along the ray with $v0$ as the center would work. However, projecting (x_3^*, y_3^*) to the boundary of the circle yields the most reasonable behavior from a visual perspective.

4.3 Lemma. *Let G be an elemental subnet as denoted in Figure 4.2 with vertex $v0$ fixed. Suppose vertex $v3$ undergoes a click and drag event from (x_3, y_3) to (x_3^*, y_3^*) as denoted in Figure 4.3. If the inequality in Equation 4.2 is satisfied, the click and drag event is mechanically admissible. If the inequality is not satisfied, the click and drag event is mechanically inadmissible.*

Proof. Certain aspects of the geometry of the elemental subnet are labeled as follows:

1. Length of line joining the fixed vertex $v0$ and the moved vertex $v3$ is d (see Figure 4.3).

2. Length of the edge(rigid-link) is l (see Figure 4.3).

3. Co-ordinates of vertex $v0$ are (x_0, y_0).

4. Co-ordinates of vertices $v1$ and $v2$ after the transformation are (x_1^*, y_1^*) and (x_2^*, y_2^*).

5. The mid-point of fixed vertex $v0$ and moved vertex $v3$ is (x_m, y_m).

6. Angle made by the line passing through fixed vertex $v0$ and moved vertex $v3$ with the X-axis is θ (see Figure 4.4).

7. Angle made by the line passing through fixed vertex $v0$ and moved vertex $v3$ with the edge joining fixed vertex $v0$ and vertex $v1$ is α (see Figure 4.4).

8. Angle made by the edge joining fixed vertex $v0$ and vertex $v1$ with the X-axis is β (see Figure 4.4).

Case 1: Without loss of generality, $d < 2l$ and the elemental subnet is in the 1st quadrant (see Figure 4.4).

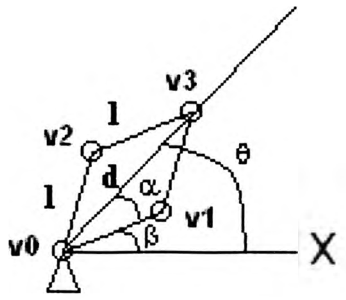


Figure 4.4 Elemental Subnet with vertex v0 fixed and vertex v3 moved in the 1st quadrant

Here x_3^* and y_3^* are displacements on the respective axes provided the distance between fixed vertex $v0$ and moved vertex $v3$ is less than $2l$, i.e.

$$\sqrt{(x_3^* - x_0)^2 + (y_3^* - y_0)^2} < 2l$$

Trigonometry yields

$$\theta = \arctan\left(\frac{y_3^* - y_0}{x_3^* - x_0}\right),$$

$$\alpha = \arccos(d/2l)$$

since $d < 2l$, and

$$\beta = \theta - \alpha.$$

Co-ordinates of vertex $v1$ are given by

$$x_1^* = x_0 + l\cos\beta$$

$$y_1^* = y_0 + l\sin\beta.$$

Co-ordinates of the mid-point of fixed vertex $v0$ and moved vertex $v3$ are

$$x_m = (x_3^* + x_0)/2$$

$$y_m = (y_3^* + y_0)/2.$$

As the subnet is in the form of a rhombus (see Figure 4.4), the edges joining the opposite vertices have the same mid-point. Therefore the co-ordinates of the vertex $v2$ are

$$x_2^* = 2 * x_m - x_1^*$$

$$y_2^* = 2 * y_m - y_1^*.$$

Case 2: Without loss of generality, $d < 2l$ and the elemental subnet is in the 2^{nd} quadrant (see Figure 4.5).

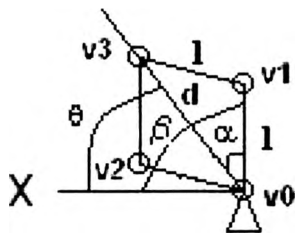


Figure 4.5 Elemental Subnet with vertex v0 fixed and vertex v3 moved in the 2^{nd} quadrant

As seen in case 1:

$$\theta = \arctan(\frac{y_3^* - y_0}{x_0 - x_3^*}),$$

$$\alpha = \arccos(d/2l)$$

since $d < 2l$. But in this case angle β is sum of the angles θ and α (see Figure 4.5).

$$\beta = \theta + \alpha.$$

Co-ordinates of vertex $v1$ are given by

$$x_1^* = x_0 - l \cos \beta$$

$$y_1^* = y_0 + l \sin \beta.$$

Co-ordinates of the mid-point of fixed vertex $v0$ and moved vertex $v3$ are

$$x_m = (x_3^* + x_0)/2$$

$$y_m = (y_3^* + y_0)/2.$$

As the subnet is in the form of a rhombus (see Figure 4.5), the edges joining the opposite vertices have the same mid-point. Therefore the co-ordinates of the vertex $v2$ are

$$x_2^* = 2 * x_m - x_1^*$$

$$y_2^* = 2 * y_m - y_1^*.$$

Case 3: Without loss of generality, $d < 2l$ and the elemental subnet is in the 3^{rd} quadrant.

This case is same as case 1 (rotation by 180 deg).

Case 4: Without loss of generality, $d < 2l$ and the elemental subnet is in the 4^{th} quadrant.

This case is same as case 2 (rotation by 180 deg).

Case 5: Without loss of generality, $d = 2l$.

That is, $d/2l = 1$ means angle $\alpha = 0$ which is possible only if the vertices $v1$ and $v2$ overlap. Hence, the subnet collapses to a line segment.

Case 6: Without loss of generality, $d > 2l$

Equation 4.2 shows that the edges have stretched. This is not admissible.

Crucial to the the success of the previous theorem is the fact that the vertex selected for the click and drag event is diagonally opposite to the fixed vertex. If either vertex $v1$ or $v2$ is chosen, the result is less satisfying.

4.4 Lemma. *Let G be an elemental subnet with vertices $\{v0, v1, v2, v3\}$ starting in the lower left-hand corner (see Figure 4.2). Suppose vertex $v0$ is fixed and vertex $v1$ undergoes a click and drag event from (x_1, y_1) in the picture and to the transformed co-ordinates (x_1^*, y_1^*). There are infinitely many mechanics-like graphical transformations T determined on the basis of (x_1^*, y_1^*).*

Proof. Certain aspects of the geometry of the elemental subnet are labeled as follows:

1. Length of line joining fixed vertex $v0$ and moved vertex $v3$ is d (see Figure 4.3).
2. Length of the edge(rigid-link) is l (see Figure 4.3).
3. Transformed co-ordinates of vertex $v3$ are (x_3^*, y_3^*).
4. Angle made by the line passing through fixed vertex $v0$ and vertex $v3$ with the X-axis is θ (see Figure 4.6).
5. Angle made by the line passing through fixed vertex $v0$ and vertex $v3$ with the edge joining fixed vertex $v0$ and moved vertex $v1$ is α (see Figure 4.6).
6. Angle made by the edge joining fixed vertex $v0$ and moved vertex $v1$ with the X-axis is β (see Figure 4.6).

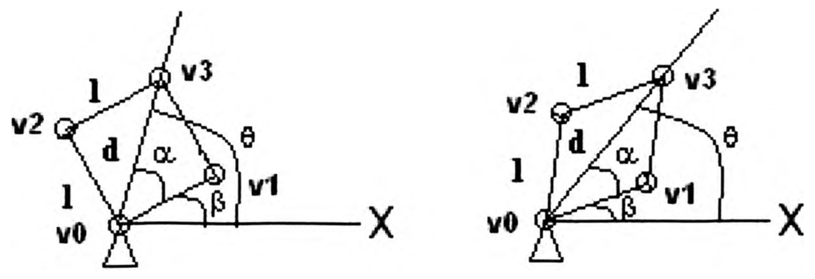
Without loss of generality, trigonometry yields (see Figure 4.6)

$$\beta = \arctan(\frac{y_1^* - y_0}{x_1^* - x_0}),$$

$$\alpha = \arccos(d/2l)$$

since $d < 2l$, and

$$\theta = \beta + \alpha.$$



(a) Scenario 1 (b) Scenario 2

Figure 4.6 Elemental Subnet with vertex v0 fixed and vertex v1 moved

Co-ordinates of vertex $v3$ are given by

$$x_3^* = x_0 + l\cos\theta = x_0 + l\cos(\beta + \alpha)$$

$$y_3^* = y_0 + l\sin\theta = y_0 + l\sin(\beta + \alpha).$$

To get the co-ordinates of vertex $v3$, the value of d is required. Different values of d or α give different co-ordinates for vertex $v3$. Hence, the configuration is ambiguous. $\square$

The non-uniqueness witnessed in Lemma 4.4 is not unusual in mechanical models. The first concern that must be addressed is whether the boundary conditions and initial state are sufficiently strong to yield unique behavior. If non-uniqueness is demonstrated, then the usual approach is to add a *selection criterion* to the model (this is a standard situation in hyperbolic conservation laws) [12]. This is a future direction for this research. Lemma 4.4 shows that selecting a vertex that is rigidly linked to a fixed vertex for a click and drag event causes the captured vertex to rotate about the fixed vertex. Meanwhile the remaining vertices can freely shear about the rotating base. One form of remedy is to fix one of the remaining vertices.

4.5 Lemma. *Let G be an elemental subnet with vertices $\{v0, v1, v2, v3\}$ starting in the lower left-hand corner (see Figure 4.1). Suppose vertices $v0$ and $v2$ are fixed and vertex $v1$ undergoes a click and drag event from (x_1, y_1) in the picture and to the* *transformed co-ordinates* (x_1^*, y_1^*). *There is a unique mechanics-like graphical transformation* T *determined by* (x_1^*, y_1^*) *(see Figure 4.7).*

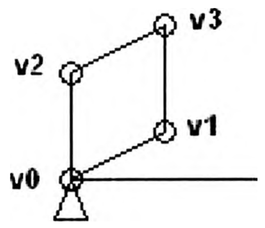


Figure 4.7 Elemental Subnet with vertices v0 and v2 fixed and vertex v1 moved

Proof. Certain aspects of the geometry of the elemental subnet are labeled as follows:

1. The mid-point of fixed vertex $v2$ and moved vertex $v1$ is (x_m, y_m).
2. Co-ordinates of vertices $v0$ and $v2$ are (x_0, y_0) and (x_2, y_2).
3. Transformed co-ordinates of vertex $v3$ are (x_3^*, y_3^*).

Without loss of generality, co-ordinates of the mid-point of fixed vertex $v2$ and moved vertex $v1$ are

$$x_m = (x_1^* + x_2)/2$$

$$y_m = (y_1^* + y_2)/2.$$

As the subnet is in the form of a rhombus (see Figure 4.7), the edges joining the opposite vertices have the same mid-point. Therefore the co-ordinates of the vertex $v3$ are

$$x_3^* = 2 * x_m - x_0$$

$$y_3^* = 2 * y_m - y_0.$$

This determines the graphical transformation.

