Interacting with a virtually deformable object using an instrumented glove

Ma Mun Chung

Thesis for the Degree of
MASTER OF PHILOSOPHY

Mechanical and Automation Engineering

Chinese University of Hong Kong

June 1998

Last minute changes

According to the valuable comments from the examiners, changes have been made to the thesis. Four sections have been added to the draft, they are,

a. Section 1.3 – Contribution

b. Section 4.6 – Repeating deformation in different orientation

c. Section 6.3 – An operation example

d. Appendix C – Derivation of **(6.2)**

In addition, an example of deforming a toothpaste with the virtual hand have been added to section 6.3 and a reference ([HUA95]) is added when the use of hand approximation is explained in section 4.4.1.

摘要

此論文描述一個利用儀器手套操控虛擬可變形物件的系統。此系統共分三部分，分別爲物件模型建立、碰撞檢測及資料通訊。

此系統應用有限元分析作爲可變形虛擬物件的線性彈性模型。虛擬物件被分解爲四面體單元。依循有限元分析的程序，虛擬物件因被虛擬手干涉的變形形狀就可計算出來。

球體樹被用作虛擬物件與虛擬手間碰撞檢測的方法，此球體樹爲一棵於不同解像層次逼近物件形狀的八分樹(octree)，而虛擬物件與虛擬手於不同解像層次的碰撞可用圓柱體與球體干涉測試，及球體與球體干涉測試的聯合結果檢測出來。

爲了改善有限元分析的效能，此程序被轉移至一部並行計算機中執行。因應此程序轉移的需要，一資料通訊演算法被導出，作爲主體計算機與並行計算機間的資料互傳工具。

當虛擬手使虛擬物件變形時，此合成系統造出合理的互動反應。

Abstract

A system for interacting with virtually deformable objects by using an instrumented glove is developed. The system is mainly divided into three parts. They are object modelling, collision detection and data communication.

Finite element analysis (FEA) is adopted for modelling the linear elastic behaviour of the deformable virtual object. The virtual objects are discretised into tetrahedral solid elements. The deformed shape of the virtual object can hence be computed when the virtual hand is manipulating the virtual object.

A sphere tree approach is adopted for the collision detection between the virtual object and the virtual hand. The sphere tree is an octree of overlapping spheres that approximates the object shape in different resolution levels. The collision between the virtual hand and the virtual object is detected at different levels by the combined result of cylinder-sphere interference tests and sphere-sphere interference tests.

In order to improve the performance of FEA, this procedure is executed on a parallel computer. A data communication algorithm is developed for data transfer between the host computer and the parallel computer.

The resulting system gives reasonable interactive response when the virtual hand is deforming the virtual object.

Declaration

I hereby declare that the Master of Philosophy thesis titled "Interacting a virtually deformable object using an instrumented glove" represents my own work. I also declare that the work reported in the thesis has not been previously included in a thesis, dissertation or report submitted to this university or to my any other institutions for a degree, diploma or other qualifications.

Ma Mun Chung

Acknowledgements

Thanks God Almighty for creating this universe and giving us Christ Jesus so that I can discover the knowledge given by Him and carry on my research. My all the praises and glory goes to His name.

My sincere appreciation goes to Prof. Hui Kin Chuen, my project supervisor, who guided my research towards the correct direction and provided full academic, spiritual and financial support. He also gives me valuable suggestions and improvements on my work.

Thanks to my thesis committee members, Prof. Y.H. Lui , Prof. H.Q. Sun and Prof. M.M.F. Yuen, for spending their time reading my thesis draft and giving me valuable comments.

I must express my acknowledge to the Chinese University of Hong Kong and the Department of Mechanical and Automation Engineering in providing the financial support and computer facilities to me.

I also thanks Mr. Mr. Lee Sau-leung, Alex, Mr. Yip Hoi-Man, Johnny, Ms. Chan Yuk-kuen, Ms. Djin Kie, Karina, Mr. Mok Wai-kit, Allan, Mr. Lee Yuk-keung, Philip and Mr. Tong Hing for their technical support in the project. Thanks Ms. Kan Yuet-lin, Ms. Chan Miu-ling, Maggie and Ms. Wong Mei-ha, Joyce for their administrative support and solve the coordination problems for me.

Thanks to my colleagues Mr. Lam Pak Chio, Eric, Mr. Yang Chi Tin, Stephen, Dr. Yueng Wai Leung, Mr. Tang Kai Hung, Duncan, Mr. Winston Sun, Mr. Chan Chun Kwong, Joseph, Mr. Fung Wai Keung and Mr. Leung Yun-yee, Martin for their accompany throughout the project

I also want to express my gratitude towards the members in my church for their prayers and encouragements when I found difficulties during the past three years. They are Ms. Chum Yuen Mei, Tomi, Ms. Lai Pui Shan, Ada, Rev. Wong Kwok Yiu, Salias, Mr. Lam Wai Sze, Stanley, Ms. Cheung Suk Ling, Shirley, Mr. Chu Man Chung, Carl, Mr. Kong Wai Kei, Joe, Mr. Tsui Yip Wing, Timothy, Ms. Wong Wai Ling, Elaine, Ms. Chan Suk Yee, Moody, Ms. Lo Suk Ching, Queenie, Mr. Chiu Wai Chor and my cell group members in my fellowship.

List of Figures

List of Tables

Table of Contents

1.Introduction

Modelling of objects in virtual environment is a fundamental problem in virtual reality (VR) applications. Objects in existing VR systems are usually assumed to be rigid bodies. The problem becomes apparent when the user interact with a virtual object. Real objects are deformable to different extend and cannot be assumed rigid when accurate object behaviour is required. Modelling of the deformed shape of a virtual object based on physical laws is thus essential.

1.1. Motivation

Computer aided design (CAD) is one of the computer applications that requires high precision. Existing CAD systems allow engineering analysis and assembly tests to be performed in addition to the visualisation of product shape.

With the rapid development in VR and CAD technology, designers can manipulate virtual products by three-dimensional input devices. However, the behaviour of the virtual product remains different from the real prototype. One improvement is to make the virtual product deformable by applying physical laws to the object since the real products are deformed when subjected to applied force.

There are several methods for incorporating physical laws to deformable virtual objects. Terzopoulos et al [TER87] proposed a method based on an elastic model. Finite difference method was used for calculating the potential energy of deformation. Kang and Kak [KAN96] proposed a hybrid finite element (FE) model using cube element inside the object and plate element on the surface of the object. Gourret et al [GOU89] proposed a method using eight nodes solid element. Unknown displacements were calculated based on given node displacements on the body. Bro-Nielsen and Cotin [BRO96] proposed to use tetrahedron model for discretization of a solid model and using condensation method to calculate the deformation of the object. Rappoport et al [RAP96] proposed a volume-

preserving solid model which determines the deformation based on free-form deformations (FFD).

In order to allow the use of instrumented glove for manipulating virtual object, collision detection and object response have to be considered. Since real-time response is critical in a virtual environment, real-time collision detection and object response is essential. Some time saving algorithms for collision detection thus have to be employed in order to attain interactive response.

Several methods have been developed for collision detection between objects. Bandi and Thalmann [BAN95] used digital differential analyser to adaptively discretise the space into voxels for collision detection. Palmer and Grimsdale [PAL96] used a three stage process for the detection, namely construction of an initial bounding box, construction of a sphere tree hierarchy and polygonal intersection tests. Smith et al [SMI95] used a bounding box octree for approximating potential collision node location. Hubbard [HUB96] used a tightly bounded sphere tree for collision detection. Liu et al [LIU91] used HSM (hierarchical sphere model) for detecting the collision of robot arm and an object. These are all stimulating insight for the collision detection algorithm developed in this thesis.

Most existing VR systems allow the use of instrumented glove without any force-feedback. This makes the interaction with virtual object very difficult. It is difficult to decide whether a virtual hand and a virtual object are in contact simply by visual observation. Evaluation of deformation with the finite element model usually requires force input. As this information is not available with an instrumented glove. The force applying on the object when the virtual hand is in contact with the object is not known. A modification of the problem is thus required.

The evaluation of deformation using the FE method is a computation intensive process. In order to improve the performance of the system, it is desirable to perform deformation computation on a high-speed computer while graphics display can be performed on an

ordinary graphics workstation. It is thus essential to develop algorithms for data transfer between computers.

The objective of this project is to develop a system that allows interaction with virtual objects in real time. Finite element method is adopted for simulating the shape of virtual object. Tetrahedral solid elements are employed for finite element computation.

Interference between the virtual hand and the virtual object in the virtual space is evaluated by approximating the virtual object with a hierarchy of overlapping spheres. Virtual fingers are approximated as cylinders and spheres. A coarse estimate of any possible interference is obtained by performing cylinder-sphere interference test and sphere-sphere interference test. Exact interference test is performed at the leaf node. The number of interference tests is thus reduced and interactive performance can be achieved.

A major time lag affecting the performance of the system is the time required for the FE analysis. In order to improve the performance of the system, the computation of deformation is performed on a parallel machine (ONYX) while the user interacts with the virtual object through a graphics workstation (an Indigo2).

1.2. Thesis Roadmap

In chapter 2, the system architecture used in this project will be introduced. The glove system, the VR system and the tracker system will be introduced individually. The integration of these components into the final system will be described briefly.

In chapter 3, the deformable model used in the system will be described. Previous works on deformable body animation are described in the first part. The virtual object model and the force-displacement relations will be introduced. The method for evaluating model displacement will be presented.

In chapter 4, the collision detection problem will be addressed. Previous works on collision detection algorithm are described in the first part, followed by a description of the technique. The method for constructing the tree structure and search method will be

introduced. Lastly, the grasping mode for attaching the virtual object to the virtual hand and the condition for performing finite element analysis will be described.

In chapter 5, the method for enhancing the speed of finite element analysis will be discussed. The algorithms for data communication and the parallel tools for improving the efficiency of the algorithms will be described.

The test results of the system will be described in chapter 6. The menu functions that control the program flow are described. Then, the hand interaction with virtual objects are described. Finally, the performance of parallel algorithm is discussed with experimental results.

In chapter 7, the conclusion of the whole project will be drawn and some suggestions for future work will be discussed.

1.3. Contribution

The contributions of this thesis are,

1. A workable system for interacting virtual deformable object using an instrumented glove is developed. In the system, the virtual objects can be deformed repeatedly by the virtual hand in different orientations.

2. Cylinders and spheres are used in the approximation of the virtual hand instead of spheres on the joints in the existing system. The whole finger segment instead of the joints can be detected in the collision detection.

3. Parallel algorithm is employed for the improvement of speed of finite element analysis. A study of the effect of parallel algorithm is presented in the thesis and estimation based on the results are given.

2 System Architecture

The system for interaction is composed of two sub-systems. The tracker system provides spatial information of the physical hand. The glove system collects and digitises information of joint angles at the finger joints. By collecting these information in Indigo2 workstation, the exact position, orientation and gesture of the physical hand in three-dimensional space is obtained. A virtual hand model can thus be constructed in the virtual world.

2.1 Tracker system

The Polhemus™ FASTRAK tracking system is used to estimate the three-dimensional position and orientation of the glove [POL94]. The tracking system uses electromagnetic fields to determine the position and orientation of a remote object, i.e., the glove in the current system. The system generates a low frequency magnetic field by a transmitter (Xmtr). The system detects the field vectors with a receiver (Recvr). The signals are then used to compute the receiver's position and orientation relative to the transmitter.

The FASTRAK system consists of a spatial information processing unit (SIPU), a receiver and a transmitter.

2.1.1 Spatial Information Processing Unit (SIPU)

SIPU is the central part of the tracker system. This is a printed circuit board that controls all the I/O of the system. The receiver input, transmitter input and power input receptacles are located on the SIPU. The processing unit provides switches for selecting the customised mode of operation for the system. Receiver selector switch, and I/O configuration switches are located on the SIPU. Please refer to the manual [POL94] for hardware detail of SPIU.

2.1.2 Transmitter (Xmtr)

The resolution of the transmitter is 0.0005 cm, and 0.025°. The instrument provides the specified accuracy when the receivers are located within 76cm of the Transmitter. Operation up to 305cm is possible with reduced accuracy.

2.1.3 Receiver (Recvr)

The static accuracy of the receiver of the system is 0.08cm RMS (root mean square value) for X, Y, and Z Receiver position, and 0.24cm RMS for receiver orientation. There is a latency of 4.0ms from centre of receiver measurement period to beginning of transfer from output port.

2.2 Glove System

In the glove system, there are 22 sensors for measuring the gesture of the physical hand. The sensors send signals to the CyberGlove Interface Unit (CGIU) that amplifies and digitises the signal of the sensors. Then, CGIU sends the digitised information to the host computer via an RS-232 interface. The computer then calculates the angle of flexure of each finger joints that will be used for the display of the virtual hand based on the hand model [VIR94].

2.2.1 CyberGlove Interface Unit (CGIU)

The CyberGlove Interface Unit (CGIU) houses amplification and digitisation circuitry for the CyberGlove. CyberGlove sensor values are amplified and digitised to data of 8-bit resolution inside CGIU. The digitised data are polled by the host, or is directed to the host automatically if a continuous sampling mode is set by a command from the host. For example, to request a single record of 22 sensor values, the host computer sends an ASCII 'G' (which stands for Glove data). All sensor values are then digitised in the CGIU and sent back to the host. The default protocol for the record returned in response to a 'G' command is: first a 'G' is echoed, followed by 22 joint data bytes, followed finally by a trailing NULL character ('/0').

A NULL character that appears in a data stream signifies termination of information. The host computer may use the trailing NULL to verify that all data was properly received. The CyberGlove sensor values are truncated at their lower limits to a value of one.

2.2.2 Bend Sensors

There are 22 sensors in the CyberGlove to capture the motions of the physical hand and finger. The sensors are located over or near the joints of the hand and wrist. The

CyberGlove is designed to best fit an average-sized hand. A property of the CyberGlove's sensor design is that whenever the sensor completely covers the arc of the joint between adjacent bone segments, the sensor will provide an output proportional to the angle between the bones, independent of where the sensor lies relative to the joint and the joint radius.

There are three bend sensors on each of the five fingers on the glove. On the thumb, there are two sensors that measure the metacarpophalangeal joint and interphalangeal joint (MPJ and IJ) (i.e., the outer two thumb joints). On the remaining four fingers, there are three bend sensors to measure the MPJ, proximal IJ (PIJ) and distal IJ (DIJ). The joints are defined as follows:

MPJ = Metacarpophalangeal Joint. This is the joint where the finger connects to the palm.

PIJ = Proximal Interphalangeal Joint. This is the next joint towards the fingertip from the MPJ.

DIJ = Distal Interphalangeal Joint. This is the outermost joint, i.e. nearest to the fingertip.

TMJ = Trapeziometacarpal Joint. This is the joint where the thumb connects to the palm.

Abduction sensors are provided for the thumb, the middle-index, ring-middle and pinkie-ring fingers. These are the horseshoe-shaped ridges sticking up from the surface of the glove. An abduction sensor measures the amount that the corresponding finger moves laterally in the plane of the palm.

The thumb has an additional sensor that measures how much the thumb rotates across the palm toward the pinkie finger. Similarly, the pinkie has a sensor that measures how much the pinkie rotates across the palm toward the thumb, i.e., the arch of the palm near the pinkie finger when the hand is cupped. Finally, there are two wrist sensors, one to measure wrist pitch and one to measure wrist yaw.

The output voltage of each sensor varies with the change in bend angle so that there is no loss of resolution at joint extremes. The linear conversion from digitised output (0-255) to degree is accomplished in the VirtualHand software using linear equation with

two parameters, a gain (slope) and an offset (y-intercept). Details can be found in [VIR94].

2.3 *Integrating the tracker and glove system*

The tracker and glove systems are integrated to give a system with position sensing and finger gesture sensing. The two systems are linked together by using the VirtualHand software a workstation. A virtual hand model is built which can be used to interact with the virtual environment displayed on the workstation screen. Calibration of the virtual hand is required to obtain proper alignment between the physical hand and the virtual hand.

A Silicon Graphics Indigo2 workstation with one R-4400 processor and 128MB memory is used for the integration.

2.3.1 System layout

The layout of the system is shown in **Figure 2.1**.

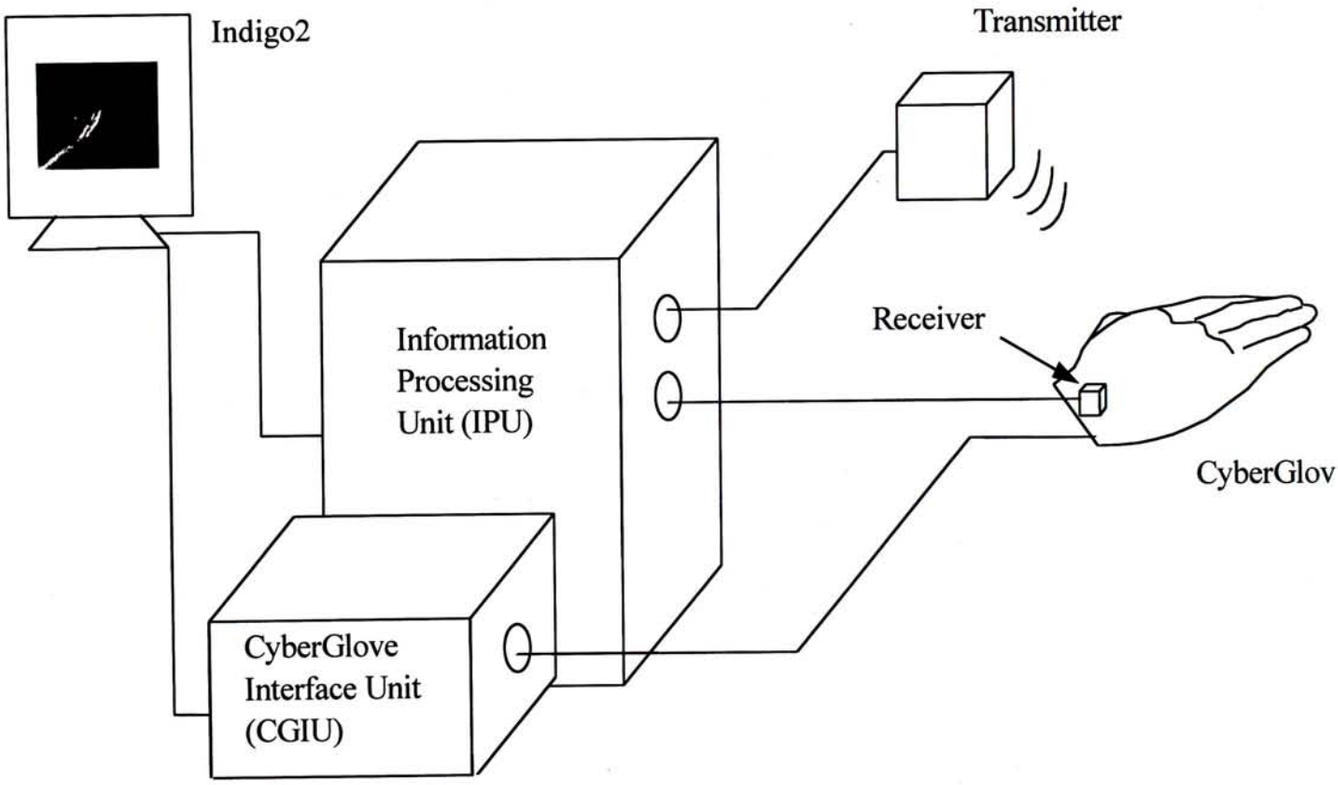


Figure 2.1: Hardware connection of the glove system

The position and orientation of the glove is detected by the three dimensional tracking system located at the wrist position of the glove. The receiver detects magnetic field vectors sent by the transmitter. The transmitter generates a low frequency magnetic field. The signal received is directed to the information-processing unit (IPU). The IPU interprets the signal received and calculate the three-dimensional position and orientation of the glove relative to the transmitter. The position and orientation values are then sent to the computer for the display of the virtual hand.

The relative positions of the fingers are determined by the joint angles between each joint of the fingers since the geometric model of the fingers is known. The sensors are responsible for determining the joint angles between the various sections of the fingers.

The digitised signal is passed to the Indigo2 workstation where the joint angles are calculated and the virtual hand is constructed and displayed. The position and orientation of the virtual hand thus follow the actual movement of the glove, allowing the hand to interact with the virtual environment, as shown in **Figure 2.2**.

Figure 2.2: The resulting hand in the virtual world

3 Deformable Model

The shape of virtual objects in the proposed system can be changed subjected to force applied by the virtual hand. The relation between the shape of the virtual object and the applied force is defined by a deformable model. Since there is no force feedback device in the current system, the applied force is not known. The applied force thus cannot be used directly in the force-displacement relation. The force-displacement relations have to be modified and a displacement-displacement relation has to be developed so that the displacement of the nodes based on the known displacements can be calculated. As an elastic model is used to model the virtual objects, the displacement-displacement relation is reduced to a system of linear equations. By inverting the relation matrix between the displacements, unknown displacements can be calculated from the known displacements that acted as input of the system. By displaying the deformed shape of the virtual object and the virtual hand, a simulation of the virtual hand deforming a virtual object can be obtained.

3.1 Elastic models in computer

In the past, computer animation usually adopted the key-frame interpolation technique to simulate deformation of objects. Artists produce static key pictures to pre-define the change in shape and motion of the deformable bodies. Animators then create intermediate pictures between key pictures (key-frame) by interpolation so as to produce continuous motion in animation. Using this technique, artists are required to produce large number of key-frames to describe the motions of the objects in the animation. No interaction with the animated figures or objects can be attained using this kind of technique.

The use of physically based model provides a solution to the above problem. Once the natural shape (shape before deformation) is developed, the deformed shape of the object can be calculated using physical laws. This is especially useful for interactive applications because the response of the virtual object and the applied force cannot be predicted.

Among various physical models for describing deformable solid objects, the simplest model is the elastic model. Elastic model contains a system of linear equations containing the force-displacement relation. Therefore, standard techniques for solving linear equations in linear algebra can be applied to develop force-displacement relation. Hence, the displacement of object nodes can be calculated from the input of external forces.

In the literature, there are several methods to model elastic objects for animation that can be used for modelling elastic object in a virtual environment. Terzopoulos et al [TER87] developed an elastic model based on energy approach. The strain energy ε(r) of a deformable solid is given by,

$$\varepsilon(r) = \int_{\Omega} \left\| G - G^0 \right\|_{\alpha}^2 da_1 da_2 da_3 \qquad \textbf{(3.1)}$$

In the relation, G is the metric tensor or first fundamental form of the deformed object defined by the relation,

$$G_{ij}(r(a)) = \frac{\partial r}{\partial a_i} \cdot \frac{\partial r}{\partial a_j} \qquad \textbf{(3.2)}$$

where r(a) is the position of a particle a, and

a is a material point in a body.

G^0 is the fundamental form of the natural, undeformed body.

$\|\bullet\|_{\alpha}$ is a weighted matrix norm.

Ω is the space where the integration takes place.

a_1, a_2, a_3 are the variables of co-ordinate axes.

i, j are the parametric indices of the material point.

The finite difference method is used for the numerical computation. For a deformable surface, the forward and backward cross difference operators are defined as,

$$\begin{aligned} D_{12}^{+}\mathbf{u}[m,n] &= D_{21}^{+}\mathbf{u}[m,n] = D_1^{+} D_2^{+}\mathbf{u}[m,n], \\ D_{12}^{-}\mathbf{u}[m,n] &= D_{21}^{-}\mathbf{u}[m,n] = D_1^{-} D_2^{-}\mathbf{u}[m,n], \end{aligned} \qquad \textbf{(3.3)}$$

where **u**[m,n] is a grid function.

D^{+}_1 is the forward difference operator.

D^{-}_1 is the backward difference operator.

D^{+}_{12} is the forward cross difference operator.

D^{-}_{12} is the backward cross difference operator.

Using the grid function $\mathbf{x}[m,n]$ to represent the continuous counterparts and applying the above difference operators for discretising functions, we have,

$$\begin{aligned} a_{ij}[m,n] &= w^1_{ij}[m,n]\left(D^+_i \mathbf{x}[m,n] \cdot D^+_j \mathbf{x}[m,n] - G^0_{ij}[m,n]\right), \\ b_{ij}[m,n] &= w^2_{ij}[m,n]\left(\mathbf{n}[m,n] \cdot D^{(+)}_{ij} \mathbf{x}[m,n] - B^0_{ij}[m,n,]\right), \end{aligned} \qquad \textbf{(3.4)}$$

where $a_{ij}[m,n]$ and $b_{ij}[m,n]$ are constitutive functions describing the elastic properties of the material,

$w^1_{ij}[m,n]$ and $w^2_{ij}[m,n]$ are the weighting functions describing the deformation resistant of the material,

B^0_{ij} is the natural curvature of the surface,

the (+) superscript indicates that the forward cross difference operator is used when $i \neq j$,

n[m,n] is the surface normal grid function,

The elastic force can then be approximated by,

$$\varepsilon[m,n] = \sum\left[- D^-_i\left(a_{ij} D^+_j \mathbf{x}[m,n]\right) + D^{(-)}_{ij}\left(b_{ij} D^{(+)}_{ij} \mathbf{x}[m,n]\right)\right] \qquad \textbf{(3.5)}$$

By assembling all the equations of elastic forces, the stiffness matrix can be constructed to define the force-displacement relation and the deformed shape of the virtual object can be calculated.

J. P. Gourret et al [GOU89,GOU91] used a finite element model based on the virtual displacement principle to represent deformable balls. Virtual displacement principle states that the internal work resulting from internal stresses is equal to the external work resulting from forces such as gravity, pressure and contacts. All components are the integral over the surface of the ball. After the ball is discretised into elements, these integrals are calculated over the element volume V_e or over element surface S_e that can be stated as follow,

$$
\begin{aligned}
\textit{internal work} &= \int_{V_e} (\textit{strain})_e (\textit{stress})_e dV_e \\
\textit{external work} &= \int_{V_e} (\textit{force per unit volume})_e (\textit{displacement})_e dV_e \\
&\quad + \int_{V_e} (\textit{force per unit surface})_e (\textit{displacement})_e dS_e \qquad \textbf{(3.6)} \\
&\quad + (\textit{concentrated force})_e (\textit{displacement})_e
\end{aligned}
$$

As each component depends on the $(displacement)_e$, each element can be assembled, and the equilibrium equation can be written in the form **KU = R**, which can be viewed as a stiffness matrix **K**, displacement **U** and external force **R**.

