Soft-Object Modelling Using A Chainmail Algorithm
Matt Dean
Computing
2003/2004

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student)
Summary

The application of Soft-Object modeling takes a key role in creating realistic, real-world simulations using computer graphics. It is needed to allow natural forces to interact with objects and entities within a simulated environment. By applying these forces to an environment (i.e., gravity, wind, user interaction) the creation of useful, measurable, consistent, realistic models becomes a reality. It allows realistic models of objects such as cloth, water, skin and many other real-world materials with “soft” properties.

There are many soft object modeling techniques available that simulate object deformation in different ways. Some use direct manipulation of an object (physically-based) and need to be given rules and laws, like in the real-world, to govern how the object will react to the deformation. Others use in-direct manipulation of objects (geometrically-based) by linking parts of the object to some other space in the environment, sometimes a cube is drawn around an object and deformation of that cube results in some relative deformation of the object. Each method has its own advantages which make it useful for certain applications. After looking into some deformation modeling techniques we look to the 3D Chainmail (Frisken) technique. More specifically, we look at an adaptation of the original 3D Chainmail called Generalized Chainmail (Yi, Brodlie).

The purpose of this project is to compare the Generalized Chainmail (Yi, Brodlie) with a possible alternative (See Section 4). Both algorithms will be implemented in C++ using the OpenGL graphics library and a visual comparison will be carried out to obtain which algorithm is the most efficient. The comparison of efficiency will be calculated by monitoring which algorithm can sustain real-time interaction when object size is increased.

The original implementation of the Generalized Chainmail used a VRML graphics engine and Java interface to allow distributed use of the application. As the Generalized Chainmail was originally implemented using a web-based interface, it is necessary to deploy the Generalized Chainmail in the same way as the modified algorithm to allow a fair, un-handicapped comparison.
# Table of Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Deformable modelling Techniques</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Finite Element Method</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Free Form Deformations</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Boundary Element Method</td>
<td>3</td>
</tr>
<tr>
<td>2 3D Chainmail</td>
<td>4</td>
</tr>
<tr>
<td>2.1 The Algorithm</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Problems</td>
<td>7</td>
</tr>
<tr>
<td>3 Generalized Chainmail</td>
<td>7</td>
</tr>
<tr>
<td>3.1 The Algorithm</td>
<td>8</td>
</tr>
<tr>
<td>3.2 Possible Alterations</td>
<td>10</td>
</tr>
<tr>
<td>4 The Natural Queue Chainmail</td>
<td>11</td>
</tr>
<tr>
<td>4.1 The Algorithm</td>
<td>11</td>
</tr>
<tr>
<td>4.2 Custom Data Structures and Classes</td>
<td>12</td>
</tr>
<tr>
<td>4.3 Implementation Methods</td>
<td>13</td>
</tr>
<tr>
<td>4.4 Development Issues</td>
<td>15</td>
</tr>
<tr>
<td>4.5 Possible Enhancements</td>
<td>17</td>
</tr>
<tr>
<td>4.6 Conclusion</td>
<td>18</td>
</tr>
<tr>
<td>5 The Graphical Interface</td>
<td>18</td>
</tr>
<tr>
<td>5.1 Objects and Elements</td>
<td>19</td>
</tr>
<tr>
<td>5.2 User Interaction</td>
<td>20</td>
</tr>
<tr>
<td>5.3 Algorithm Graphics and Representations</td>
<td>20</td>
</tr>
<tr>
<td>5.4 Problems Occurred</td>
<td>21</td>
</tr>
<tr>
<td>6 Efficiency Evaluation</td>
<td>22</td>
</tr>
<tr>
<td>6.1 The Performance of Natural Queue Chainmail</td>
<td>23</td>
</tr>
<tr>
<td>6.2 Efficiency against Generalized Chainmail</td>
<td>23</td>
</tr>
</tbody>
</table>
1. Deformable Modeling Techniques

Deformable modeling allows computer graphics applications to construct objects that can be molded and deformed in much the same way as a real-world instance of that object would. These deformations can be applied to an object to show how that object would react under certain pressures and forces. This can be very useful when trying to simulate a circumstance that is difficult or dangerous to create in real life, such as training surgeons correct operating procedures, or simulating how certain structures react to earthquakes. It is also useful for copying or mimicking how an object reacts in real life so allow realistic simulations for the real world. This application has a large area of research in the field of virtual reality, and in the computer gaming industry where developers are always striving to create games with greater realism.

All deformable modeling techniques fall into one of the two following categories: physically-based deformation and geometrically-based deformation. These categories show how the deformations are applied to the object.

Physically-based deformations are deformations that involve direct manipulation of an object by some physical force. This allows deformation according to internal and external forces. A force of some kind will interact with a single point or an area of points on an object and that interaction will lead to a deformation of some kind, depending on the laws that govern the object and environment. Using this technique allows more realism in any transformation as this is how objects are deformed in the real world. Techniques such as Finite Element Method (See Section 1.1), Boundary Element Method (See Section 1.3) and Chainmail methods (See Sections 2, 3 and 4) all fall into this category.

Geometrically-based deformations involve manipulation of control points surrounding the object and leads to a geometric deformation based on how these control points are transformed. This techniques uses no physical laws to govern how an object can be transformed, the deformations are translate straight from the control points. The main technique that involves this type of deformation is the Free Form Deformation (See Section 1.2).

1.1. Finite Element Method

The finite element method is used in many different applications and areas of study. It is used in other areas such as Optical Waveguide Theory, Electromagnetics and Geo-technical Engineering plus many more. The one area that relates to the work in this report is that of
deformation modeling. An example of how finite element method can be used for deformation modeling is show by Bro-Nielsen and Cotin 1998.

When using the finite element method to create model deformations, the object to be deformed is divided into subdivisions that each contains a number of elements. These elements are able to be deformed in a restricted way, allowing computation of the entire subdivision of elements. From this, an equation is made that relates the deformation with the objects elastic force. Once the equations are made, they are then solved using a number of different techniques. The finite element method only works on unstructured meshes, such as tetrahedral meshes.