Even in the elemental subnet, there is a fundamental difficulty in allowing the mouse to interact with a picture of a graph and hope for it to respond in mechanics-like fashion. There is a potential for non-uniqueness depending on vertex selected by the user. However, these lemmas provide conditions that yield more acceptable behavior. The next step is to consider the situation of nets composed of several elementary subnets.

5 SQUARE NETS OF ORDER 3

This section is concerned with the square net of order 3, composed of 4 elementary subnets. The question at hand is determining the number of vertices that must be fixed to yield a mechanics-like response when the user selects a vertex and drags it to a new location.

5.1 Lemma. *For a simple mechanical net of order N, if a vertex is between two fixed vertices, then it is immovable.*

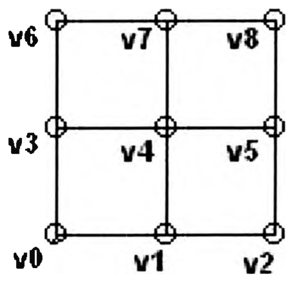


Figure 5.1 Square net of order 3

Proof. Without loss of generality, suppose vertices $v0$ and $v2$ are *fixed* (see Figure 5.1). It is known that all the edges in are rigid-links. Due to the edges being rigid-links vertex $v1$ can only move along the circumference of two circles, one with center $v0$ and other with center $v2$. These two circles have single point in common, vertex $v1$, this leaves vertex $v1$ immovable. This extends easily to arbitrary situation.

5.2 Theorem. *For a square net of order 3, if every row and every column contains a fixed vertex, then the square net is locked.*

Proof. Let G be a *square net of order 3* as denoted in Figure 5.1. Without loss of generality, suppose vertices $v0, v4$ and $v8$ are fixed. Apply Lemma 4.1 to the elemental subnet with vertices $v0, v1, v3$ and $v4$ having vertices $v0$ and $v4$ fixed. Vertices $v1$ and $v3$ cannot have a graphical transformation that can bring any change to either the shape or the position of the elemental subnet. Similarly, apply Lemma 4.1 to the elemental subnet with vertices $v4, v5, v7$ and $v8$, having vertices $v4$ and $v8$ fixed. Vertices $v5$ and $v7$ cannot move. Finally, as all the other vertices in the square net are fixed, vertices $v2$ and $v6$ cannot move. This method can be readily applied to any other combination of fixed vertices satisfying the hypothesis.

5.3 Theorem. *For a square net of order 3, if all the vertices of an elemental subnet are fixed and a vertex diagonally opposite to the fixed elemental subnet undergoes click and drag event, then there is a unique mechanics-like graphical transformation.*

Proof. Let G be a square net of order 3 as shown in Figure 5.1. Without the loss of generality, fix all the vertices of the elemental subnet $v0, v1, v3, v4$, then click and drag diagonally opposite vertex $v8$ (see Figure 5.2).

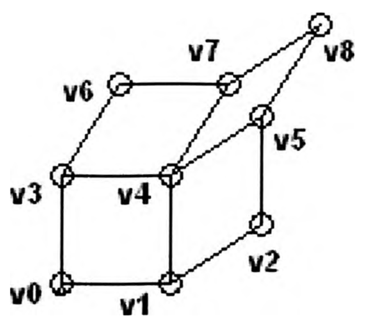


Figure 5.2 Square net of order 3 having Elemental Subnet with vertices v0, v1, v3 and v4 fixed and vertex v8 moved

Apply Lemma 4.3 to the elemental subnet with vertices $v4, v5, v7$ and $v8$ to get the transformed co-ordinates of vertices $v5$ and $v7$. Then apply Lemma 4.5 to the elemental subnet with vertices $v1, v2, v4$ and $v5$ to get the transformed co-ordinates of vertex $v2$. Similarly, applying Lemma 4.5 to the elemental subnet with vertices $v3, v4, v6$ and $v7$ results in the transformed co-ordinates of vertex $v6$.

Crucial to the the success of the previous theorem is the fact that the vertex selected for the click and drag event is diagonally opposite to the fixed elemental subnet. Consider a case where vertex selected for the click and drag event is not diagonally opposite to the fixed elemental subnet.

5.4 Theorem. *For a square net of order 3, if all the vertices of an elemental subnet are fixed and a vertex not diagonally opposite to the fixed elemental subnet undergoes click and drag event, then there are infinitely many mechanics-like graphical transformations.*

Proof. Let G be a square net of order 3 as shown in Figure 5.1. Without loss of generality fix all the vertices of the elemental subnet $v0, v1, v3, v4$, then click and drag vertex $v7$. Apply Lemma 4.5 to the elemental subnet with vertices $v3, v4, v6$ and $v7$ to get the co-ordinates of vertex $v6$. Now move to the elemental subnet with vertices $v4, v5, v7$ and $v8$, it has a fixed vertex $v4$ and a moved vertex $v7$, there are infinitely many transformations possible that can be determined on the basis of the transformed co-ordinates of vertex $v7$ (this case has been already discussed see Lemma 4.4).

5.5 Theorem. *For a square net of order 3, if all the vertices in two consecutive rows or two consecutive columns are fixed and a free vertex undergoes click and drag event, then there is a unique mechanics-like graphical transformation.*

Proof. Let G be a square net of order 3 as denoted in Figure 5.1. Here there are two different cases that need to be proved;

Case 1: Without loss of generality, all the vertices of the first two columns of the square net of order 3 are fixed (see Figure 5.3).

This leaves three vertices that can be moved around, namely $v2, v5$ and $v8$. If vertex $v2$ is moved, apply Lemma 4.5 to the elemental subnet with vertices $v1, v2, v4$ and $v5$, to get the transformed co-ordinates of vertex $v5$. Similarly, applying Lemma 4.5 to the elemental subnet with vertices $v4, v5, v7$ and $v8$, results in the transformed co-ordinates of vertex $v8$.

Case 2: Without loss of generality, all the vertices of the first two rows of the square net of order 3 are fixed (see Figure 5.4).

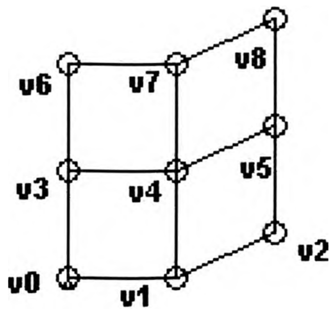


Figure 5.3 Square net of order 3 with all the vertices of first two columns fixed

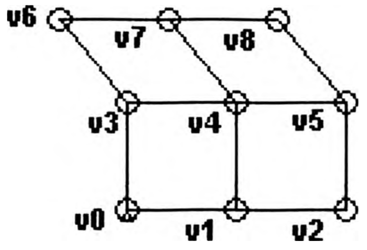


Figure 5.4 Square net of order 3 with all the vertices of first two rows fixed

This leaves three vertices that can be moved around, namely $v6, v7$ and $v8$. If vertex $v6$ is moved, apply Lemma 4.5 to the elemental subnet with vertices $v3, v4, v6$ and $v7$ to get the transformed co-ordinates of vertex $v7$. Similarly, applying Lemma 4.5 to the elemental subnet with vertices $v4, v5, v7$ and $v8$, results in the transformed co-ordinates of vertex $v8$.

6 SQUARE NETS OF ORDER N

This section addresses the question on the nets of arbitrary order. Analysis of this situation is difficult and some questions remain open. However, *a priori* conditions yielding locked nets are established. These are discussed in this section.

6.1 Theorem. *Let $\mathcal{R}$ be a square net of order N, if every row and every column contains a fixed vertex, then the square net is locked.*

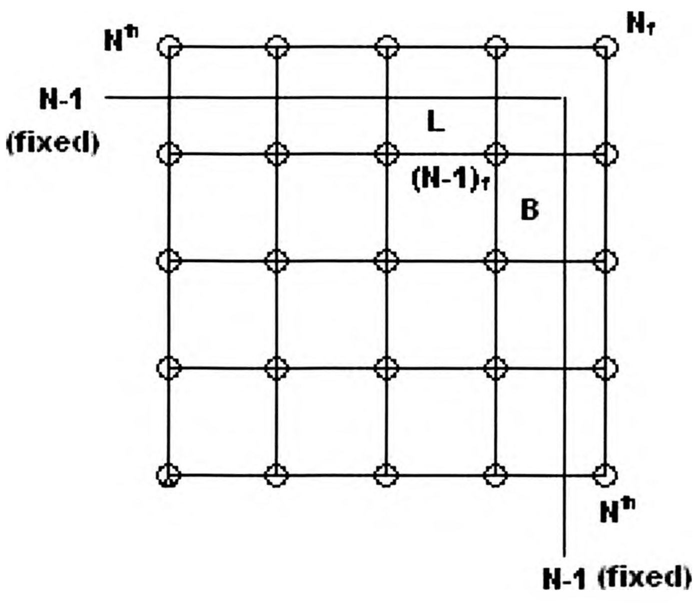


Figure 6.1 Square net of order N Locked

Proof. This has been proved for the square nets of order 2 and 3 (see Lemma 4.2 and Theorem 5.2). Assume that this is true for a square net of order $N-1$; it will be proved true for a square net of order N. Addition of a row and a column of vertices to a square net of order $N-1$ gives a square net of order N (see Figure 6.1).

Fix vertex common to the N^{th} column and the N^{th} row and call it vertex N_f and also fix vertex common to the $(N-1)^{th}$ column and the $(N-1)^{th}$ row and call it vertex $(N-1)_f$ (see Figure 6.1). The elemental subnet having diagonally opposite vertices N_f and $(N-1)_f$ is now *locked.* Hence, its other two vertices cannot move (see Lemma 4.2).

The elemental subnet (denoted by letter B in Figure 6.1) exactly below the fixed elemental subnet is *locked* as three of its vertices are fixed. Similarly, the elemental subnet (denoted by letter L in Figure 6.1) left to the fixed elemental subnet is also *locked* as three of its vertices are fixed. This can be extended to all the remaining elemental subnets of the square net of order N.

7 OBJECT-ORIENTED MODEL

The major motivation for an object-oriented model is software reuse [13]. Furthermore, object-oriented modeling is well-suited to graphics problems for the following reasons:

1. The need to organize and describe basic graphic objects and representing relationships such as part-whole by inter-object references [14].

2. Communication among the parts of a graphic representation is needed to perform graphic operations. Message passing among the objects representing the graphic components is a very direct way to carry out graphical operations [14].