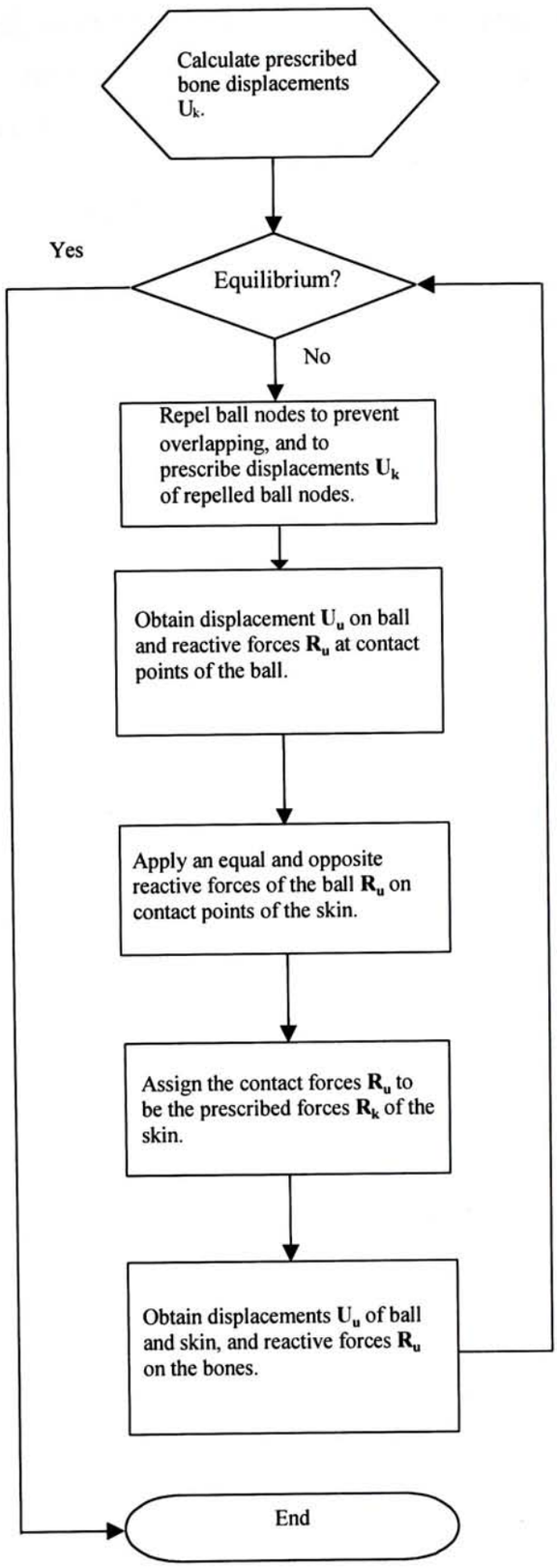


Figure 3.1: Flowchart of contact forces and displacement calculation

To simulate the deformation of the object and the virtual hand, Gourret have developed an iterative method to calculate the deformation of the object based on displacement input, as shown in **Figure 3.1**

In the calculation prior to the iterative process, the displacement of the hand is calculated based on kinematics. The displacements are put into the known displacement vector ($\mathbf{U_k}$) of the hand.

In the iteration loop, the nodes of the ball is firstly repelled against the hand to prevent overlapping of the ball and the hand. The resulting node displacements after repelling are put into the known displacement vector ($\mathbf{U_k}$) of the ball. By partitioning the stiffness matrix and rearranging the linear equations, unknown values, $\mathbf{U_u}$ and $\mathbf{R_u}$, of the ball can be obtained by the equations,

$$\begin{cases} \mathbf{U_u} = \mathbf{K_{11}}^{-1} \cdot (\mathbf{R_k} - \mathbf{K_{12}} \cdot \mathbf{U_k}) \\ \mathbf{R_u} = \mathbf{K_{12}} \cdot \mathbf{U_u} + \mathbf{K_{22}} \cdot \mathbf{U_k} \end{cases} \tag{3.7}$$

Applying the reactive force ($-\mathbf{R_u}$) to the skin, the force becomes known forces $\mathbf{R_k}$ of the hand. Solving the linear equations using known displacement $\mathbf{U_k}$ of the skin obtained by repelling nodes, the node displacements of the skin can be calculated. Afterwards, the reactive force on the skin can be calculated.

The iteration terminates when the reactive forces on the bones are in equilibrium with the forces on the ball.

Kang and Kak [KAN96] used a hybrid model for discretization of the virtual objects and analyse the objects with linear elastic models. For each virtual object model, cubic elements are used in the core of the object and cylindrical shell element layers are used on the surface of the object. A sample meshing for a virtual cylinder is shown in **Figure 3.2**.

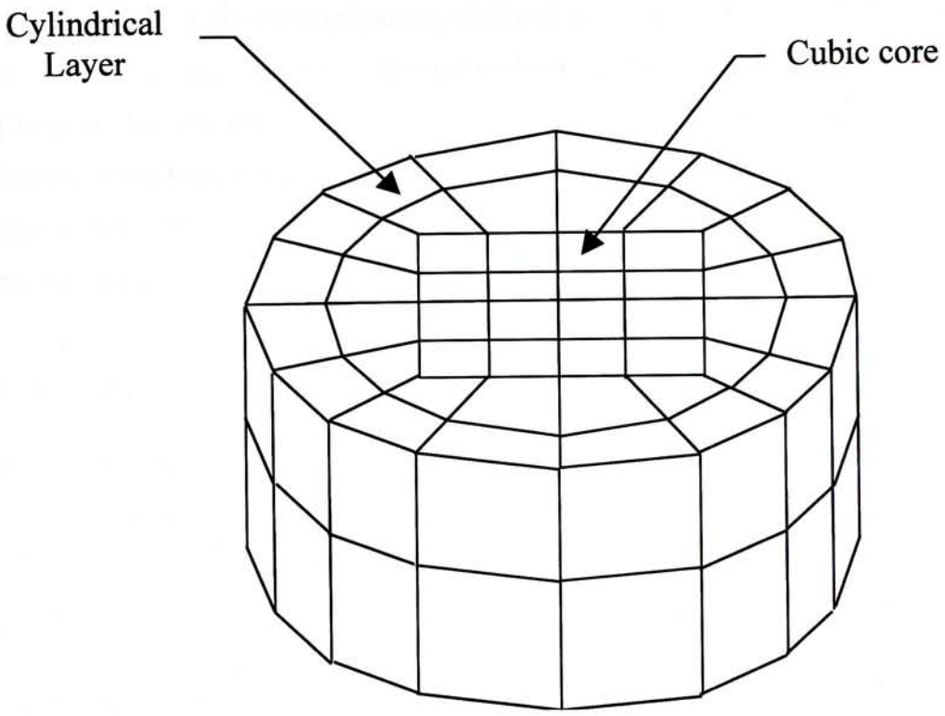


Figure 3.2: Cylindrical Sample of hybrid model

The hybrid model can reduce the time for finite element analysis (FEA) by reducing the number of nodes of a virtual object. For the cubic core, the elements are coarser than the shell elements on the surface of the object. The surface is described by the finer shell elements on the surface of the object. The shape of the object is calculated by performing FEA using plate elements. This helps to produce a finer surface detail with less computational effort.

Rappoport et al [RAP96] addressed the volume preservation problem in free form deformation. In general, solid models should have a constant volume during deformation. Uzawa algorithm is used for non-linear optimisation, with an objective function based on deformation energy. The input of the algorithm include a set of primitives defined with node points, the primitive volume sizes and a set of linear constraints based on continuity of the control points. The resulting algorithm is a control point configuration closest to the given one such that the volume of the solid is conserved while the linear constraints are obeyed. Hence, a solid can be deformed with constant volume while still obeying the physical law of minimum energy within the solid. After careful tuning, the algorithm becomes interactive for manipulations of solid elements.

Bro-Nielsen and Cotin [BRO96] proposed a finite element based approach for real-time deformation applications. The approach uses volumetric solid elements for discretization

of the virtual objects. This discretization method computes the 3-dimensional volumetric behaviours of a real solid object. Condensation method borrowed from Gourret et al [GOU89] is used for the calculation of the object nodes displacements and the unknown external forces. This method produces real-time simulation of solid deformation based on linear elastic finite elements using a Silicon Graphics ONYX computer for analysis, and the SGI Performer graphics library for display.

3.2 *Virtual object model*

In the proposed system, linear elastic model is used for simulating the deformable behaviour of virtual objects by finite element method (FEM). Tetrahedral solid elements are used for analysing the relationship between the external forces and the nodal displacement of the object.

A tetrahedral solid element is shown in **Figure 3.3**.

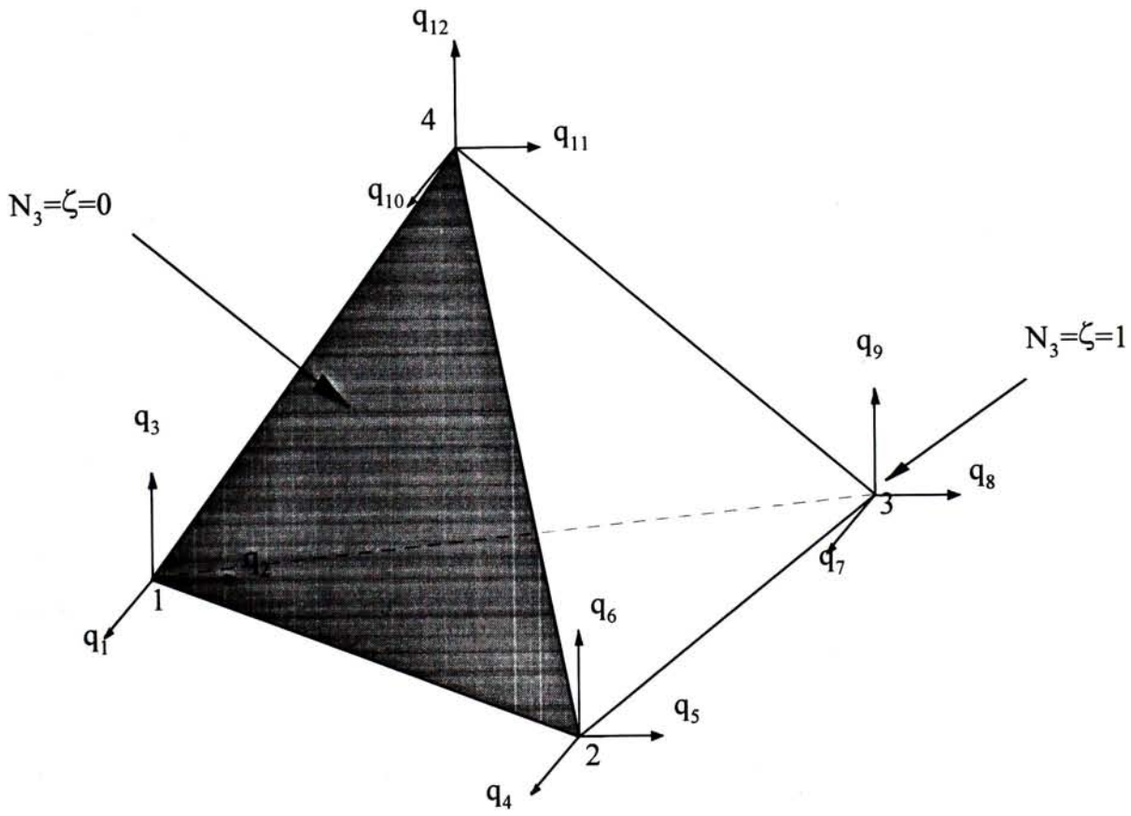


Figure 3.3: Tetrahedral element

In the figure, q_1 to q_{12} are the displacements of the respective nodes, q_1 to q_3 are the displacement of node 1 along the x, y, z-axis respectively. There are four shape functions, namely, N_1 to N_4, defined as follow,

$$N_1 = \xi \quad N_2 = \eta \quad N_3 = \zeta \quad N_4 = 1 - \xi - \eta - \zeta \tag{3.8}$$

The value of the shape function (N_i) is equal to 1 at the vertex (vertex i) and is equal to 0 on the opposite face of the respective vertex, as shown in **Figure 3.3**. For example, the value of N_3 is equal to 1 at node 3 and is equal to 0 at the opposite face (the shaded face in **Figure 3.3**) of node 3. Since the opposite face of node 3 contains nodes 1, 2, and 4, the value of N_3 is equal to zero at nodes 1, 2 and 4. In other words, N_3 is zero at the other nodes of the tetrahedral element.

3.3 Force displacement relationship

The force displacement relationship defines the relation between the external applied force to the nodal displacement of the virtual object. Since the current model is a linear elastic model, the relationship between the applied forces and the displacement of object nodes are linear. Hence, a system of linear equations can be set up for solving the unknown nodal displacements based on the applied force. The coefficients of the linear equations can be summarised into a matrix called stiffness matrix. This relationship is applicable to each element of the virtual object.

3.3.1 Stress-strain relationship

Based on the three dimensional elastic model, there are two parameters for measuring the material properties for isotropic material. They are modulus of elasticity (E) and Poisson's ratio (ν) [CHA91]. Considering an elemental cube inside a body, strains can be expressed in terms of stresses by Hooke's law,

$$\varepsilon_x = \frac{\sigma_x}{E} - \nu\frac{\sigma_y}{E} - \nu\frac{\sigma_z}{E}$$

$$\varepsilon_y = -\nu\frac{\sigma_x}{E} + \frac{\sigma_y}{E} - \nu\frac{\sigma_z}{E}$$

$$\varepsilon_z = -\nu\frac{\sigma_y}{E} - \nu\frac{\sigma_y}{E} + \frac{\sigma_z}{E}$$

$$\gamma_{xy} = \frac{\tau_{xy}}{G} \qquad \textbf{(3.9)}$$

$$\gamma_{yz} = \frac{\tau_{yz}}{G}$$

$$\gamma_{xz} = \frac{\tau_{xz}}{G}$$

where the modulus of rigidity, G is given by,

$$G = \frac{E}{2(1+\nu)} \qquad \textbf{(3.10)}$$

ε_x, ε_y, ε_z and σ_x, σ_y, σ_z are nominal strain and stress of the element in nominal x, y and z direction respectively, and

γ_{xy}, γ_{yz}, γ_{xz} and τ_{xy}, τ_{yz}, τ_{xz} are the shear strain and stress of the element respectively.

Adding all the nominal strains gives the following relation,

$$\varepsilon_x + \varepsilon_y + \varepsilon_z = \frac{(1-2\nu)}{E}(\sigma_x + \sigma_y + \sigma_z) \qquad \textbf{(3.11)}$$

Using this relation to solve for the stresses, the following relation is obtained,

$$\sigma = \mathbf{D}\varepsilon \qquad \textbf{(3.12)}$$

where σ is the stress applying on the element given by

$$\sigma = \left[\sigma_x \quad \sigma_y \quad \sigma_z \quad \tau_{xy} \quad \tau_{yz} \quad \tau_{xz}\right]^T, \qquad \textbf{(3.13)}$$

ε is the strain of the element given by

$$\varepsilon = \left[\varepsilon_x \quad \varepsilon_y \quad \varepsilon_z \quad \gamma_{xy} \quad \gamma_{yz} \quad \gamma_{xz}\right]^T, \qquad \textbf{(3.14)}$$

D is a (6×6) symmetric matrix given by,

$$\mathbf{D}=\frac{E}{(1+\nu)(1-2\nu)}\begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5-\nu & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5-\nu & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5-\nu \end{bmatrix} \tag{3.15}$$

3.3.2 Stiffness matrix formulation

The strain-displacement relations is given by,

$$\varepsilon=\left[\frac{\partial u}{\partial x}\quad \frac{\partial v}{\partial y}\quad \frac{\partial w}{\partial z}\quad \frac{\partial v}{\partial z}+\frac{\partial w}{\partial y}\quad \frac{\partial u}{\partial z}+\frac{\partial w}{\partial x}\quad \frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right]^{T} \tag{3.16}$$

where ε is the strain of the element, and

u, v, w are the displacements in x, y, z directions respectively.

Using **(3.8)**, the displacement $\mathbf{u}=[u\ v\ w]^{\mathrm{T}}$ at arbitrary position $\mathbf{x} = [\mathrm{x\ y\ z}]^{\mathrm{T}}$ within the element can be written in terms of the unknown nodal values as,

$$\mathbf{u}=\mathbf{Nq} \tag{3.17}$$

where $$\mathbf{N}=\begin{bmatrix} N_1 & 0 & 0 & N_2 & 0 & 0 & N_3 & 0 & 0 & N_4 & 0 & 0 \\ 0 & N_1 & 0 & 0 & N_2 & 0 & 0 & N_3 & 0 & 0 & N_4 & 0 \\ 0 & 0 & N_1 & 0 & 0 & N_2 & 0 & 0 & N_3 & 0 & 0 & N_4 \end{bmatrix}, \tag{3.18}$$

q is the local displacement vector of the nodes given by

$$\mathbf{q}=[q_1\quad q_2\quad q_3\quad q_4\quad q_5\quad q_6\quad q_7\quad q_8\quad q_9\quad q_{10}\quad q_{11}\quad q_{12}]^{T}, \tag{3.19}$$

In addition, the arbitrary position **x** at which the displacement **u** is interpolated can be expressed as,

$$\begin{aligned} x &= N_1x_1+N_2x_2+N_3x_3+N_4x_4 \\ y &= N_1y_1+N_2y_2+N_3y_3+N_4y_4 \\ z &= N_1z_1+N_2z_2+N_3z_3+N_4z_4 \end{aligned} \tag{3.20}$$

where x_i, y_i, z_i are the co-ordinates of the ith vertex along x, y and z-axis respectively, for i = 1, 2 , 3, 4.

Substituting **(3.8)** into **(3.20)** and using the notation $x_{ij} = x_i - x_j$, $y_{ij} = y_i - y_j$, $z_{ij} = z_i - z_j$, equation **(3.20)** becomes,

$$\begin{aligned} x &= x_4 + x_{14}\xi + x_{24}\eta + x_{34}\zeta \\ y &= y_4 + y_{14}\xi + y_{24}\eta + y_{34}\zeta \\ z &= z_4 + z_{14}\xi + z_{24}\eta + z_{34}\zeta \end{aligned} \tag{3.21}$$

Using chain rule for partial derivatives, the relationship for a displacement derivatives with respect to different basis can be obtained, take u as an example, the relation is,

$$\begin{Bmatrix} \frac{\partial u}{\partial \xi} \\ \frac{\partial u}{\partial \eta} \\ \frac{\partial u}{\partial \zeta} \end{Bmatrix} = \mathbf{J} \begin{Bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \\ \frac{\partial u}{\partial z} \end{Bmatrix} \tag{3.22}$$

where **J** is the Jacobian of the transformation, which is given by,

$$\mathbf{J} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix} = \begin{bmatrix} x_{14} & y_{14} & z_{14} \\ x_{24} & y_{24} & z_{24} \\ x_{34} & y_{34} & z_{34} \end{bmatrix} \tag{3.23}$$

Hence, the following relation can be found,

$$\begin{Bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \\ \frac{\partial u}{\partial z} \end{Bmatrix} = A \begin{Bmatrix} \frac{\partial u}{\partial \xi} \\ \frac{\partial u}{\partial \eta} \\ \frac{\partial u}{\partial \zeta} \end{Bmatrix} \tag{3.24}$$

where **A** is the inverse of the Jacobian **J**, which is given by,

$$\mathbf{A} = \mathbf{J}^{-1} = \frac{1}{\det \mathbf{J}} \begin{bmatrix} y_{24}z_{34} - y_{34}z_{24} & y_{34}z_{14} - y_{14}z_{34} & y_{14}z_{24} - y_{24}z_{14} \\ z_{24}x_{34} - z_{34}x_{24} & z_{34}x_{14} - z_{14}x_{34} & z_{14}x_{24} - z_{24}x_{14} \\ x_{24}y_{34} - x_{34}y_{24} & x_{34}y_{14} - x_{14}y_{34} & x_{14}y_{24} - x_{24}y_{14} \end{bmatrix} \tag{3.25}$$

where det **J** is the determinant of Jacobean **J**.

Combining **(3.16)**, **(3.17)** and **(3.24)** the relation between strain and displacement is obtained, take $\frac{\partial u}{\partial x}$ as an example, from **(3.24)**,

$$\frac{\partial u}{\partial x} = A_{11}\frac{\partial u}{\partial \xi} + A_{12}\frac{\partial u}{\partial \eta} + A_{13}\frac{\partial u}{\partial \zeta} \tag{3.26}$$

where A_{ij} is a element of matrix **A** in the *i*th row and the *j*th column.

From **(3.17)**,

$$\begin{aligned} u &= N_1 q_1 + N_2 q_4 + N_3 q_7 + N_4 q_{10} \\ &= \xi q_1 + \eta q_4 + \zeta q_7 + (1 - \xi - \eta - \zeta) q_{10} \end{aligned} \tag{3.27}$$

where q_i is the *i*th element in the local displacement vector **q** of the tetrahedral element.

Hence, **(3.26)** becomes,

$$\begin{aligned} \frac{\partial u}{\partial x} &= A_{11}(q_1 - q_{10}) + A_{12}(q_4 - q_{10}) + A_{13}(q_7 - q_{10}) \\ &= A_{11}q_1 + A_{12}q_4 + A_{13}q_7 - (A_{11} + A_{12} + A_{13})q_{10} \\ &= A_{11}q_1 + A_{12}q_4 + A_{13}q_7 - \tilde{A}_1 q_{10} \end{aligned} \tag{3.28}$$

where $\tilde{A}_1 = A_{11} + A_{12} + A_{13}$.

Similarly, other equations can be obtained and the strain-displacement relation becomes,

$$\varepsilon = \mathbf{B}\mathbf{q} \tag{3.29}$$

where **B** is a (6×12) matrix given by,

$$\mathbf{B}=\begin{bmatrix} A_{11} & 0 & 0 & A_{12} & 0 & 0 & A_{13} & 0 & 0 & -\widetilde{A}_1 & 0 & 0 \\ 0 & A_{21} & 0 & 0 & A_{22} & 0 & 0 & A_{23} & 0 & 0 & -\widetilde{A}_2 & 0 \\ 0 & 0 & A_{31} & 0 & 0 & A_{32} & 0 & 0 & A_{33} & 0 & 0 & -\widetilde{A}_3 \\ 0 & A_{31} & A_{21} & 0 & A_{32} & A_{22} & 0 & A_{33} & A_{23} & 0 & -\widetilde{A}_3 & -\widetilde{A}_2 \\ A_{31} & 0 & A_{11} & A_{32} & 0 & A_{12} & A_{33} & 0 & A_{13} & -\widetilde{A}_3 & 0 & -\widetilde{A}_1 \\ A_{21} & A_{11} & 0 & A_{22} & A_{12} & 0 & A_{23} & A_{13} & 0 & -\widetilde{A}_2 & -\widetilde{A}_1 & 0 \end{bmatrix} \quad \textbf{(3.30)}$$

where $\widetilde{A}_1 = A_{11} + A_{12} + A_{13}$, $\widetilde{A}_2 = A_{21} + A_{22} + A_{23}$, $\widetilde{A}_3 = A_{31} + A_{32} + A_{33}$.

The element strain energy is given by,

$$\begin{aligned} \mathbf{U}_e &= \frac{1}{2}\int_e \varepsilon^{\mathrm{T}}\mathbf{D}\varepsilon dV \\ &= \frac{1}{2}\mathbf{q}^{\mathrm{T}}\mathbf{B}^{\mathrm{T}}\mathbf{D}\mathbf{B}\mathbf{q}\int_e dV \\ &= \frac{1}{2}\mathbf{q}^{\mathrm{T}}V_e\mathbf{B}^{\mathrm{T}}\mathbf{D}\mathbf{B}\mathbf{q} \\ &= \frac{1}{2}\mathbf{q}^{\mathrm{T}}\mathbf{k}^{\mathbf{e}}\mathbf{q} \end{aligned} \quad \textbf{(3.31)}$$

where the element stiffness matrix $\mathbf{k}^{\mathbf{e}}$ is given by,

$$\mathbf{k}^{\mathbf{e}} = V_e\mathbf{B}^{\mathrm{T}}\mathbf{D}\mathbf{B} \quad \textbf{(3.32)}$$

V_e is the volume of the element given by

$$\frac{1}{6}|\det\mathbf{J}| \quad \textbf{(3.33)}$$

Using the result of **(3.27),** the force-displacement relation in the element becomes,

$$\mathbf{f}^{\mathbf{e}} = \mathbf{k}^{\mathbf{e}}\mathbf{q}^{\mathbf{e}} \quad \textbf{(3.34)}$$

where $\mathbf{f}^{\mathbf{e}}$ is the force vector describing the force adding on the element given by

$$\mathbf{f}^{\mathbf{e}} = [f_1 \quad f_2 \quad f_3 \quad f_4 \quad f_5 \quad f_6 \quad f_7 \quad f_8 \quad f_9 \quad f_{10} \quad f_{11} \quad f_{12}]^T. \quad \textbf{(3.35)}$$

By assembling the elemental stiffness matrix together, the global force-displacement relation is obtained,

$$\mathbf{f}^{\mathbf{g}} = \mathbf{k}^{\mathbf{g}}\mathbf{q}^{\mathbf{g}} \quad \textbf{(3.36)}$$

where $\mathbf{f}^g$ is the global force vector applied to the object with N nodes, given by,

$$\mathbf{f}^g = \begin{bmatrix} f_0^g & \cdots & f_{3N-1}^g \end{bmatrix} \tag{3.37}$$

$\mathbf{q}^g$ is the global displacement vector of the object, given by,

$$\mathbf{q}^g = \begin{bmatrix} q_0^g & \cdots & q_{3N-1}^g \end{bmatrix}^T . \tag{3.38}$$

$\mathbf{k}^g$ is the global stiffness matrix, given by,

$$\mathbf{k}^g = \sum \mathbf{k}^e . \tag{3.39}$$

The process of assembling of global stiffness matrix is described in 3.5.2.

3.4 Solving the linear system

From **(3.36)**, external forces can be calculated by multiplying the displacements with the global stiffness matrix or the inverse relation can be obtained,

$$\mathbf{q}^g = \left(\mathbf{k}^g\right)^{-1} \mathbf{f}^g \tag{3.40}$$

Unknown nodal displacements can thus be computed from the input of external forces. However, since there is no force feedback devices in the current system, the displacement vector cannot be computed by **(3.40)**.

Since there are some known displacements with unknown forces and there are some unknown displacements with known forces, the complementary relations of forces and displacements make the number of equations sufficient for solving the unknowns. Since the number of equations is equal to the number of unknowns, all the unknowns can be solved by rearranging the system of equations and the order of variables.

By considering the relation between the virtual hand and the virtual object, there is no external applied force at the nodes having no constraint or no contact with the virtual hand. Therefore, all the known external forces are zero.

After the rearrangement, the relation between displacements and forces is given by relation **(3.41),**

$$\begin{bmatrix} \mathbf{0} \\ \mathbf{F}_u^g \end{bmatrix} = \begin{bmatrix} \mathbf{K}_{11}^g & \mathbf{K}_{12}^g \\ \mathbf{K}_{21}^g & \mathbf{K}_{22}^g \end{bmatrix} \begin{bmatrix} \mathbf{Q}_u^g \\ \mathbf{Q}_k^g \end{bmatrix} \tag{3.41}$$

where $\mathbf{F}_u^g$ is the unknown force vector, $\mathbf{Q}_u^g$ is the unknown displacement,

$\mathbf{Q}_k^g$ is the known displacement, and

$\mathbf{K}_{11}^g$, $\mathbf{K}_{12}^g$, $\mathbf{K}_{21}^g$, $\mathbf{K}_{22}^g$ are the submatrices of the rearranged global stiffness matrix $(\mathbf{k}^g)$.

Multiplying the matrix partitions and the displacement vectors, a system of linear equations can be found,

$$\begin{cases} \mathbf{0} = \mathbf{K}_{11}^g \mathbf{Q}_u^g + \mathbf{K}_{12}^g \mathbf{Q}_k^g & \textbf{(3.42a)} \\ \mathbf{F}_u^g = \mathbf{K}_{21}^g \mathbf{Q}_u^g + \mathbf{K}_{22}^g \mathbf{Q}_k^g & \textbf{(3.42b)} \end{cases}$$

Rearranging equations, **(3.42)** gives,

$$\begin{cases} \mathbf{Q}_u^g = -\left(\mathbf{K}_{11}^g\right)^{-1} \mathbf{K}_{12}^g \mathbf{Q}_k^g & \textbf{(3.43a)} \\ \mathbf{F}_u^g = \mathbf{K}_{21}^g \mathbf{Q}_u^g + \mathbf{K}_{22}^g \mathbf{Q}_k^g & \textbf{(3.43b)} \end{cases}$$

Unknown displacements and forces can hence be found by relation **(3.43)**.