The finite element method allows extremely high accuracy physical deformations to take place on two dimensional and three dimensional objects. In applications where high accuracy is needed the finite element method can delivery this required level of accuracy. One of the main problems with this technique though, is that as the level of accuracy, number of nodes and the degrees of freedom increases in the finite element model, the amount of computation needed also increases. To allow real-time interaction with a large finite element model, it must have extreme large computational resources (CPU time and Memory) at its disposal. As the speed and size of computing resource technology increase the finite element method become more feasible as a useful accurate modeling solution. It is possible to use the finite element method for smaller, less accurate tasks but with other techniques available that can deliver less accurate solutions with less processing power needed the finite element method is better suited for the higher accuracy tasks. The need for enormous computational power is largely down to the mathematical complexity of the algorithm itself. Other techniques such as Free Form Deformations (See Section 1.2) and Chainmail Methods (See Sections 2, 3, and 4) are examples of less mathematically complex alternatives.

1.2. Free Form Deformations

Free form deformations are one of the main geometrically-based deformation methods available in deformation modeling. This method involves covering an object with a lattice of control points that are used to deform the object in question. The free form deformations allow objects to be deformed in relation to the lattice mesh that the object is embedded inside. Each point on the lattice (called a control point) is linked to some point on the object and deformation of that control point results in an equivalent deformation of the point that it is link to.
This technique can allow large objects to be transformed in real-time as users interact with the lattice's control points instead of direct manipulation of the object. This method of deformation is useful in such areas as object design in the cartoon, film and computer gaming industry, where realistic response from an object is not always needed.

One of the main disadvantages of this technique is that realistic deformations cannot take place as the force interacting with the object never gets direct access to manipulate the object. This can mean that free form deformations are not able to be used for many graphic applications when high accuracy or real-world forces, such as gravity, wind, rain, earthquakes or even sometimes user interaction, are needed. See Fig 1.2a for an example.

![Fig 1.2a: Example of a free-form deformation on a cartoon car.](image)

1.3. Boundary Element Method

The boundary element method has many uses outside of the field of deformation modeling; it is used in the field of solving partial differential equations. It has a lot in common with the finite element method (See Section 1.1), but has one obvious difference. The boundary element method only carries out its calculations in relation to the boundaries of the object.

If for example a regular sphere is being deformed, this means that the algorithm carries out its calculations on the surface of the sphere only. Instead of subdividing the whole object (internal and external) into divisions (as with the finite element method), it only divides the surface of the sphere into subdivisions. This gives the boundary element method a great computational advantage over similar techniques, such as the finite element method and the finite difference method, as it only needs to due it's computations for the elements on the surface of the object.
One of the main problems with this approach is that if any internal elements exist within the object, they will not be processed along with the other elements on the surface. This means that accurate transformations on volumetric meshes cannot be carried out using this method. This technique can be used on volumetric meshes but only the surface elements will be affected by the algorithm, causing possible inconsistencies to the internal elements inside the object.

2. 3D Chainmail

3D Chainmail was first proposed by Frisken 1997, and is described as a “fast deformable modeling technique” (Li, Brodlie 2003). An object is defined by a number of points (called Elements) and the links between the elements are controlled by certain stiffness parameters that are predefined and stored within the object. Movement of elements in relation to each other is controlled by these parameters which cause the moved elements to drag or push their neighbors in a similar way in which chainmail armor works (See fig. 2a)

![Diagram of 3D Chainmail](image)

This approach differs greatly from most of the other approaches mentioned in Section 1 as instead of carrying out a small number of very complex calculations, the 3D Chainmail carries out a large number of simple calculations. Which means very large objects (sometimes tens of thousands of elements in size) can be created and simulated in real-time on even the most basic machines.
2.1. The Algorithm

All chainmail algorithms are carried out in two main steps. The first involves the movement of the originally moved element. This movement is then propagated onto its neighbors, and spreads though the whole object to each neighbor of every element. The second step is a relaxation step that centralizes each element in relation to its neighbors. This step is important to minimize the objects energy. To demonstrate how this works I will be using a two dimensional example (See fig. 2.1a)

The elements in this object are linked in a simple square mesh, so each element has at most four neighbors. Each neighbor for each element is categorized as either top, bottom, left or

Fig. 2.1a: 2D Mesh containing 9 Elements

Fig. 2.1b: Original positions of elements E and C.
right. Suppose element E is moved to a different position, its four neighbors A, B, C and D could all need to be moved due to E's position change. In this case E is known as the sponsor for any possible moves made by its neighbors. Each element now need to be added to one of four queues depending on its relation to E. Element A is added to the top list (as it is above E), B to the bottom list, C to the right list, and D to the left list. The right list is the first to be processed, so we look at element C. The initial positions of E and C are shown in Fig 2.1b. The area of valid regions for C, based on the new position of E, which we shall call E*, are shown in Fig. 2.1c. This region is calculated using the stiffness values of the link between E and C. These values are the compression (named cmpX), stretch (named strX) and the shear (named shrY). Different values for these parameters control the stiffness of the object. If C lies within the valid region defined by these values, no change is made to C's position and the left queue is then processed. If C lies outside of the boundaries of the valid region, C is moved to the nearest point, called C*, within the valid region. Also C's neighbors are added to the appropriate queues.

To update element C's position the following calculations are used:

\[
\text{if}(x(C) - x(E) < \text{cmpX}) \\
\quad \text{then } x(E) = x(A) + \text{cmpX} \\
\text{else if}(x(C) - x(E) > \text{strX}) \\
\quad \text{then } x(E) = x(A) + \text{strX} \\
\]

\[
\text{if}(y(C) - y(E) < -\text{shrY}) \\
\quad \text{then } y(E) = y(A) - \text{shrY} \\
\text{else if}(y(C) - y(E) > \text{shrY}) \\
\quad \text{then } y(E) = y(A) + \text{shrY} \\
\]

Now that C has been moved to its new position C* and its neighbors have been added to the appropriate queues, the right queue continues to be processed in this way until it is empty. Once the right queue is empty, the left queue is then processed and then the top, and then the bottom in turn.

