Faculty of Computer Science and Information Technology
University of Malaya
Kuala Lumpur

Cloth Simulation

By

Nor Azlina Binti Mohamad Salleh
WEK 000266

Project Supervisor : Mr Amirrudin Bin Haji Kamsin
Project Moderator : Mrs Nornazlita Binti Hussin

A project report submitted to the
Faculty of Computer Science and Information Technology
University of Malaya
Kuala Lumpur

Dissertation submitted in partial fulfillment of the requirement for the
Degree of Bachelor of Computer Science
Session 2003/2004

Cloth Simulation

WXES 3181

ABSRTACT

With cloth simulation, as in most of Computer Graphics, the challenge lies in making the artifact on the screen look realistic. In particular, the cloth's movements and interactions with its surroundings ought to look familiar to the viewer, who should even be able to recognize different types of cloth simply by watching the simulation's behaviour.

In the literature, realistic cloth movement has been obtained using a network of particles connected by springs (in that case, the system is referred to as a *mass-spring system*) or other natural forces. With such a model, perturbations at one cloth vertex are propagated to the other areas of the cloth as dictated by the system of differential equations that links the forces acting on all the particles.

Any cloth simulation system needs efficient numerical methods for integrating the equations that describe the mechanical behaviour of the discrete representation of the cloth. Choosing the adequate method should be done with full knowledge of the advantages and weaknesses of the main techniques.

Most simple simulators use *explicit integration*, which is easy to implement and fairly fast for small pieces of cloth. Freecloth uses *implicit integration*, which is harder to implement but gives much better results for large pieces of cloth.

Implementation was done in OpenGL. It has been compiled on Linux and under Windows, and it should be straightforward to port to other operating systems.

ACKNOWLEDGEMENT

In the name of God, Most Gracious, Most Merciful, AlhamduliLlah, thanks to **ALlah**, The Almighty for giving me the strength and confidence to complete this project.

First and foremost I would like to express my gratitude to the Faculty of Computer Science and Information Technology (FCSIT), University of Malaya for giving me an opportunity to do my final year project. FCSIT has provided valuable and good facilities and commitments to assist me in completing my final year project report successfully and smoothly.

My greatest thank you and appreciation to Mr. Amirrudin Bin Kamsin, my remarkable supervisor for whom I respect and salute to the utmost degree. Thank you for giving this golden opportunity to do this project under his privilege supervision. Your numerous and varied tasks as well as guidance has contributed a lot in my self-improvement.

Special thanks to Mrs. Nornazlita Binti Hussin, my thesis moderator for taking off attending my VIVA presentation and putting efforts in his valuable query and questions to me. Without him, I would not have made this thesis report a success.

My most gratitude also goes to my colleagues, coursemates and housemates, for their comments and advice throughout this project. Thank you very much for your suggestions and ideas which had further enhance the value of Cloth Simulation.

Last but foremost , my warmest thank you to my parents , Mr. Mohamad Salleh Bin Selamat and Mrs. Suleha Bin Selama, for all the supports materially and non-materially, for raising me since birth up till now. Also to my siblings, very much gratitude. Families are forever..

Finally , millions of thanks to all who have assisted me during this project officially or non- officially.

"So, verily, with every difficulty, there is relief:

Verily, with every difficulty there is relief."

The Holy Quran,

Chapter 94 (Al Sharh), Verses 5 & 6

Table of Contents

List of Figures

List of Tables

Cloth Simulation

Chapter 1: Introduction

CHAPTER 1 : INTRODUCTION

1.1 Computer Graphics

Computer Graphics is concerned with theories and techniques to input, output, generate, transform, manipulate and transmit pictures with the aid of computers. It objects are artifacts obtained by synthesis. In contrast, picture or image processing deals with images obtained from the real world, it is field quite distinct from computer graphics although some overlap exists.

Graphics are natural and efficient way of communicating information. Computer graphics is an applied branch of computer science. Computer graphics is applied in Computer Aided Design (CAD), presentation graphics, computer arts, entertainment, education and training (simulator), image processing and Graphical User Interface (GUI).

The topics of computer graphics include output primitives, two-dimensional transformation, two-dimensional viewing, three-dimensional object representation, three-dimensional transformation, visible surface detection and surface rendering. Three-dimensional graphics is a graphics system in which a three-dimensional data set is converted to two-dimensional for viewing.

1.2 History of Computer Graphics

The middle 50's to early 60's was the 'beginning' era. The early 60's to the late 60's was the 'gee whiz, look what aerospace and automotive is doing' era and the late 60's to the early 70's was the 'let's form a new graphics company' era.

>> *1950s*

In 1950, the first computer-driven display, attached to Massachusetts Institute of Technology's (MIT), was used to generate simple pictures. This display made use of a cathode-ray tube. During 1950s, interactive computer graphics made little progress because the computers of that period were so unsuited to interactive use. These computers were number crunchers.

>> 1960s

In the spring of 1963, Dr Ivan Sutherland a MIT described Sketchpad with some of the seminal data-structure work laying the software-theoretical basis for computer graphics. The system uses a light pen to draw pictures on the display. The promoted interactive computer graphics. Around 1963, Steve Coons began developing surface-patch techniques for computer graphics modelling. In early 1960s, there were beam-forming techniques, dot raster systems and line raster system.

IBM organized a program call Demand in an effort to evolve CAD and CAM techniques. There were inefficient operating systems and high cost of development and implementation of interactive graphics applications. The applications in 1960s included CAD in the aircraft and textile industries, management information system, simulations, process control, graphics arts and computer generated movies.

>> 1970s

The early systems use keyboards and light pens. Graphic tablets, digitizers and touch sensitive device were included later. Early system require the user to create their own software. In 1960s, a wide variety of proprietary package were available. Since early 1970s, complete turnkey system have became available. Throughout 1960s and early 1970s, computer graphics devices were considered 'expensive toys'. In late 1960s and early 1970s, a number of new computer graphics companies were organized. In the early 1970s, the development of semiconductor and integrated memory hardware lower the size and cost of pixel image storage systems and raster devices were available commercially. In the mid 1970s, multi-console systems were available.

1.3 Introduction to Computer Simulation

The goal of this project is to implement and study cloth-simulation. Cloth simulation is a subject of significant interest to the Computer Graphics (CG) community. Computer network simulation is the discipline of designing a model of an actual or theoretical physical system. After designing, the model is executed on a digital computer.

Computer Simulation is becoming an increasingly popular method for network performance analysis. It is the process of designing and executing experiments with a model of actual or imaginary system using computer. A typical network simulator can provide the programmer with the abstraction of multiple threads of control and inter-thread commutation. It usually comes with a set of predefined modules and user friendly graphical user interface (GUI). The objective of computer simulation is to experimentally test and analyze a system without building an actual system.