In the current application, it is only required to obtain the nodal displacement. Hence, it is only necessary to compute the unknown displacement in **(3.43a)**. In addition, $\mathbf{K}_{12}^g$ is known after the rearrangement of the equations. Only $(\mathbf{K}_{11}^g)^{-1}$ is required to compute and the unknown displacements. (The process of computing the matrix inverse is described in Appendix A)

The rearrangement improves the computation efficiency in two ways. The size of $\mathbf{K}_{11}^g$ is smaller than $\mathbf{k}^g$ so that the time that is required for computing inverse is reduced. The size of $(\mathbf{K}_{11}^g)^{-1}\mathbf{K}_{12}^g$ is smaller comparing with $(\mathbf{k}^g)^{-1}$ in **(3.40)** so that the computation of unknown displacement becomes more efficient.

3.5 Implementation

There are three problems in the implementation of the modelling algorithm. The first problem is the development of a special data structure for providing efficient data extraction. The second problem is the process of global stiffness matrix formulation and the assembly of global stiffness matrix. The third problem is how to re-assemble the nodal displacement vector so that the result of modelling virtual object can be used for displaying the deformed object.

3.5.1 Data structure

The data structure for the virtual objects is specially designed for the ease of retrieving relevant information for the analysis of the virtual object. There are three lists under the data structure of a virtual object. They are summarised in **Table 3.1**.

List no.	Data type	Description
1	(N×3) floating points	Vertex co-ordinates
2	(M×4) integers	Element nodes
3	(P×3) integers	Display list

Table 3.1: Data structure of a virtual object

The first list is the vertices of the object. All the virtual objects in the system are polyhedral objects. This facilitates the modelling and rendering of the virtual object in the virtual environment. For each vertex, there is a 3D-array that contains the x, y and z local co-ordinate of the object. Therefore, the first list is an N by 3 array of floating points where N is the number of vertices in the object.

The second list describes the elements of the virtual object. As tetrahedral elements are used, there are four vertices for each element. The indices to the corresponding vertices are represented by four integers. Therefore, the second list is an M by 4 array of integers, where M is the number of tetrahedral elements in the virtual object.

The third list is the display list of the object. The list is to be used for rendering and displaying the virtual object. The surfaces of the virtual objects in the system are triangular facets. The three vertices of a facets have to be specified in counter-clockwise order so as to give a consistent surface normal for rendering. Therefore, the third list is a P by 3 array of integers, where P is the number of faces to be displayed.

3.5.2 Global stiffness matrix formulation

From **(3.34)**, the force-displacement relation in element j is given by,

$$\mathbf{f}^e = \mathbf{k}^e \mathbf{q}^e$$

Multiplying the matrix with the vector, a system of linear equations is found.

$$
\begin{aligned}
f_0^e &= K_{0,0}^e q_0^e + K_{0,1}^e q_1^e + \cdots + K_{0,11}^e q_{11}^e \\
f_1^e &= K_{1,0}^e q_0^e + K_{1,1}^e q_1^e + \cdots + K_{1,11}^e q_{11}^e \\
&\vdots \\
f_{11}^e &= K_{11,0}^e q_0^e + K_{11,1}^e q_1^e + \cdots + K_{11,11}^e q_{11}^e
\end{aligned}
\tag{3.44}
$$

Taking $f_0{}^e$ as an example, replacing the indices in the equation with global indices,

$$
f_{3*h(j,1)}^g = K_{3*h(j,1),3*h(j,1)}^g q_{3*h(j,1)}^g + K_{3*h(j,1),3*h(j,1)+1}^g q_{3*h(j,1)+1}^g + \cdots + K_{3*h(j,4)+2}^g q_{3*h(j,4)+2}^g \tag{3.45}
$$

where $h(j,i)$ is the index in the jth row and the ith column of the element list.

(see section 3.5.1 for detail description of the element list)

As there are N nodes in a virtual object, the element force-displacement relation can be expanded to a system of 3N linear equations. Expressing **(3.44)** in matrix form with replaced index like the example in **(3.45)**, the relation becomes,

$$\begin{bmatrix} 0 \\ \vdots \\ 0 \\ f^g_{3*h(j,1)} \\ f^g_{3*h(j,1)+1} \\ f^g_{3*h(j,1)+2} \\ 0 \\ \vdots \\ 0 \\ f^g_{3*h(j,2)} \\ f^g_{3*h(j,2)+1} \\ f^g_{3*h(j,2)+2} \\ 0 \\ \vdots \\ 0 \\ f^g_{3*h(j,3)} \\ f^g_{3*h(j,3)+1} \\ f^g_{3*h(j,3)+2} \\ 0 \\ \vdots \\ 0 \\ f^g_{3*h(j,4)} \\ f^g_{3*h(j,4)+1} \\ f^g_{3*h(j,4)+2} \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & \cdots \\ \vdots & \cdots & \vdots & \vdots & \vdots & \vdots & \cdots & \cdots \\ \vdots & \cdots & \vdots & 0 & 0 & 0 & \cdots & \cdots \\ \vdots & \cdots & \vdots & K^g_{3*h(j,1),3*h(j,1)} & K^g_{3*h(j,1),3*h(j,1)+1} & K^g_{3*h(j,1),3*h(j,1)+2} & \cdots & \cdots \\ \vdots & \cdots & \vdots & K^g_{3*h(j,1)+1,3*h(j,1)} & K^g_{3*h(j,1)+1,3*h(j,1)+1} & K^g_{3*h(j,1)+1,3*h(j,1)+2} & \cdots & \cdots \\ \vdots & \cdots & \vdots & K^g_{3*h(j,1)+2,3*h(j,1)} & K^g_{3*h(j,1)+2,3*h(j,1)+1} & K^g_{3*h(j,1)+2,3*h(j,1)+2} & \cdots & \cdots \\ \vdots & \cdots & \vdots & 0 & 0 & 0 & \cdots & \cdots \\ \vdots & \cdots & \vdots & \vdots & \vdots & \vdots & \cdots & \cdots \\ \vdots & \cdots & \vdots & 0 & 0 & 0 & \cdots & \cdots \\ \vdots & \cdots & \vdots & K^g_{3*h(j,2),3*h(j,1)} & K^g_{3*h(j,2),3*h(j,1)+1} & K^g_{3*h(j,2),3*h(j,1)+2} & \cdots & \cdots \\ \vdots & \cdots & \vdots & K^g_{3*h(j,2)+1,3*h(j,1)} & K^g_{3*h(j,2)+1,3*h(j,1)+1} & K^g_{3*h(j,2)+1,3*h(j,1)+2} & \cdots & \cdots \\ \vdots & \cdots & \vdots & K^g_{3*h(j,2)+2,3*h(j,1)} & K^g_{3*h(j,2)+2,3*h(j,1)+1} & K^g_{3*h(j,2)+2,3*h(j,1)+2} & \cdots & \cdots \\ \vdots & \cdots & \vdots & 0 & 0 & 0 & \cdots & \cdots \\ \vdots & \cdots & \vdots & \vdots & \vdots & \vdots & \cdots & \cdots \\ \vdots & \cdots & \vdots & 0 & 0 & 0 & \cdots & \cdots \\ \vdots & \cdots & \vdots & K^g_{3*h(j,3),3*h(j,1)} & K^g_{3*h(j,3),3*h(j,1)+1} & K^g_{3*h(j,3),3*h(j,1)+2} & \cdots & \cdots \\ \vdots & \cdots & \vdots & K^g_{3*h(j,3)+1,3*h(j,1)} & K^g_{3*h(j,3)+1,3*h(j,1)+1} & K^g_{3*h(j,3)+1,3*h(j,1)+2} & \cdots & \cdots \\ \vdots & \cdots & \vdots & K^g_{3*h(j,3)+2,3*h(j,1)} & K^g_{3*h(j,3)+2,3*h(j,1)+1} & K^g_{3*h(j,3)+2,3*h(j,1)+2} & \cdots & \cdots \\ \vdots & \cdots & \vdots & 0 & 0 & 0 & \cdots & \cdots \\ \vdots & \cdots & \vdots & \vdots & \vdots & \vdots & \cdots & \cdots \\ \vdots & \cdots & \vdots & 0 & 0 & 0 & \cdots & \cdots \\ \vdots & \cdots & \vdots & K^g_{3*h(j,4),3*h(j,1)} & K^g_{3*h(j,4),3*h(j,1)+1} & K^g_{3*h(j,4),3*h(j,1)+2} & \cdots & \cdots \\ \vdots & \cdots & \vdots & K^g_{3*h(j,4)+1,3*h(j,1)} & K^g_{3*h(j,4)+1,3*h(j,1)+1} & K^g_{3*h(j,4)+1,3*h(j,1)+2} & \cdots & \cdots \\ \vdots & \cdots & \vdots & K^g_{3*h(j,4)+2,3*h(j,1)} & K^g_{3*h(j,4)+2,3*h(j,1)+1} & K^g_{3*h(j,4)+2,3*h(j,1)+2} & \cdots & \cdots \\ \vdots & \cdots & \vdots & 0 & 0 & 0 & \cdots & \cdots \\ \vdots & \cdots & \vdots & \vdots & \vdots & \vdots & \cdots & \cdots \\ 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & \cdots \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ q^g_{3*h(j,1)1} \\ q^g_{3*h(j,1)+1} \\ q^g_{3*h(j,1)+2} \\ 0 \\ \vdots \\ 0 \\ q^g_{3*h(j,2)} \\ q^g_{3*h(j,2)+1} \\ q^g_{3*h(j,2)+2} \\ 0 \\ \vdots \\ 0 \\ q^g_{3*h(j,3)} \\ q^g_{3*h(j,3)+1} \\ q^g_{3*h(j,3)+2} \\ 0 \\ \vdots \\ 0 \\ q^g_{3*h(j,4)} \\ q^g_{3*h(j,4)+1} \\ q^g_{3*h(j,4)+2} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \textbf{(3.46)}$$

Adding all the element force-displacement relations together, the global force-displacement relation results,

$$\begin{bmatrix} F^g_0 \\ \vdots \\ F^g_{3N-1} \end{bmatrix} = \begin{bmatrix} K^g_{0,0} & \cdots & K^g_{0,3N-1} \\ \vdots & \ddots & \vdots \\ K^g_{3N-1,0} & \cdots & K^g_{3N-1,3N-1} \end{bmatrix} \begin{bmatrix} Q^g_0 \\ \vdots \\ Q^g_{3N-1} \end{bmatrix} \quad \textbf{(3.47)}$$

where $F_i g$ is a global force vector element, **(3.48)**

$K^g_{i,j}$ is an element of the global stiffness matrix, and **(3.49)**

$Q_i{}^g$ is a global displacement vector element. **(3.50)**

(3.36) gives the matrix equation of this relation.

3.5.3 Re-assemble of nodal displacements

In **(3.43)**, all the displacement vectors are column vectors. After the unknown displacement vector is computed, the nodal displacements are computed by re-assembling into an array with three columns. The element displacement vectors are arranged so that three consecutive displacement vectors are in a group. The three components of the displacement vectors of the *ith* node are then copied into the *i*th entry of an array with three columns. The process is illustrated in **Figure 3.4**.

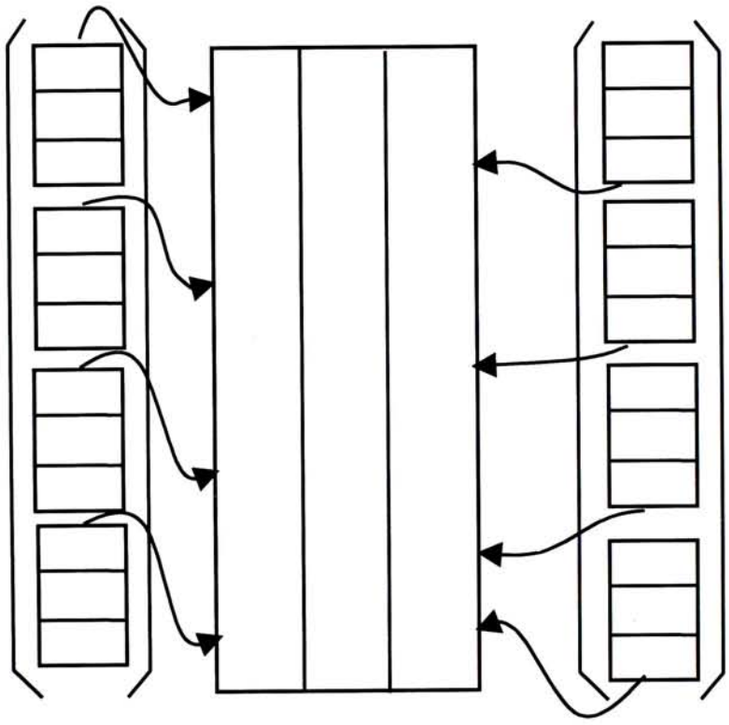


Figure 3.4: Process of displacement re-assembly

Then, the displacements are added to the original mesh points. The deformed shape of the virtual object is then computed. **Figure 3.5** gives an example of displaying the deformed shape with the hand.

Figure 3.5a - An object attached to the virtual hand

Figure 3.5b - Fingers collided with the object and the object undergo small deformation

Figure 3.5c - Fingers moved further resulting in large deformation of the object.

4. Collision Detection

The points of contact between the virtual hand and the virtual objects are required for estimating the response of the virtual object. Therefore, interference test has to be performed between the virtual object and the virtual hand. However, if the interference test is performed for each pair of facets of the virtual object and that of the virtual hand, the computation time required will not be acceptable for interactive applications. Since most of the face pairs are not colliding, the time required for collision detection can be reduced by restricting the interference test to the pairs of object faces which may possibly intersect.

4.1 Related work

There are several methods for interactive collision detection application. Youn and Wohn [YOU93] suggested a hierarchical object (HO) for detecting collision between complex objects. An object is first divided into several major parts and the parts are further sub-divided so that the object is represented by a hierarchy of object parts. **Figure 4.1** is a human HO example.

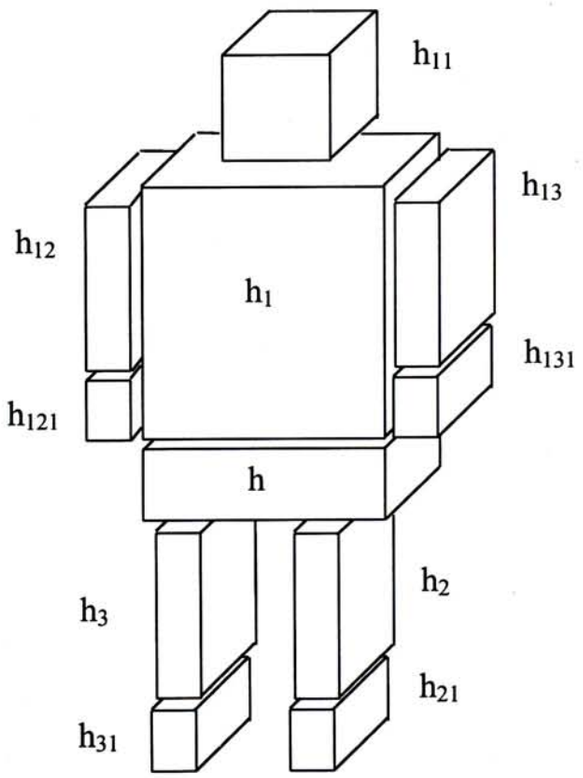


Figure 4.1: A human hierarchical object

In the object, hip (h) is the root of the HO. As the trunk and the legs are connected to the hip (root of HO), the descendent of hip is the trunk (h_1), left leg (h_2) and right leg (h_3). Similarly, the head (h_{11}), left arm (h_{12}) and the right arm (h_{13}) is the descendent of the trunk. Left forearm (h_{121}) and right forearm (h_{131}) is the descendent of left arm and right arm respectively.

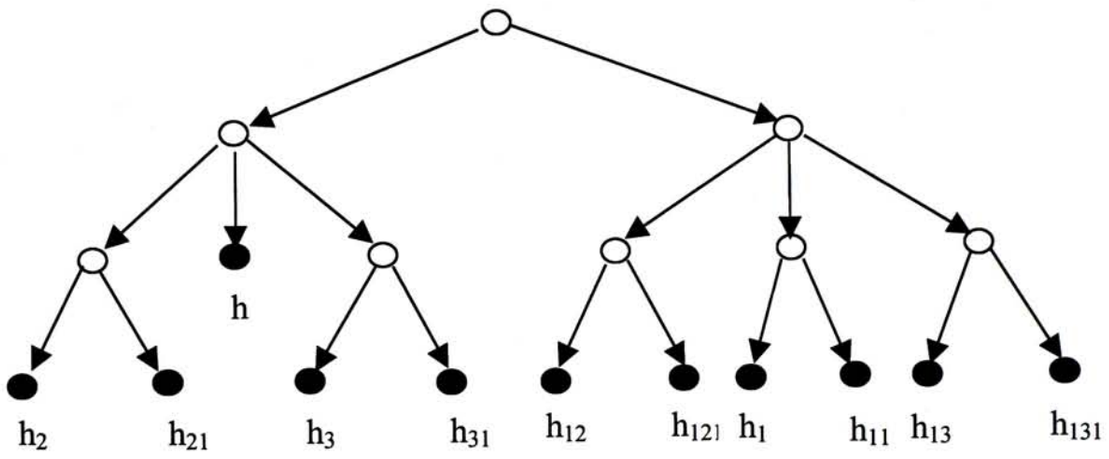


Figure 4.2: C-tree of the human HO

Using the HO, a C-tree is developed for locating the region of collision and reduces the number of comparisons required for detecting the collision of the HO and other virtual objects. There are two types of nodes in the tree. The filled nodes represent pointer to a part of the human HO. The empty nodes represent intermediate nodes within the hierarchy and there is no valid value within such nodes. The C-tree representing the human in **Figure 4.1** is shown in **Figure 4.2** as an example.

Bandi and Thalmann [BAN95] used digital differential analyser as an adaptive spatial subdivision technique and digitise a virtual object into voxels. Based on an octree structure, the virtual object is subdivided into voxels for collision detection tests. During collision detection tests, the virtual object voxels are supersampled so as to increase the number of tests required to be performed. This technique is used when the objects are close to each other. The voxels of the virtual objects are supersampled at a higher level of resolution than an octree does. If the objects are found to be not colliding with each other before the highest level of resolution is reached, they are not considered to be colliding.

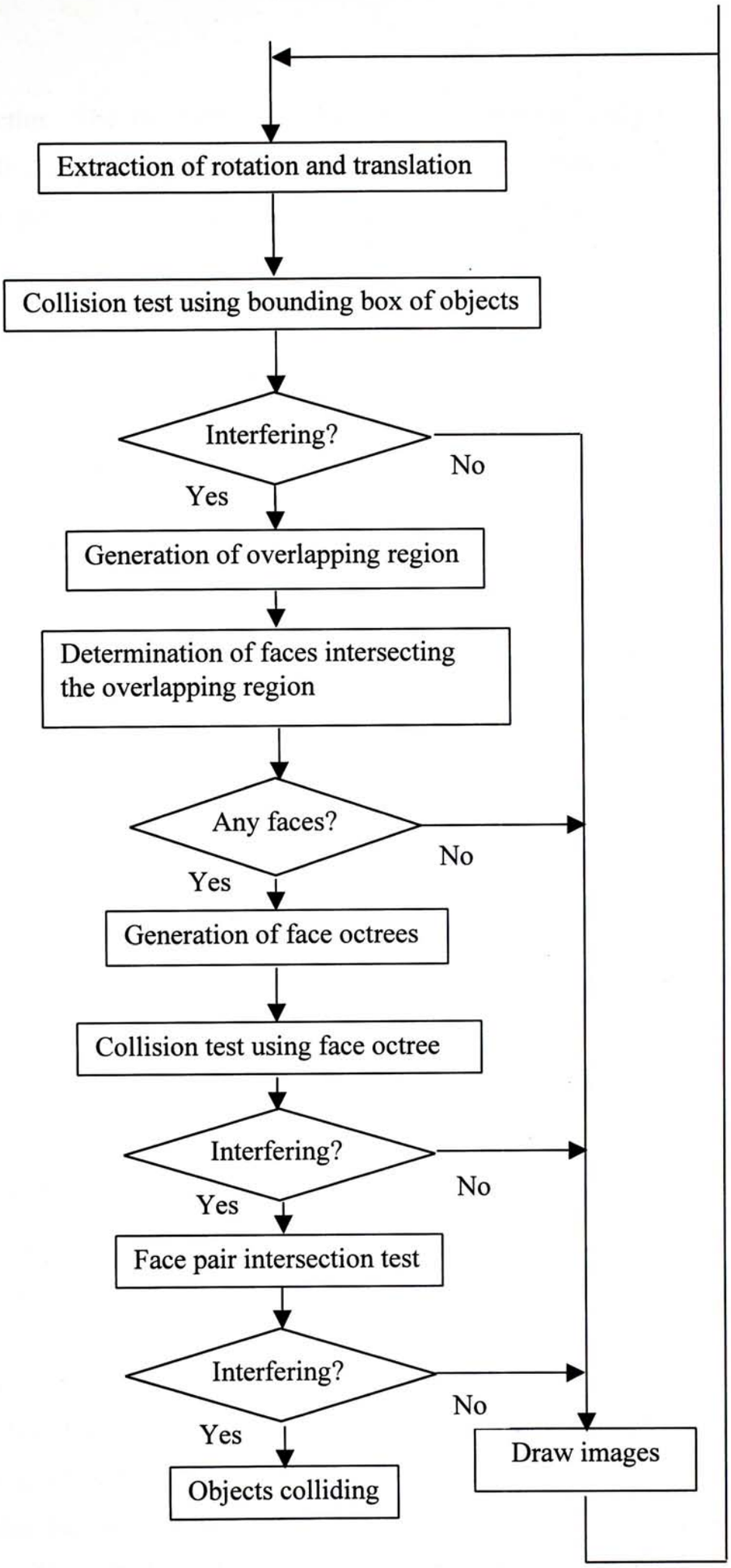


Figure 4.3: Collision detection procedure

Smith et. al. [SMI95] used bounding boxes to approximate a virtual object for

collision detection. The overlapping region of two objects is considered only to reduce the collision detection test. Faces intersecting within the region are determined to exhaust the possible intersecting faces. Octree subdivision is performed so that overlapping region is subdivided and the region contains different object faces are sorted out. Face-face intersection tests are performed for faces of the sorted regions. The colliding faces are then located. The process is shown in **Figure 4.3**.

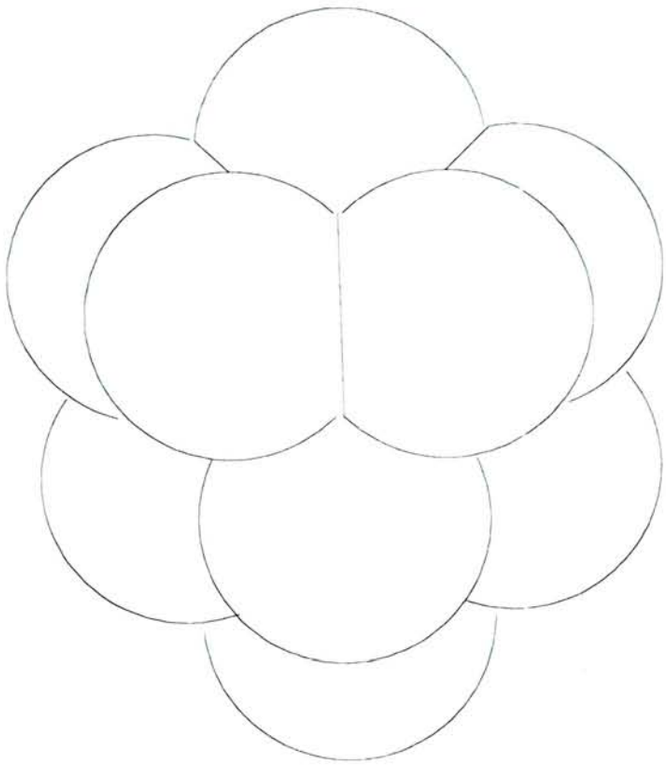
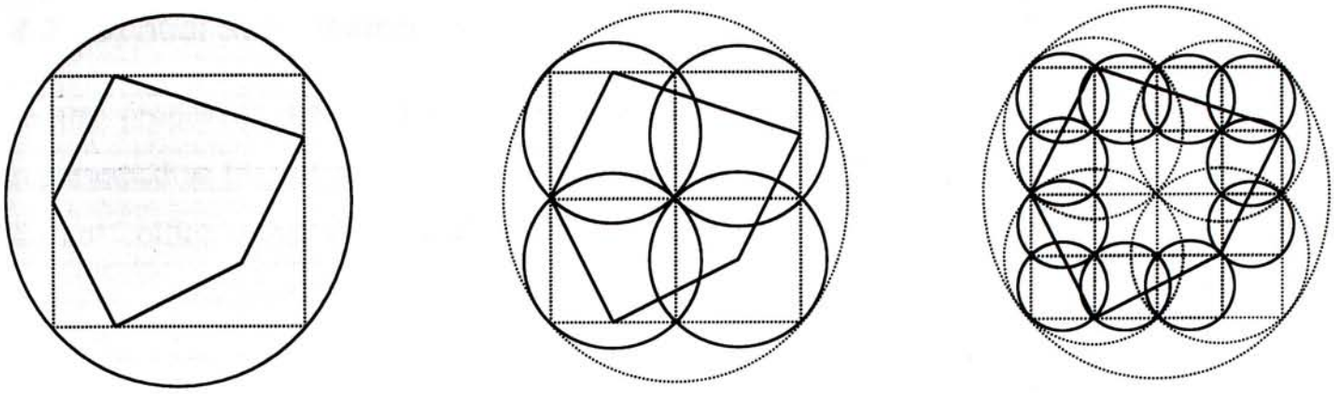


Figure 4.4: Decomposition of a sphere

Several researchers used hierarchical sphere tree structure for approximating objects in collision detection algorithm. Liu et. al. [LIU91] used a solid model called Hierarchical Sphere Model (HSM) for approximating the objects in the virtual environment. The root node of the tree is a sphere that surrounds the object to be approximated. Then, for each level, the corresponding sphere is decomposed into 13 overlapping descendent spheres, as shown in **Figure 4.4**. The descendent spheres are divided into three types, namely "white" node, "black" node and "mix" node. Among the three types, white nodes represent nodes that are outside the object. Black nodes represent nodes that are inside the object whereas mix nodes are nodes that are on the surface of the object. Both black and white nodes are leaf nodes of the tree and only mix nodes require further subdivision. As all the nodes are spheres, collision can be detected by performing sphere-sphere intersection test.



a. Level 1 of tree **b.** Level 2 of tree **c.** Level 3 of tree

Figure 4.5: A 2D analogy of a sphere tree

Palmer et. al. [PAL96] used a three-stage process for efficient collision detection. In the first stage, a global bounding volume of the object is used as a reference for constructing the sphere at the root node of the hierarchy. In the second stage, potentially colliding regions are detected by searching down the sphere tree. The sphere tree is a tree of overlapping spheres constructed with an octree-type spatial subdivision technique. A 2D analogy is shown in **Figure 4.5**. The final stage in the process is to perform face interference tests when the spheres at the leaf nodes of the objects collide with each others.

Hubbard [HUB96] used medial-axis surfaces to construct sphere-tree for approximating the objects that may collide with each other. The tree is constructed such that a progressively refining accuracy of object approximation can be attained. This technique can support time-critical collision detection algorithms that trade time and accuracy for collision detection. Using medial-axis surfaces for construction of the sphere hierarchy, a tight bound of object approximation can be attained. The nodes are then merged to create the ancestor nodes. Hence, a multi-resolution approximation of the object is obtained. This technique can be used to create a progressively refining approximation of the virtual objects. This can improve the accuracy of the collision detection but the computation overhead for constructing the hierarchy is increased substantially (about 400 times slower than the construction of octree).