A big part of the success of the algorithm is that every element in the object gets processed at most once. For example, when element C's neighbors get added to the queues element E
would not be added to the left queue as it has already been updated in the algorithm. Also all once the elements in the right queue have been processed, no more neighbors are added to that queue.

Once this chainmail step has been complete, it is followed by a relaxation step that adjusts element positions, according to where their neighbors are, to reduce the energy inside the object. Each element is adjusted so that it is an equal distance from each of its neighbors. To avoid object shrinkage (See Frisken-Gibson, 1999) it is necessary to move each neighbor towards the element by an amount that is equal to the optimal link length between itself and the element. This part of the algorithm is carried out when any free processing time is available (when CPU is idle) so not to interrupt or delay any processing of the chainmail step.

The algorithm can be easily extended into three dimensions. The main difference is that each element has six neighbors instead of four and the appearance of the z-axis to make the number of calculations significantly greater.

2.2. The Problems

The main problem with the 3D Chainmail is that it is restricted to deformations on regular rectangular meshes (in 2D and 3D). Each element must have neighbors above, below, to the left or to the right. This means that the initial state of the object must be able to have its elements classified in relation to one and other (top, bottom, left and right), making the ability to use this algorithm on arbitrary meshes impossible.

An alternative to using this 3D Chainmail algorithm is the use of an adaptation, called Generalized Chainmail (Li, Brodlie 2003). This algorithm supports much greater flexibility when modeling meshes for deformation as it allows the same algorithm to be used for deforming surface and volumetric meshes. It allows each element to have any number of neighbors and also gives the ability to introduce a cutting functionality that allows links between elements to be broken.

3. Generalized Chainmail

Generalized chainmail is an extension of the original 3D Chainmail algorithm (See section 2). Many aspects of the extended algorithm are similar to the original, such as the functionality to model rectangular meshes in two and three dimensions, and the relaxation step used to minimize objects energy. Although the generalized chainmail has these similarities with Frisken’s chainmail it also has many other abilities to improve on what the
original 3D chainmail algorithm could achieve.

One of the main advantages of this algorithm is that it is possible to embed a mesh of any dimension, into a space of higher dimension. For example:

- A 1D mesh of elements, as a 1D line, or as a curve on a 2D plane, or in a 3D space
- A 2D mesh of elements, as a triangular mesh, lying in a 2D plane or as a surface in 3D space.
- A 3D mesh of elements, as a tetrahedral mesh, in 3D space.

As well as the above examples, the rectangular meshes used by the original chainmail algorithm can also be modeled using this algorithm.

One of the most relevant differences is how the softness and shearing parameters are calculated (See section 3.1). In the generalized chainmail these parameters are expressed relative to the length of the original link between two elements, rather than as absolute values used globally by every element in the object. This allows each link to have independent properties instead of all the links in the object having exactly the same softness and shearing parameters.

3.1. The Algorithm

As with the 3D Chainmail (See Section 2.1) the algorithm is split into two steps, the chainmail step and the relaxation step.

For the first step we will use a 2D example. Fig 3.1a shows the original positions of elements A and B. Fig 3.1b shows the reaction of a move A to A* causing a move of B to B*, where B* is the nearest point in the valid region.

Each element has two separate values, x and y, which denote where about on respective axis the element lies. If the example was using three dimensions there would be a z value to consider for the z-axis.
So we say $A$ is at point $(x_A, y_A)$ and $B$ is at point $(x_B, y_B)$. We also define:

$$x\text{Vect} = |x_B - x_A|; \quad y\text{Vect} = |y_B - y_A|;$$

We also define our softness parameters, $cmp$ to denote compression and $str$ to denote stretch, which control how the links between elements can be stretched and compressed. A final value $shr$ controls the amount of shearing that is allowed. These values create a definition of a valid region, $R$, for element $B$ (see area in grey in fig 3.1b), after a move of $A$ to $A^*$ $(x_A^*, \ y_A^*)$.

We can define the valid region $R$ as follows:

$$R = \{ (x, y) : xMin \leq x \leq xMax; \ yMin \leq y \leq yMax \}$$

where (first we say that $x_B > x_A$ and $y_B > y_A$):

$$xMin = x_A^* + (cmp \times x\text{Vect} - shr \times y\text{Vect})$$
$$xMax = x_A^* + (str \times x\text{Vect} + shr \times y\text{Vect})$$
$$yMin = y_A^* + (cmp \times y\text{Vect} - shr \times x\text{Vect})$$
$$yMax = y_A^* + (str \times y\text{Vect} + shr \times x\text{Vect})$$

The size of the region $R$ is defined in terms of the original lengths between the elements $A$ and $B$, that is the un-deformed positions of $A$ and $B$ of when the object is first created, before any deformations have taken place. This region is fixed throughout every subsequent
deformation, to constrain the movement of B relative to A. The information on region R for each element in relation to its neighbors is stored within the object.

If we go back to the example used in section 2 (See Fig 3.1c), we see a 2D mesh containing nine elements. We can use this example to demonstrate the iterations that are used in Generalized Chainmail.

Firstly, element E is moved to a new position. The \((x,y)\) values of E are updated. The neighbors of element E are added to a queue of elements that are to be processed for possible adjustment. We call this queue the waiting list. Each neighbor is added to the front of the waiting list. At this stage the elements A, B, C and D are added to the waiting list. Then for each element in the waiting we carry out the following stages:

1) Check the processing state of the element.
2) If it is set to 'updated', it is removed from the waiting list and we go to the next element in the list.
3) If it is set to 'not updated', the position is checked with the region R, with respect to its sponsoring element. If it lies outside R it is moved to the nearest point inside the region and its neighbors are added to the front of the waiting list. Finally the processing state of the element is changed to 'updated' to indicate its position has been updated already.