Object-oriented modeling not only reduces development time but also cost of maintenance [13]. Jacobson [15] says object-oriented programming has changed the nature of software design from being an art or a craftsmanship to being an industrial process.

7.1 CLASS DESIGN

The Unified Modeling Language (UML) [16] was used extensively in providing an object-oriented model of the graphical nets. Classes were designed in such a way that they could be implemented in any object-oriented language. Various classes and their details are discussed here. Class names have been capitalized.

Class GUI is the user interface class (see Figure 7.1). Events related to the user interface like *mousePressed, mouseDragged, mouseReleased* and *paint* are handled here. The purpose of this class is to pass on the user interactions to the other classes, it has no role to play in the implementation of the theorems. The user interface and the theorem logic have been kept independent of each other. The GUI class extends the APPLET class and overrides *paint* method. It also uses MOUSEADAPTER and MOUSEMOTIONADAPTER classes to trap the mouse movements. It has a dependency relationship with the GRAPHICS class as it uses *drawOval and drawLine* functions in the *paint* method.

Class POINT is the basic building block of the net classes. It encapsulates the functionality of a *vertex.* It has attributes for the co-ordinates, type (fixed or

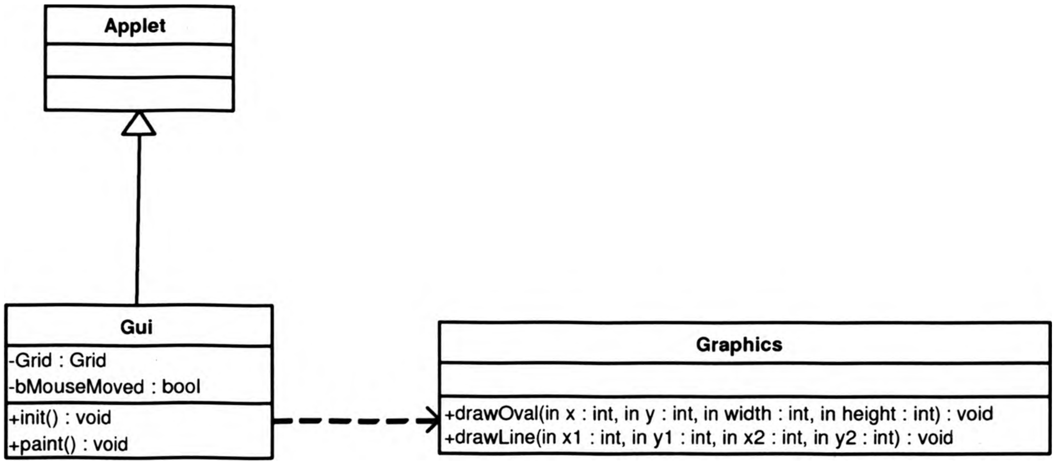

Applet
Gui
-Grid : Grid
-bMouseMoved : bool
+init() : void
+paint() : void
Graphics
+drawOval(in x : int, in y : int, in width : int, in height : int) : void
+drawLine(in x1 : int, in y1 : int, in x2 : int, in y2 : int) : void

Figure 7.1 Class diagram showing relationship of class GUI with the other Java classes

movable) and status (moved or not moved). Apart from the accessor functions, it has specific functions to set its location, check whether it has been selected, get the distance between the two vertices, and find the quadrant.

Class GRID is the most important class as it models the interconnections of the points. It has the capability needed to build the net using the class POINT and transform the net depending on the user interaction. It has attributes like an array of points and vertex selected. It has specific functions to get length of the link, build the net, find the selected vertex, and transform the net.

Class GLOBAL holds the parameters that specify the initial state of the net. All the state parameters have been kept here to avoid modifying other classes when the configuration of the net is to be changed. Any changes made to the parameter values in the GLOBAL class are automatically reflected in all other classes. It models the state parameters like the order of the net, the size (the total number of vertices in the net), the co-ordinates of the left most vertex from the bottom and the distance between the two vertices (length of *rigid-link*).

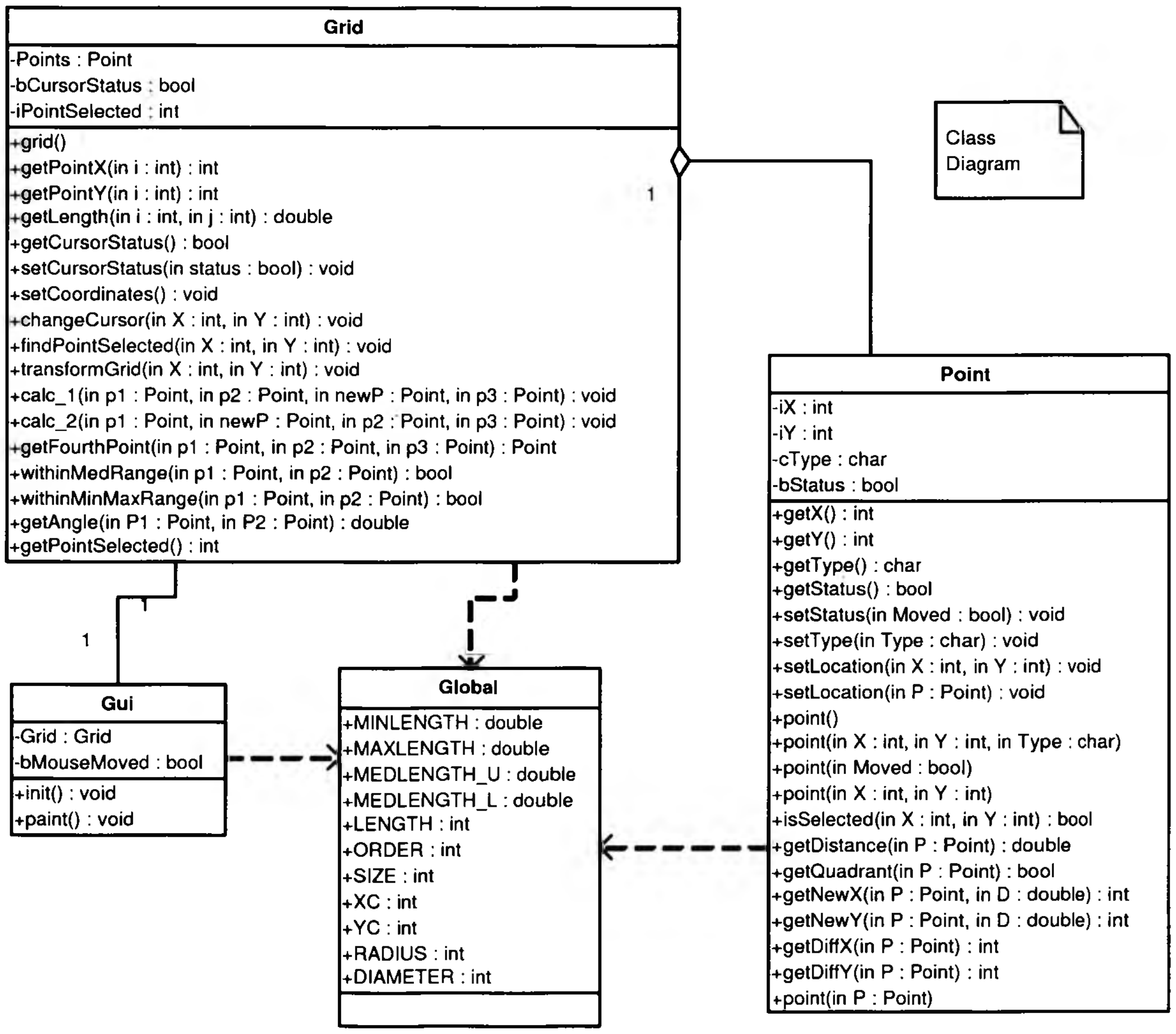

Grid
-Points : Point
-bCursorStatus : bool
-iPointSelected : int
+grid()
+getPointX(in i : int) : int
+getPointY(in i : int) : int
+getLength(in i : int, in j : int) : double
+getCursorStatus() : bool
+setCursorStatus(in status : bool) : void
+setCoordinates() : void
+changeCursor(in X : int, in Y : int) : void
+findPointSelected(in X : int, in Y : int) : void
+transformGrid(in X : int, in Y : int) : void
+calc_1(in p1 : Point, in p2 : Point, in newP : Point, in p3 : Point) : void
+calc_2(in p1 : Point, in newP : Point, in p2 : Point, in p3 : Point) : void
+getFourthPoint(in p1 : Point, in p2 : Point, in p3 : Point) : Point
+withinMedRange(in p1 : Point, in p2 : Point) : bool
+withinMinMaxRange(in p1 : Point, in p2 : Point) : bool
+getAngle(in P1 : Point, in P2 : Point) : double
+getPointSelected() : int
1
1
1
Class Diagram
Point
-iX : int
-iY : int
-cType : char
-bStatus : bool
+getX() : int
+getY() : int
+getType() : char
+getStatus() : bool
+setStatus(in Moved : bool) : void
+setType(in Type : char) : void
+setLocation(in X : int, in Y : int) : void
+setLocation(in P : Point) : void
+point()
+point(in X : int, in Y : int, in Type : char)
+point(in Moved : bool)
+point(in X : int, in Y : int)
+isSelected(in X : int, in Y : int) : bool
+getDistance(in P : Point) : double
+getQuadrant(in P : Point) : bool
+getNewX(in P : Point, in D : double) : int
+getNewY(in P : Point, in D : double) : int
+getDiffX(in P : Point) : int
+getDiffY(in P : Point) : int
+point(in P : Point)
Gui
-Grid : Grid
-bMouseMoved : bool
+init() : void
+paint() : void
Global
+MINLENGTH : double
+MAXLENGTH : double
+MEDLENGTH_U : double
+MEDLENGTH_L : double
+LENGTH : int
+ORDER : int
+SIZE : int
+XC : int
+YC : int
+RADIUS : int
+DIAMETER : int

Figure 7.2 Class diagram showing various classes and their relationships

7.2 CLASS DIAGRAM

Class diagrams [16] are the most common diagrams found in modeling object-oriented systems. A class diagram shows a set of classes, interfaces, collaborations and their relationships. Class diagrams are used to model the *static* design view of a system.

Various classes and their relationships are shown in Figure 7.2. Class GLOBAL has a relationship of type *dependency* with all the other classes, since all the other classes are dependent on it and it is independent of them. Class GRID and class POINT are related as *aggregation*, since a net consists of more than one (aggregation) vertex. Multiplicity of the relationship is one to many, i.e one GRID object is composed of many POINT objects. Class GUI and class GRID are *associated* as the navigation is from class GUI to class GRID and vice versa. Also multiplicity of the relationship is one to one, i.e one GRID object is related to one GUI object.