4.2 Spatial Subdivision

In the proposed system, the virtual hand and the virtual objects are polyhedra represented as lists of vertices and faces. However, most face, edge and vertex pairs do not collide either because they are very far apart from each other or they are

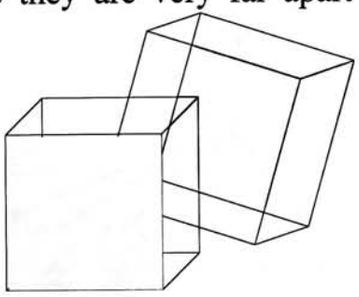


Figure 4.6: The shaded faces are unlikely to collide

geometrically impossible to have collision (e.g., the shaded face pair in **Figure 4.6**). These trivial pairs are eliminated in an efficient way such that only those possibly colliding pairs are tested for intersection.

One of the effective ways to eliminate those trivial pairs is to construct a hierarchy that subdivides the virtual objects into several parts in different resolutions. By searching down the hierarchy from the root node, collision is detected at different resolution levels. The region for performing interference test is reduced by going down the hierarchy. Only the finest region, i.e. the lowest level of the hierarchy, is tested for interference.

Apart from the search for collision detection, the time for constructing the hierarchy have to be considered as well. Most existing algorithms [PAL96,HUB96,LIU91,MOR88] assumed the colliding bodies to be rigid so that the tree can be constructed offline. Therefore, the time for constructing the hierarchy does not affect the interactive performance of the system. Unfortunately, the assumption is not applicable for deformable objects. The shape of a deformable object changes over time and the hierarchy has to be changed in order to represent the object geometry correctly. The time for constructing the hierarchy is thus critical for interactive performance [SMI95].

Taking both the times for efficient tree search and effective construction of the hierarchy into account, a sphere octree structure is adopted. A typical octree structure is shown in **Figure 4.7**. At each node of the tree, there are eight descendent nodes

(including some NULL nodes). If all the descendent nodes of a node are NULL, the node is a leaf node. In the sphere tree, each level (n) represents a different level of approximation of the virtual objects by a collection of spheres. In other words, the collection of nodes in the same level (n) within the hierarchy represents the collection of spheres at the same level of resolution approximating the virtual object.

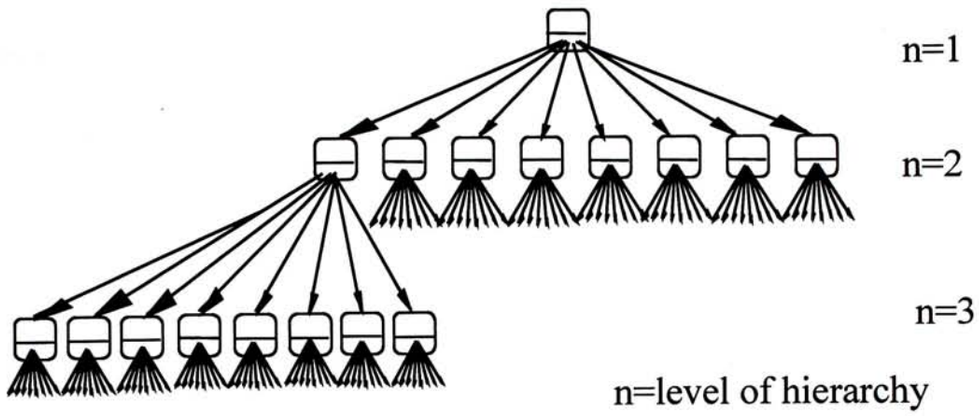


Figure 4.7: An octree structure

4.3 Hierarchy construction

The sphere tree is constructed by means of spatial subdivision. A bounding sphere enclosing the virtual object completely is constructed. This is the root node of the hierarchy. A tree is constructed by overlapping spheres that divides the current sphere into eight equal regions. Nodes are retained if the nodes represent spheres on the surface of the object, and are discarded if the spheres are not on the surface of the object. This subdivision process is repeated until a predefined number of polygon vertices of the object lie within one single sphere. In the proposed system, only one vertex is allowed in a sphere since the goals of the collision detection algorithm is to locate the colliding point between the object and the fingertips. The pseudo-code of the algorithm is shown in **Figure 4.8**.

Tree_structure is a pointer to a node. (The definition of a node is described in 4.3.1)

Tree_structure **root_node** =NULL;
Tree_structure **parent_node**;
Tree_structure **current_node**;
integer **n**; (the maximum number of vertices in the sphere)

procedure hierarchy (**parent_node**, **current_node**, **root_node**)
{
integer **i**;

Construct bounding sphere if it is the root node, i.e. if root_node = NULL.

Calculate the radius and centre of **current_node**.

if (the number of vertices in the sphere > **n**)
 {
  for **i** = 1 to 8 by 1
   Create the ***ith***_**child_node** *of the* **current_node**.
  for **i** = 1 to 8 by 1
   hierarchy (**current_node**, ***ith***_**child_node**, **root_node**)
 }

if (the number of vertices ≤ **n**)
 Save the vertex information of the object in the **current_node**.

if (there is no vertex in the **current_node**)
 Discard **current_node**.

return **root_node**.

}

Figure 4.8: Pseudo-code of sphere hierarchy construction algorithm

4.3.1 Data structure

There are two types of information stored in a node of the hierarchy: they are pointers to the descendent nodes and the data describing the characteristic of the current node, as shown in **Figure 4.9**.

Eight pointers are stored in a node pointing to its descendent nodes. One exception is a leaf node where all the pointers to the descendent nodes are NULL.

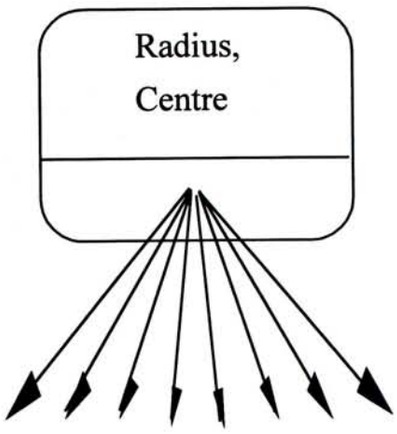


Figure 4.9: A node structure

In addition to the pointers to the descendent nodes, there is a data region for describing the characteristics of the node. The characteristics represented in the data region is summarised in **Table 4.1**.

Characteristics	Data Type	Description
Radius	A floating point number	The radius of the sphere.
Centre	A 3D vector	The centre position of the sphere.
Maximum length	A floating point number	The maximum length of a bounding box. See section 4.3.3.
Vertex	An integer number	This data is valid only in the leaf node. See section 4.3.3.

Table 4.1: Characteristic description of data in each node

In the sphere tree, each node represents a sphere in virtual space. A sphere can be accurately described by its radius and its centre position, where the radius describes the size of the sphere and the centre position describes the location of the sphere.

The maximum length describes the length of the bounding cube in a node. The bounding cube is the largest cube enclosed by the sphere in the node. An example is shown in **Figure 4.10**. As the space is subdivided based on the bounding cube (See section 4.3.3), the maximum length is used for evaluating the centres of the spheres of the descendent nodes.

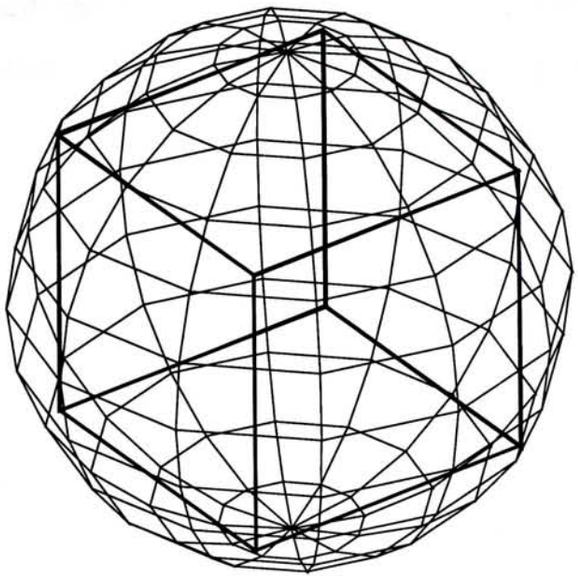


Figure 4.10: The bounding box and bounding sphere

In order to perform tests other than tree search (e.g. retrieving information for finite element analysis), the position of the polygon vertex enclosed in the leaf node is stored as well (See section 4.3.3).

4.3.2 Initialisation

Initialisation of a sphere tree is the construction of the root node of the tree. The root node of the tree is the bounding sphere of the virtual object. In order to ensure that the virtual object is enclosed completely by the sphere, a bounding box is first created by locating the vertices of the object with the maximum and minimum co-ordinate values. The box is then covered by the smallest sphere enclosing it.

The centre of the sphere at the root node can be found by the equation,

$$\left(\frac{x_{max} + x_{min}}{2}, \frac{y_{max} + y_{min}}{2}, \frac{z_{max} + z_{min}}{2} \right) \qquad \textbf{(4.1)}$$

where the subscript max and min represents the maximum and minimum values of the vertices co-ordinates.

The length of the bounding box is calculated by the difference between the maximum and minimum value of the vertices co-ordinates. The radius of the sphere is determined by the maximum length among the three lengths of the box.

$$\text{max\_len} = \max\{x_{\max} - x_{\min}, y_{\max} - y_{\min}, z_{\max} - z_{\min}\} \quad \textbf{(4.2)}$$

where max_len is the length of the bounding cube.

A bounding cube with its length equals the maximum length of the bounding box and its centre locates at the centre of the sphere is constructed. The radius of the root node sphere is determined from the length of the bounding cube by the equation,

$$\text{Radius} = \text{max\_len}\sqrt{\frac{3}{4}} \quad \textbf{(4.3)}$$

where max_len is the length of the bounding cube.

Result of the initialisation step is illustrated in **Figure 4.10**.

4.3.3 Expanding the hierarchy

Similar to the construction of an octree, each edge of the bounding cube is subdivided into two equal parts. Hence, the cube is divided into 8 or 2^3 descendent cubes as shown in **Figure 4.11**. For each bounding cube, the smallest possible sphere is created to enclose the cube. Therefore, eight overlapping spheres are created for the second

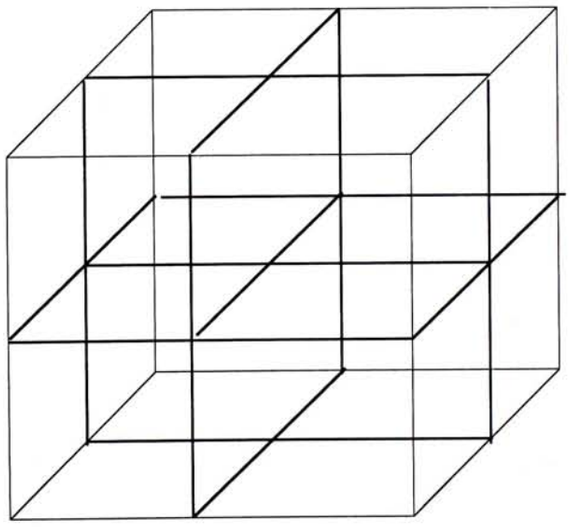


Figure 4.11: Spatial subdivision of bounding box

level of approximation, as shown in **Figure 4.12**.

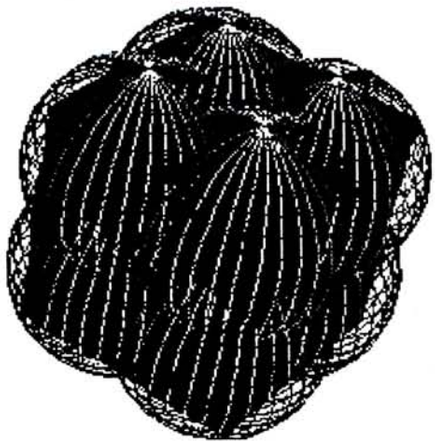


Figure 4.12: Sphere in the second level of hierarchy

As the space is subdivided evenly, the location of the spheres in the second level can be calculated from the vertices of the cube.

In the spatial subdivision process, the bounding cube is subdivided into eight equal descendent cubes. As all the cubes are axis aligned, the length of the descendent cubes are half the length of the cube. Similarly, the co-ordinates of the centre of the descendent cube are offset by ¼ of the length of the cube. Hence, the centre of the descendent cubes can be summarised by the equation,

$$\begin{cases} x_2 = x_1 \pm \dfrac{\text{max\_len}_1}{4} \\ y_2 = y_1 \pm \dfrac{\text{max\_len}_1}{4} \\ z_2 = z_1 \pm \dfrac{\text{max\_len}_1}{4} \end{cases} \tag{4.4}$$

where (x_1, y_1, z_1) is the centre of the current bounding cube,

(x_2, y_2, z_2) is the centre of the descendent bounding cube, and

max_len_1 is the length of the bounding cube.

Observing that the centre of each descendent cube is equi-distance from the corner vertices of the cube, the centre of the descendent cube is the centre of the descendent sphere.

The length of the descendent bounding cube is given by,

$$\text{max\_len}_2 = \frac{\text{max\_len}_1}{2} \tag{4.5}$$

where max_len_2 is the length of the descendent cube, and

max_len_1 is the length of the cube.

From **(4.3)** and **(4.5)**, the radius of the descendent sphere is given by,

$$\begin{aligned} \text{Radius}_2 &= \text{max\_len}_2 \sqrt{\frac{3}{4}} \\ &= \frac{\text{max\_len}_1}{2} \sqrt{\frac{3}{4}} \end{aligned}$$

$$\Rightarrow \text{Radius}_2 = \frac{\text{Radius}_1}{2} \tag{4.6}$$

where max_len_2 is the length of the descendent cube,

max_len_1 is the length of the current cube,

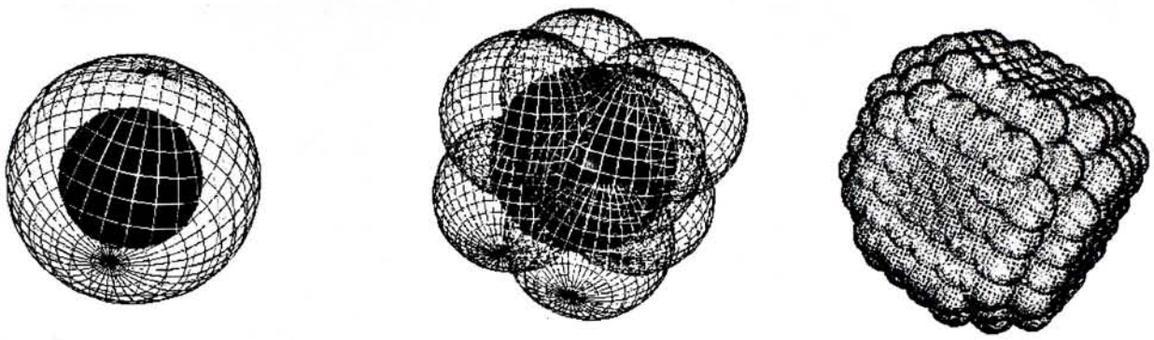
Radius_2 is the radius of the descendent sphere, and

Radius_1 is the radius of the sphere.

The spatial subdivision is repeated recursively in a similar way until the termination condition is reached. As illustrated in **Figure 4.8**, there are two termination conditions for the process. The first termination condition is when the descendent sphere does not lie on the surface of the object, i.e. there is no polygon vertex enclosed by the sphere. The node will be discarded when this condition is satisfied. The second termination condition is when there is only one polygon vertex enclosed by the descendent sphere. This implies that the node is a leaf node. The corresponding vertex number is stored in the leaf node. As the termination conditions are associated with

the number of vertices in the bounding sphere, the test for the termination condition is attained by comparing the distance between each polygon vertex of the object and the centre of the sphere with the radius of the bounding sphere. By counting the number of vertices in the sphere, the termination condition can be detected.

The result of sphere tree construction is shown in **Figure 4.13**.



(a) Level 1 (b) Level 2 (c) Leaf nodes

Figure 4.13: Construction of a sphere tree

4.4 Collision detection

The collision detection algorithm is performed in two stages. In the first stage, the algorithm searches the sphere tree to locate possible collision region. This is achieved by performing cylinder-sphere interference test (finger segments are approximated by cylinders). Since the fingertips are approximated by spheres, additional sphere-sphere interference test has to be performed for detecting interference between the distal segments and the object. When a fingertip approaches further to the object and finally goes into the sphere of the leaf node of the sphere tree, exact interference test is performed for detecting interference between the fingertip and the surface of the object. This is the second stage of the algorithm. Details of the algorithm are discussed in the following sections.

4.4.1 Hand approximation for collision detection

The hand model, which is assumed to be a rigid object, is approximated by a union of cylinders and spheres. An example of the approximated hand model is shown in **Figure 4.14**.

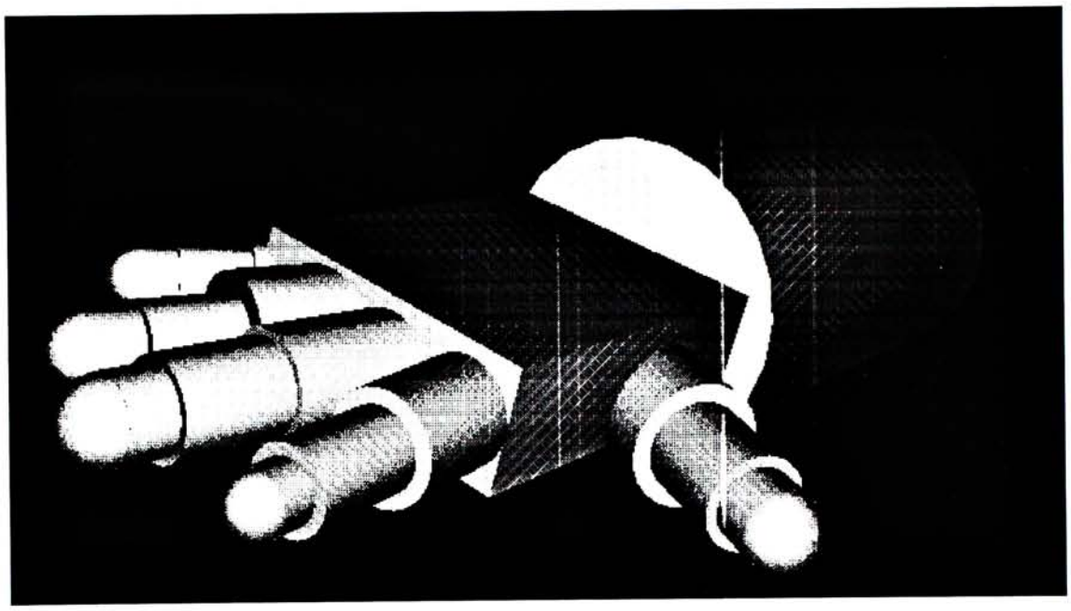


Figure 4.14: The simplified hand model

The simplified hand here is different from the hand approximation taken by Huang et. al. [HUA95] as the hand can approximate the whole finger segment instead of the joints of the fingers.

In **Figure 4.14**, the metacarpophalangeal, proximal interphalangeal and distal interphalangeal segments are represented by cylinders. The sizes of the cylinders are selected so that they are the smallest cylinders enclosing the respective finger segments. At each fingertip, a hemisphere is attached to the distal end of the finger segment. In addition, one vertex on the faceted model of each fingertip is selected to be the point on the finger to be in contact with the virtual object. The box enclosing the palm is not considered in the collision detection algorithm.

Since the approximation of the virtual object and virtual hand only involve cylinders and spheres, tests for cylinder-sphere interference and sphere-sphere interference are sufficient for locating possible collision.

4.4.2 Interference tests

There are two types of interference tests performed on different parts of the finger colliding with the virtual object. For the metacarpophalangeal and proximal interphalangeal segment of the finger, the cylinder-sphere interference test is used since the finger segments are approximated by a cylinder. On the distal interphalangeal segment, both cylinder-sphere interference and sphere-sphere interference tests are performed. The results of both tests are combined to determine the interference result between the fingers and the virtual object.

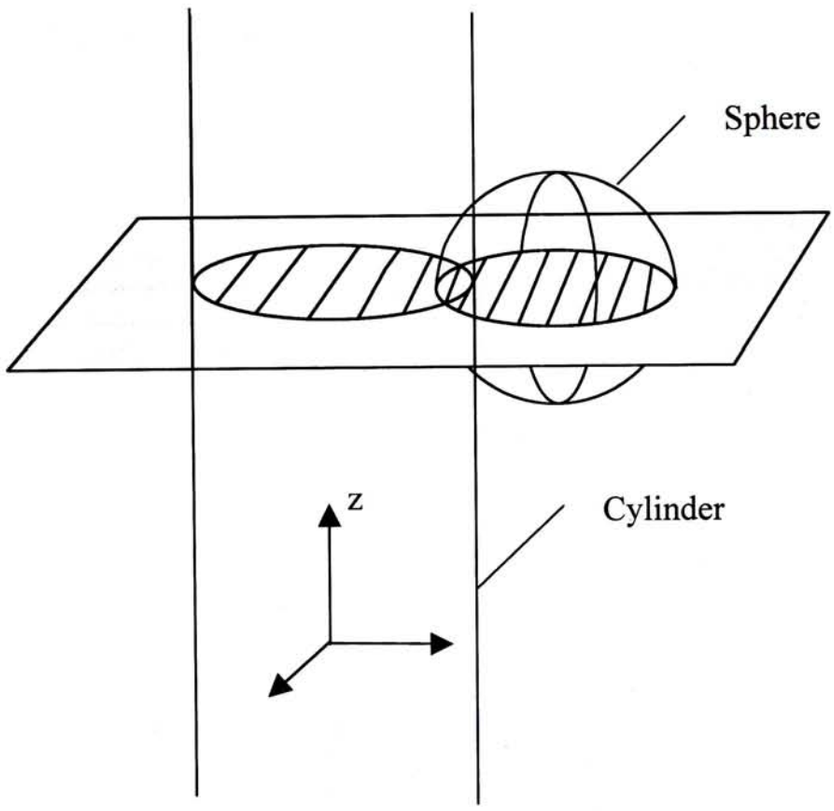


Figure 4.15: Sphere intersecting the cylindrical surface

Cylinder-sphere interference test

As shown in **Figure 4.15**, interference between the cylinder and the sphere is detected by estimating the minimum distance between the cylindrical surface and the sphere. Assuming the centre of cylinder is the origin and the axis of the cylinder is the z-axis, the minimum distance is calculated by **(4.7)**,

$$d_{\min} = \sqrt{s_x^{\,2} + s_y^{\,2}} \tag{4.7}$$

where $d_{\min}$ is the minimum distance between the cylindrical and the sphere,

s_x and s_y are the x and y co-ordinates of the centre of the sphere.

If the minimum distance ($d_{\min}$) is larger than the sum of the radius of the cylinder (r_c) and the radius of the sphere (r_s), then they are not colliding with each other. Otherwise, further test is required.

For sphere that intersect with the cylindrical surface, three cases for detecting interference with the bounds of the cylinder have to be considered depending on the position of the sphere along the z-axis. An example for testing against the upper bound is illustrated in **Figure 4.16-18**. Similar tests are performed against the lower bound of the cylinder.

Denote s_z as the z co-ordinate of the centre of the sphere,

cu_z as the z co-ordinate of the upper bound of the cylinder, and

cl_z as the z co-ordinate of the lower bound of the cylinder.

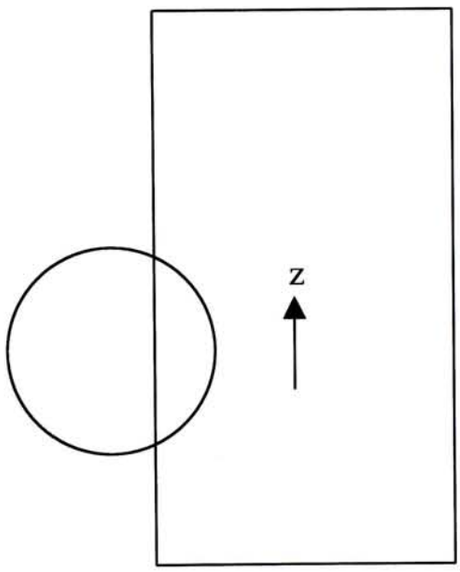


Figure 4.16: Sphere located between cylinder

Case 1: $cl_z \leq s_z \leq cu_z$ (**Figure 4.16**)

This is the same as intersecting the sphere with the infinite cylindrical surfaces. Since the sphere is found to intersect with the cylindrical surface before this test is

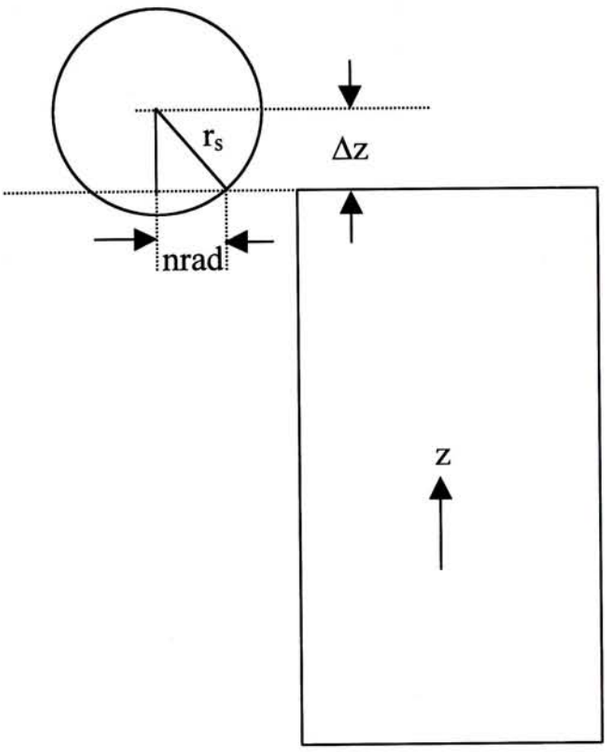


Figure 4.17: $\Delta z \leq r_s$

performed, the sphere must collide with the cylinder.

Case 2: $\Delta z \leq r_s$ (**Figure 4.17**)

Δz is defined by the difference in z co-ordinates between the centre of the sphere and the upper bound of the cylinder, i.e.

$$\Delta z = s_z - cu_z \tag{4.8}$$

In this case, the semi-chord length (*nrad*) of the intersecting circle as shown in **Figure 4.17** is given by **(4.9)**,

$$nrad = \sqrt{r_s^2 - \Delta z^2} \tag{4.9}$$

The semi-chord is constructed by intersecting the plane containing the upper bound of the cylinder and the sphere. Considering the plane containing the upper bound of the

cylinder, the semi-chord is actually the radius of a circle which intersects the cylinder where the sphere intersects the cylinder.