This algorithm continues until the waiting list is empty.

**3.2. Possible Alterations**

The algorithm was originally unsatisfactory from a visual aspect as the chain reaction generated was uneven as the propagation starting from one element to its neighbor carried on straight through the object before a second neighbor is processed (See Fig 3.2a). A suggested solution was to alter the algorithm so that instead of adding potential neighbors to the front of the waiting list, we add them to the back. This would create a spiral effect as each element around the object was updated in turn (see Fig 3.2b), rather than as soon as it is added to the waiting list.
Other possible alterations would be to check if an element's processing state is 'updated' before adding it to the waiting list to avoid any messy list removal problems later. This would also keep the size of the list down to a minimum size and not take up so much processing memory.

4. Natural Queue Chainmail

Natural Queue Chainmail is a similar alternative to Generalized chainmail. The algorithm for both techniques works in the same way with some small exceptions. The idea behind creating this algorithm is to examine whether these small differences have any impact on the algorithms efficiency when dealing with deformations on objects with any number of component elements.

In this section we discuss how this algorithm works, the data structures and classes used in creating the algorithm and then look to the implementation methods that will be used to implement both the Natural Queue Chainmail and Generalized Chainmail for testing. Also we mention all the dimensional issues based around implementing the algorithm and also any problems that were encountered while in the processes of implementation.

4.1. The Algorithm

To see how the softness and shear parameters are calculated for each link between elements in the object, see section 3.1, which describes how both techniques, Generalized Chainmail and Natural Queue Chainmail, set up the stiffness properties of each link in the object. The Natural Queue Chainmail has the same algorithm the Generalized Chainmail with the following changes:
1) When elements neighbors are to be added to the waiting list, they are appended onto the back of the waiting list.

2) An elements processing state is checked before the element is added to the waiting list. If the elements processing state is 'updated' then the element is not added to the list. If it is 'not updated', it is added to the list.

Other than these two points the algorithm is the same as the one used in Generalized Chainmail. Although there won't be a noticeable difference in the outcome of any deformations made using either technique, note that by changing these two aspects, the differences in visual feedback from watching the deformations take place will be noticeable (See Fig 3.2a and 3.2b).

4.2. Custom Data Structures and Classes

When creating the data structures and classes for the implementation of both generalized chainmail and natural chainmail it was important to consider where it was necessary to use a similar structure to that in the generalized chainmail paper, Yi and Brodlie 2003. As the generalized chainmail was to be the same as the one in the paper, the best option would be to stick to that design, and make a few minor alterations.

The data structures and classes used in this implementation for both the generalized chainmail and the natural queue chainmail are as follows:

An object stored as the class:

```
Object = (list of element, list of softness properties , object dimension, space dimension, compression factor, stretch factor, shear factor )
```

This class is used to represent the object in the simulated environment. It contains a list of all the elements that make up the object, as well as softness data for each of the links connecting each element to its neighbor(s). This is stored inside the object so that the calculations needed to deform the object are worked out when the object is constructed, therefore reduces the number of calculations needed during any deformations performed on the object at run-time.
An element stored as the class:

\[
\text{Element} = \{ \text{identifier, original position (x,y), current position (x,y), processing state, list of neighbors} \}
\]

This class is used to represent an element. A number of elements connected together make up an object. An element contains its identifier, which is a unique integer that is used to distinguish elements from one and other. The original position variable is used to calculate the softness parameters between itself and its neighbors, it's the position the element starts at, before any movements are made. The current position variable is position the element is at, at any given time, after every move this variable is updated. The processing state (see sections 3.1 and 4.1) is here to allow the algorithm to see if the element has already being processed.

A waiting list type stored as a data structure:

\[
\text{Waiting List Data} = \{ \text{element, sponsor} \}
\]

The Waiting List Data data type is used to allow the algorithm to see which element sponsored an element's entry into the waiting list. Instead of just storing an element's identifier in the waiting list, this data type is used. This is important to allow the object to know which links softness parameters need to be used when calculating the new position of an element.

A softness data type stored as a data structure:

\[
\text{Softness Data} = \{ \text{sponsor, neighbor, valid region} \}
\]

The Softness Data data type is used to store a softness property for a single link between two objects. A list containing this data type is used to store all the softness parameters for each link, between two elements, in an object. The valid region variable is a numerical representation of the area R calculate in section 3.1

\section{4.3. Implementation Methods}

After the design phase of the algorithms is complete it is important that all implementation methods are considered thoroughly and the most appropriate for the application is chosen.
When choosing the methods that will be used for the implementation first you must consider why the implementation is being created and which methodology can deliver the required outcome. As we know, we are creating this implementation to compare the visual efficiency of Generalized Chainmail and Natural Queue Chainmail, to see if small modifications to the former technique results in a more efficient algorithm for the latter technique.

To allow the simulations to run as fast as possible the implementation of both techniques will be done using C++ as the base language. This allows the application to be run straight from the compiled code without any need for another level of processing after compilation (as is required in the Java approach, Li and Brodlie 2003), causing much faster execution of the application. By using C++ we are hoping that the lower level of extraction will help run larger simulations (thousands of elements is size) in real-time, giving us a much better opportunity to examine the visual results of the two algorithms.

To create the graphics of the simulation, we will be using the OpenGL graphics library. This also allows a faster approach to the graphics that the original Generalized Chainmail implementation, which used VRML to model the simulation. VRML, being a mark-up language needs no compilation but does need pre-processing when a VRML script is executed making execution speed much slower. The OpenGL code will be compiled in with the C++ code to allow a faster execution of the application. “OpenGL is close enough to the hardware so that programs written with OpenGL will run efficiently”, E Angel.

The structure of the application will follow an Object-Oriented approach for representing data structures in the code. Each necessary data structure will be created using a class, and instantiated as an object (See section 4.2). Using this method will allows a more realistic approach to object representation for the developer and anyone else wanting to use the code after the project is complete. As regards the graphical outcome of the OO approach, there should not be too much visual difference from any other approach that could be used.