Simulation process is divided into three main fields. They are Model Design, Model Execution and Model Analysis. Computer Simulator is developed using certain programming languages.

1.4 Objectives

Objective to develop this cloth simulation are :

i. To achieve a real-time cloth simulation.

ii. To simulate the cloth under gravity and under wind affect.

iii. Using mass and spring damper for increase the realism in real-time graphic simulation.

iv. To provide information regarding internal and external force of cloth. The system not only offers information but the information is presented in such a way that it facilities user understanding about the cloth.

v. To attract the interest in fashion designer who wants to learn briefly about cloth simulation and to define the user about the latest computer technology especially for 3D and virtual reality.

1.5 Scope

In this section, the scope of the project will be defined.

i. To present the internal and external force of cloth.

ii. Simulate the deformation of the cloth in under gravity, under wind effect.

iii. Showing information about the cloth structure.

vi. To show the mathematical method in this project.

1.6 Target User

i. **Computer graphics animators** - who have been engrossed only with the draping and shape of cloth from an animators point of view for the sheer pleasure of animating a natural phenomena. They wanted to produce models which exhibit cloth like behaviour.

ii. **Fashion designer** - this was mainly inspired by apparel designers, interior designers etc who were interested in the behaviour of different types of cloth which was part of their professional work.

iii. **Public user** - for the public users especially for those who interested to learn about design cloth

1.7 Project Outcome

Basically, for the expected outcome project usually fix before the development has been done. Therefore, the factor must be identify before it. The factor are time bound to finish the project, technology and other resources. These are the outcome of the simulation project:

i. Present the cloth in different situation such as under gravity and under wind affect.

ii. The system can implement the basic function and fulfil the criteria such as stable, consistence, and reliable.

The system allowed the upgrading in future and the new reliable function can be add.

1.8 Schedule

1.8.1 Planning

The planning of the project development :

		WXES 3181														
ID	Task Name	Start	Finish	Duration	Mar-03				Apr-03				May-03			
					1	2	3	4	1	2	3	4	1	2	3	4
1	Feasibility Study					▨										
2	Literature Study						▨	▨								
3	Requirement Analysis							▨	▨							
4	System Design									▨	▨					
5	Program Design										▨	▨				
6	Documentation					▨	▨	▨	▨	▨	▨	▨	▨			

Table 1.1 : First Semester

		WXES 3182														
ID	Task Name	Start	Finish	Duration	Jun-03				Jul-03				Aug-03			
					1	2	3	4	1	2	3	4	1	2	3	4
1	Development & Implementation				▨	▨										
2	Functional Unit Testing						▨	▨	▨	▨						
3	Integration Testing									▨	▨	▨				
4	System Testing											▨	▨			
5	Delivery Testing													▨		
6	Maintenance														▨	
7	Documentation				▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	

Table 1.2 : Second Semester

1.8.2 Deliverables and Milestone of Development Stage

Stage of Development	Deliverables / Milestone
Feasibility Study	Project proposal Supervisors approval
Literature Study	Scope of literature study Methods of literature study Material Collection Ended Material read and processed
Requirement Analysis	Requirement Analysis Plan Requirement Report System Specification
System Design	System Architecture design System Abstract view
Program Design	Logical functional design State / Activity / Data flow diagrams
Development & Implementation	Logical design learning Coding Internal & External documentation
Functional Unit Testing	Unit testing plan Unit acceptance Documents verifications
System Testing	System testing Plan System acceptance
Delivery	System deliverables document User guide
Maintenance	Maintenance Plan

Table 1.3 : Deliverables and Milestone

Cloth Simulation

Chapter 2: Literature Review

CHAPTER 2 : LITERATURE REVIEW

2.1 Introduction

The background study about the knowledge and information gained to development this project is the meaning of literature review. Through this, the writer can get a better understanding on the development tools that can be used to develop a project. The writer can also get a better idea on the development methodologies used while developing a project. Besides that, it allows the developer to study existing or past develop projects and find out the weakness and strength of it.

2.2 Overview

The goal of this project is to implement and study cloth-simulation. Cloth simulation is a subject of significant interest to the Computer Graphics (CG) community. CG characters need to wear clothes, which in-turn need to be animated to make the characters more realistic. Since clothes don't have active controllers built in, their behavior can be classified as being passive response to the environment. This means that the behavior of cloth is relatively well-defined once the external forces acting on the cloth are known - as opposed to say a human who can exert forces and torques to *control* his/her motion as desired. The modeling of such passive systems is best accomplished through physics based simulation.

Cloth behavior is determined by collisions between the cloth and the body and self collisions within the cloth itself. Many different kinds of models have been used in animation of cloth and similar deformable surfaces. In this simulation the writer is looking forward to use a physical model based on Newton's motion equation applied to a particle system. This model allows us to handle each element of the cloth separately for manipulation of position, speed and direction. The simulation involves a cloth suspended in air with an initial position. Would have to provide properties of the cloth such as ***structural, bending and shear*** which in essence models different types of cloth fabrics. As well as providing external properties which affect the behavior of the cloth such as gravity and wind forces. Objects can also be placed in the scene by the user such as hanging on two fixed point and resting on a chair which the cloth can interact with. As the animation proceeds the cloth will behave according to its internal physical properties, the external properties of the environment and the how it responds to objects in the scene.