As both the sphere-plane intersection and the upper bound of the cylinder are circles, the distance d_{min} (obtained by **4.7**) between the centre of these circles can be used for detecting interference. If d_{min} is larger than the sum of the radius of the cylinder and the circle (r_c+$nrad$), the sphere and the cylinder are not colliding. Otherwise, the

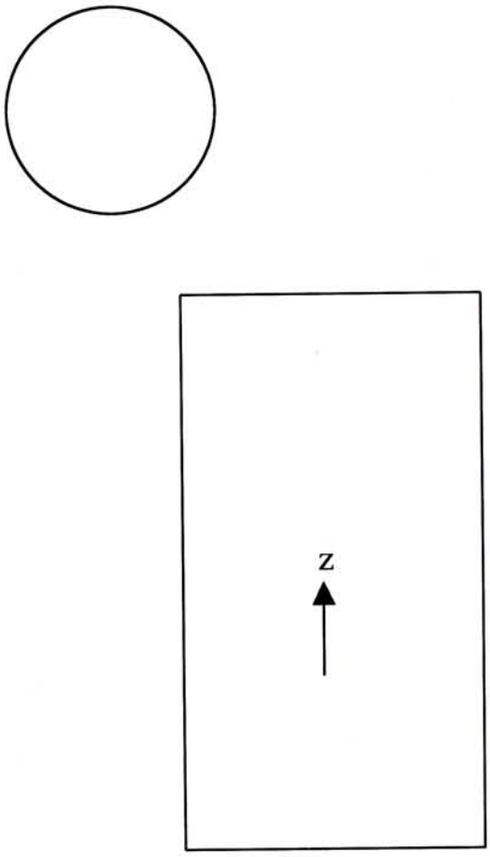


Figure 4.18: $\Delta z > r_s$

sphere and the cylinder collide with each other.

Case 3: $\Delta z > r_s$ (**Figure 4.18**)

In this case, the sphere is well above the plane containing the upper bound of the cylinder. The sphere does not collide with the cylinder even it collides with the infinite cylindrical surface.

Sphere-sphere interference test

As spheres are orientation free, the only information required to test the interference between spheres is the distance, d, between the centre of the spheres, as shown in **Figure 4.19**. If d is larger than the sum of the radii of the spheres, the spheres are not colliding. Otherwise, they collide with each other.

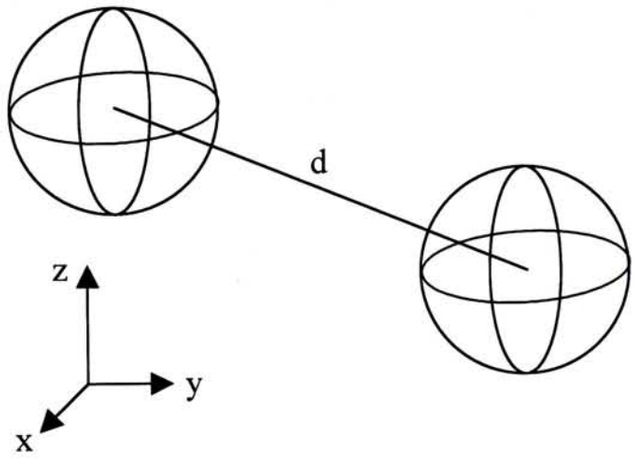


Figure 4.19: Sphere-sphere intersection test

4.4.3 Searching the hierarchy

For each finger segment, interference test between the cylinders approximating the finger and the bounding sphere of the virtual object is performed. If the interference test gives positive result, the finger segment is tested with the descendent spheres of the bounding sphere. Whenever there is interference between the cylinder approximating the finger segment and the sphere of a node, the corresponding descendent nodes are retrieved and tested. If positive result persists when the sphere representing a leaf node in the hierarchy is tested, the finger segment is reported to be "near the virtual object" and the exact interference test is performed. Otherwise, the finger segment is reported to be "not colliding with the virtual object".

4.4.4 Exact interference test

As described in section 4.4.3, the finger segment is near the virtual object if the cylinder and sphere approximating the object collides with the sphere at a leaf node of the hierarchy. The distal interphalangeal segments are assumed to deform the virtual object. Therefore, further interference test between the fingertip and the object vertex has to be performed. The test ensures that the virtual object is only deformed when there is collision between the fingertip and the object.

For each finger, a point on the distal interphalangeal segment is assumed to be the fingertip. A vector **D** is constructed from the simple vertex of the leaf node to the fingertip. In addition, a normal **N** at the vertex of the leaf node of the virtual object is constructed by averaging all the neighbouring face normal of the vertex. The vectors are shown in **Figure 4.20**. Scalar product is performed on the two vectors,

$$R = \mathbf{D} \bullet \mathbf{N} \tag{4.10}$$

where R is the result of the scalar product.

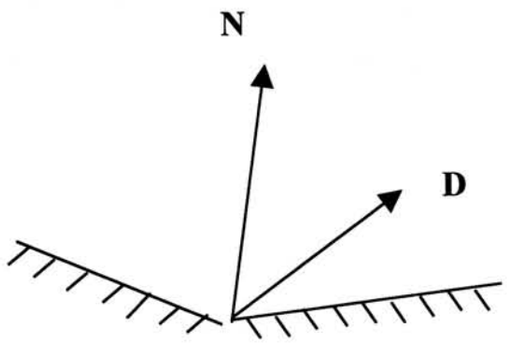
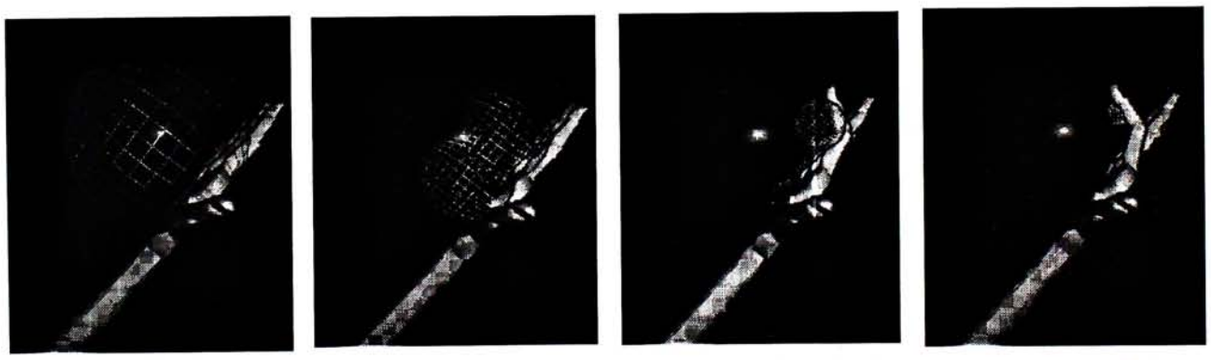


Figure 4.20: Point normal and displacement

If the result R is greater than zero, **D** is in the same direction as **N** and the fingertip is outside in the virtual object. Hence, there is no collision between the finger and the object. Otherwise, the fingertip penetrates into the virtual object.

An example of the collision detection process is shown in **Figure 4.21**.



(a) First level collision (b) Second level collision (c) Finger collided with a non-leaf node (d) Finger collided with leaf node

Figure 4.21: Process of collision detection

4.5 *Grasping mode*

In the proposed system, there is no depth information about the position of the hand and the virtual object. It is difficult for the user to grasp an object in the virtual environment effectively. In the absence of gesture recognition and stereo visualisation, some rules have to be developed so that the object can be grasp by the hand in an effective way.

4.5.1 Conditions for Finite Element Analysis (FEA)

Since it is assumed that the virtual object is deformed by the fingertips only, there are thus five points exerting external forces to the virtual object. In the collision detection process, it is essential to detect simultaneously the five points where the fingertips are in contact with the object so that FEA can be performed.

4.5.2 Attaching condition

It is difficult to manipulate the virtual hand to touch the object with five fingertips simultaneously. In order to simplify the problem, the object is first attached to the hand when three fingers collide with the object simultaneously. The three fingers are the first finger, the middle finger, and the ring finger. The contact points for the thumb and the last finger are detected after attaching the object to the hand.

The attachment is made by keeping the relative position and orientation of the object with respect to the hand at the instance of contact. An object attached to the virtual hand is shown in **Figure 4.22**.

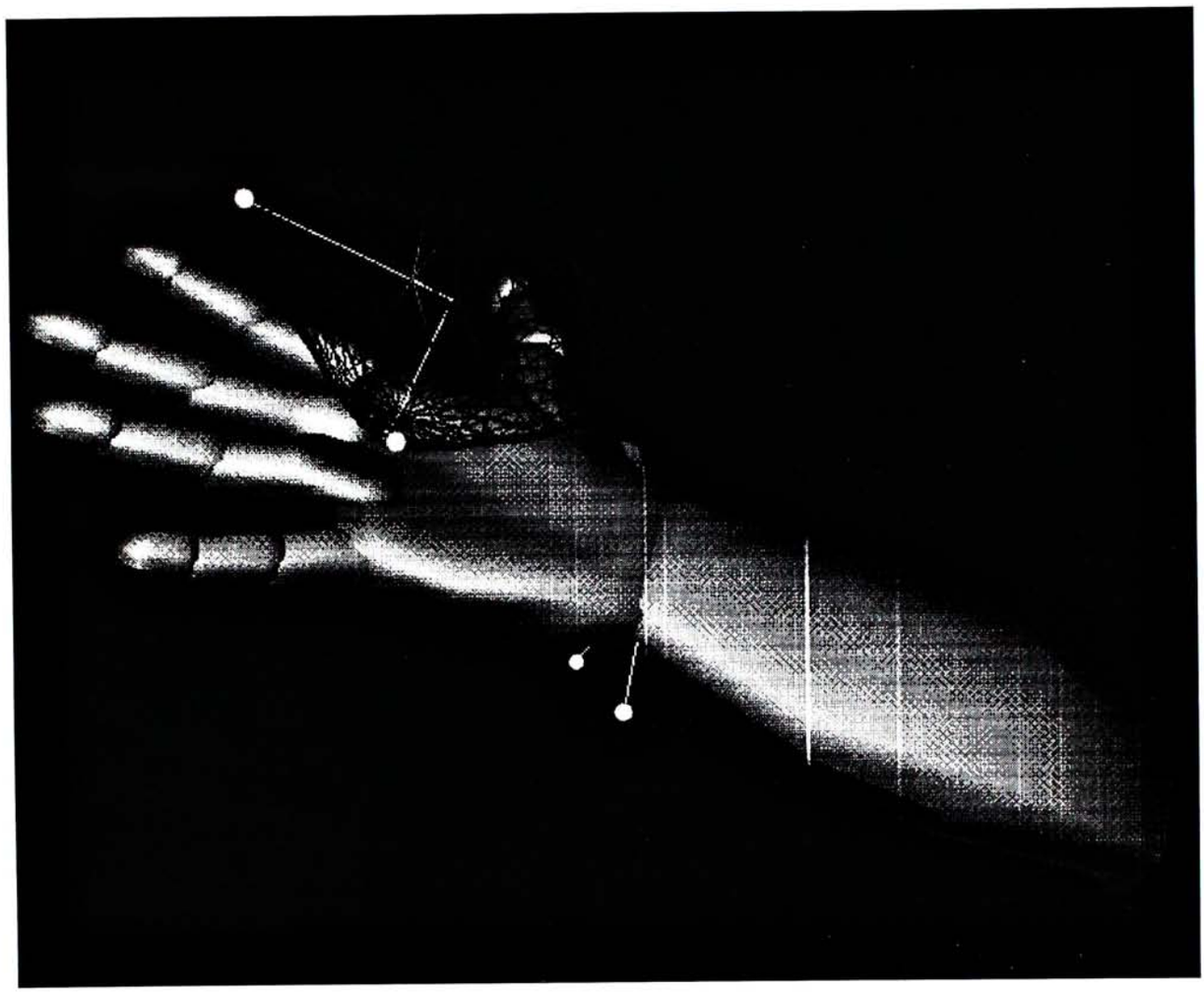


Figure 4.22: Relative position and orientation of the object with respect to the palm.

4.5.3 Collision avoidance

In the grasping operation, the object is assumed rigid so that no finger is allowed to penetrate the object.

Whenever the finger segments collides with the object, penetration into the object may occur. This is avoided by recalling the position of the finger segment just before the collision. The finger segments are then displayed at the position and orientation just before collision with the object. Furthermore, the previous positions and orientations of the lower parts are recalled when the upper part of the finger collides with the virtual object. For example, the positions and orientations of the proximal interphalangeal segment and the metacarpophalangeal segment have to be recalled when the distal interphalangeal segment of the finger collides with the object, as shown in **Figure 4.23**.

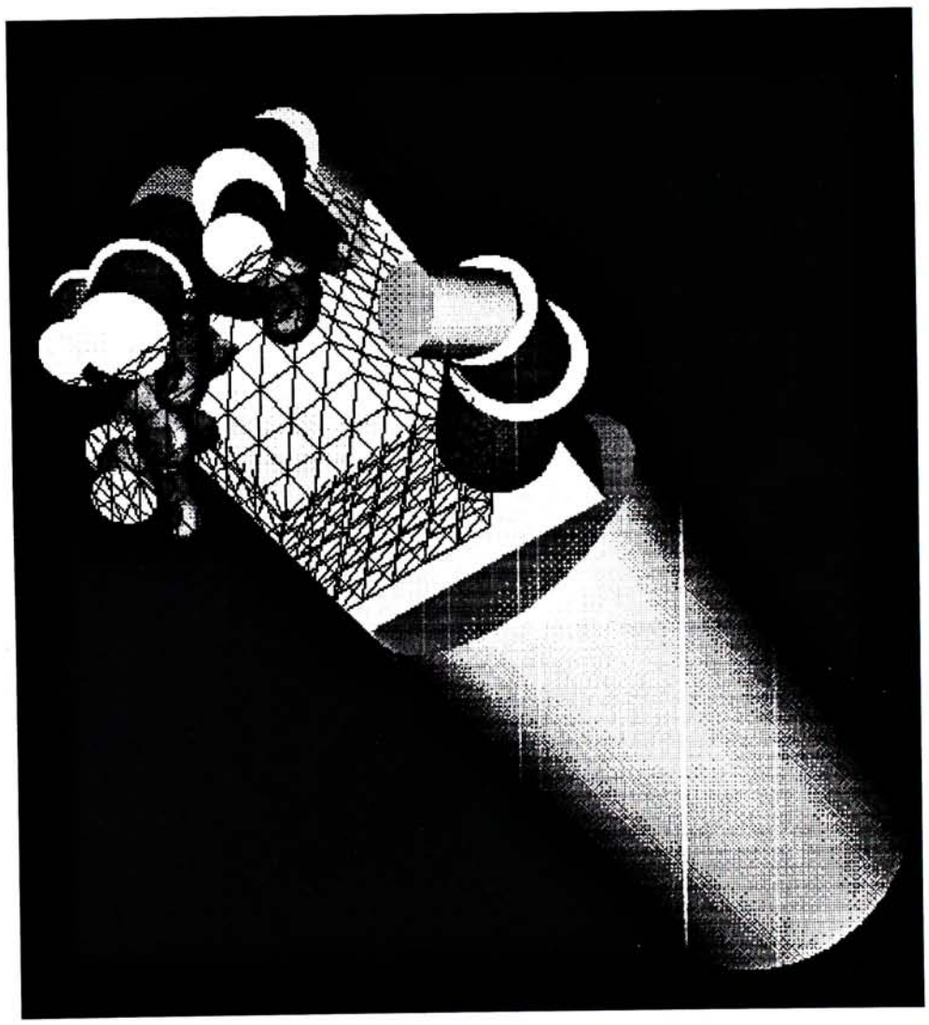


Figure 4.23: The effect of collision avoidance for the approximated hand

4.6 *Repeating deformation in different orientation*

When the collision detection algorithm is applied to the system and the tree is constructed in run-time, the virtual object can be deformed by the virtual hand repeatedly in different orientation. The process is shown in **Figure 4.24** to **Figure 4.29** using a virtual strawberry as an example.

Figure 4.24: The virtual hand deforming an attached cube

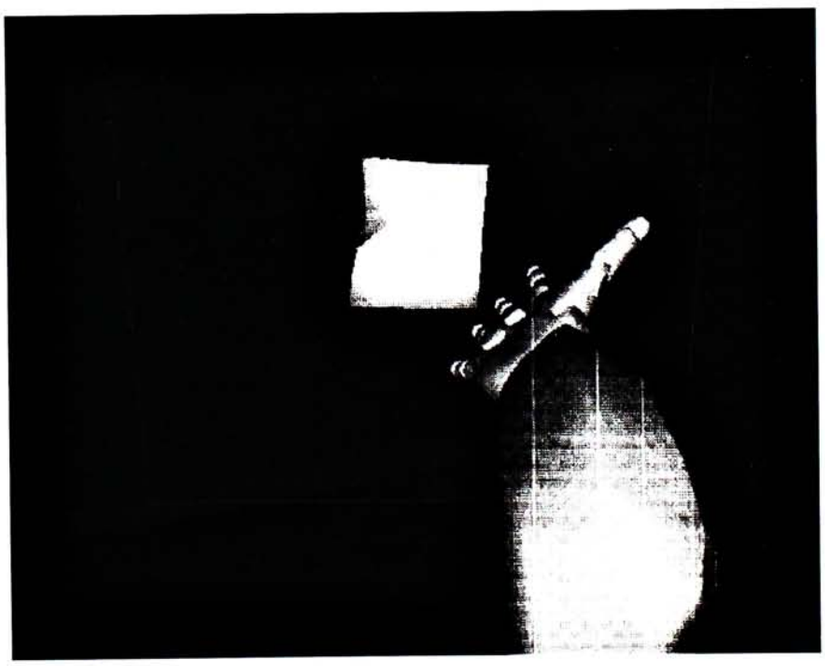


Figure 4.25: The deformed cube is detached from the hand

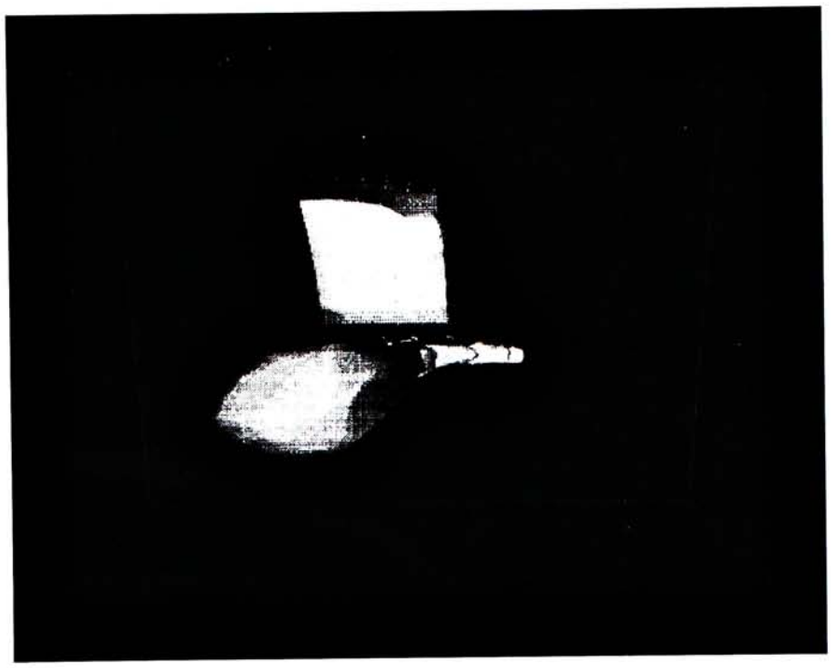


Figure 4.26: The hand tries to approach to the new object from another direction

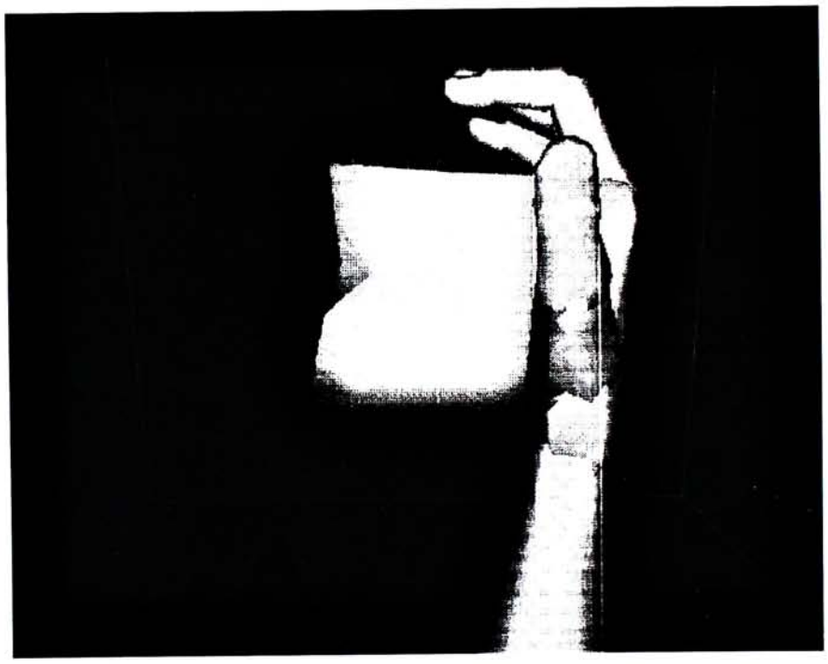


Figure 4.27: The new object is attached to the hand again with another orientation

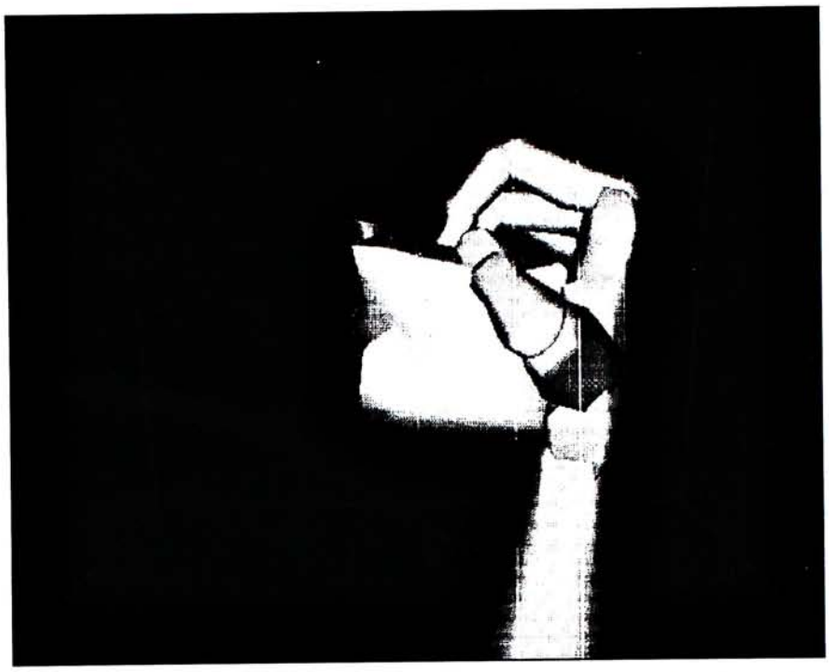


Figure 4.28: The new object is deformed again by the hand

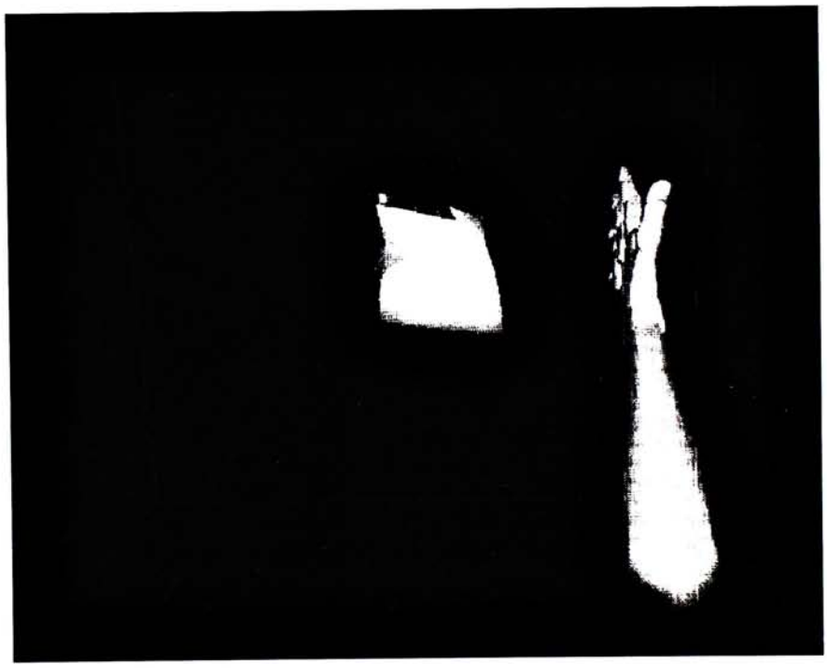


Figure 4.29: The object is detached from the hand for the second time.

5 Enhancing performance

By comparing the time required for various processes in the system (**Figure 5.1**), the finite element analysis (FEA) as described in chapter 3 is found to be the bottleneck. The time required for executing the FEA procedure on an SGI Indigo2 workstation is 280 seconds for a virtual object with 267 nodes. Among the different stages in the finite element analysis process, matrix inversion takes the longest time for execution. With a virtual object composed of 267 nodes, the time required for matrix inversion amount to 99% of the total execution time of the FEA process (**Figure 5.2**).

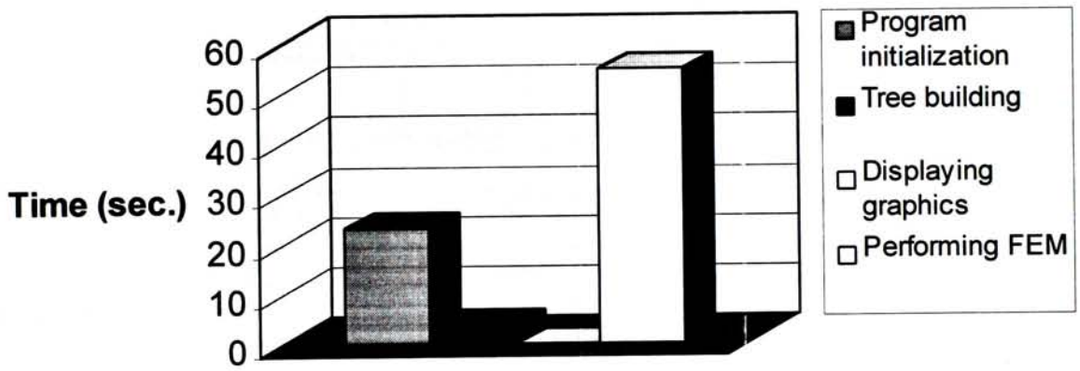


Figure 5.1: Time for various procedures in system

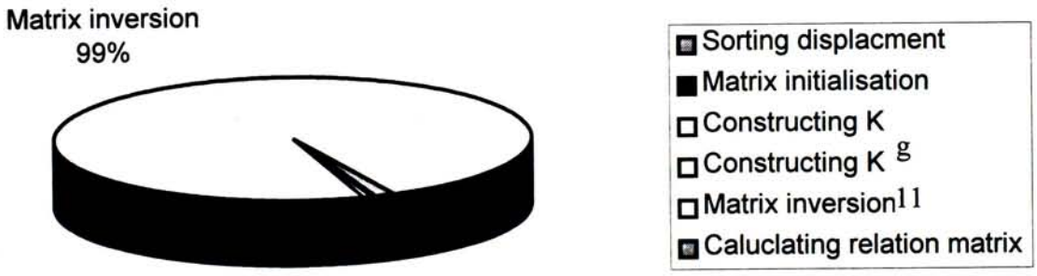


Figure 5.2: Comparison of execution time of procedures in FEA

In order to reduce the execution time of the matrix inversion procedure, one approach is to perform the FEA procedure on a parallel machine. This requires parallelising the FEA procedure for more efficient execution on a parallel computer.

In the proposed system, the matrix inversion procedure is executed on an ONYX Power Challenge 10000 machine (ONYX). The ONYX Power Challenge 10000 machine is a workstation with four R10000 processors. These four processors allow parallel processing of the FEA procedure. However, the system performance can only be improved if the computation power of the processors is properly utilised. In addition, data communication procedures have to be developed for transferring data between the Indigo2 and the ONYX machine.