When developing the code for the implementation of this project it is necessary to take a 'step by step' approach. To start the implementation, the Natural Queue Chainmail will need to be implemented. To begin, a one dimensional simulation will be developed (see Fig 4.3a). After this has been successfully finished, the creation of a simulation of a one dimensional object in two dimensional spaces (see Fig 4.3b). After completion of that, a creation of a two dimensional mesh in two dimensional space will be made (see Fig 4.3c). These simulations will successfully represent the Natural Queue Chainmail technique. Then finally the creation
of the Generalized Chainmail must take place to allow a comparison of the two algorithms. This will be done using a two dimensional mesh in two dimensional space (see Fig 4.3c).

![Fig 4.3a: 1D line, in 1D space, can be deformed along the x axis only.](image1)

![Fig 4.3b: 1D line in 2D Space, can be deformed along the x and the y axis.](image2)

![Fig 4.3c: 2D Mesh in 2D Space, can be deformed along the x and the y axis.](image3)

When creating some of the meshes for the third and fourth steps of the implementation it is important that the element link information is not hard coded into the source of the application, as the amount of data that would be needed will be a lot larger than the previous simulations. The method that will be used for storing the link data (which element neighbors which element) involves storing the data in a data file that is dynamically created by another application. This application works by taking the number of elements on the x axis and the number of elements on the y axis, and creating a data file for a rectangular mesh with those parameters. The data files will be formatted to allow the chainmail applications to read in the data and construct the objects from that data.

The data files will be formatted as follows:

Name: 3x3.dat (for a 3 element by 3 element mesh)

File content:

```
0 2 1 3
1 3 2 0 4
2 2 1 5
3 3 4 6 0
```
Format of each row:

```
elementID no_of_neighbs(N) neighb1 .... neighbN
```

E.g.

Element 0 has two neighbors, which are elements 1 and 3

The above data file would create a 3 element by 3 element rectangular mesh (see Fig 4.3c). Please note it is possible to modify the output from the simulator to add links and remove links by changing the content of the data file.

### 4.4. Development Issues

Using the 'step by step' approach to development (see Section 4.3) gives us the opportunity to pin-point any issues that may arise to hinder the development process. These issues were a major part of the development of this implementation and, to aid any further progression from this project, are contained in this section of the report.

When initially developing the first step of the implementation phase of this project, a small issue with the algorithm arose when developing the one dimensional simulation (see Fig 4.3a). The interaction between the user and the elements in the object was extremely static. Any alteration of the field of view, or perspective of the simulation resulted in errors when interacting with the elements. Trying to solve this problem would take time away from the development of the simulations and it was decided that dynamic interaction was not going to be needed at this stage of the project.

When developing the two dimensional mesh (see Fig 4.3c), during the organization of the waiting list for the algorithm (see section 4.1) it was noticed that the standard list container (used by C++, from the list.h header file) only allowed access to individual elements by the iteration through each element in the list. This implied that every time details of an element (stored in a list by the object) were needed, a search of all the elements in the list would need to be carried out. Doing this each time an elements details are needed, would create a processing overhead, which could maybe be avoided using a container like an associative
map, where elements could be referenced by an identifier. This was not noticed earlier as the size of the object was not larger enough to effect any real-time interaction. To keep the similarities with the original algorithm it was decided that the linked list was the best way to go, and therefore was used to store the details of the waiting list.

After the creation of the Generalized Chainmail and Natural Queue Chainmail simulations were complete it was noticed that the speed in which each of these simulations executed was significantly lower than the original simulation used for Generalized Chainmail. It only allowed the possibility for small 2D simulations to be constructed and simulated in real-time (only 49-64 elements mesh on an Intel P4 processor with excess of 512Mb of RAM). The reasoning for this was discussed with Y Li, the original creator of Generalized Chainmail, and the following areas were mentioned:

1) The calculation of the area $R$ for each link between elements should be calculated and stored at run-time. This avoids re-calculation every time the area $R$ is needed, which in turns removes more processing time from the whole algorithm.

2) The custom GLUT (included in the OpenGL library) shapes should not be used to represent the elements of an object. As these take more time to render and therefore take processing power that would be needed for the chainmail algorithms.

3) Avoid adding any lighting and effect to the simulation. This all adds to the processing time of each screen refresh which adds more processing time to the simulation.

Even after all these points were taken into consideration and the applications were updated with the new methods, there was very little improvement when larger scale objects were simulated. The simulation of 8x8 and 9x9 meshes can be carried out at real-time, but 2D 81 element objects is in no close to the simulations that were being used in the original implementation of Generalized Chainmail which comprised of a 3D 349 element object.

The addition of the relaxation step into both algorithms caused some elements to jump to different places in the space when elements at the far side of the object were deformed. This was observed to be a random side-effect of the relaxation step when it was implemented. As the relaxation step is an important part of the chainmail algorithm this side-effect would need to be resolved to allow a more accurate result from the chainmail simulations. Although it
caused this visual 'glitch' there was no disruption to the speed of the simulations as the relaxation step was introduced into the idle function provided by OpenGL, this function executes whenever the application is idle and there are no other processes/functions being executed.

4.5. Possible Enhancements

From looking at the above issues it is clear to see where the application could be enhanced to allow a more efficient and accurate solution. If this project was to be enhanced after completion, here are some recommended enhancements:

1) The use of an associative map (using the map.h header file), or a custom built list container to allow individual access to list elements, to represent the waiting list in the algorithm. This could increase the efficiency of the code by making the iterative searches through the lists irrelevant.

2) The use of a standard structure instead of using the OO approach could make accessing the data stored within the objects and elements quicker as selector and mutator methods might not be necessary. This in turn could lead to a faster implementation of the algorithm.

3) An implementation of the Natural Queue Chainmail in 3D. From the implementation of the 2D simulation an adaptation to 3D would need very little extra work. Although with the efficiency of the 2D simulations at present, the 3D simulations would struggle to run in real-time with any object larger than approx. 81 elements.