2.2.1 How cloth simulation work?

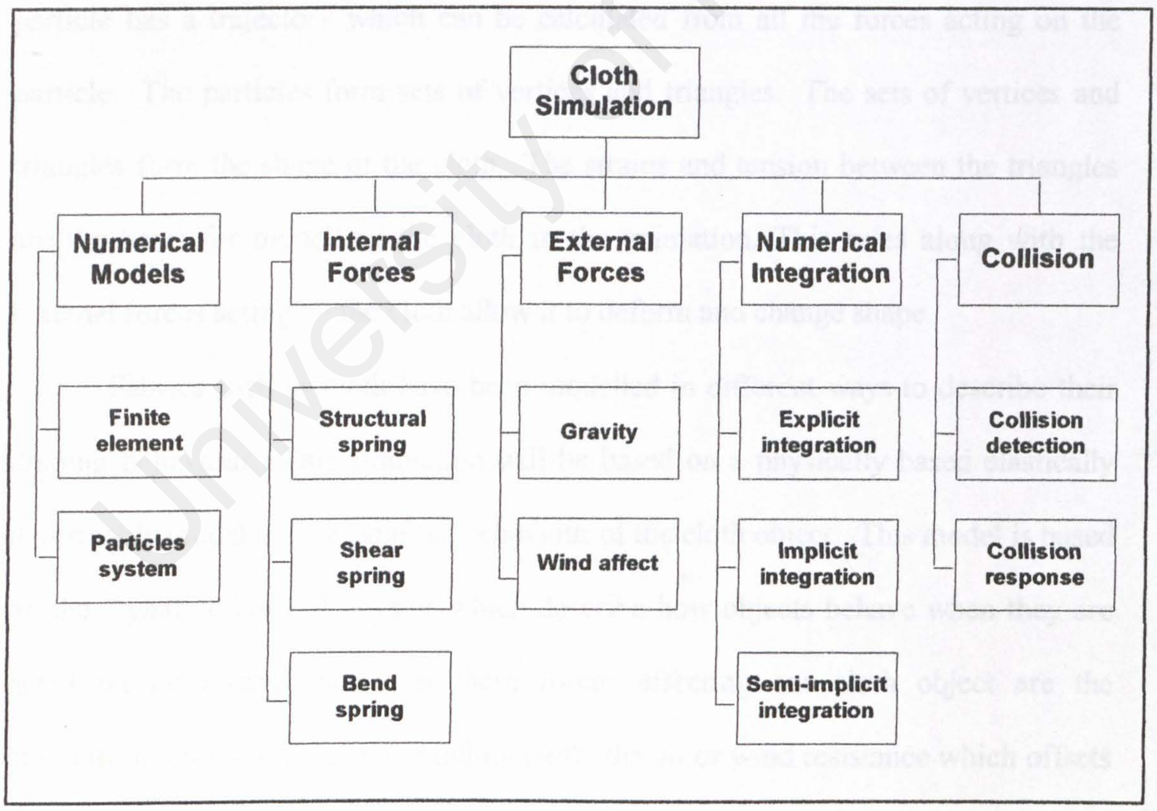


Figure 2.1 : How cloth simulation work

The cloth object can be described by its properties and how it behaves in the real world. We can describe the cloth as have certain properties such as tension, shearing and bending. To describe the cloth, these properties model the internal stresses and strains of the cloth and can be controlled to model different types of cloth from ones that are soft to rigid, or can bend to some degree. There are also external forces that can be modelled for the cloth, such as gravity and wind resistance. The cloth is suspended by holding points on the cloth at various points, this determines the orientation and shape of the cloth in space. Threads in cloth may be stretched or relaxed. The points on the cloth are affected by the stress and strain rules which deform and stretch the cloth. The points on the cloth can be treated as a system of particles each with a 3D position in space, direction and a velocity. Each particle has a trajectory which can be calculated from all the forces acting on the particle. The particles form sets of vertices and triangles. The sets of vertices and triangles form the shape of the cloth. The strains and tension between the triangles are the basis for modelling the cloth in the animation. This rules along with the external forces acting on the cloth allow it to deform and change shape.

Fabrics such as cloth have been modelled in different ways to describe their draping behaviour. This simulation will be based on a physically based elastically deformable model to represent the behaviour of the cloth object. This model is based on the Dynamic laws of physics which describe how objects behave when they are acted on by forces. Some of these forces affecting our cloth object are the gravitational force acting on the falling cloth, the air or wind resistance which offsets the full effect of gravity and the frictional forces such as a cloth sliding off a box. The basic laws controlling the cloth's behaviour are based on the Newton's laws of motion. We will primarily be using Newton's second law of motion that describes the acceleration of an object depends on the forces acting on the object and it's mass. Often several forces are acting on an object at one time the net force is the vector sum of all of them. For each control point in our cloth we sum all the forces acting on the cloth to determine the acceleration including all the internal and external forces of the cloth.

2.2.2 Numerical models

Several models have been proposed to modelize a cloth structure. They can be divided into two categories:

- *Finite element based models*

  Finite element based models are the most accurate, but only work well for static rendering. No dynamic forces (for instance wind) are included in the simulation. Extending this model for including some dynamic forces may be possible, but quite difficult though. Furthermore, these methods require high computational time.

- *Particle system based models*

  These models are discrete representations of smooth surfaces. Here the cloth is represented by a finite number of control points called *nodes*. Easier to implement than finite elements, this model also gives rather correct results. Besides they are less time-consuming.

The goal of this project was to achieve a real-time cloth simulation, so it used the **particle system based model**.

2.2.3 Internal force representation

The cloth model will be composed of masses and springs, where the cloth model is an array of masses which form the control points for the cloth.

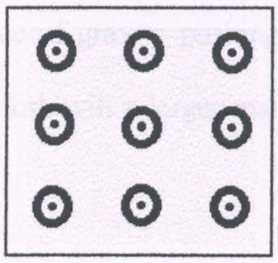


Figure 2.2 : The vertex / mass

Each mass is linked to it's neighbour by springs. There are three different kinds of springs:

Structural springs handle compression and traction stresses. Structural springs connect adjacent horizontal and vertical vertices. ***Shear springs*** handle shear stresses. Shear springs connect vertices diagonally. ***Bend springs*** handle stresses from the cloth bending. Flexion springs connect every other vertex in the horizontal and vertical directions.

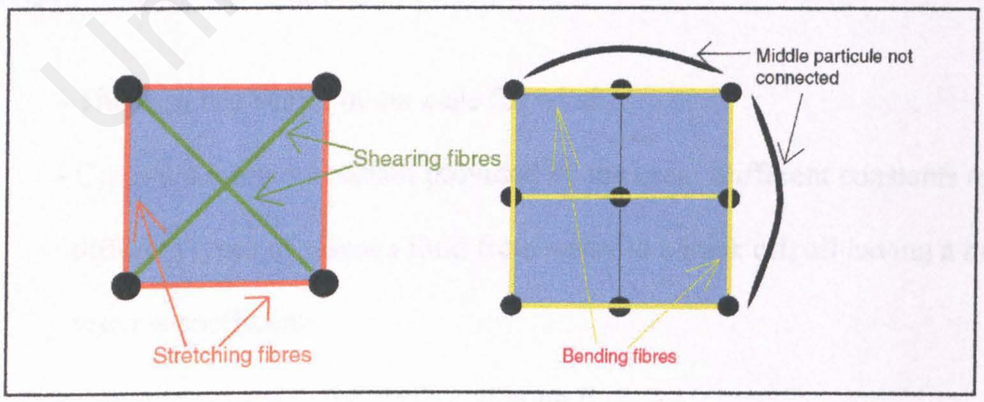


Figure 2.3 : Structural, Shear and Bend springs

2.2.4 External force representation

External forces that will be modelled will be a gravitational force, wind resistance and a damping force.

The *force due to gravity* is a force that acts on all objects. This cloth suspended in the air will fall due to the force of gravity pushing downwards to the earth. Since this force depends on mass a cloth will a larger mass will fall faster than one with a smaller mass.