7.3 SEQUENCE DIAGRAMS

An interaction diagram [16] shows an interaction, consists of a set of objects and their communication, models the messages that may be dispatched among the objects. A sequence diagram [16] is an interaction diagram that emphasizes the time ordering of messages. The sequence diagrams are used to model the *dynamic* aspects of a system.

The following scenarios were selected as they form a complete sequence, starting from clicking on a vertex to selecting it, then dragging it around, and finally, releasing the mouse to transform the net. The GLOBAL object has not been shown in the sequence diagrams as there is no message passing after the construction of the net. The communication between the GRID object and the POINT object is iterated, in the sequence diagram a single POINT object will be shown. Object names have been capitalized and their operations are italicized.

7.3.1 Start-up operation The sequence that takes place when the APPLET starts is shown in Figure 7.3. GUI starts the sequence by calling *init.* GUI passes a message to GRID to call its constructor. GRID calls *setCoordinates* to build the net. GRID passes a message to POINT to call its constructor. GRID passes a message to POINT to call *setLocation* to set its location. GUI calls the *paint* to print the net on the screen. GUI passes messages to GRID to call *getPointX* and *getPointY* for getting the x and the y co-ordinates of a point. GRID passes messages to POINT to call *getX* and *getY* to get the x and the y co-ordinates. The sequence ends with the display of the net on the screen.

7.3.2 Mouse is clicked to select a vertex The sequence that takes place when the user clicks on a vertex to select it is shown in Figure 7.4. GUI starts the sequence by passing a message to GRID to call *findPointSelected* to find the point selected.

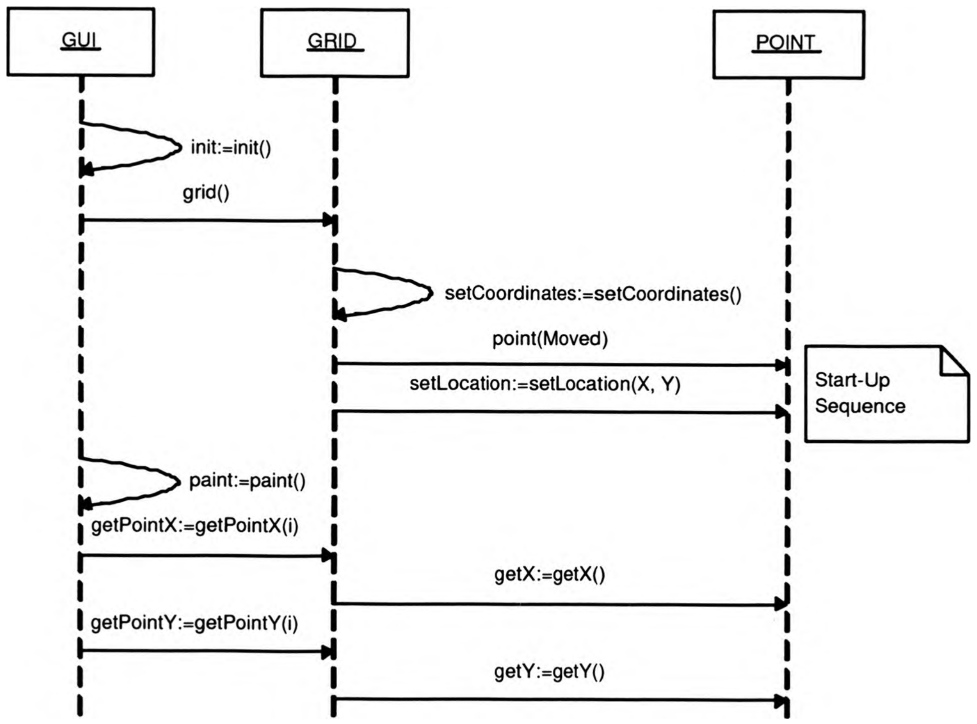


Figure 7.3 Sequence diagram showing start-up operation

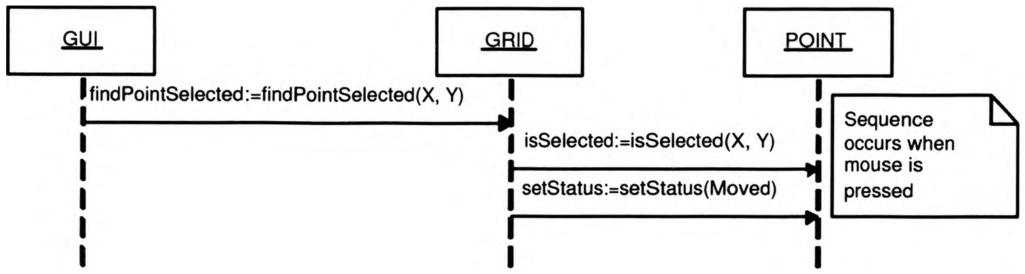

GUI
GRID
POINT
findPointSelected:=findPointSelected(X, Y)
isSelected:=isSelected(X, Y)
setStatus:=setStatus(Moved)
Sequence occurs when mouse is pressed

Figure 7.4 Sequence diagram showing mouse is clicked to select a vertex

GRID passes a message to POINT to call *isSelected* to check whether the point is selected. GRID passes a message to POINT to call *setStatus* to set the status to selected.

7.3.3 Mouse is dragged after selecting a vertex The sequence that takes place when the user moves the selected vertex is shown in Figure 7.5. GUI starts the sequence by passing a message to GRID to call *changeCursor* to change the cursor. GRID calls *withinMinMaxRange* to check the boundary conditions. GRID passes a message to POINT to call *getDistance* to get the distance between the two points. The sequence ends when GUI passes a message to GRID to call *getCursorStatus* and depending on the status, cursor is changed on the screen.

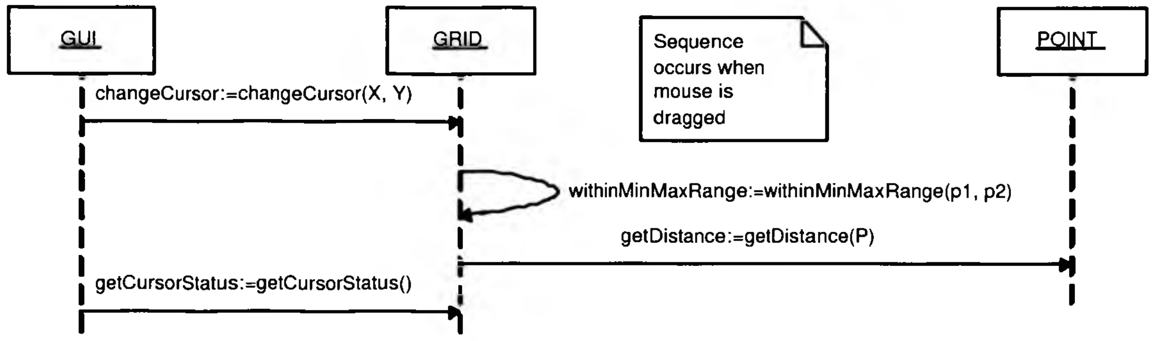

GUI
GRID
Sequence occurs when mouse is dragged
POINT
changeCursor:=changeCursor(X, Y)
withinMinMaxRange:=withinMinMaxRange(p1, p2)
getDistance:=getDistance(P)
getCursorStatus:=getCursorStatus()

Figure 7.5 Sequence diagram showing mouse is dragged after selecting a vertex

7.3.4 Mouse is released after selecting and moving a vertex The sequence that takes place when the user releases the mouse after selecting and moving a vertex is shown in Figure 7.6. GUI starts this sequence by passing a message to GRID to call *transformGrid* to transform the net. GRID passes a message to POINT to call *getStatus* to get the status of the point. GRID calls *calc_2* to calculate the transformed co-ordinates of the points. GRID checks the boundary conditions by calling *withinMinMaxRange* and passes a message to POINT to call *getDistance* to get the distance between two points. GRID calls *getAngle* to calculate the angle and passes messages to POINT to call *getDiffX* and *getDiffY* to get the difference between the x and the y co-ordinates of two points. GRID passes a message to POINT to call *getDistance* to get the distance between two points. GRID passes a message to POINT to call *getQuadrant* to get the quadrant of the point. GRID passes a message to POINT to call *getNewX* and *getNewY* to get the new co-ordinates of the point. GRID passes a message to POINT to call *setLocation* to set its location.