4) An implementation of a cutting function for Natural Queue Chainmail is also a possible enhancement. This could easily be added to the application by adding a User Interface which allows a user to change tools, from deformation to cutting. When in cutting mode the user would be able to cut through existing links, which would mean appending the list that’s contained in the object which stores the information on the links.

5) An implementation of a possible application is also a possible enhancement for the algorithm. For information on possible applications see section 7.

6) The addition of a dimensional implementation, where, depending on the objects dimensions, the way in which the object is constructed and modeled is altered. This would make the transition from 1D to 2D to 3D much simpler.

By changing the algorithm (as mentioned in point 1 and 2 above), there would also be the
need for a new comparison with the Generalized Chainmail. If the simulation could sustain a larger number of elements at real-time, this would allow a better comparison between the two techniques.

4.6. Conclusion

The aim of this project is to compare the efficiency of the Generalized Chainmail algorithm and the Natural Queue Chainmail Algorithm. For this there will need to be an implementation of both algorithms that can be used side-by-side to compare the visual results of the two techniques. Each will need to be able to model different sized simulations of deformation. Even though the size of the simulations in this project may be restricted, the requirements to carry out the comparison have all been met. This will allow a fair, even efficiency comparison to be carried out and the results will be able to be collected (see section 6.2). The algorithm described in this section is a new alternative to the Generalized Chainmail and the test as to whether it is more efficient can now be carried out. See section 6 for the evaluation of the results of this algorithm and the generalized chainmail algorithm.

5. The Graphical Interface

During the implementation of this project it has been necessary to develop a graphical interface to represent the deformations to an object. This shows the users, visually, how the object deformations take place, how the elements respond to interaction, how the links restrict deformations between two elements in relation to the sponsor and help the user carry out the deformations using interaction from the mouse. The graphical interface was also adapted to help the development process by including a 'debugging mode'. This allowed the user of the application to view the valid regions (named R in the example in section 3.1) for each element's neighbor. See section 5.3 for more details. All the graphics used in this implementation were constructed especially for this project.

5.1. Objects and Elements

When developing the code for the implementation, the use of an OO approach helped when creating the graphical representation of the object and elements as each object and element contained its own draw function that was in control of drawing itself. This allowed each instance of each class to be drawn in the same way depending on the properties of that class.

In an example, using the Object class, the draw function cycles through the list of elements contained in the object, each element is drawn in-turn, along with a link connecting it to each of its neighbors. This gives the graphical image of the entire object, which are all its elements
connected to their neighbors by links (see Fig 5.1a).

As you can see from the above images, the elements are represented as dots and the links are represented as the white lines that connected the elements together. The object is the entire mesh, and is in control of any deformations that are made by the user at any time. Fig 5.1b and 5.1c show two deformations on the object. Fig 5.1b shows the top left element being deformed by the user, this results in a chain reaction causing its neighbors to follow. Fig 5.1c shows the center element being deformed by the user the result is that the elements below are push away and the elements above are dragged along. The restrictions applied by the object mean that the object has two boundary states. One is fully stretched, where each link is at its maximum stretch (see Fig 5.1d). The second is fully compressed, where each link is at its maximum compression (see Fig 5.1e).
5.2. User Interaction

To allow this simulation to be understood, visually, it was always an important issue to allow users to make interactions instinctively. This meant that direct manipulation of the elements in the object was an important choice. The possibility of indirect manipulation was discussed, the use of an element selector and an element deformer interface was the main alternative. As the simulation modeled direct deformations onto the object, it was decided that the direct interaction method was required to fulfill the needs of the application.

The direct interaction method chosen was a way of interacting with the software application, by the user directly choosing the element to deform, usually with the mouse, and then carrying out the deformation with no interval necessary. This allows a 'point, click and drag' interaction to take place. Even novice users should be able to carry out the model deformations without any prior knowledge of the system. Although, there is no text prompts or error messages as part of the program. This could cause problems if the user is unfamiliar with the system.

As the system is implemented using OpenGL to simulate the graphics, the OpenGL control functions were used also, to control the keyboard and mouse inputs. This allowed direct contact with the objects in the simulation and made the input processing much easier.

5.3. Algorithm Graphics and Representations

The graphical implementation of the simulation could be run without the feature described within this section although these tools did aid the development of the project. The addition of this 'debugging mode' allows the user to view a graphical representation of the algorithm, (see sections 2.1 and 3.1).

The debugging mode allows the developer and the user to see how the algorithm works in more detail. Rather that seeing only the deformations of the object it is possible to view how the deformations are calculated in real-time (on small object only). This gives the user and developer a greater understanding of how each deformation take place. See Fig 5.3a to see the simulation with debugging mode turned on.
As you can see from the above image, the center element is selected, each of its four neighbors have the valid region defined. As each neighbor is within their valid region the selected element (sponsor) can move a small amount without affecting any of its neighbors.

When carrying out the efficiency test on the two graphical applications the actual graphics will be altered to allow a faster execution of the simulation. This will involve removing all the debugging tools, removing all the element markers, and leaving just the links. Although this will make interaction more difficult it will be necessary to allow the algorithms to compute quicker with less interference from the graphics of the simulation.

5.4. Problems occurred

During the development of the graphic of the algorithm, there were a few problems that can about. The main problem in the beginning was difficulty in debugging errors from the graphical and algorithmic code. This was solved by the introduction of the 'debugging mode' (see section 5.3). This allowed the use of data analysis to solve problems. If the simulation wasn't acting as expected the debugging mode helped to find out why. The original technique used for this was the use of a command line debugger. During the development of the 1D simulation, this technique worked fine as there wasn't much data to be analyzed. The waiting lists were shorter; there were few elements to analyze. Once the 2D Mesh was in development this technique was almost useless as the size of the output from the simulation was enormous. This is when the graphical debugging mode was suggested.