$F_{gr}(P_{i,j}.) = \mu g$

- μ is the mass of the cloth for point $P_{i,j}$
- g is the acceleration of gravity, this is usually 9.8 m/s^2

The *force of air resistance or wind* is a vector exerting a force on a object depending on the normal to the surface. This force is greatest when the normal and the wind vector are perpendicular since this gives the greatest cross sectional area and greater air resistance. Wind or air are a specific case of a type of Viscous fluid which can also be water or an oil. The viscous force for wind in our model is:

$F_{vi}(P_{i,j}.) = C_{vi}[n_{i,j}(u_{fluid} - v_{i,j})]\ n_{i,j}$

- The u_{fluid} is a vector in our case for wind
- C_{vi} is a a viscous constant provided by the user. Different constants model different types of viscous fluid from water to a thick oil, oil having a higher viscous coefficient.
- $n_{i,j}$ the normal to the surface at point $P_{i,j}$

The *force due to viscous damping*. This force models the loss of mechanical energy of the cloth.

$F_{dis}(P_{i,j}.) = - C_{dis}v_{i,j}$

- C_{dis} is a damping coefficient given by the user

- $v_{i,j}$ is the velocity of point $P_{i,j}$

Again summing up all the external forces gives a net external force vector.

The two internal and external force vectors are summed to give the final force acting on the point $P_{i,j}$, this force is used in Newton's motion equation to find the acceleration of point $P_{i,j}$, which is substituted back into the velocity equation. Now we know the velocity for $P_{i,j}$ and we can find the displacement of $P_{i,j}$ at time $t + \Delta t$.

The *force due to viscous damping*. This force models the loss of mechanical energy of the cloth.

$F_{dis}(P_{i,j}.) = -C_{dis}v_{i,j}$

- C_{dis} is a damping coefficient given by the user

- $v_{i,j}$ is the velocity of point $P_{i,j}$

Again summing up all the external forces gives a net external force vector.

The two internal and external force vectors are summed to give the final force acting on the point $P_{i,j}$, this force is used in Newton's motion equation to find the acceleration of point $P_{i,j}$, which is substituted back into the velocity equation. Now we know the velocity for $P_{i,j}$ and we can find the displacement of $P_{i,j}$ at time $t + \Delta t$.

2.2.5 Fundamental Dynamic Equation

Cloth with nodes (masses) can use the Newton Fundamental Dynamic Equation : This equation is a **second order differential equation** where *m* is the mass, *F* the forces and *x* the acceleration. At this point, what we have to do is to retrieve the velocity and the position of each node each time step. To achieve this, we must integrate this differential equation.

$$\boxed{\ddot{x} = \frac{1}{m}F}$$

2.2.6 Numerical integration methods

In order to integrate the previous equation, we will use numerical integration methods. These methods are splitted into two categories:

- *Explicit Integrations*

  These methods are based on Taylor expansion of the previous equation : **Euler** (1st order), **Mid-point** (2nd order), **Runge-Kutta** (4th order). They are relatively easy to implement but are submitted to instability.
- *Implicit Integrations*

  These methods reformulate the previous equation into a system of equations. These methods are very stable but difficult to implement.

- ***Explicit integration methods***

First of all, note that the fundamental dynamic equation is applicable to each node. Each one has a position, a velocity and forces represented by vectors. For this project, writer use *Euler* explicit integration. It consists in approximating the previous equation by a **first order Taylor expansion**. Note that the dynamic fundamental equation, which is a second order differential equation, was splitted into two **first order differential equations.** This method is very simple to implement but very instable. For a stable system, we must choose a **small time step** and **small spring constants**.

$$\boxed{\begin{cases} v(n+1) = v(n) + \dfrac{dt}{m} F(n) \\ x(n+1) = x(n) + dt * v(n) \end{cases}}$$

To resume,

Strong points

- low computational time

- easy to implement

Weak points

- small time step

- small spring constant

- over-elongation problem

▪ ***Explicit integration methods : the over-elongation problem***

As with explicit methods, it cannot use high spring constants, cloth fibres are over-elongated when submitted to external forces (see figure below). To prevent this issue, used some post-correction methods. These methods intervene just after the integration step and work directly on positions and/or velocities.

Position correction:

The algorithm first checks the distances between spring. For over-elongated springs (springs whose length is bigger than the maximum allowed), it changes the node position as illustrated in the figure below. These two steps are iteratively repeated until no over-elongated springs remain. This algorithm, first proposed works well in simple situations but fails when too many collisions occur between cloth and other objects, because the method does not converge.

Velocity correction:

Another approach is to correct the velocity of each node. First, decompose the velocity into two components : the normal component and the tangential component. In other words, suppose two nodes linked by a spring. Only the velocity component projected on the line passing through the two nodes is responsible for over-elongation.

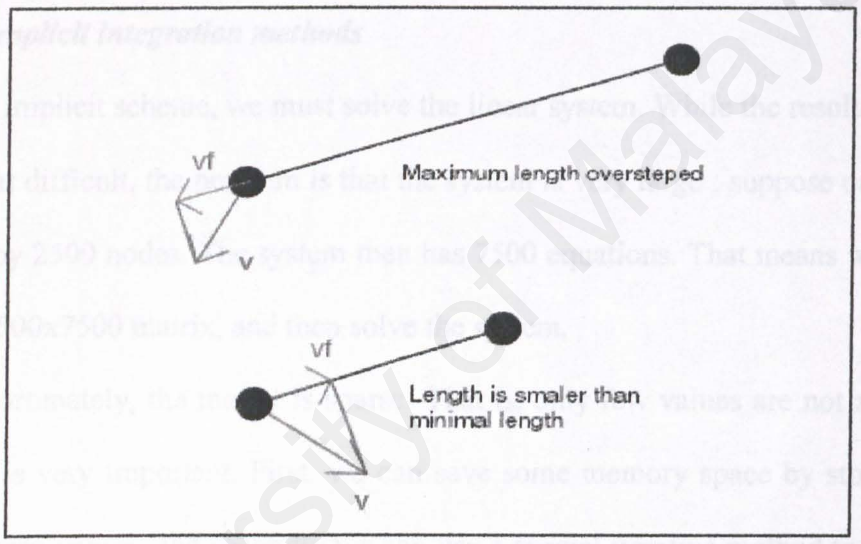

vf
Maximum length oversteped
v
vf
Length is smaler than minimal length
v

Figure 2.4 : Explicit integration methods : the over-elongation problem

- ***Implicit integration methods***

To obtain an implicit scheme, the writer introduce a little change in the 1st order Taylor expansion used by Euler integration. In the equation below, in the first line, forces are not expressed any more at time *t* but at time *t+1*. This little change forces the force field to be coherent at time *t+1*. Therefore assume that there will not be any unstability.