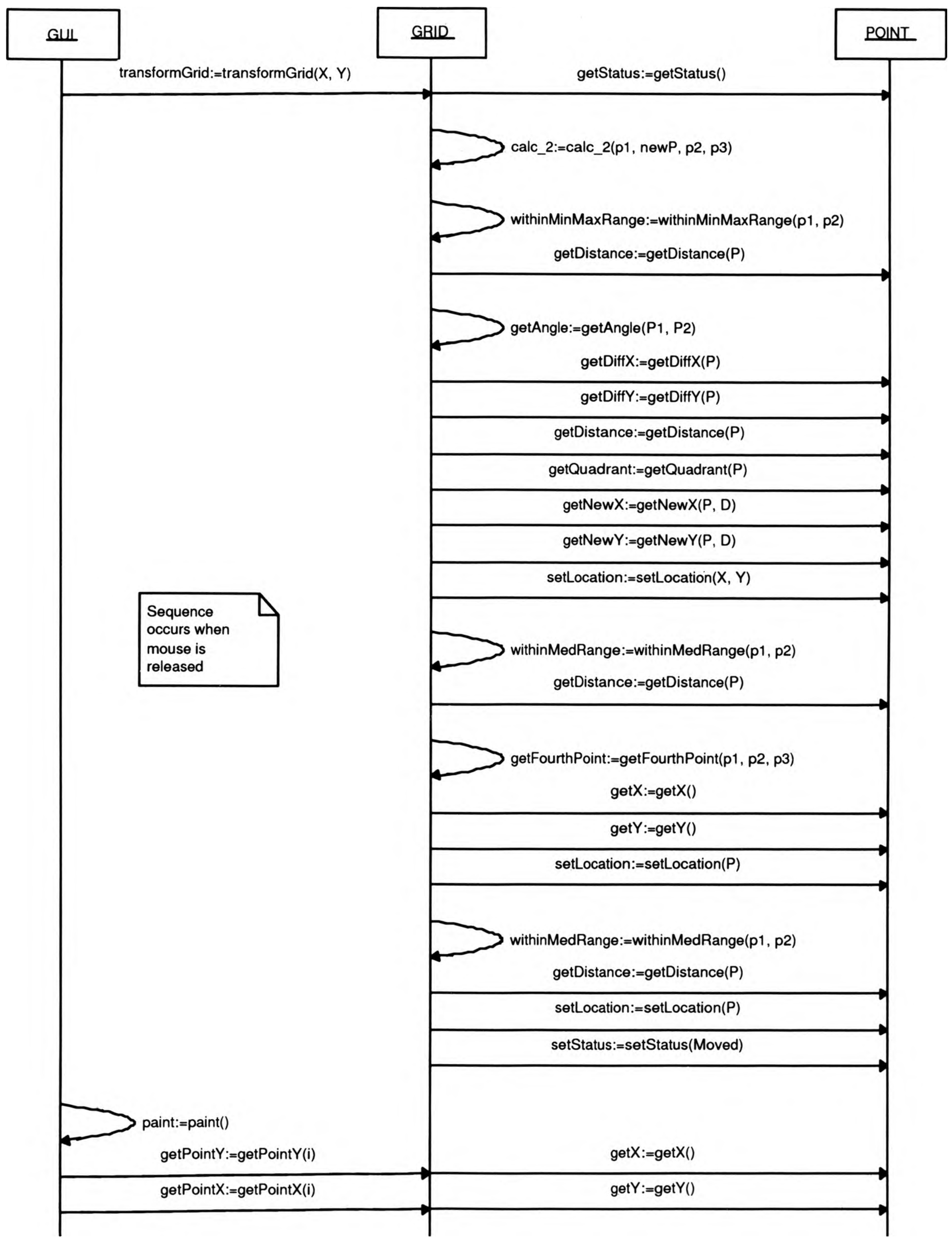

GUI
GRID
POINT
transformGrid:=transformGrid(X, Y)
getStatus:=getStatus()
calc_2:=calc_2(p1, newP, p2, p3)
withinMinMaxRange:=withinMinMaxRange(p1, p2)
getDistance:=getDistance(P)
getAngle:=getAngle(P1, P2)
getDiffX:=getDiffX(P)
getDiffY:=getDiffY(P)
getDistance:=getDistance(P)
getQuadrant:=getQuadrant(P)
getNewX:=getNewX(P, D)
getNewY:=getNewY(P, D)
setLocation:=setLocation(X, Y)
Sequence occurs when mouse is released
withinMedRange:=withinMedRange(p1, p2)
getDistance:=getDistance(P)
getFourthPoint:=getFourthPoint(p1, p2, p3)
getX:=getX()
getY:=getY()
setLocation:=setLocation(P)
withinMedRange:=withinMedRange(p1, p2)
getDistance:=getDistance(P)
setLocation:=setLocation(P)
setStatus:=setStatus(Moved)
paint:=paint()
getPointY:=getPointY(i)
getPointX:=getPointX(i)
getX:=getX()
getY:=getY()

Figure 7.6 Sequence diagram showing mouse is released after selecting and moving a vertex

GRID checks the boundary conditions by calling *withinMedRange.* GRID passes a message to POINT to call *getDistance.* GRID calls *getFourthPoint* to get the co-ordinates of the fourth point by using the mid-point rule. GRID passes a message to POINT to call *getX* and *getY* to get the co-ordinates of three points. GRID passes a message to POINT to call *setLocation* to set its location. GRID checks the boundary conditions by calling *withinMedRange.* GRID passes a message to POINT to call *getDistance* to get the distance between two points. GRID passes a message to POINT to call *setLocation* to set its location. GRID passes a message to POINT to set its status using *setStatus.*

GUI calls the *paint* to print the net on the screen. GUI passes messages to GRID to call *getPointX* and *getPointY* for getting the x and the y co-ordinates of a point. GRID passes messages to POINT to call *getX* and *getY* to get the x and the y co-ordinates. The sequence ends with the display of transformed net on the screen.

8 IMPLEMENTATION OF THE OBJECT MODEL

An implementation of the object model described in Section 7 was done using Java. In this section, the nature of the graphical interactions is demonstrated in a variety of settings. Beginning with the elemental subnets, the demonstration is expanded through the various sizes.

8.1 ELEMENTAL SUBNETS

Figure 8.1 displays the elemental subnet in the initial state. An elemental subnet consists of four vertices $v0, v1, v2$ and $v3$ with vertex $v0$ fixed in this case. The smaller circle in the figure represents the path on which vertices $v1$ and $v2$ can move and the larger circle represents the area in which vertex $v3$ can move while remaining mechanically admissible. By clicking and dragging a vertex, there are various cases as discussed below. Note that the cursor changes in order to indicate that a vertex can be moved to a new location.

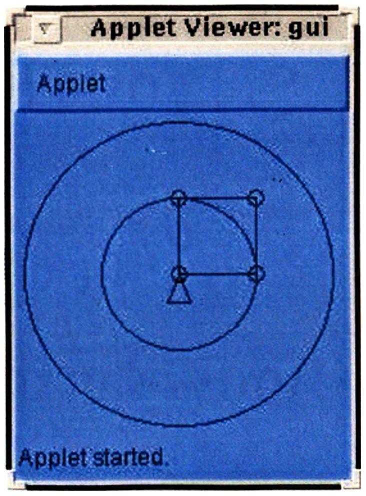


Figure 8.1 Elemental Subnet with vertex v0 fixed

8.1.1 Vertex v0 fixed and vertex v3 moved Figure 8.2 displays the elemental subnet with vertex $v0$ fixed and vertex $v3$ moved. The change in the location of vertex $v3$ transforms the subnet, making vertices $v1$ and $v2$ change their locations from their initial positions (see Figure 8.1). Vertex $v3$ can only be moved within the larger circle. Here Lemma 4.3 is applicable and shows that the user can drag vertex $v3$ to any point within the larger circle and remain mechanically realistic.

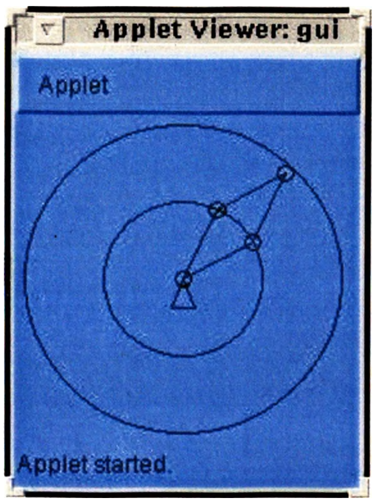


Figure 8.2 Elemental subnet with vertex v0 fixed and vertex v3 moved

8.1.2 Vertices v0 and v2 fixed and vertex v1 moved Figure 8.3 displays the elemental subnet with vertices $v0$ and $v2$ fixed and vertex $v1$ moved. The change in the location of vertex $v1$ transforms the subnet, making vertex $v3$ change its location from initial position. Vertex $v1$ can only be moved along the circumference of the smaller circle. Here Lemma 4.5 is applicable and shows that the user can drag vertex $v1$ to any point on the smaller circle and remain mechanically realistic.

8.1.3 Vertices v0 and v1 fixed and vertex v2 moved Figure 8.4 displays the elemental subnet with vertices $v0$ and $v1$ fixed and vertex $v2$ moved. The change in the location of vertex $v2$ transforms the subnet, making vertex $v3$ change its location from initial position. Vertex $v2$ can only be moved along the circumference of the

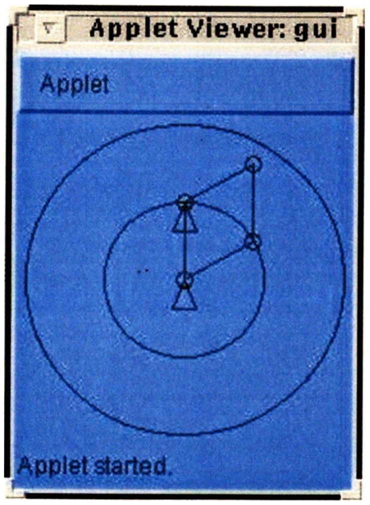


Figure 8.3 Elemental subnet with vertices v0 and v2 fixed and vertex v1 moved

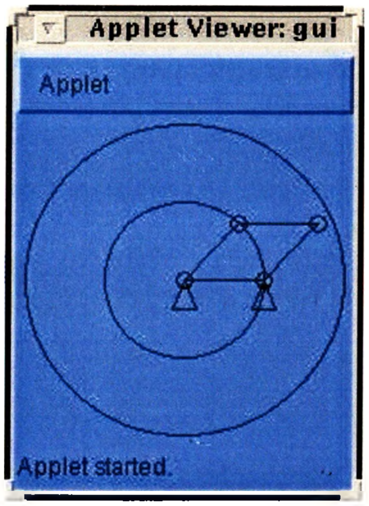


Figure 8.4 Elemental subnet with vertices v0 and v1 fixed and vertex v2 moved

smaller circle. Here Lemma 4.5 is applicable and shows that the user can drag vertex $v2$ to any point on the smaller circle and remain mechanically realistic.

8.2 SQUARE NETS OF ORDER N

Figure 8.5 displays the square net of order $N = 10$ in the initial state. The following cases were selected for the demonstration of click and drag event interactions.