Another problem with the graphics of the algorithm was the start position of the object. The
initial deformation causes the object to skew into a small random arrangement of elements, then once carrying the deformation further the object reshaped itself back to the original arrangement. This causes problems when starting the simulation and even after rigorous checks of the algorithm, graphics and all source code, the solution to this problem was still not found.

6. Efficiency Evaluation

The entire aim of the project is to evaluate the efficiency of the new Natural Queue Chainmail algorithm. Firstly on stand-alone performance, and how it can handle the interactions of simple deformation modeling using 2D models. Secondly, the performance against the Generalized Chainmail Algorithm, to visually analyze the performances of the two algorithms side by side. This will allow us to decide which approach offers greater efficiency for modeling soft object deformations.

In the first section we discuss the performance of the Natural Queue Chainmail in regards to how it should work. If all the actions and interactions made, to and, by the object in the simulation are feasible. We also mention any short-fallings of the algorithm, and say how this could be avoided in future implementations.

In the second section we compare the Natural Queue Chainmail approach with Generalized Chainmail in a visual efficiency comparison. The comparison will be based entirely on visual output, with no mathematical calculations used to compute algorithms efficiency. The comparison will involve using each implementation of the each technique (Natural Queue Chainmail and Generalized Chainmail) modeling a simple 2D simulation of a rectangular mesh that will be deformed by random user input. In each stage of the comparison the size of the mesh will be increased by the same amount for each simulation. The tests start using a 4x4 Mesh, and will go up in single units (4x4, 5x5, 6x6 etc.). Once either simulation no longer runs in real-time the comparison will end. The simulation that can no longer sustain real-time interaction will be stopped, and the other simulation will carry on until this simulation cannot sustain real-time interaction. The results from this comparison are discussed in sections 6.2 and 6.3.

6.1. The Performance of Natural Queue Chainmail

To test the performance of the Natural Queue Chainmail, the algorithm was created and then a graphical representation of this was implemented to allow a visual simulation to be seen of how the algorithm works. From what is described in the algorithm (see sections 3.1 and 4.1)
the written definition of the algorithm allows us to know how the graphics simulation should act. So therefore we can analyze the performance of the implementation, in regards to what it should be doing.

When the simulation first starts you see the preset, un-deformed object. The first attempt at deformation causes skewing to the object (see section 5.4 and Fig 5.4a). This should not happen but at present the reason for this is unknown. After this initial 'bug' the simulation starts to execute as normal. As each element is deformed, the other elements in the object also deform in the correct chain reaction effect as described in the algorithm. Occasionally some elements will alter position by an incorrect amount causing inaccuracies, but always return to their original position after another deformation. When the relaxation step is removed from the algorithm this error no longer takes place. As the relaxation step, defined in Li and Brodlie 2003, is based around triangular meshes, it is possible that this has some unwanted effects on the rectangular meshes that have been modeled by the simulations in this project. A possibility of looking into a different approach to the relaxation step, or even to use Frisken's 3D Chainmail relaxation step, is a definite requirement for any future advances to this algorithm.

Overall the Natural Queue Chainmail implementation works well and allows easy 2D mesh deformations to be carried out. It also allows an easy transition from the 2D mesh simulations to a 3D mesh simulation with very little modification of the code. The efficiency of the algorithm, against Generalized Chainmail, is discussed in section 6.2.

### 6.2. Efficiency Against Generalized Chainmail

When running the two simulations side by side on smaller simulation it is difficult to see any differences in the way the two approaches handle the deformations. Both can run small simulation (3x3, 4x4, 5x5) 2D meshes in real-time on an Intel P3 650Mhz Laptop with 128Mb RAM using an 8Mb ATI Rage Mobility Graphics Card. Once the number of elements goes beyond 36 elements the speed of the simulation is greatly reduced on this system.

The larger simulation were carried out using an AMD Athlon XP 2.1Ghz PC with 512Mb RAM using a 64MB ATI Radeon 9000 Graphics Card. Using this system it was possible to achieve simulations of up-to 64 elements in size, that was using an 2D 8x8 mesh. Both techniques allowed the 2D 8x8 Mesh to be deformed by a user in real-time. Going much beyond the 8x8 mesh was difficult for both simulations and each one had very little advantage over the other.
The largest simulation that was able to be carried out was a 10x10 mesh, which is 100 elements in size. This simulation was carried out using the Natural Queue Chainmail, and although didn't sustain real-time interaction constantly throughout the test, it did allow deformations to be carried out with a minimum of delays. The Generalized Chainmail implementation allowed 9x9 meshes to run in much the same way. The 81 elements in this mesh could be deformed in near-real-time, but short delays did sometimes interrupt the interaction when large movements were made.

### 6.3. Evaluation Conclusion

With the evaluation of the two approaches complete and the implementations of each simulation been tested it is still difficult to see which algorithm is more efficient for simulating the larger model deformation simulations. In the tests that were carried out on each simulation, there was very little visual difference between either. Even when reaching the the highest meshes tested, the visual results were still similar. The outcome of all the tests showed that the Natural Queue Chainmail could sustain higher mesh sizes, but the margin between Natural Queue Chainmail's maximum size and Generalized Chainmail maximum size was very slim, only 19 elements in size. As the mesh size gets larger it is possible that this difference will become exponentially greater, but without further tests using, either faster computing power, or more efficient code, this cannot be proven.

If the implementation method of both simulation are compared to the original Generalized Chainmail implementation, the level of efficiency is much lower. If the Natural Queue Chainmail was implemented using the same implementation methods as the original Generalized Chainmail's implementation, the visual efficiency of both approaches could be measure with a lot greater accuracy. An alternative to this would be to alter the source code that was used for these simulations to make them more efficient.

### 7. Further Applications

The Natural Queue Chainmail algorithm, along with other deformable modeling techniques, have many possible applications. The structured, physical-based deformation techniques used in the algorithm allow it to be integrated into a number of different graphics applications. Two of the most obvious applications for this are Real Object Modeling and Image Warping. This section discusses how the algorithm could be used in these two areas and why the use of the algorithm would be a good idea. Although it wasn't the purpose of this project to create a possible application for the algorithm, this could have been a possible extension if there had
been more time, or if some of the problems that occurred had taken less time to solve.