Theorically, this scheme is stable whatever the time step and the spring constants are. The main drawback is that we have to compute forces at time *t+1*, and cannot be computed directly. To solve this problem, we can use another first order

Taylor expansion to express forces at time $t+1$. Once done, we can reformulate the equation to obtain the final result specified in figure below.

This is a linear equation system. We have to solve it in order to find the velocity, and then the position of each node.

$$\boxed{\begin{cases} v(n+1) = v(n) + \dfrac{dt}{m} F(n+1) \\ x(n+1) = x(n) + dt * v(n) \end{cases}}$$

- ***Implicit integration methods***

Now for implicit scheme, we must solve the linear system. While the resolution itself is not that difficult, the problem is that the system is very large : suppose our cloth is defined by 2500 nodes. The system then has 7500 equations. That means we have to store a 7500x7500 matrix, and then solve the system.

Fortunately, the matrix is sparse. That is, only few values are not zero. This property is very important. First, we can save some memory space by storing only the non-zero values, and secondly we can use a special iterative method to solve the linear system : *the conjugate gradient*. This iterative method is designed to exploit the sparsity property. It uses only vector-matrix products. Obviously, the matrix-vector product implementation exploit the sparsity described before.

$$\boxed{(I - hM^{-1}\frac{\partial f}{\partial v} - h^2 M^{-1}\frac{\partial f}{\partial x})\Delta v = hM^{-1}(f_0 + h\frac{\partial f}{\partial x}v_0)}$$

To resume

Strong points

- stable whatever the chosen time step

- high spring constants allowed

Weak points

- storing a big matrix

- solving a linear system

- real-time execution difficult to obtain

- ***Semi-Implicit integration methods***

In this project, the writer will only present the main ideas of this method. In the integration process, instabilities are due to high frequencies. The idea is to filter the force field to attenuate high frequencies. Here is a general description of the method :

i. *Compute the inverse matrix of linear forces*

This step is only made at the beginning of the simulation. This inverse matrix remains unchanged during all the simulation.

ii. *Filter the force field by the previously obtained matrix*

It is a standard matrix-vector product.

iii. *Make an Euler integration step to retrieve velocities and positions*

As we filtered the force field, instability is no more an issue.

iv. *Post-correction step*

This step is used to preserve the angular momentum.

This method gives good results, but it is usable for small system only because of the matrix inversion step at the beginning of the simulation, which is very expensive.

2.2.7 Collision handling

To improve the realism of simulation, it includes cloth-object collisions. The collision processing is performed in two steps. The first step is the collision detection, and the second is the collision response.

- ***Collision detection***

- ***Collision response***

2.3 Review On Existing System

2.3.1 Syflex - The Cloth Simulator

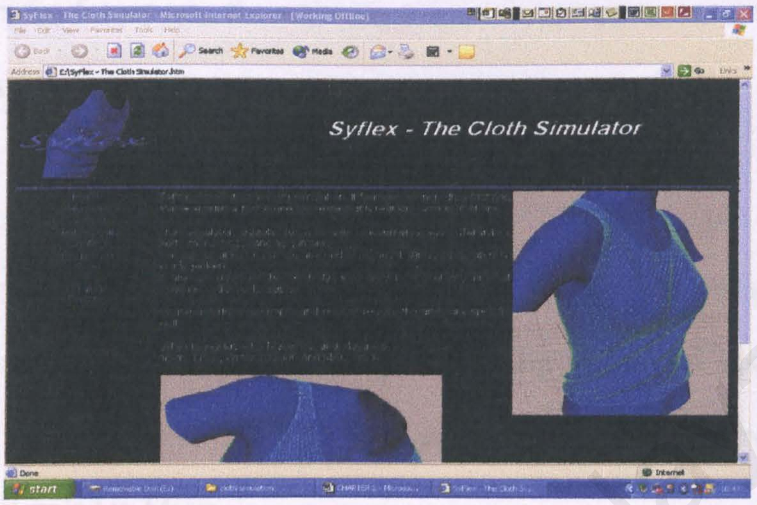


Figure 2.5 : Syflex – The Cloth Simulator

Syflex is the latest in cloth simulation. It features an incredibly fast and stable simulator that allows to create highly realistic cloth animations. The simulator adapts to whichever movements a characters perform: running, dancing, jumping and others. It allows an artist to animate any kind of clothes: t-shirts, pants, aprons, skirts, jackets. It also provides all the flexibility necessary to model any kind of material: cotton, silk, leather and others material.

Its user interface is simple and doesn't require the artist any specific skill. Syflex is available for Maya 4.0, and Maya 4.5, on Irix, Linux, Windows 2000 and MAC OS X.

Features For Syflex :

Speed

Syflex is an extremely fast cloth simulator. Syflex is an order of magnitude faster than any other cloth simulator. The following animation was computed at a rate of 50 frames per second on a PIII 933 MHz.

Stability

Syflex technology provides for a perfectly stable simulation. Whatever the constraints or the forces are, the software always computes an accurate animation, without vibrations.

Ease of use

Syflex is easy to use. It doesn't require any special fashion-designer skill from the animator. There is no need to model the cloth using flat panels and stitch them together: the artist just uses any traditional polygonal object. Moreover, Syflex is seamlessly integrated in Maya.

Cache

The simulation can be saved in files, allowing to re-play it quickly. It also allows to re-simulate just one or many frames, without recomputing all the frames. Blend together cache files, choosing which part of the cloth to keep for each cache.

Nails, Pins

Any vertex of the cloth can be nailed. These nails may be animated as any other object. Any vertex can be pinned to another static or moving object.

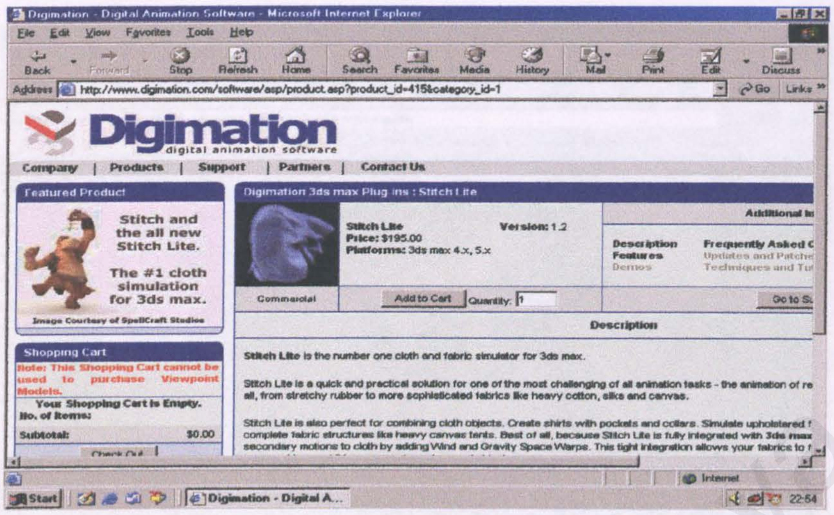
Collisions

Collisions of the cloth with any static or moving object are computed accurately. To optimize collisions, the user can specify which faces may collide. Self-collisions are also available.