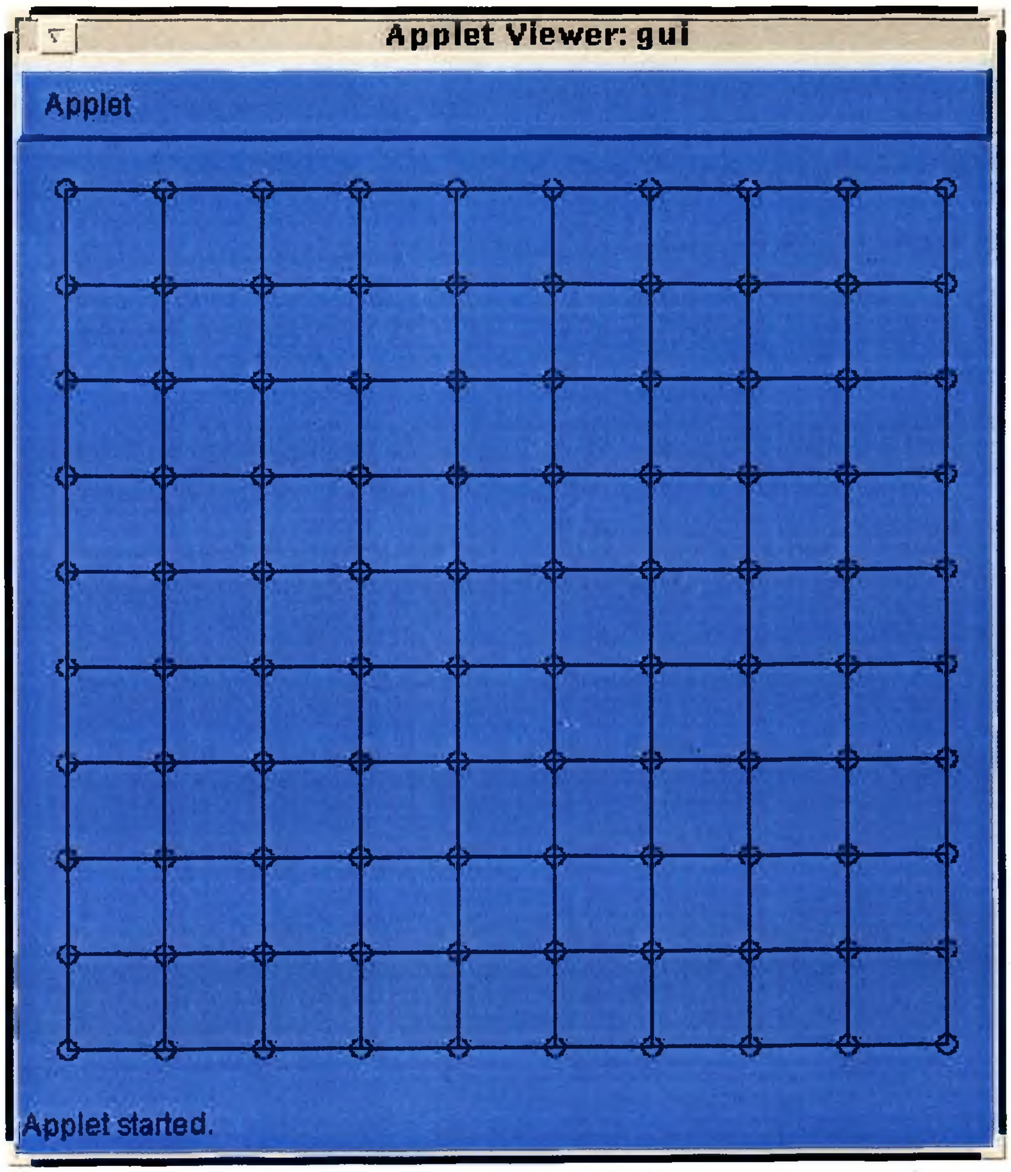


Figure 8.5 Square net of order N

8.2.1 All the vertices in square net of order N-1 fixed and vertex $v_{(N^2-1)}$ moved

Figure 8.6 displays the square net of order N with all the vertices in the square net of order $N-1$ fixed in the initial state. Small circles in the figure represent the path that a vertex on the N^{th} column or the N^{th} row has to follow and the larger circle represents the area in which vertex $v_{(N^2-1)}$ can move while remaining mechanically admissible.

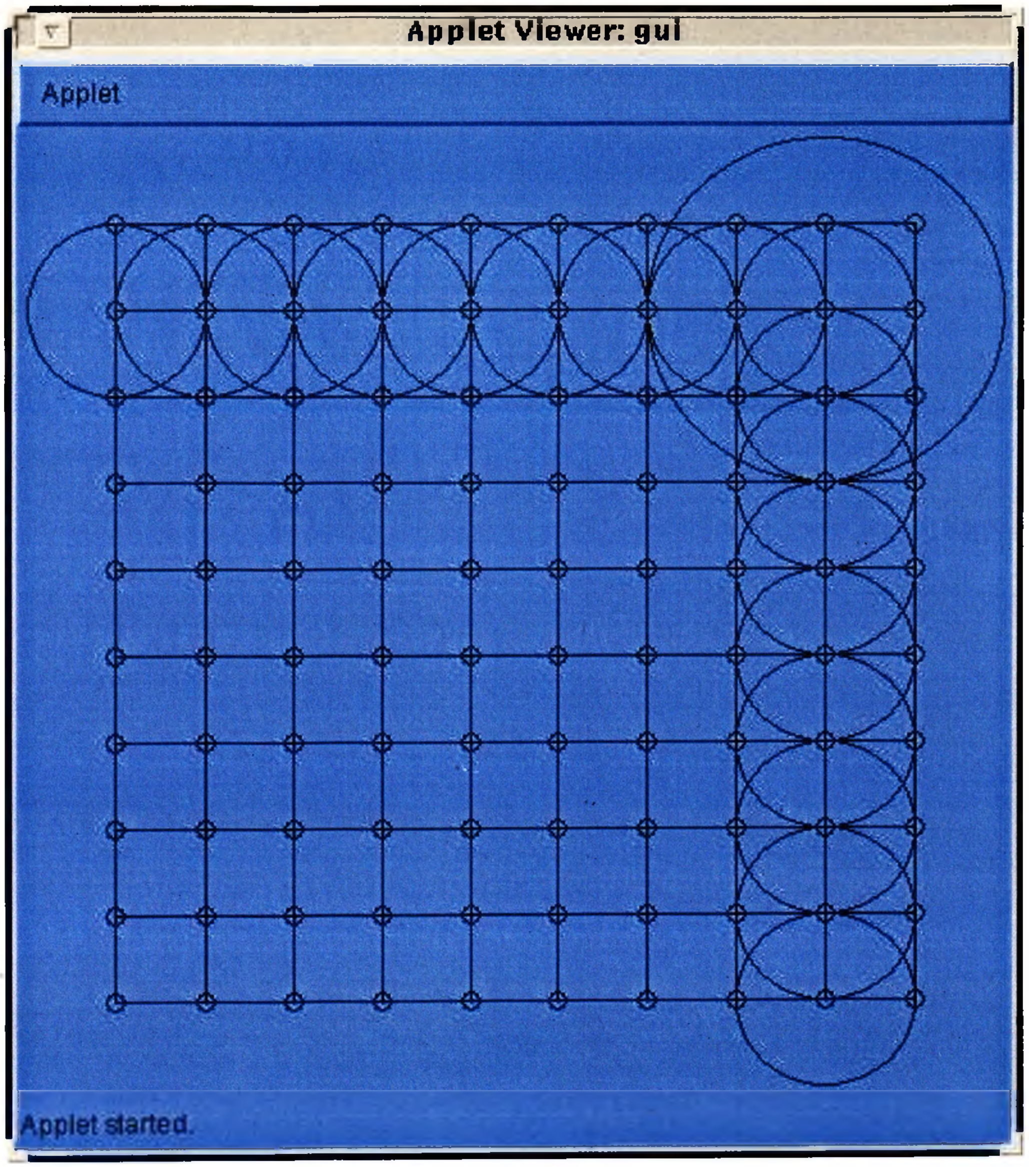


Figure 8.6 Square net of order N before vertex $v_{(N^2-1)}$ is moved

When vertex $v_{(N^2-1)}$ is moved, it transforms the square net, making all the vertices in N^{th} column and N^{th} row change their locations from their initial positions (see Figure 8.7). Here Theorem 5.3 is applicable and shows that the user can drag vertex $v_{(N^2-1)}$ to any point within the larger circle and remain mechanically realistic.

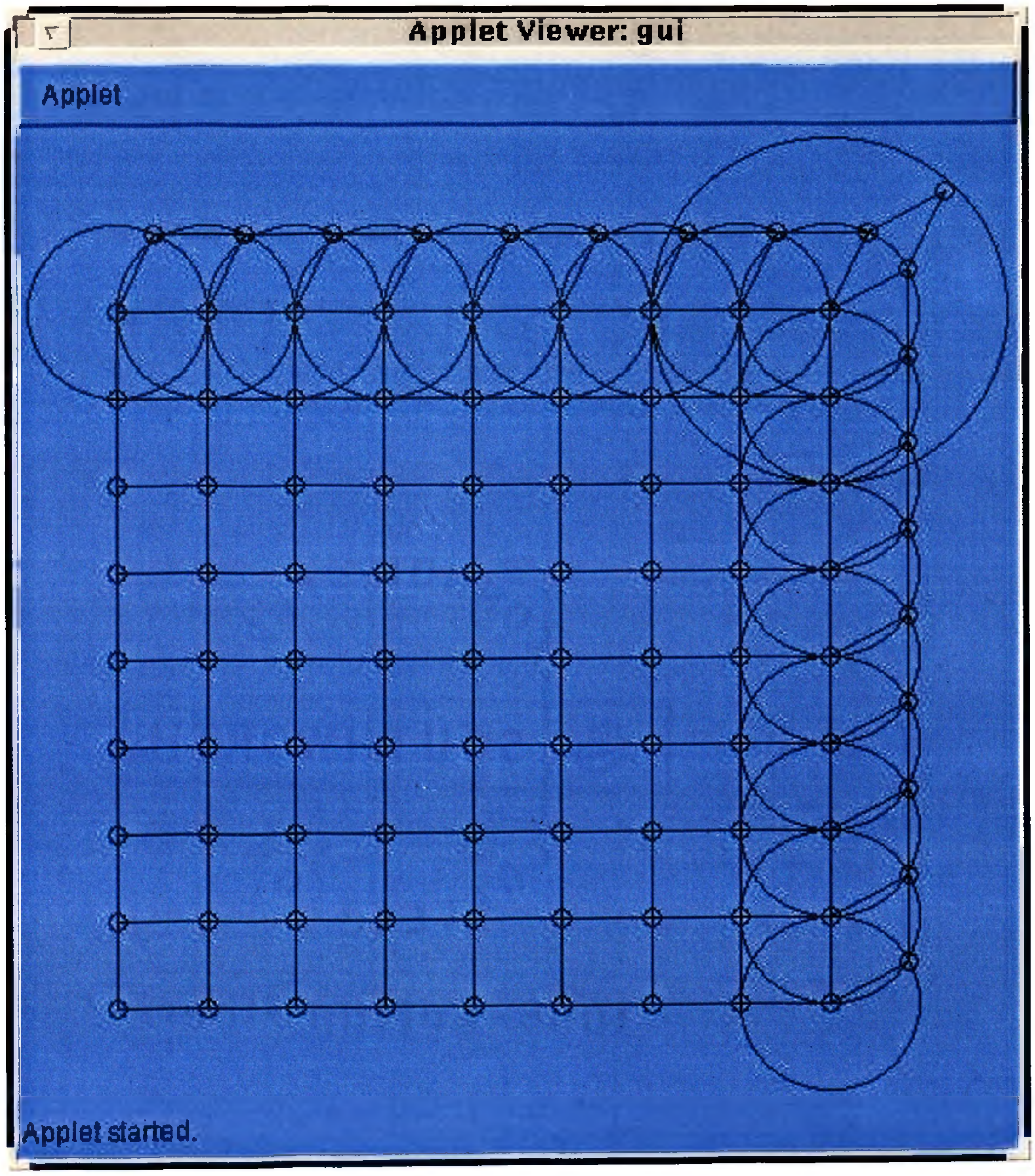


Figure 8.7 Square net of order N after vertex $v_{(N^2-1)}$ is moved

8.2.2 All the vertices in N-1 columns are fixed and a vertex in N^{th} column is moved   Figure 8.8 displays the square net of order N with all the vertices in $N-1$ columns fixed in the initial state. Circles in the figure represent the path that a vertex on the N^{th} column can follow while remaining mechanically admissible.