5.1 Data communication

The data communication between Indigo2 and ONYX is accomplished by using the client-server model, the TCP/IP Internet protocol and the Berkeley socket.

5.1.1 Client-server model

The standard client-server model [STE91] is adopted in the current system. A server is a process waiting to be contacted by a client process so that the server can perform the task requested by the client.

In the data communication procedure of the current system, the server process is started on the ONYX machine before the client process in the Indigo2 starts. After initialisation, the server is put to sleep waiting for a client process to awake it. The client process on the Indigo2 is initialised to establish a connection. The client process then sends a connection request across the network to the server. After the connection is established, the client sends input data to the server. The server receives the data and performs the FEA procedures. When the server finishes performing the FEA procedures, data are directed back to the Inidgo2. The server then goes to sleep again and waits for the next client request. This model can be described as a connection-oriented model as shown in **Figure 5.3**.

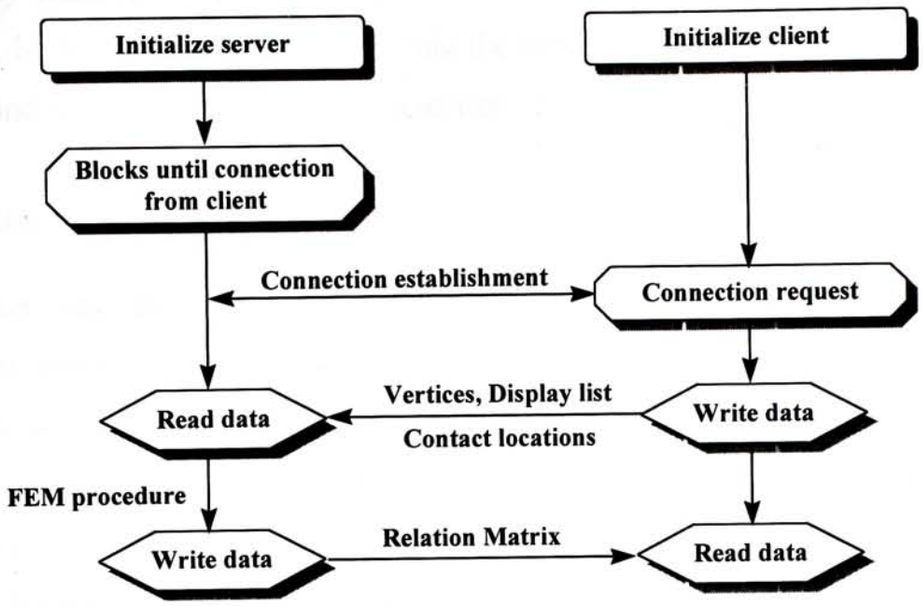


Figure 5.3: Connection-oriented Data Transfer

5.1.2 Internet protocol suite

The use of TCP/IP protocol originates from a military research project in US. It is developed by a DARPA-funded research (DARPA is Defence Advanced Research Projects Agency) that has led to an interconnection of many different individual networks into a single large network called Internet.

TCP and IP are actually two protocols in different layers of a communication model. Internet Protocol (IP) is the protocol that provides the packet delivery service. Transmission Control Protocol (TCP) is a connection-oriented protocol that provides a reliable, full-duplex, byte stream for a user process. Most Internet application programs use TCP. Since TCP uses IP in the network layer, the entire Internet protocol suite is often called TCP/IP protocol family.

5.1.3 Berkeley socket

The Berkeley socket used in the current system was first provide with the 4.1cBSD (Berkeley Software Distribution) for the VAX in 1982. The current interface corresponds to the original 4.3BSD release from 1986.

Socket is a form of interprocess communication provided by 4.3BSD that maintain communication between processes on different systems. In the socket, the server is initialised by first creating an endpoint, then binding the address, specifying the queue

and finally waits for connection from client. It also initialises the client by creating an endpoint, binding the address, connecting the server. The socket also handles the data transfer and the termination of the connection.

5.1.4 Checksum problem

The socket may read or write fewer bytes than requested, but the system does not report any error messages. Actually, this is not an error if the buffer limit of the socket in the kernel is reached. All the caller has to do is to invoke the read or write system call again and read or write the remaining bytes. In the current system, a checkpoint is set after each read and write process and remedial action, such as re-invoking the read/write call, processed after each checksum error.

5.2 Use of parallel tool

Parallelising the FEA procedures is accomplished by using the IRIS Power C compiler [GRA00]. The IRIS Power C is a C compiler that analyses sequential codes to determine where loops can be parallelised automatically. The existing sequential codes are recompiled so that the codes can be run efficiently on the multiprocessing ONYX computers. In order to enhance the efficiency of the system, the parallel codes are optimised using different skills so that the processors can be utilised effectively during run time.

5.2.1 Parallel code generation

Using IRIS Power C, the parallel codes are generated automatically. The process of parallel code generation is summarised in **Figure 5.4**.

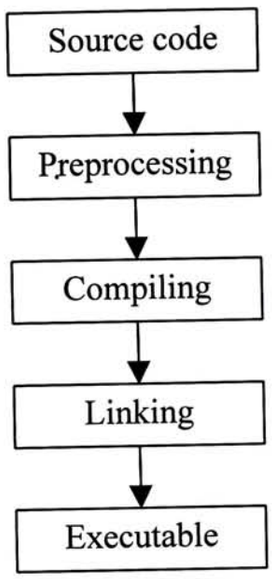


Figure 5.4: Flowchart of parallel executable generation

In the preprocessing stage, the Power C Analyzer (PCA, a C code optimisation pre-processor) manipulates the sequential C code by,

1. Parses the source into an internal representation.
2. Performs data dependency analysis and transformations
3. Generates C source code from the internal representation.
4. Produces C code with parallel directives.

After preprocessing, the multiprocessing compiler generates the object file by,

1. Identifies parallel directives.
2. Rewrites the parallel codes with explicit runtime calls.
3. Processes the C code.
4. Generates the executable object.

After linking the multiprocessing libraries, an executable file is created which can be run on the ONYX system. The executable will adapt to the number of processors present in the system being used.

5.2.2 Optimising parallel code

As most of the optimiser and compilers do, the default options of the multiprocessing compiler are conservative. This prevents undesirable effects resulting from aggressive code optimisation such as elimination of useful codes. In the current system, the most aggressive optimisation option is used. The optimisation at this level generally seeks the highest-quality generated code even if it requires extensive compilation time. In addition, the optimised code may degrade performance occasionally. From the result of the experiments, however, this option proved beneficial to the performance of the system. (See chapter 7 for detail.)

In addition, parallel directives can be used to further optimise the performance. A data-dependency analysis is performed when PCA is running. During the analysis, PCA looks for for-loops with the property that each iteration is data independent. When PCA finds a loop that has the property of data independence, parallel for-loop directive is inserted. Otherwise, the iterations will be performed sequentially. When PCA encounters a nested for-loop, it assumes data-dependency of all the outer loops so that the parallel for-loop directives are only inserted to the innermost loop.

Revisiting the Gauss-Jordan elimination algorithm (**Figure 3.8**), data independent loops are not just the innermost loops for performing row operation. The outer loop (j-loop) is also data independent as the update of row j only depends on the subtraction of the row elements with a constant. The parallel for-loop can be extended to the j-loop. This extension of parallel for-loop improved the performance of FEA by about five times. The time for performing FEA is reduced to 13 seconds for modelling a 267-node object. A 20 times sped up is obtained since the original time for the FEA procedure is 280 seconds for modelling the same object.

6 Implementation and Results

The current system is implemented on a SGI Indigo2 workstation with a R4400 processor, except the finite element analysis (FEA) procedure which is implemented on a SGI Power Challenge 10000 ONYX machine with four R10000 processors.

The system allows the virtual hand to interact with the virtual object by approaching and grasping it from different angles and orientations. The virtual hand is allowed to deform the grasped virtual object until a desired shape is created. Other functions are provided for inspecting the deformed virtual object using a mouse. The virtual object model can be saved or retrieved through the menu functions.

Experiments are performed to evaluate the performance of the FEA procedures. The results of experiments showed that performing the FEA procedure on the ONYX is not always beneficial. The number of vertices of the virtual object has to be limited in order to achieve interactive response.

6.1 Supporting functions

In the proposed system, supporting functions that are not directly related to the interaction are provided through a set of pop-up menu. For example, a "read file" process is invoked by selecting a menu option or by typing in a command from a terminal. Though the process may not be directly related to the interaction between the hand and the virtual object, it is necessary for recording and retrieving the shape of the deformed object.

The pop-up menu in the current system is invoked by clicking the right mouse button. There are several options on the pop-up menu, namely, "*read file*", "*keep shape*", "*save as*" and "*exit*". The appearance of the pop-up menu in the graphics window is shown in **Figure 6.1**.

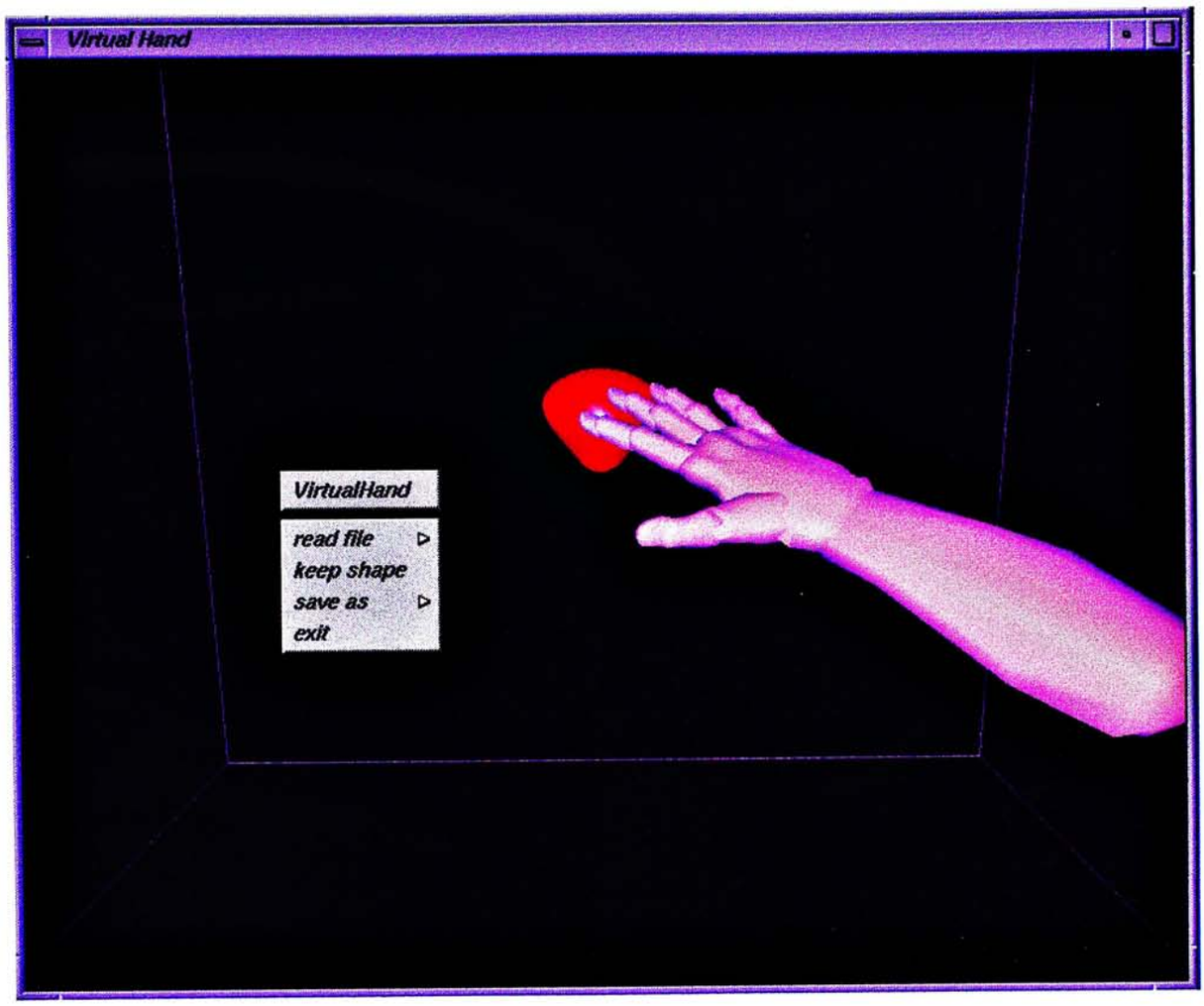


Figure 6.1: Pop-up menu in the graphics window

6.1.1 Read file

The function of this option is to read in a file specified by the user in the command prompt. The file being read contains vertices, elements and display information of the virtual object. Then, a sphere tree for collision detection between the virtual object and the virtual hand is constructed. (See chapter 4 for detail) The virtual object is then displayed in the virtual environment.

6.1.2 Keep Shape

If the shape of the deformed object is to be used as an undeformed object, the shape of the object can be kept in memory by choosing the option "*keep shape*" in the pop-up menu. The sphere-tree for collision detection will be reconstructed. Then, the object with its new shape will be detached from the virtual hand and will be returned to the origin while the vertex list is saved in the memory. The new object can be rotated by clicking and dragging the left mouse button.

6.1.3 Save as

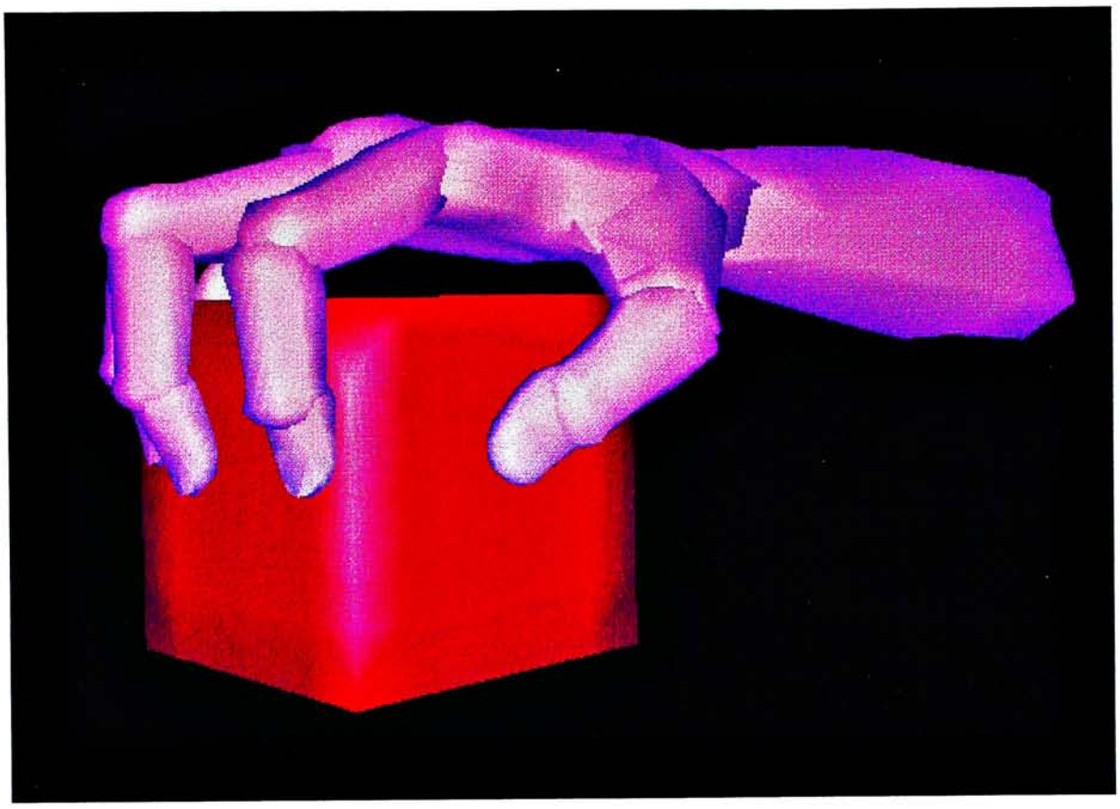
When the "*save as*" option is invoked, the vertex, element and display list of the deformed object are saved to a file. The system will convert the current virtual object model into text format which is then written to a file. If the file to be saved does not exist, the process will create one. If the file exists, the process will overwrite the file by the current information.

6.1.4 Exit

This is a process invoked by selecting the option "*exit*" in the main menu. This process controls the termination of the program and closes all the graphics windows. The program will be terminated immediately after all the graphics windows are closed.

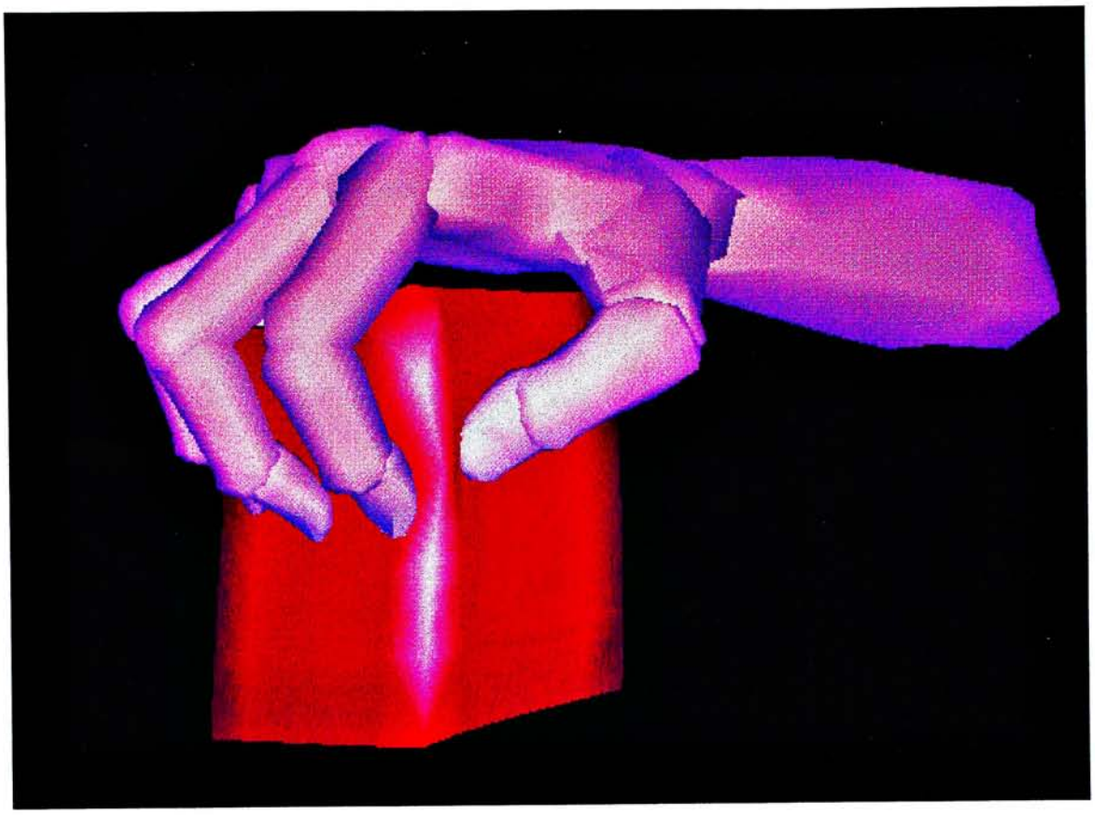
6.2 Visual results

Figure 6.2 to **Figure 6.4** shows the snapshots of the virtual hand deforming different virtual objects.

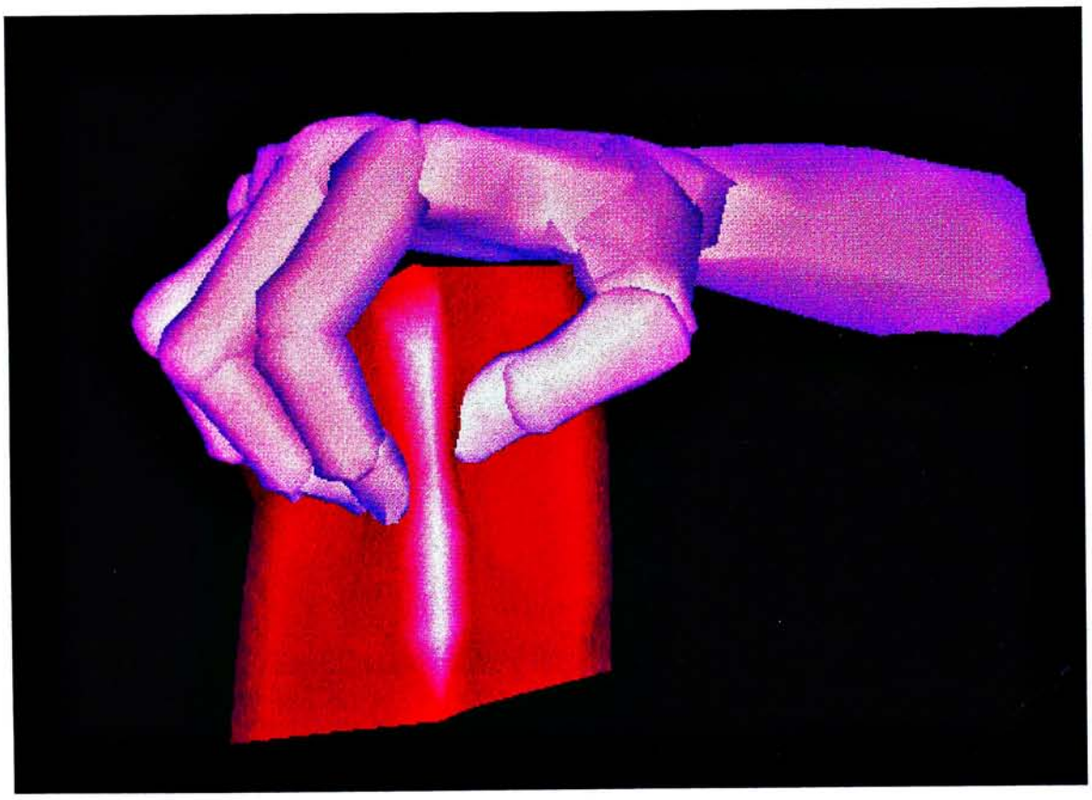


(a) Cube with no deformation

(b) Cube with small deformation

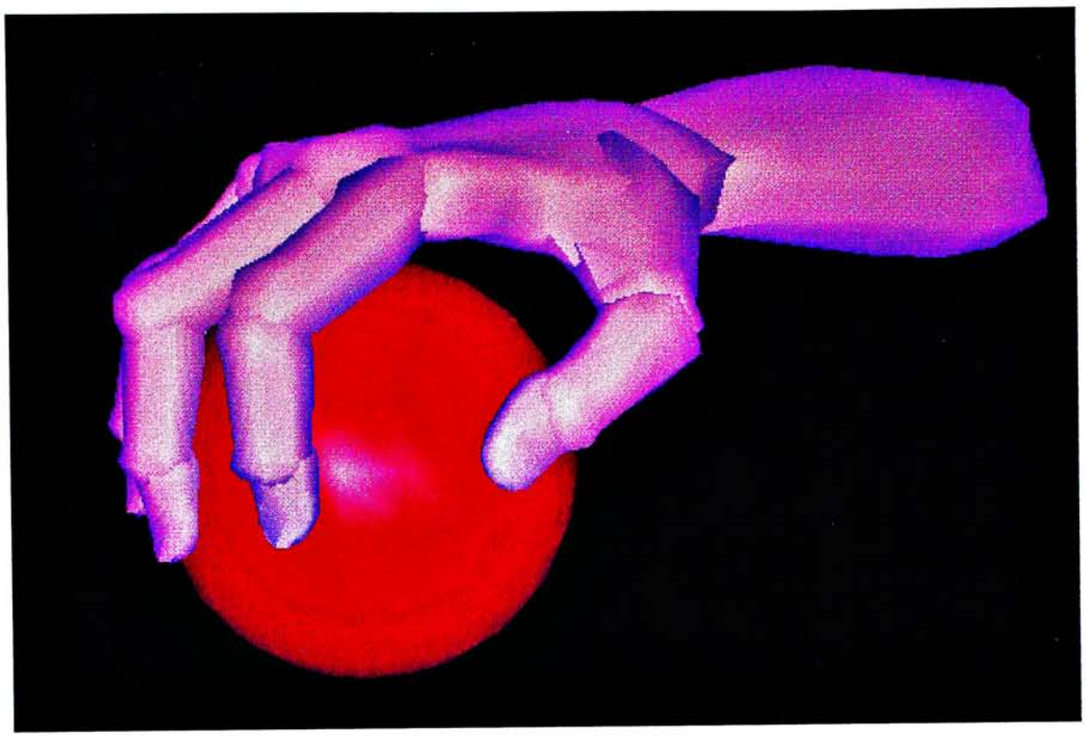


(c) Cube with medium deformation

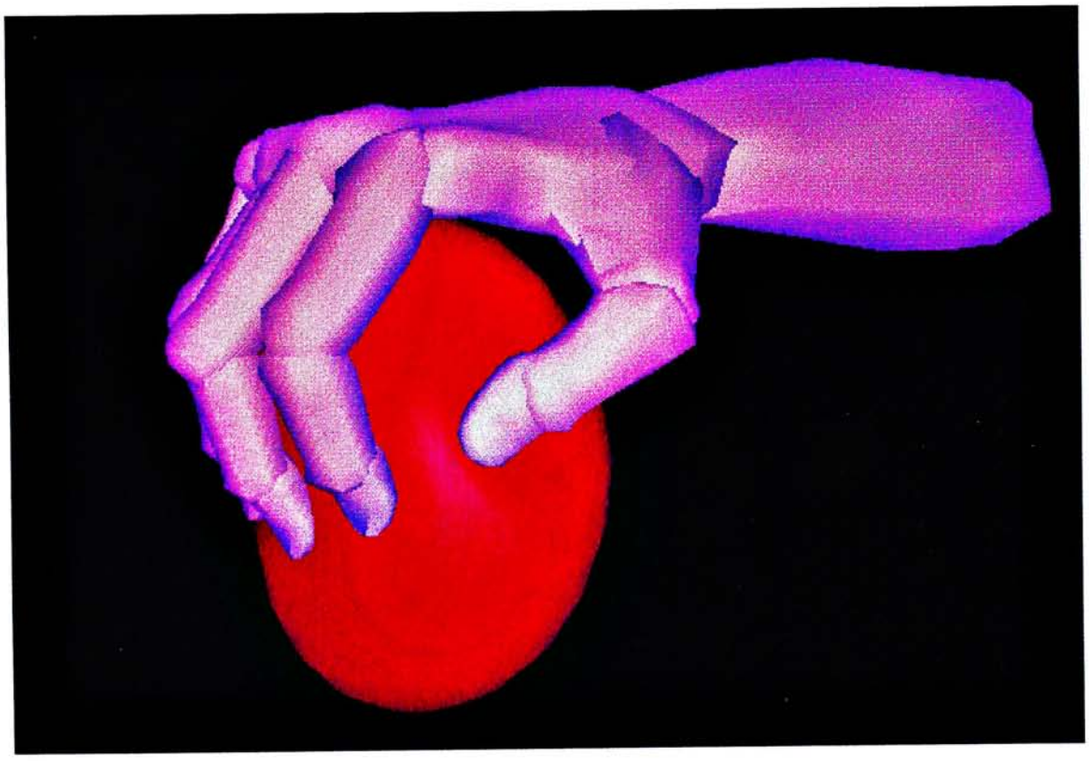


(d) Cube with large deformation

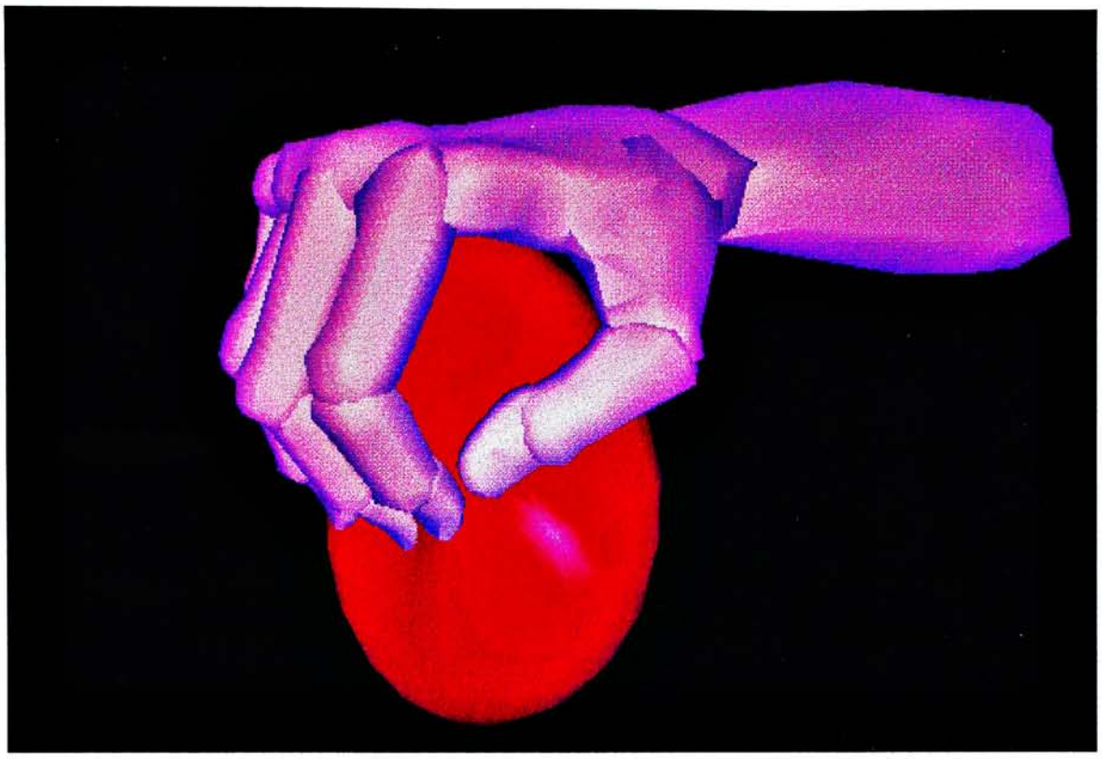
Figure 6.2: The interaction with a virtual cube