SDK

Syflex can be customized by adding new forces and constraints.

2.3.2 Stitch Lite - The Cloth Simulator


Digimation - Digital Animation Software - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites Media History Mail Print Edit Discuss
Address http://www.digimation.com/software/asp/product.asp?product_id=415&category_id=1
Go Links
Digimation digital animation software
Company | Products | Support | Partners | Contact Us
Featured Product
Stitch and the all new Stitch Lite.
The #1 cloth simulation for 3ds max.
Image Courtesy of SpellCraft Studios
Shopping Cart
Check Out
Additional In
Description
Features
Demos
Frequently Asked
Updates and Patche
Techniques and Tu
Commercial
Add to Cart
Go to S
Start
Digimation - Digital A...
Internet
22:54

Digimation 3ds max Plug ins : Stitch Lite

Stitch Lite Version: 1.2
Price: $195.00
Platforms: 3ds max 4.x, 5.x
Quantity: 1

Note: This Shopping Cart cannot be used to purchase Viewpoint Models.
Your Shopping Cart is Empty.
No. of Items:
Subtotal: $0.00

Description

Stitch Lite is the number one cloth and fabric simulator for 3ds max.

Stitch Lite is a quick and practical solution for one of the most challenging of all animation tasks - the animation of re
all, from stretchy rubber to more sophisticated fabrics like heavy cotton, silks and canvas.

Stitch Lite is also perfect for combining cloth objects. Create shirts with pockets and collars. Simulate upholstered f
complete fabric structures like heavy canvas tents. Best of all, because Stitch Lite is fully integrated with 3ds max
secondary motions to cloth by adding Wind and Gravity Space Warps. This tight integration allows your fabrics to f

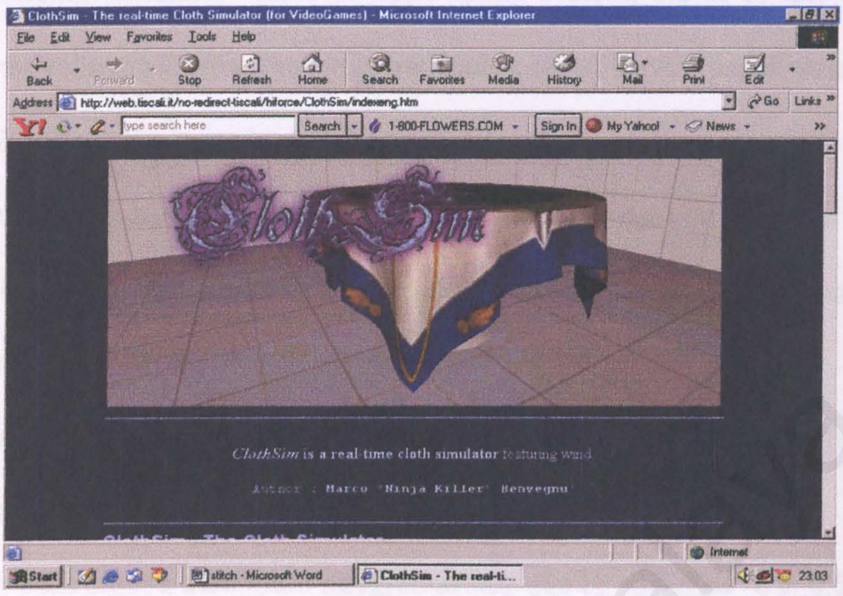
Figure 2.6 : Stitch Lite – The Cloth Simulator

Stitch Lite is the number one cloth and fabric simulator for 3ds max. Stitch Lite is a quick and practical solution for one of the most challenging of all animation tasks - the animation of realistic fabric. Stitch Lite can handle it all, from stretchy rubber to more sophisticated fabrics like heavy cotton, silks and canvas.

Stitch Lite is also perfect for combining cloth objects. Create shirts with pockets and collars. Simulate upholstered furniture, drapery, bedding, even complete fabric structures like heavy canvas tents. Best of all, because Stitch Lite is fully integrated with **3ds max**, it easily applies dynamic secondary motions to cloth by adding Wind and Gravity Space Warps. This tight integration allows fabrics to flow, fold, bunch and gather just would expect it to characters and objects move.

More advanced controls include vertex level control over the dynamic and static properties of the cloth. Some of these controls include air resistance, and the ability to attach parts of the cloth to other objects. These powerful controls allows a character to grab a scarf without relying on the simulation alone. There are also controls for positioning the cloth by grabbing and pulling it during a simulation.

2.3.3 ClothSim - The Cloth Simulator


ClothSim - The real-time Cloth Simulator (for VideoGames) - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites Media History Mail Print Edit
Address http://web.tiscali.it/no-redirect-tiscali/hiforce/ClothSim/indexeng.htm
Go Links
type search here Search 1-800-FLOWERS.COM Sign In My Yahoo! News
ClothSim
Internet
Start stitch - Microsoft Word ClothSim - The real-ti...
23:03

ClothSim is a real-time cloth simulator featuring wind

Author : Marco 'Ninja Killer' Benvegnu'

Figure 2.7 : ClothSim – The Cloth Simulator

With *ClothSim* we can watch various cloth models (tablecloths, flags, garments and others) blowing in the wind, and we can freely move around them: we can examine each single wrinkle under every angulation. But *ClothSim* isn't limited to this: we can change the wind speed, stop the time, and ask for a series of statistical information's. In particular, we will see the the astonishing frame rates which this powerful implementation is able to achieve.

Features :

- Fast.
- Numerical stability assured.
- Physically correct.
- Precise and fast normals generation (only one cross product for each vertex).
- A personal technique for the computation of wind forces, 2 times faster than the conventional method.
- Rendering via TriangleStrips.

- Texturing / Lighting.
- *ClothSim* simulates whatever mass-spring system, cloths of any form, and even volumetric structures (e.g. molecules, tents, leaves, paper, strings, ropes, cobwebs, jelly, hair via skeletal animation).
- Enhanced tablecloth: the elements (mass and springs) on the table are fixed, and so they leaved the scene at all. Moreover, the triangles' orientation is customized for the folds of a tablecloth.
- Variable integration step: animation speed is unrelated to the frame rate. Timescale is physically accurate.
- For each cloth a maximum integration step can be specified (e.g. 0.5s); in this way the integrator will never exit the region of absolute stability, even in stress conditions (swapping / loading); this guarantees the total absence of "explosions" (an aged obstacle in the explicit integration of stiff systems).
- Vibration free rest condition (hard to obtain, especially with variable step).
- OOProgramming for the maximum flexibility, portability and reusability of the code.
- Specifically studied to dress quality videogames.
- Time stats.
- Smooth movements and mouse filtering (fundamental when both strafing and aiming).