7.1. Real Object Modeling
As this algorithm was designed and tested for modeling soft objects, to represent object deformations, an obvious application for the algorithm would be to model soft objects in real world simulations. There are a lot of possible algorithms that can represent soft object deformations (see section 1), the main advantage this algorithm has over other techniques is that it can offer a good level of accuracy, with very little computation power, while deforming the object directly. This application of the algorithm could be used in 3D modeling packages such as 3ds Max or Maya as an object modeling tool.

7.2. Image Warping
This application for the algorithm allows the approach to be used outside the object modeling area of computer graphics. The idea here is to have a normal image, for example as a jpeg, as part of a 2D environment. The user could then apply a custom mesh over the image. The mesh could then be deformed. The image would be attached to the mesh at the element points, and the deformation of the mesh would lead to a deformation of the image. Although this doesn't use the algorithm for the designed purpose, it is a feasible possibility as an application. This application of the algorithm could be used in image design packages (e.g. The GIMP) as an image deformation technique.

8. Conclusion and Future Developments
8.1. Project Conclusion
After completion of the project it is clear to see that, although the development of the simulations did create some delay in the overall development of the project, the important parts of the project (simulation creation, efficiency evaluation and graphical representations) were completed and a result was gained from the project. The outcome of the project is that both algorithms are of similar efficiency and that either approach to the Chainmail technique for soft-object modeling works well. The efficiency of the applications that were used to simulate the model were not efficient enough to carry out large model deformation but this did not stop a visual analysis from being carried out, so didn't hinder getting the results from the process in any way. Although the results from this were not on models as large as at first anticipated, there was still a fair test carried out to gain the results.

8.2. Future Developments
This project created and implemented a 2D version of the Natural Queue Chainmail
Algorithm. From this there is large scope for possible developments to carry on from where this project finishes. The developments can be categorized into two categories. First there are categories that are to do with further study into the efficiency of the Natural Queue Algorithm. The second category is the use of the Natural Queue Chainmail for a possible application.

The completion of this project gained a small insight into the efficiency of Natural Queue Chainmail but as the source code for the implementation wasn't at optimum efficiency, the results were not 100% accurate. This accuracy could be gained by using more efficient source code to build an implementation of Natural Queue Chainmail (see section 4.4).

The simulations built in this project were two dimensional with the purpose of easy adaptation to upper dimensions. Another possible development would be to take this to three dimensions. Or even develop code that could be run using any dimension. This would give a better look into the overall output of the algorithm. Creating a three dimensional model would show if this algorithm works well in three dimensions. As the Generalized Chainmail approach worked well in 3D, it would be assumed that this could be done, but without actual testing, there is no proof of this.

While the efficiency comparison of the algorithm was carried out in this project, the creation of an application for the algorithm was not. This could be done to show how the algorithm could be used in the computer graphics industry. As was mentioned in previous sections, there are a massive number of possible applications for the algorithm and any of these could be implemented to show the algorithms practical use.
Appendix A – Project Reflection

After six to seven months, involving up to 400 hours of work on this project, I have learned many new lessons and gain a number of skills that will help me when trying to progress in the computing industry. Before the project started I was unsure as to what I could achieve, and what result would come out of doing this a large project such as this one on my own. Now that the project is finished I am satisfied with the results that have been return and feel more confident in my own abilities. The progress through the project helped me develop my computing skills. The main areas I feel have been affect greatly by this project are the following:

- **Programming Technique** – As this project was heavily programming based, my technique has improved over the last 6 months..

- **Programming Knowledge** – I gain a large amount of knowledge about programming computer graphics and also some new C++ skills.

- **Planning and Organising Skills.** – After planning the project before the start, keeping to the deadline was a bit of a struggle. But after 3 to 4 months of doing this, it became more natural for me. Also the appending of the deadline to suit any problems encountered was a major issue in this project.

- **Self-Discipline and Self-Motivation** – Keeping to the deadlines was difficult when there is no one else but you relying on the work being done. This takes self-discipline to keep the project going at a regular speed. Also being motivated to do the work can be a big issue, especially when you’ve hit a problem and the solution isn’t clearly seen.

Now that the project is finished I also have gained a respect for team working on such large tasks. If this project had been done in a team, it would have progressed far beyond the scope of the project itself and much more practical work could’ve been carried out.

Overall I feel the projects conclusion is not exactly what was planned at the start of the project, but I feel that the progression through the project has been plagued with development issues. I made allowances from this and although I wanted to carry out some further tasks in the project (such as a 3D model and the implementation of an application) it was decided that this would not be in the best interest of the project as the report writing would suffer.
Appendix B – Project Schedule

<table>
<thead>
<tr>
<th></th>
<th>11/10</th>
<th>11/11</th>
<th>11/12</th>
<th>11/01</th>
<th>11/02</th>
<th>11/03</th>
<th>11/04</th>
<th>11/05</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1D Model</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mid-Project Report</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2D Model</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3D Model</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comparisons and Extensions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final Report</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: Project Schedule

**Research** - Carried out on time, involved reading into other soft object modelling techniques that could be used in replacement of the chainmail algorithms. Also involved reading into the 3D and Generalized Chainmail methods.

**1D/2D/3D Models** – 1D model carried out on time. 2D Model encountered problems, and ran over by a month. 3D Model was not created as the comparisons section was to use the 2D simulations to compare the algorithms. This section involved creating the simulations of the algorithms.

**Mid-Project Report** – Started late, handed in on time. This was the half way milestone for the project before Christmas. This was started later to allow the 2D model to be start early.

**Comparison and Extensions** – The comparison was completed on time, although it did use the 2D model not the 3D model that was previously planned. No extensions to the project were carried out due to lack of time.

**Final Report** – The report was started on time, and finished on time. That is this document.