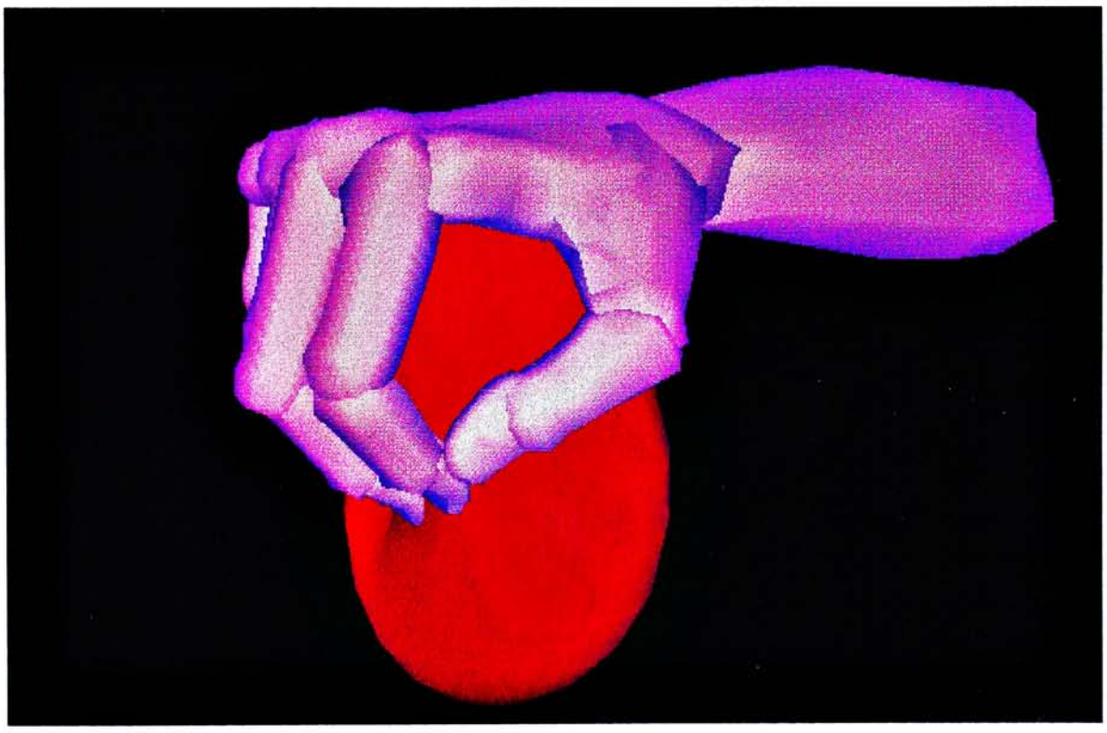
(a) Sphere with no deformation



(b) Sphere with small deformation

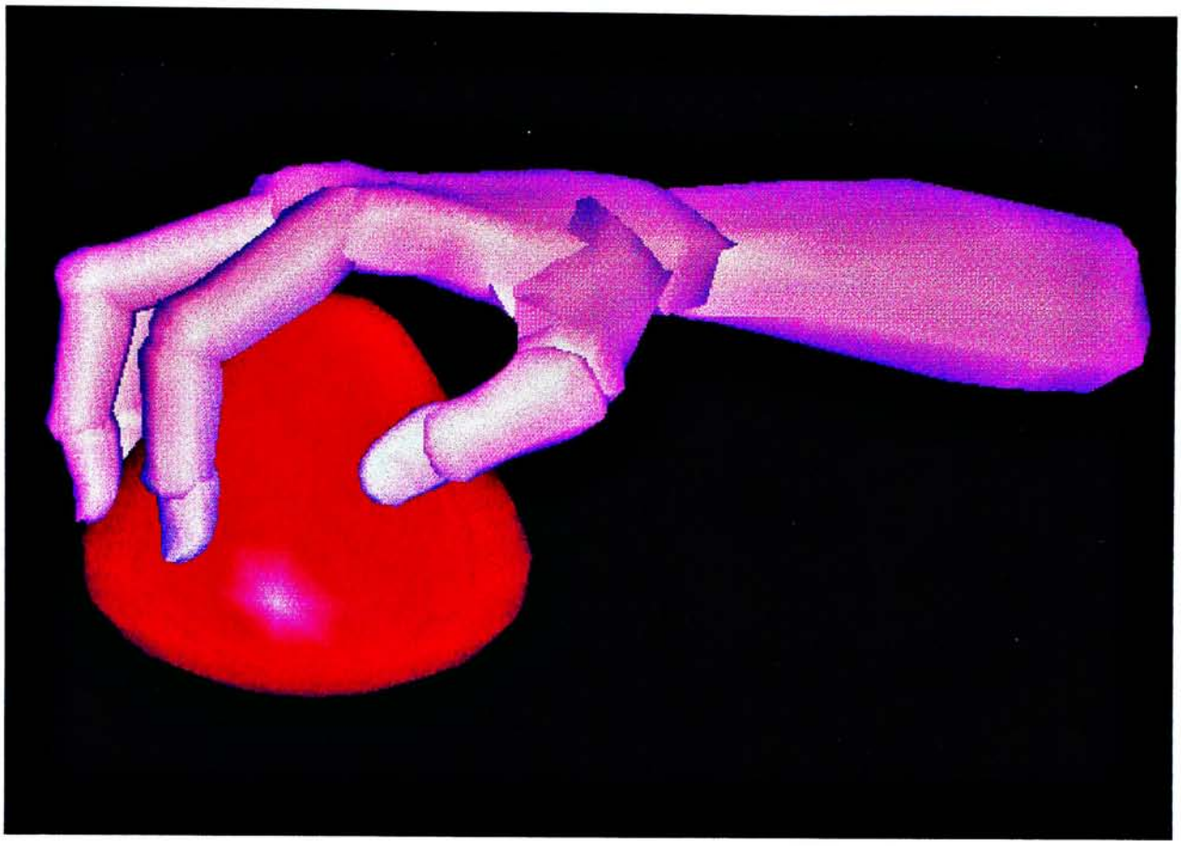


(c) Sphere with medium deformation

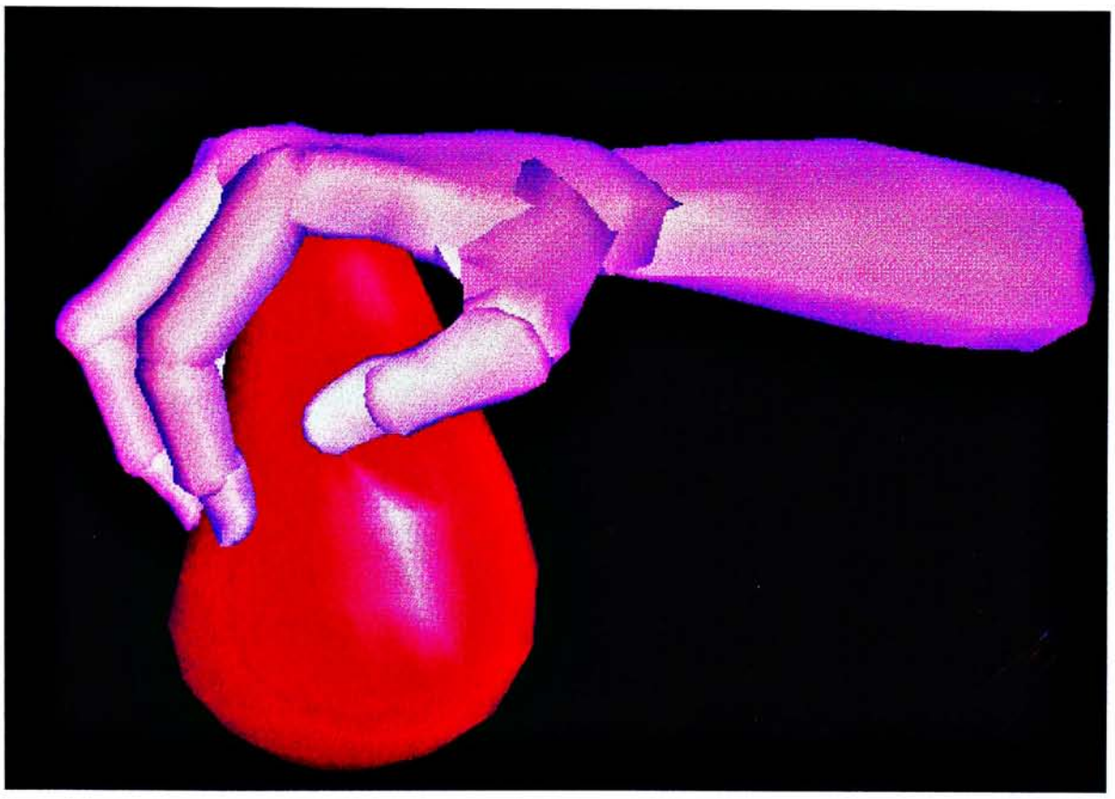


(d) Sphere with large deformation

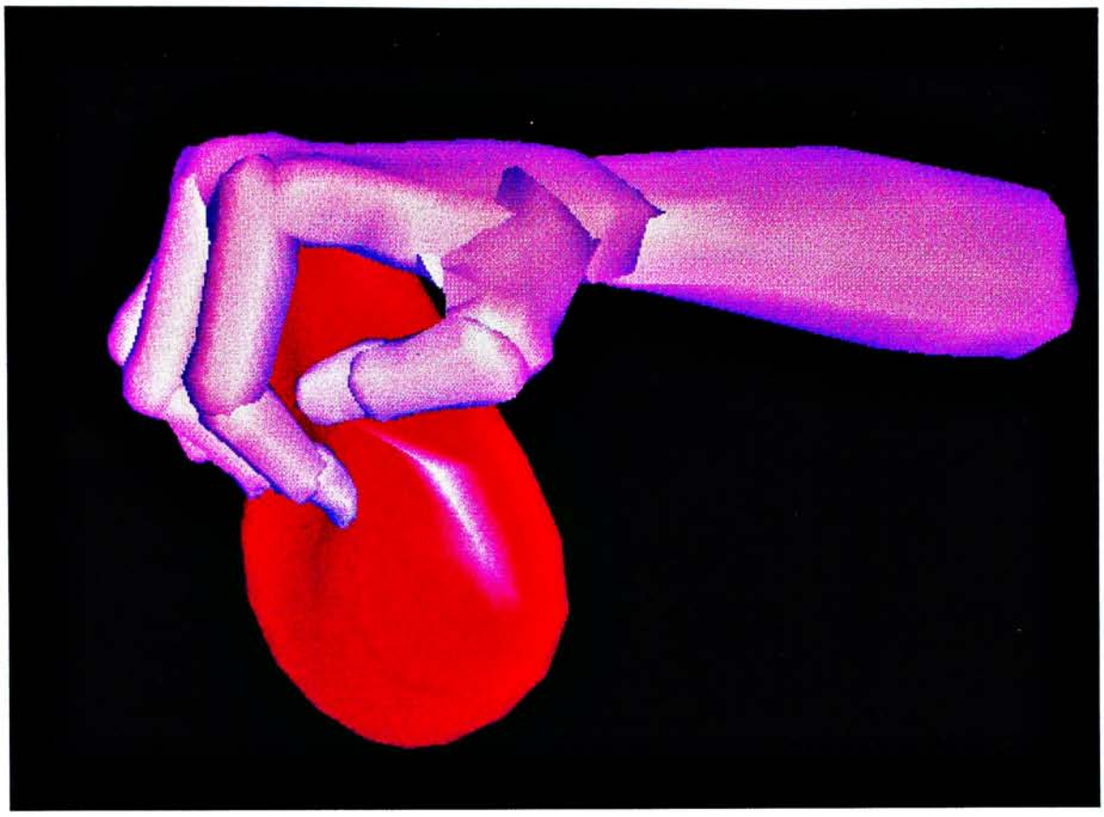
Figure 6.3: The interaction with a virtual sphere



(a) A strawberry without deformation



(b) A strawberry with medium deformation



(c) A strawberry with large deformation

Figure 6.4: The interaction with a virtual strawberry

(a) A toothpaste without deformation

(b) A toothpaste is bent by the hand

(c) A toothpaste is squeezed by the hand

Figure 6.5 The interaction with a toothpaste

6.3 *An operation example*

Figure 6.6 to 6.12 shows an example of operation of the system performing repeating deformation, read and write action, and the keep shape function.

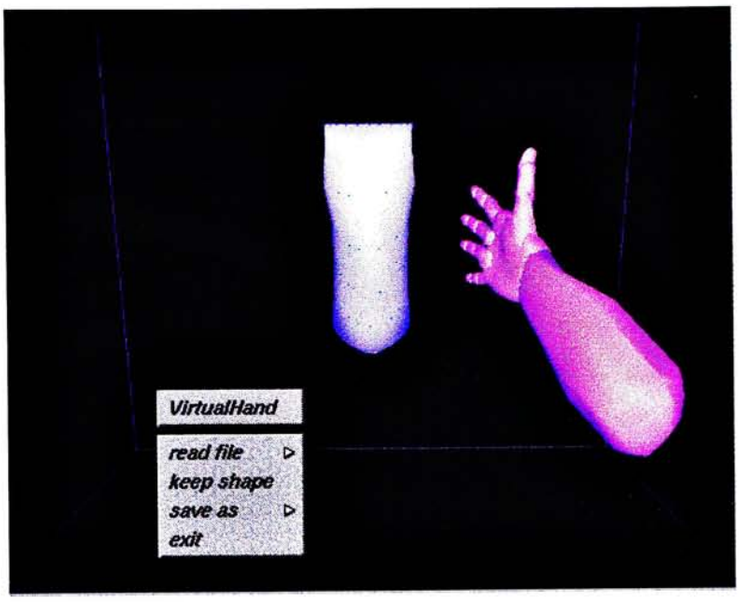


Figure 6.6: A toothpaste is imported from the read file option of the pop-up menu.

Figure 6.7: The system will prompt the user to enter an object name for the object to be retrieved

Figure 6.8: The hand tries to approach to the object

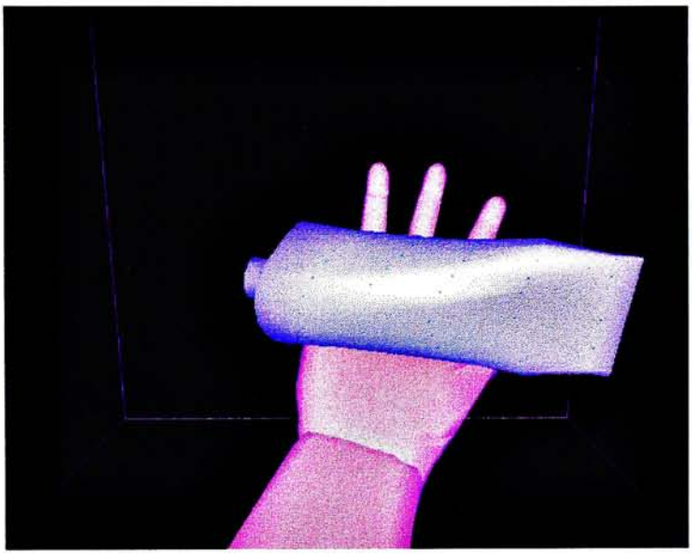


Figure 6.9: The object is attached to the hand

Figure 6.10: The object is deformed by the hand

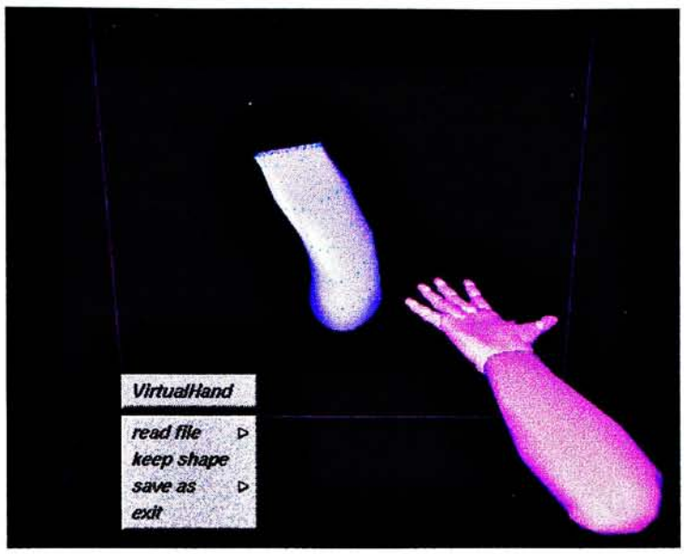


Figure 6.11: The shape of the new object is saved by the save as option in the pop-up menu

Figure 6.12 The system will prompt the user to enter a name for the object

Figure 6.13 The hand tries to approach to the new object from another direction

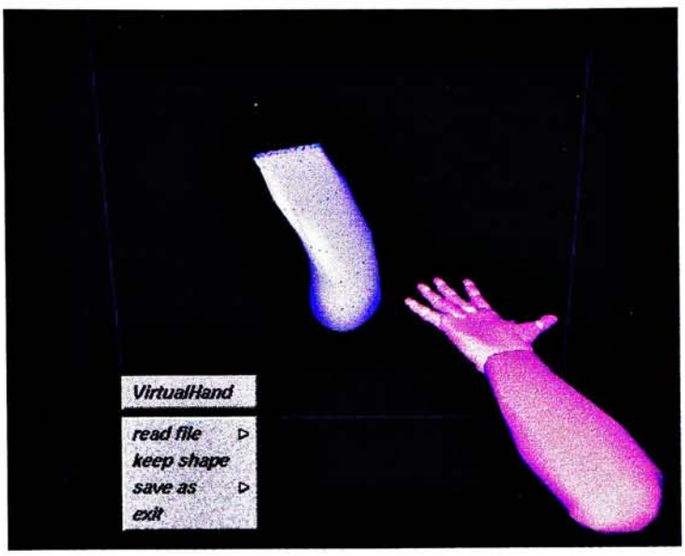


Figure 6.11: The shape of the new object is saved by the save as option in the pop-up menu

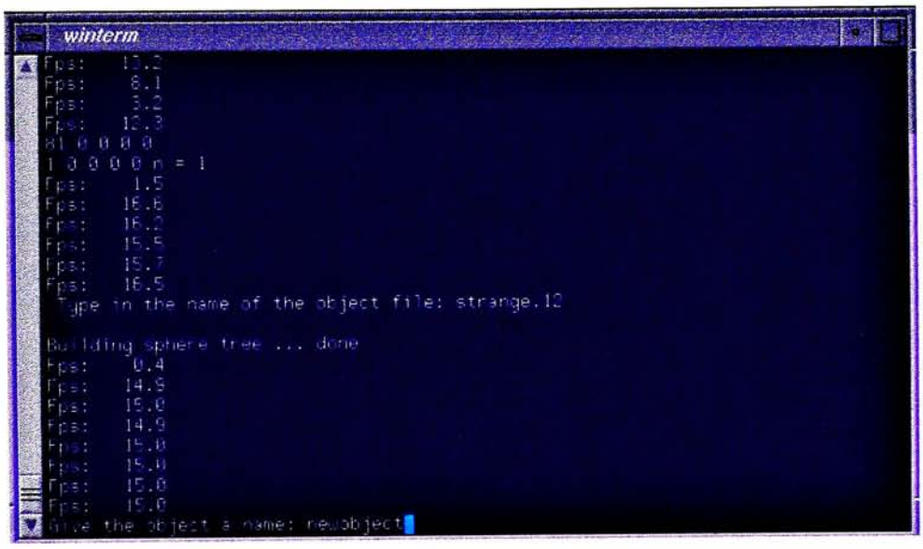


Figure 6.12 The system will prompt the user to enter a name for the object

6.4 *Performance of parallel algorithm*

Comparisons have been made to compare the performance for different optimisation techniques and different number of object vertices.

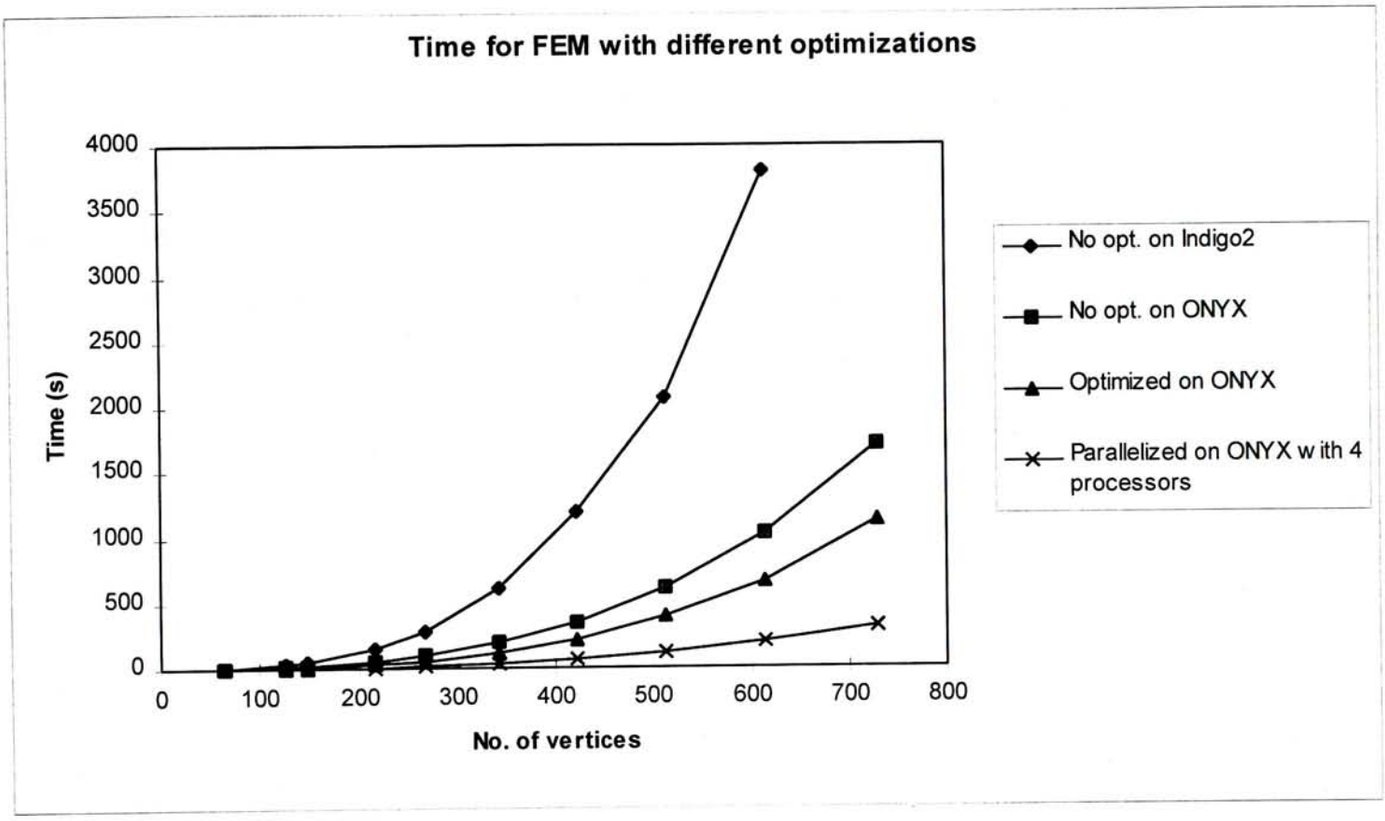


Figure 6.13: The comparison of FEA computation time

From **Figure 6.13**, it is observed that the computation time required on the ONYX system is much shorter than that of the Indigo2 when the codes are run sequentially with the same optimisation level. The difference is due to the hardware limit of Indigo2. The computation time required for executing the parallel codes is shorter than that of the optimised sequential codes on the ONYX system.

The shape of the virtual object does not affect the computation time of the algorithm. In the computation of matrix inverse, the computation time depends on the matrix size and hence the number of vertices of the virtual object.