2.4 Review on Journal

2.4.1 Large Steps in Cloth Simulation

This paper presents a cloth simulation system that can stably take large time steps. The system combines a new technique for enforcing constraints on the cloth particles with an implicit integration method. The simulator discretizes the cloth as a triangular mesh. It uses a simple continuum formulation to derive the internal cloth forces that model operations like local anisotropic stretch or compression. It also includes a unified treatment of damping forces. A modified conjugate gradient method is used to solve the large sparse matrix generated at each time step of the implicit method. Due to the modification, the constraints are enforced exactly at each iteration step. The modified conjugate gradient method converges at a similar rate to unmodified CG.

Summary

Physically based cloth animation is of interest to computer animation and poses a significant challenge because of the computational and mathematical complexity. The main challenges in cloth animation are the following: numerical instability due to stiffness of the material, high resolution requirement to show the realistic wrinkling and folding of cloth, constraint maintenance between cloth and solids or between cloth and cloth. The simulation system presented here provides a framework that meets all of the above challenges making it viable and useful for generating realistic animations of non-trivial cloth models.

Representation

Cloth is modelled as a triangular mesh of particles. Each particle has an associated position, velocity, and force acting on it. The simulator computes the position and velocity for each particle at each time step.

Implicit Integration Method

The stability of this cloth simulation system results from the use of an implicit integration method rather than explicit method like Euler's method. Explicit methods require small time steps to maintain stability and typically perform poorly on stiff systems. For system with some very stiff components, the entire simulation can grind to a halt. On the other hand the implicit method is based an implicit *backward Euler* method using a Taylor series expansion to convert the resulting non-linear system into a large unbanded sparse linear system. Without constraints, this linear system can be converted to a symmetric, positive definite system and solved using a conjugate gradient method with a running time of roughly $O(n_{1.5})$. While the running time for an explicit method, like Euler, is O(n), the implicit method permits the use of much larger time steps without compromising stability. This in turn allows higher resolution models to be simulated without increasing the overall running time.

Forces

Cloth's material behaviour is often described in terms of a scalar potential energy E. The quadratic energy formulation with arbitrarily large stiffness is chosen because it fits well with the numerical model of the simulation system. Simple and intuitive continuum-based force models are developed that allow the system to support operations like local anisotropic stretching, shrinking, and bending. Damping forces are easily derived using this energy formulation and integrated into the system.

Modified Conjugate Gradient Method

The linear system with mass modifications cannot be directly solved using the conjugate gradient method, because the modified mass matrix is singular so the system cannot be trivially converted to a symmetric system. Instead, the traditional CG method is modified to include a filter that guarantees the constraints exactly at each step of the iteration. This simple modification also has the side benefit of allowing the system to compute the force needed to release the constraint. The modified CG method converges at a rate of $O(n_{1.5})$ empirically.

Collision

The system doesn't introduce a new collision detection method, instead it does have a quick and robust way of handling penetration. Penetrations between cloth/cloth particles are handled using a penalty force. For cloth to solid penetration, the system makes an alteration to the position that is integrated into the ODE. In combination with mass alteration, the system has total control of the velocity and position of the constrained particle.

2.4.2 Devil in the Blue-Faceted Dress: Real-Time Cloth Animation

This journal describes methods of dynamic simulation using mass and spring systems. These techniques dramatically increase the realism in real-time graphic simulation. One of dynamic simulation's key benefits is that it creates a scaleable game experience.

Traditional Cloth Animation in Games

Cloth animation is tricky. Even in the world of high-end computer graphics, it's difficult to get right. Most of the time, it's wise to avoid the whole issue. Anyone who has ever created a female character in a skirt is familiar with the problem of the legs poking through the cloth mesh during animation. This is pretty difficult to fix, especially if animation requires a variety of motions. Unfortunately, it's also a really obvious animation problem that any end user can spot. Most loose clothing doesn't look natural in digital art because it's static. It doesn't move along with the body. It's possible to morph the shape of the skirt to match the motion of the character, but this requires quite a bit of detailed animation work. Likewise, deforming the skirt with a bone system can be effective, but not necessarily realistic.

The Latest Springy Fashions

The mass and spring dynamics simulation proved effective for simulating soft body objects in real time. Thought it should be possible to use these techniques to create a cloth simulation. Mass-spring model consists of a square grid of particles, each with unit mass and at (initially) unit distance away from the adjacent particles in its row and column. Holding the masses in the grid formation are three types of springs, as seen in the diagram below. Vertices that are adjacent along column or row lines are connected by **structural springs**, and springs that connect two vertices at opposite corners of a grid square are **shear springs**. **Bend springs** extend in the same directions as structural springs but connect nodes that have one node between them.

Problems to Avoid and Ignore

The simulation has a couple of problems. The first is that the way to simulate cloth realistically is to use a lot of points in the simulation. This takes more computation time. High-end animation programs rely on a great number of particle points for realism. Of course, in other fields, hour-long render times are perfectly acceptable. This is another good area for scaling game performance. If the system is running quite fast, subdivide the cloth patches a little more. Game players with a white-hot system should have smooth-looking cloth.

Another problem is that each spring acts independently. This means that each spring can be stretched to a great extent. In many cases, the amount of stretch can exceed 100 percent. This is not very realistic. Actual fabric will not stretch in this manner. The problem have is that using linear springs when fabric actually displays a nonlinear spring behaviour. As the amount of stretch increases, the strength of the spring increases also. The fabric will also stretch to some limit and then if the force continues, it will rip. Increasing the spring strength dynamically can lead to instability problems just like any other stiff spring problem. The suggestion is checking the amount of stretch in each spring, and if it exceeds a set deformation limit, the springs are adjusted to achieve this limit. While his agree this solves a definite problem, a second pass through the springs is costly.

2.5 Technological Review

Before any processes is started, as developers should review the technology available to them, so they may choose the most ideal one to be used. There is more than one aspect of technology to be review on, we may have programming languages, platform and object reuse technologies. All of these have their pros and cons. Developers of the project should have those pros and conts reviewed to ensure the quality of the output developed.

2.5.1 Programming Language Overview

2.5.1.1 Microsoft Visual Basic 6.0

Visual Basic programming language is fairly simple and uses common English words and phrases for the most part. Visual Basic has evolved from the simplest programming language for Microsoft Windows to an exceedingly complex development environment, capable of delivering virtually anything from tiny utilities to huge n-tier client/server applications.

Control are tools on the Toolbox window that you can place on a form to interact with the user and control the program flow.