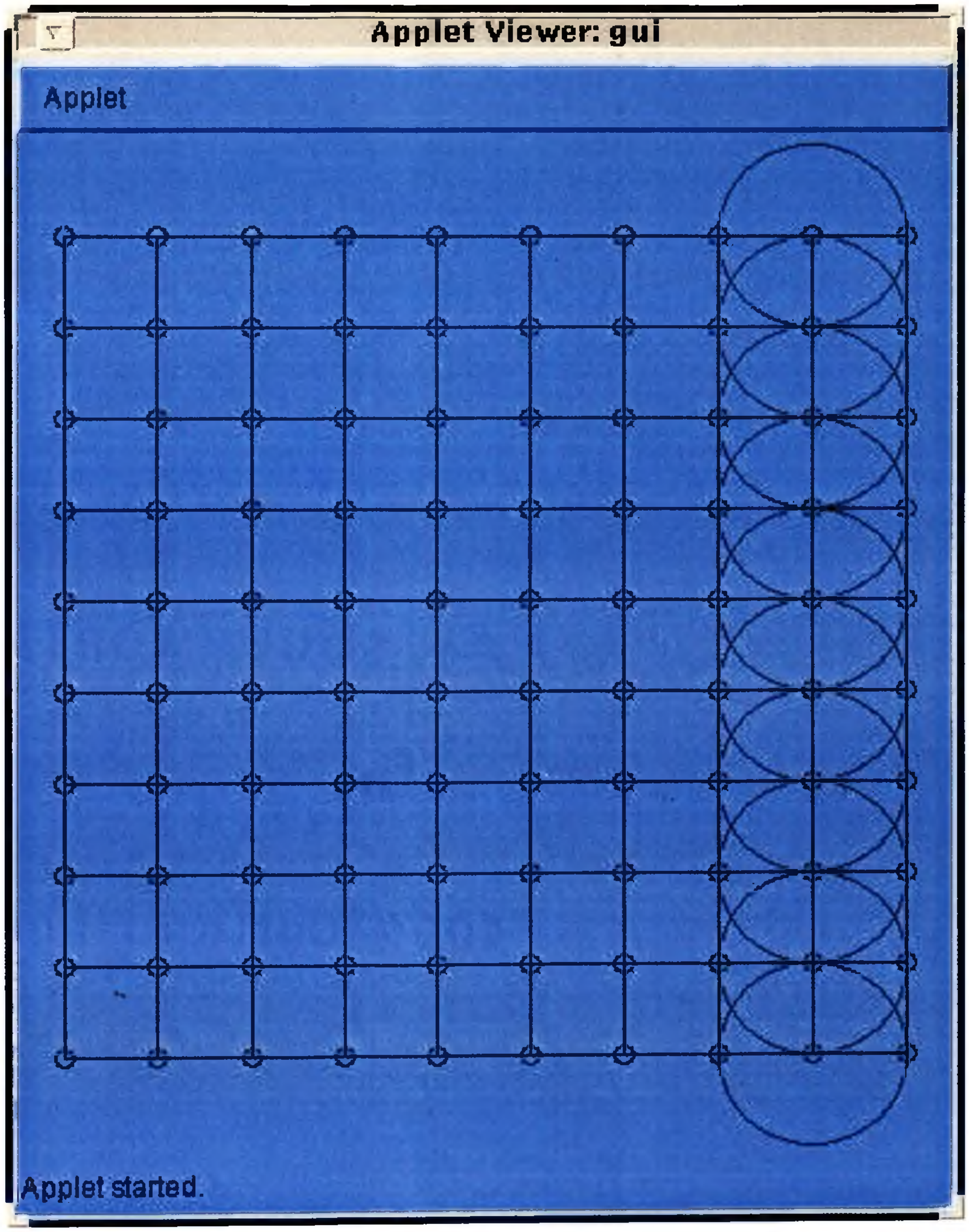


Figure 8.8 Square net of order N with all the vertices in N -1 columns fixed

When a vertex in the N^{th} column is moved, it transforms the square net, making all the vertices in the N^{th} column change their locations from their initial positions (see Figure 8.9). Here Theorem 5.5 is applicable and shows that the user can drag a vertex in the N^{th} column to any point on the circle and remain mechanically realistic.

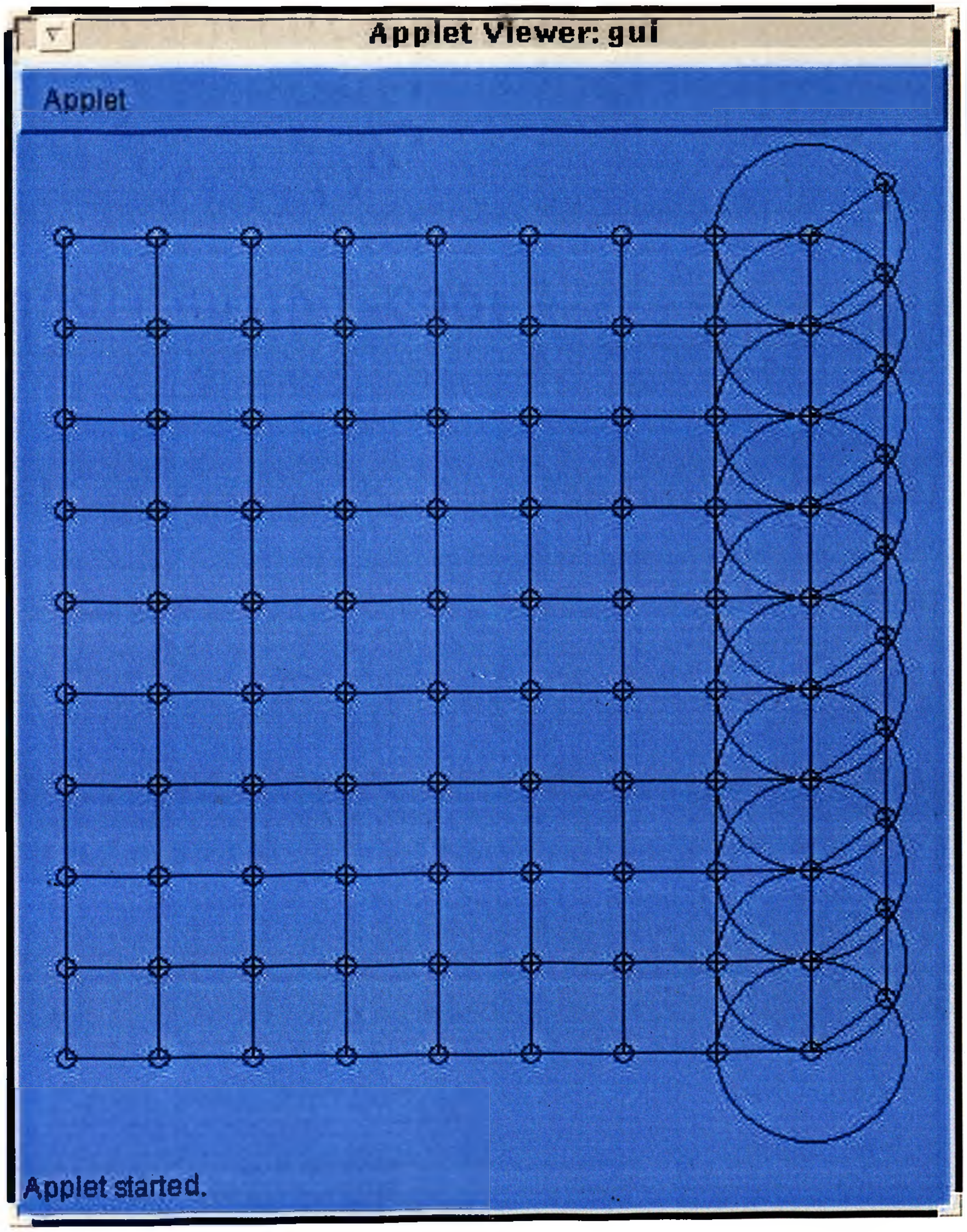


Figure 8.9 Square net of order N after a vertex in N^{th} column is moved

8.2.3 All the vertices in N-1 rows are fixed and a vertex in N^{th} row is moved

Figure 8.10 displays the square net of order N with all the vertices in $N-1$ rows fixed in the initial state. Circles in the figure represent the path that a vertex on the N^{th} row can follow while remaining mechanically admissible