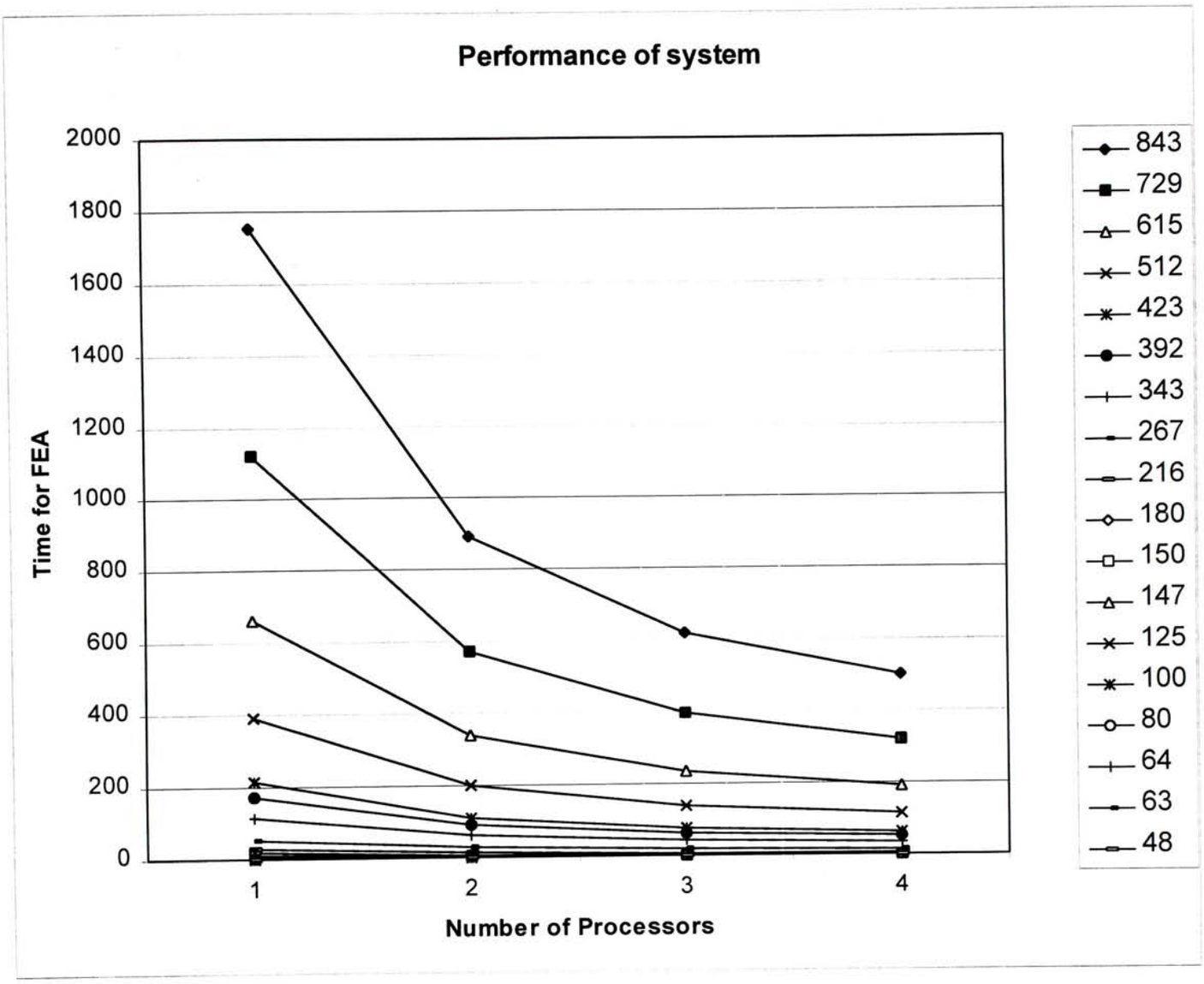


Figure 6.14: The comparison of computation time for different number of processors

From **Figure 6.14**, the computation time required for FEA decreases with increasing number of processors. The rate of reduction of computation time decreases with the increasing number of processors.

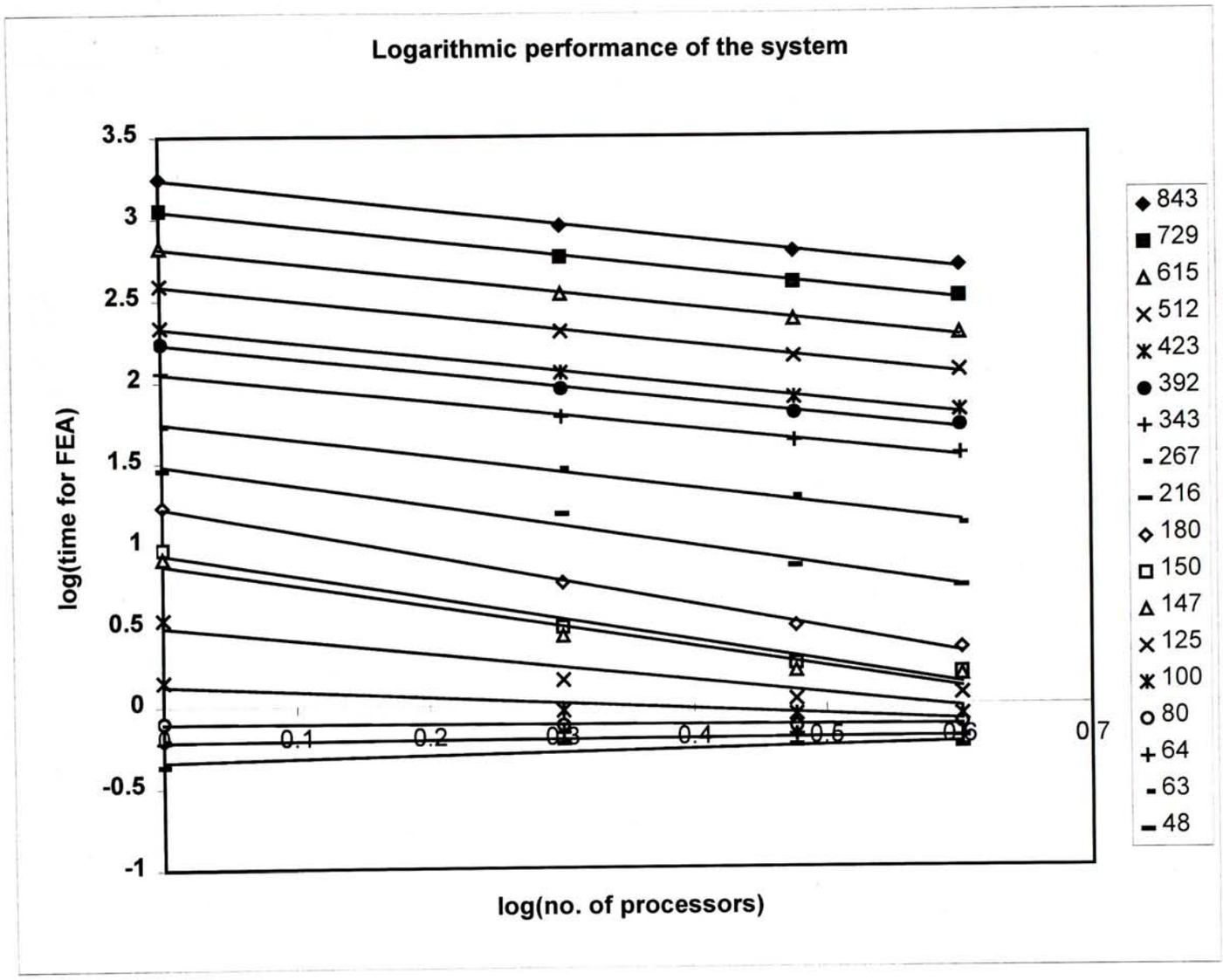


Figure 6.15: Logarithmic relation of computation time and number of processors

When the logarithm of the time for FEA and the number of processors is plotted, linear relationships are observed.

Another observation is that the trendlines are nearly parallel for the series representing the number of vertices larger than 300. In **Figure 6.16**, the slopes of trendlines are plotted against the number of vertices.

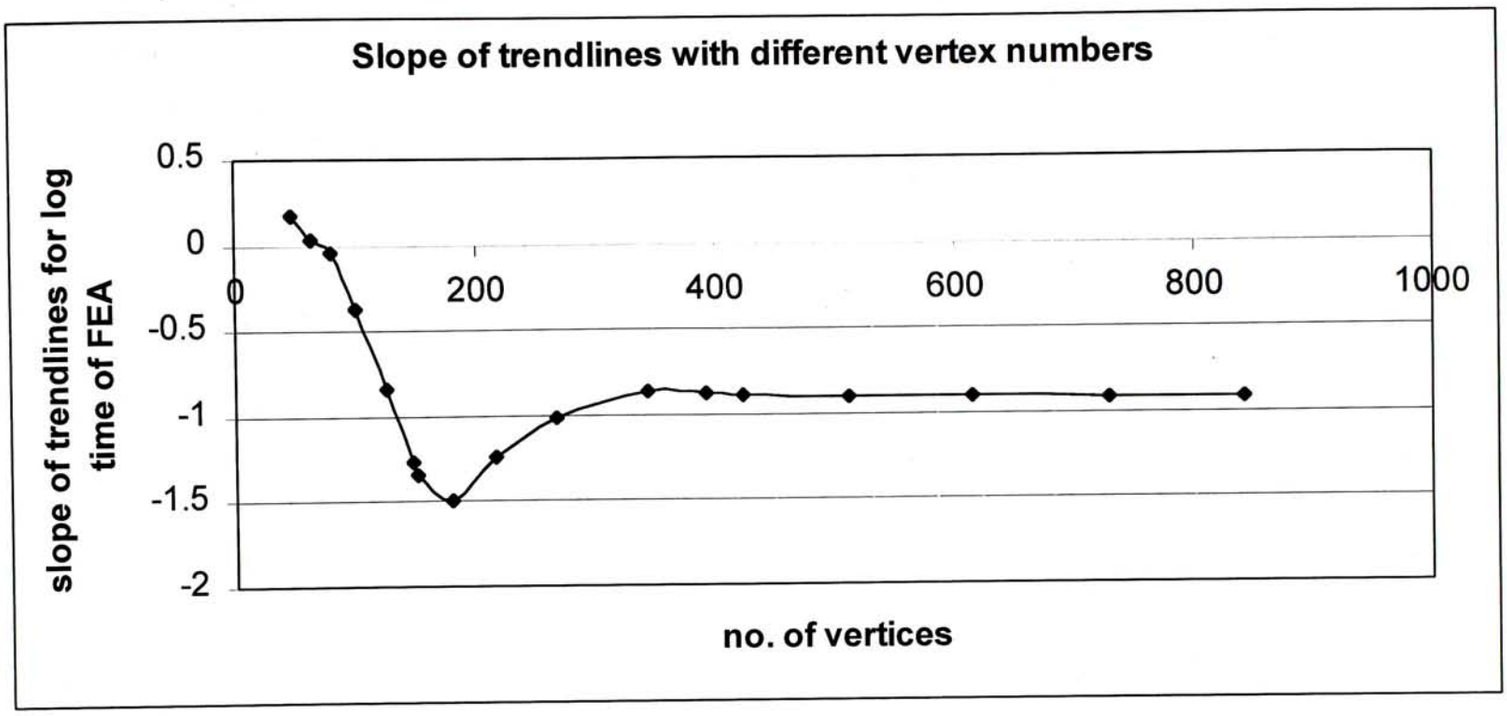


Figure 6.16: Relation of slope of trendlines in **Figure 6.15** and the number of vertices

The slope of the trendlines decreases from 0.18 to a minimum value of about 1.5 when the number of vertices increases from about 50 to 200. Then the slope increases again and reaches a steady value of about 0.9 when the number of vertices is larger than 400.

When the log-log relation between the computation time of FEA and the number of processors used in the procedure is approximated by a straight line, the actual function approximating the time and number of processors in normal scale is given by, (derivation see Appendix B)

$$T = kN^{m} \tag{6.1}$$

where T is the computation time of FEA,

k is a constant,

N is the number of processors,

m is the slope of the trendline when the time- processor relation is plotted in log scale.

From **Figure 6.16**, the value of m is nearly zero when the number of vertices is small. From **(6.1)**, the value of N^m is about one when m is about zero and the value of T will be independent of the value of N. In other words, the computation time is independent of the number of processors used in the procedure. Hence, the parallel algorithm is not efficient when the number of vertices is small.

When the number of vertices is larger than 100, the value m decreases and becomes negative. The negative value of m, from **(6.1)**, implies a decrease in time for performing FEA when the number of processors is increased. Therefore, the efficiency of the parallel algorithm continues to improve with increasing number of vertices. The improvement of efficiency is due to the declining effect of computation overhead for parallel processing when the number of vertices increases.

When the number of vertices is larger than 400, the value of m comes to a steady state and is equal to -0.9 approximately. If the time for FEA procedure is inversely proportional to the number of processors used, the value of m should be equal to -1. The value -0.9 implies that the time-processors relation is nearly inversely proportional since the time complexity for performing FEA is given by,

$$\frac{2n^2(3n-1)}{N}$$

where n is the size of the global stiffness matrix $\mathbf{k}^g$.

N is the number of processors.

For fixed n, the time complexity is thus $O\left(\frac{1}{N}\right)$, an inversely proportional relation.

To sum up, several variables are affecting the performance of the FEA procedures,

- Parallel algorithm – the algorithm becomes more efficient when there are more parallel loops in the procedure.
- Computation overheads for parallel processing – this is a competing factor with the parallel algorithm, the computation overhead increases with the increase in the number of parallel loops in the procedure.

- Number of processors – the efficiency of the system is expected to increase with increasing number of processors. However, this is not always true. There is not much improvement when the number of processors increases to certain level. Fortunately, the decay of improvement is predictable in the current system. The number of processors used for the task can be optimised using the existing trend of performance. For example, performing FEA for an object with 216 vertices within 0.5 seconds, 27 processors should be used by extrapolating the log-log relation trendline with 216 vertices in **Figure 6.15**.

7. Conclusion and Future Work

7.1 Conclusion

A system for interacting with virtual objects in the virtual world by using an instrumented glove has been developed. The system is found to provide interactive response in the manipulation of virtual objects. The performance of the most time consuming task of finite element analysis (FEA) is improved by porting the analysis algorithm to the Power Challenge 10000 multi-processor computer (ONYX).

Tetrahedral solid elements are adopted for modelling the deformable virtual object. The stiffness matrix is partitioned and the linear equations describing the force-displacement relation is rearranged so that deformation of the object can be estimated according to movement of the fingers. Gauss-Jordan elimination is adopted for computing the inverse of the relation matrix.

A sphere tree is employed for detecting collision between the virtual object and the virtual hand. Collision is detected by locating the leaf nodes of the tree where sphere of the node collides with an approximation of the virtual hand. Decision rules for attachment of the virtual object to the virtual hand are developed.

A data communication algorithm is also developed for distributing the tasks of FEA computation to the parallel machine. This algorithm transfers data between the client computer, an Indigo2 machine, and the server computer (the ONYX). The matrix inversion procedures are parallelised by using IRIS Power C. The speed for performing FEA is 22 times faster than the serial version running on the Indigo2 for 200 to 300 node.

Experiments showed that the time for performing FEA is inversely proportional to the number of processors used. The system performance is thus expected to be improved if more processors are available in the server computer (the ONYX). The time for performing FEA decreases exponentially when more processors are added to the system.

7.2 Future Work

The algorithm adopted for matrix inversion in the FEA procedure is the Gauss-Jordan algorithm. It is expected that performance of the system can be improved by taking the band-symmetric matrix property into consideration, e.g., using LU-decomposition for matrix inversion.

In the current system, parallelisation relies on the compiler of the ONYX system that mainly attains data parallel operations. It is envisaged that higher degree of parallelism can be attained by parallelising the algorithms.

The same approach can be extended to include elasto-plastic and plastic behaviour of virtual objects. In this case, model capable of describing the behaviour of the virtual object under large deformation has to be used. However, this will lead to non-linear equations so that iterative methods have to be adopted for solving the equations. This will have adverse effect on the performance and accuracy of the system.

Different analysis methods, e.g., finite difference method, may be adopted to explore possible alternatives for modelling deformable object. Head mount and force feedback devices may be employed to improve the user-interface.

Reference:

[BAN95] Srikanth Bandi and Daniel Thalmann, "An Adaptive Spatial Subdivision of the Object Space for Fast Collision Detection of Animated Rigid Bodies", in Eurographics '95, pp.C-239-270.

[BRO96] Morten Bro-Nielsen, Stephane Cotin, "Real-time Volumetric Deformable Models for Surgery Simulation using Finite Elements and Condensation", pp. C57-C66, Eurographics '96, volume 15, number 3, 1996.

[CHA91] Tirupathi R. Chandrupatla, Ashoh D. Belegindu, "Introduction to Finite Elements in Engineering", Prentice Hall Inc., New Jersey, 1991.

[GOU89] Jean-Paul Gourret, Nadia Magnanat Thalmann, Daniel Thalmann, "Simulation of Object and Human Skin Deformations In a Grasping Task", pp. 21-30, Computer Graphics, volume 23, number 4, 1989.

[GOU91] Jean-Paul Gourret, Nadia Magnanat Thalmann, Daniel Thalmann, "Modeling of contact deformations between a synthetic human and its environment", pp. 514-520, Computer Aided Design, volume 23, number 7, 1991.

[GRA00] David Graves, "IRIS Power C User's Guide", Silicon Graphics, Inc., Mountain View, California.

[HUA95] Zhiyong Huang, Ornan Boulic, Nadia Magnenat Thalmann, Daniel Thalmann, "A Multi-sensor Approach for grasping and 3D Interaction", conference proceedings of CGI'95, June 1995, Leeds UK, pp. 235-254.

[HUB96] Philip M. Hubbard, "Approximating Polyhedra with Spheres for Time-Critical Collision Detection", ACM Transactions on Graphics, volume 15, number 3, July 1996, pp.179-210.

[HUI97] K. C. Hui, M. C. Ma, "Interacting with a virtually elastic object", pp.243-248, Proceedings of International Conference on Maunfacturing Automation, number 1, 1997.

[KAN96] HoSeok Kang, Avi Kak, "Deforming virtual objects interactively in accordance with an elastic model", pp. 251-262, Computer Aided Design, volume 29, number 4, 1996.

[LIU91] Yun-Hui Liu, Saguru Arimoto and Hiroshi Noborio, "A New Solid HSM and Its Application to Interference Detection between Moving Objects", Journal of Robotic Systems, 1991, pp. 39-54.

[MCL92] Patricia McLendon, "Graphics Library Programming Guide", Silicon Graphics Inc., Mountain View, California, 1992.

[MOR88] M. Moore and J. Wilhelms, "Collision Detection and Response for Computer Animation", Computer Graphics (Proceedings of Siggraph), volume 22, number 4, August, 1988, pp.289-298.

[PAL96] I. J. Palmer and R. L. Grimsdale, "Collision Detection for Animation using Sphere-Trees", Computer Graphics Forum, volume 14, number 2, pp.105-116.

[POL94["Polhemus 3Space® Users Manual", Polhemus Incorporated Colchester, Vermount, 1994.

[RAP96] Avi Rappoport, Alla Sheffer, Michel Bercovier, "Volume-Preserving Free-Form Solids", IEEE Transactions on Visualization and Computer Graphics, volume 2, number 1, 1996.

[SMI95] Andrew Smith, Yoshifumi Kitamura, Haruo Rakemura and Fumio Kishino, "A Simple and Efficient Method for Accurate Collision Detection Among Deformable Polyhedral Objects in Arbitrary Motion", Proceedings of IEEE Virtual Reality Annual International Symposium '95, pp.136-145.

[STE91] W. Richard Stevens, "UNIX® Network Programming", Prentice Hall International, New Jersey, 1991.

[TER87] Demetri Terzopouluos, John Platt, Alan Barr, Kurt Fleischer, "Elastically Deformable Models", pp. 205-214, Computer Graphics, volume 21, number 4, 1987.

[THA95] Nadia Magnanat Thalmann, Daniel Thalmann, "Finite elements in task-level animation", pp. 227-242, Finite Elements in Analysis and Design, volume 19, 1995.

[VIR94] "CyberGlove™ User's Manual", Virtual Technologies, Palo Alto, 1994.

[YOU93] Ji-Hoon Youn and K. Wohn, "Realtime Collision Detection for Virtual Reality Applications", in Proceedings of IEEE Virtual Reality Annual International Symposium '93, pp.415-421.

Appendix A - Matrix inversion

From **(3.40)**, $\mathbf{K_{11}}$ is required to be inverted so that the unknown displacements can be computed by **(3.40)**. In this thesis, Gauss-Jordan elimination is used to compute the inverse of $\mathbf{K_{11}}$. The algorithm is summarised in **Figure A.1**.

```
n×n_matrix      global;
n×n_matrix
procedure matrix_inverse(global)
{
integer  i, j, k;
n×n_matrix      inverse;

Initialise inverse as identity matrix

for i = 1 to n by 1
 Normalise row i of global by dividing it with diagonal element global(i,i)
 Divide row i of inverse by global(i,i)
 for j = 1 to n by 1
   if (i ≠ j)
     1. Eliminate ith element of row j in global by:
        Row j of global = row j of global - (global(j,j)•row i of global)
     2. Row j of inverse = row j of inverse – (global(j,j)•row i of inverse)
   end if
 end for (j-loop)
end for (i-loop)

return inverse;
}
```

Figure A.1: Pesudo-code of Gauss-Jordan elimination

Before the elimination, an identity matrix (matrix 2) is created for computing the inverse. This matrix is written side by side with the stiffness matrix as shown in **Figure A.2**.

$$\left[\begin{array}{cccc|cccc} x_{11} & x_{12} & \cdots & x_{1n} & 1 & 0 & \cdots & 0 \\ x_{21} & x_{22} & & x_{2n} & 0 & 1 & & 0 \\ \vdots & & \ddots & \vdots & \vdots & & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} & 0 & 0 & \cdots & 1 \end{array}\right]$$

Figure A.2: Matrix initialization

Two processes are performed repeatedly from the first row (row 1) to the last row (row n) of both matrices, which are illustrated in **Figure A.3**.

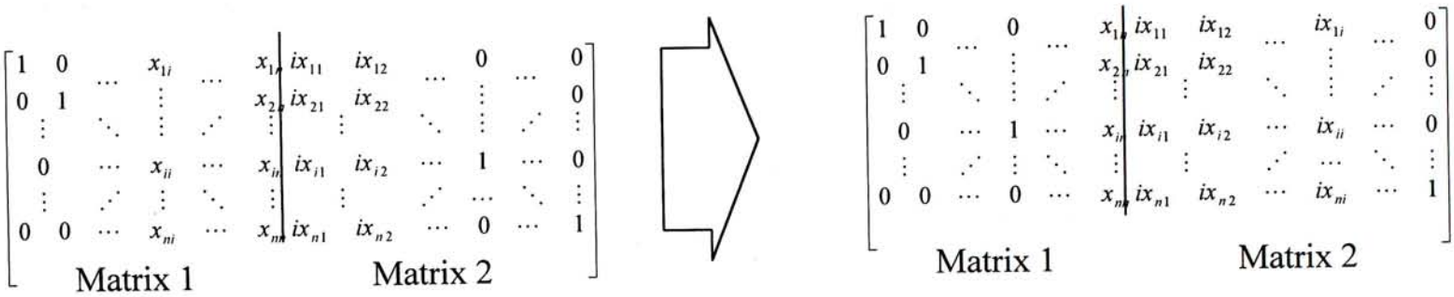


Figure A.3: Row operations and normalisation

a. Normalise the current row (row *i*) of matrix 1 by dividing the row elements by the diagonal element at column *i* of matrix 1. Divide each elements of row *i* in matrix 2 by the same value, i.e. the of diagonal element of matrix 1 of row *i*. The equation describing this process is given as,

$$x_{ij} = \frac{x_{ij}}{x_{ii}}$$
$$ix_{ij} = \frac{ix_{ij}}{x_{ii}} \qquad \textbf{(A.1)}$$

where $j = 1, 2, \ldots, n$ is the column number of the matrices.

b. Eliminate the other elements of column *i* in matrix 1 by row operation. For example, if row *k* is to be eliminated, the first step is to multiply row *i* by the value of the element at column *i*, row *k* (x_{ki}). Then, subtract the whole row from row *k*. The element at column *i* in row *k* (x_{ki}) will be eliminated. Perform the same operation for matrix 2. The equation describing this process is,

$$x_{kj} = x_{kj} - x_{ki} \times x_{ij}$$
$$ix_{kj} = ix_{kj} - x_{ki} \times ix_{ij} \qquad \textbf{(A.2)}$$

where $j = 1, 2, \ldots, n$ is the column number of the matrices.

$k = 1, 2, \ldots, i\text{-}1, i\text{+}1, \ldots, n$ is the row number of the matrices.

When the processes are completed, matrix 1 will be eliminated to an identity matrix. Meanwhile, matrix 2 will be the inverted matrix of the global stiffness matrix.

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & \\ 0 & 1 & \cdots\ 0 & \mathbf{K_{11}^{-1}} \\ \vdots & & \ddots\ \vdots & \\ 0 & 0 & \cdots\ 1 & \end{array}\right]$$

Figure A.4: Final status of matrix 1 and 2

Appendix B - Derivation of **(6.1)**

Suppose the time of performing FEM is T,

the number of processors is N,

the slope of a trendline in **Figure 6.13** is m,

the constant of the actual time-processors relation is k,

the intercept of a trendline in **Figure 6.13** is $c = \log k$.

From **Figure 6.13**, the equation of trendlines are given by,

$$\log T = m \log N + c \tag{B.1}$$

Rearranging **(B.1)**,

$$\begin{aligned} \log T &= \log N^m + c \\ &= \log N^m + \log k \\ &= \log\left(N^m \times k\right) \\ &= \log kN^m \end{aligned} \tag{B.2}$$

Anti-loging both sides, the relation becomes,

$$T = kN^m$$

which is the equation **(6.1)**.

Appendix C – Derivation of (6.2)

The time complexity for performing FEA in parallel machine can be derived from the parallel algorithm itself, which is summarised in **Figure C.1**.

Denote n - the size of the global stiffness matrix $\mathbf{k}^g$.

N - the number of processors.

G – a n×n matrix

R – the resulting matrix (becomes inverse of **G** finally)

```
n×n matrix  procedure matrix_inverse(G)
{
Initialise R as identity matrix
for i = 1 to n by 1
# DO in parallel loops with N processors
    Normalise row i of G by dividing it with diagonal element G(i,i)        (2n)
    Divide row i of R by G(i,i)                                             (2n)
    for j = 1 to n by 1
      if (i ≠ j)
        1. Eliminate ith element of row j in G by:                          (3n)
           Row j of G = row j of G - (G(j,j)•row i of G)                    (n-1)   (n)
        2. Row j of R = row j of inverse – (G(j,j)•row i of R)              (3n)
      end if
    end for (j-loop)
# END OF parallel loops with N processors
end for (i-loop)
return R;
}
```

Figure C.1: Pesudo-code of current matrix inverse algorithm

From the figure, there are 3 operations performed in the row operation of the matrix **G** and the matrix **R** in the j-loop. Since there are n elements for a row in each matrix (**G** and **R**), there are 3×n operations for one row operation step in the j-loop. As there are two such row operation steps in the j-loop, the number of operations are added

together and the total number of operations for each iteration in the j-loop is 6n operations.

In the j-loop, the elimination process is omitted for the normalized row as it is already updated. Therefore, the number of iteration that row operations are performed is n-1 and the total number of operation performed in the nested j-loop is,

$$(n-1)(6n) \tag{C.1}$$

Apart from the j-loop, the i-th row in the matrix **G** is normalized. The step takes 2 operations and there are 2n operations. Similarly, updating the i-th row in matrix **R** takes 2n operations. The number of steps taken is,

$$4n \tag{C.2}$$

As the j-loop and the steps of updating the i-th row are in the i-loop, the steps are added together and the total number of steps is,

$$(n-1)(6n) + 4n = 6n^2 - 2n \tag{C.3}$$

The operations in the i-loop are performed in parallel with N processors, the time taken for the operations should be divided by the number of processors and becomes,

$$\frac{6n^2 - 2n}{N} \tag{C.4}$$

Simplifying **(C.4)**, the expression becomes,

$$\frac{2n(3n-1)}{N} \tag{C.5}$$

The outermost loop in the elimination process is the i-loop. There are n iterations in the loop which are performed sequentially. The time complexity for the whole elimination process is thus,

$$n \times \frac{2n(3n-1)}{N}$$
$$= \frac{2n^2(3n-1)}{N}$$