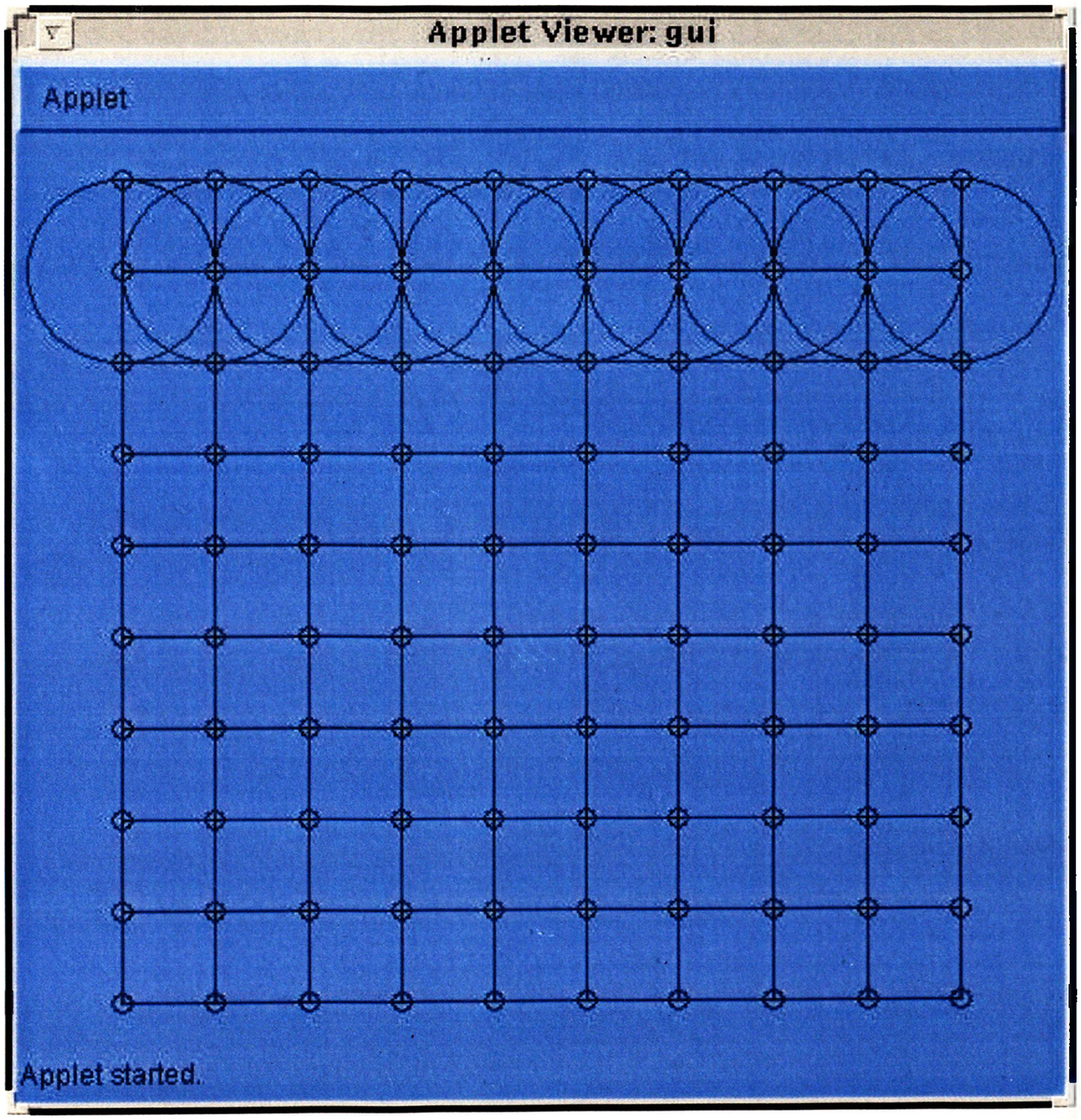


Figure 8.10 Square net of order N with all the vertices in N -1 rows fixed

When a vertex in N^{th} row is moved, it transforms the square net, making all the vertices in N^{th} row change their location from their initial position (see Figure 8.11).

Here Theorem 5.5 is applicable and shows that the user can drag a vertex in the N^{th} row to any point on the circle and remain mechanically realistic.

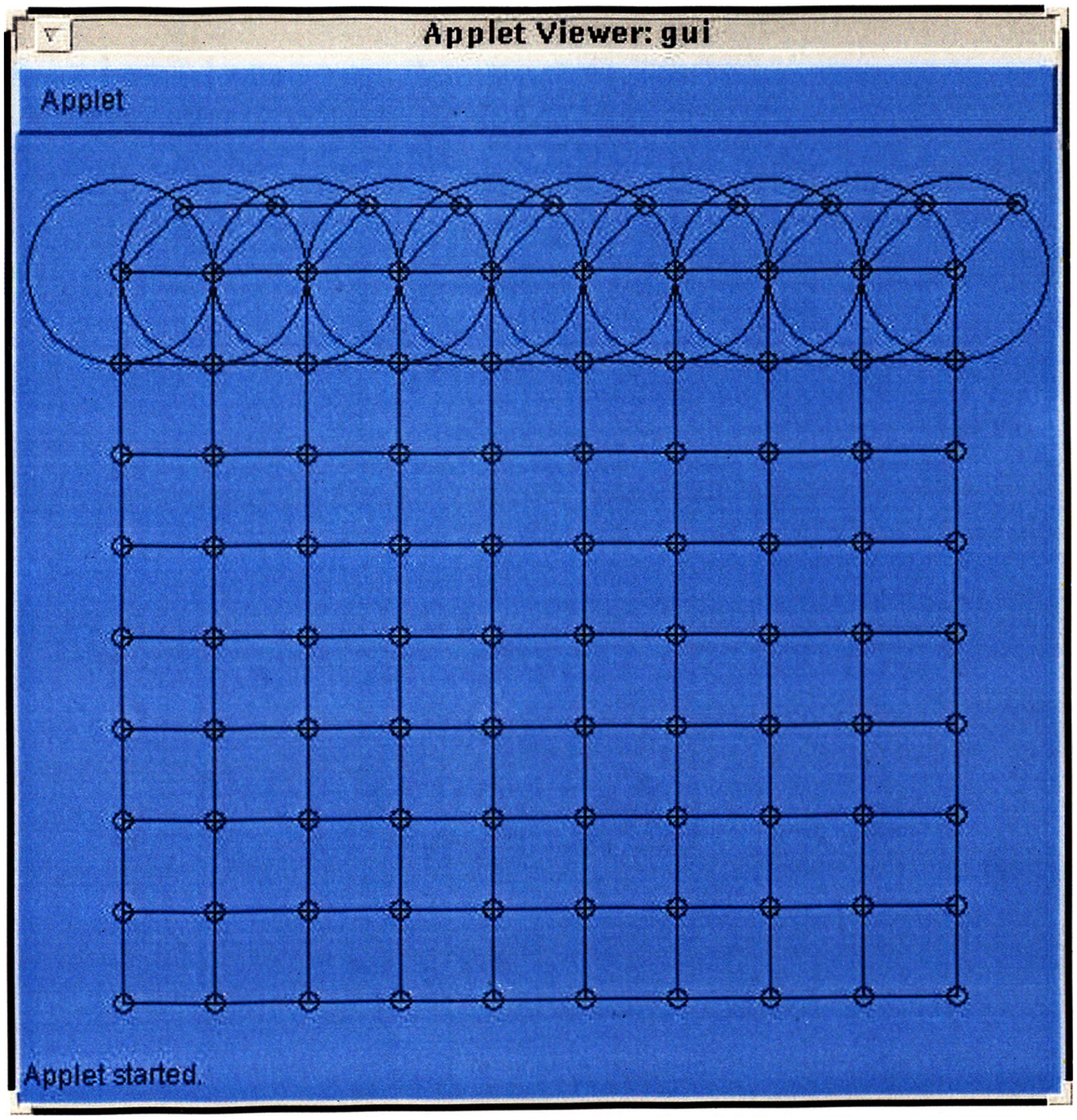


Figure 8.11 Square net of order N after a vertex in N^{th} row is moved

9 CONCLUSIONS

9.1 SUMMARY

This research was started with defining various terms that were needed to build and analyze the mathematical model. Then the behavior of a square net of order 2 was studied which provided the foundation for what is observed in the larger nets. This behavior was successfully extended to square nets of order 3 and also to square nets of order N.

An object-oriented model was developed using UML. This model is capable of handling various cases related to the square nets of different orders including the square net of order N. The model was designed in such a way that it can be implemented in any object-oriented language. An object-oriented model was implemented in Java and was tested on square nets of different orders.

9.2 FUTURE WORK

The present work is based on the behavior of square nets. The terms defined and theorems proved can very well be used to extend the mathematical model to incorporate rectangular nets. Providing $N \times N$ solutions for all the cases of square nets would be a logical direction to extend this work.

An object-oriented model has been provided for handling nets of graphical objects. It would be interesting if this model is extended to incorporate the needs of web-sites that allow image modification and record the changes made.

BIBLIOGRAPHY

[1] Greg J. Badros, Alan Borning and Peter J. Stuckey. *The Cassowary Linear Arithmetic Constraint Solving Algorithm.* ACM transactions on computer-human interaction, volume 8, number 4, December 2001, 267-306.

[2] Greg J. Badros, J. Nichols and Alan Borning. *SCWM-The Scheme Constraints Window Manager.* Proceedings of the AAAI spring symposium on smart graphics, March 2000.

[3] Michael G. Hilgers. *Results of Honda Initiation Grant BLUEPRINT A BIKE.* Technical Report CSC-02-02, Department of Computer Science, University of Missouri - Rolla, September 2002.

[4] Bashforth Byron and Yee-Hong Yang. *Physics-Based Explosion Modeling.* Graphical models and image processing, volume 63, number 1, February 2001, 21-44.

[5] J. F. O' Brien and J. K. Hodgins. *Graphical Modeling and Animation of Brittle Fracture.* Computer Graphics (proceedings Siggraph '99), August 1999, 137-146.

[6] H. N. Ng and R. L. Grimsdale. *Computer Graphics Techniques for Modeling Cloth.* IEEE computer graphics and applications, 16, number 5, September 1996, 28-41.

[7] J. Weil. *The Synthesis of Cloth Objects.* Computer Graphics (proceedings Siggraph '86), volume 20, number 4, August 1986, 49-54.

[8] P. L. Tchebychev. *Sur la coupe des vetements.* Assoc. franc. pourl' avancement des sci., congres de paris, 1878, 154-155.

[9] T. Agui, Y. Nagao and M. Nakajma. *An Expression Method of Cylindrical Cloth Objects - An Expression of Folds of a Sleeve using Computer Graphics.* Trans. Soc. of Electronics, Information and Communications, volume J73-D-II, number 7, 1990, 1095-1097.

[10] Michael G. Hilgers and A.C. Pipkin. *Elastic Sheets with Bending Stiffness.* Q. JI Mechanics applied Mathematics, volume 45 , 1992, 59-75.

[11] Sherman R. Alpert. *Graceful Interaction with Graphical Constraints.* IEEE computer graphics and applications, March 1993, 82-91.

[12] Peter D. Lax. *Hyperbolic Systems of Conservation Laws and the Mathematical Theory of Shock Waves.* Society for industrial and applied mathematics, 1973.

[13] Ralph E. Johnson and Brian Foote. *Designing Reusable Classes.* Journal of object-oriented programming, volume 1, number 2, June/July 1988, 22-35.

[14] Robert L. Young. *An Object-Oriented Framework for Interactive Data Graphics.* OOPSLA Proceedings, October 1987, 78-90.

[15] Ivar Jacobson. *Object Oriented Development in an Industrial Environment.* OOPSLA Proceedings, October 1987, 183-191.

[16] Grady Booch, James Rumbaugh and Ivar Jacobson. *The Unified Modeling Language User Guide.* Addison Wesley Longman, Inc., 1999.

VITA

Rakesh Kumar Bajaj was born on August 17, 1977 at Hyderabad, Andhra Pradesh, India. Rakesh received his primary and secondary education at Hyderabad. He did his B.E. in Computer Science and Engineering at Vasavi College of Engineering under the Osmania University, Hyderabad and graduated in June of 1998. After graduation he joined Computer Maintenance Corporation (CMC) Limited, Hyderabad as Systems Integration Engineer. During his two year stay at CMC he worked on various projects ranging from software development to IT consultancy.

In January of 2001 he left CMC to begin his graduate studies in the department of Computer Science at the University of Missouri - Rolla, where he was supported by two semester long quarter-time teaching assistantship for assisting students in Programming Methodology Laboratory (C++), an introductory course.