Microsoft Visual Basic 6.0, the latest and greatest incarnation of the old BASIC language, gives a complete Window application development system in one package. Visual Basic lets ones write, edit, and test Windows applications. In addition, VB includes tools ones can use to write and compile help files, ActiveX controls, and even Internet applications.

2.5.1.2 Microsoft Visual C++

Microsoft has built into its C++ development environment to enable developers to create very advanced applications for the Windows and NT platforms. When Microsoft's developers first came up with the idea behind Visual C++, they decided to take their world-class C++ compiler and create a development environment and set of tools that would enable developers to create Windows applications with a level of ease and speed that was unheard of among C++ development environment. Since that first version, Microsoft has continued to improve the tools that are a part of Visual C++ to make it even easier to cerate Windows applications.

2.5.1.3 OpenGL

OpenGL is the premier environment for developing portable, interactive 2D and 3D graphics applications. Since its introduction in 1992, OpenGL has become the industry's most widely used and supported 2D and 3D graphics application programming interface (API), bringing thousands of applications to a wide variety of computer platforms. OpenGL fosters innovation and speeds application development by incorporating a broad set of rendering, texture mapping, special effects, and other powerful visualization functions.

Developer-Driven Advantages

Industry standard

An independent consortium, the OpenGL Architecture Review Board, guides the OpenGL specification. With broad industry support, OpenGL is the only truly open, vendor-neutral, multiplatform graphics standard.

Stable

OpenGL implementations have been available for more than seven years on a wide variety of platforms. Additions to the specification are well controlled, and proposed updates are announced in time for developers to adopt changes. Backward compatibility requirements ensure that existing applications do not become obsolete.

Reliable and portable

All OpenGL applications produce consistent visual display results on any OpenGL API-compliant hardware, regardless of operating system or windowing system.

Evolving

Because of its thorough and forward-looking design, OpenGL allows new hardware innovations to be accessible through the API via the OpenGL extension mechanism. In this way, innovations appear in the API in a timely fashion, letting application developers and hardware vendors incorporate new features into their normal product release cycles.

Scalable

OpenGL API-based applications can run on systems ranging from consumer electronics to PCs, workstations, and supercomputers. As a result, applications can scale to any class of machine that the developer chooses to target.

Easy to use

OpenGL is well structured with an intuitive design and logical commands. Efficient OpenGL routines typically result in applications with fewer lines of code than those that make up programs generated using other graphics libraries or packages. In addition, OpenGL drivers encapsulate information about the underlying hardware, freeing the application developer from having to design for specific hardware features.

Well-documented

Numerous books have been published about OpenGL, and a great deal of sample code is readily available, making information about OpenGL inexpensive and easy to obtain.

2.5.1.4 Java

Java designer started with C++, removed numerous constructs, changed some, and added a few others. The resulting language provides much of the power and flexibility of C++, but in a smaller, simpler, and safer language.

Java, like many programming languages, was designed for an application for which there appeared to be no satisfactory existing language. In the case of Java, however, it was actually a sequence of applications, the first of which was the programming of the embedded consumer electronic devices, such as toaster, microwave ovens, and interactive TV system. It may not seem that reliability would be an important factor in the software for a microwave oven. If an oven dad malfunctioning software, it probably would not pose a grave danger to anyone and probably would not lead to large legal settlements. Realibility is an important characteristic of the software in consumer electronic product.

2.5.2 Software Overview

2.5.2.1 MAYA

" MAYA's simulation software supports many element types; lump masses, beams, triangular and quad shells, bricks, wedges, and most important, tetrahedral elements which are needed for the really complicated geometry - our geometry just isn't rectangular " (***Tom Hatfield,*** Operations Manager Motorola Inc.)

MAYA offers several technologies which are completely integrated within MAYA's software and make simulation modelling easier and more effective. Technology that allows you to build complex assemblies of parts and still maintain individual thermal/flow models on each of these parts or sub-assemblies. MAYA provides the tools to build individual thermal/flow models and then connect or couple them into a thermal/fluid simulation assembly.

2.5.2.2 3D Studio Max

3D Studio MAX is one of the most powerful desktop 3D graphics programs available today. It is used for a wide variety of commercial and artistic applications, including architecture, computer games, film production, web design, forensics, scientific visualization and virtual reality.

One of the most exiting aspects of 3D graphics is animation. To animate literally means to give life. In MAX, animation is easily achieved by changing an object over time.

2.5.2.3 Adobe Photoshop 6.0

As the industry standard for digital image manipulation software, Adobe Photoshop has revolutionized the photography and prepress industries and has provided commercial and fine artists with an exciting new medium for photographic editing. Adobe has integrated into Photoshop a design based upon traditional photo manipulation technique, where tools and processes directly correspond with those used in 'physical' photography. Photoshop introduces features and enhancements which go far beyond the capabilities of the darkroom technician, thanks to digital technology; yet through and interface based on traditional technique, Adobe ensure a relevant, familiar but powerful program environment. Covering a few of the general elements of the program will give us a better perspective on how it works, allowing us to cover technicalities in more efficient terms. The pixels represent a unit of color information, all changes in Photoshop occur at a two-dimensional level. The closest Photoshop comes to the versatility afforded object-based software is through the use of layers, which are not unlike cells used in the physical graphics industry.

2.5.3 Operating System Overview

For any development of the application that runs on PCs, the platform or more specific, the operating system is always taken into accounts. It tells how far a developer needs to develop, and what has already been taken care of.

2.5.3.1 Microsoft Windows 2000 as Network Operating System

Windows 2000 Professional is the latest commercial version of microsoft's evolving windows operating system. Microsoft emphasize that Window 2000 Professional is evolutionary from Windows NT 5.0 and 'Build on NT Technology'. The change, both fundamental and cosmetic, have made Windows 2000 faster, more reliable, heavier-duty and easier to use.

Windows 2000 Professional is designed to appeal to small business and professional users as well as to the more technical and large business market for which the NT was designed.

Windows 2000 is reported to be more stable than Windows 98/NT system. A significant new feature is Microsoft's Active Directory, which enables a company to set up virtual private network in order to encrypt data locally or on the network and to give users access a shared files in a consistent way from any network computer.

2.5.3.2 UNIX Operating System

UNIX is an increasing popular operating system that traditionally used on minicomputers and workstations in the academic community. UNIX is now available on personal computer and the business community has started to choose it for its openness [Gottschalk, 1996].

UNIX, like any other operating system, is a layer between the hardware and the application that run on the computer. It has functions that manage the hardware and functions that manage the executing application. Besides that, UNIX includes the traditional operating system components.

In addition, a standard UNIX system includes a set of libraries and a set of application. It includes the file system and process control and a set of libraries. One of the greatest strength of UNIX is the consistent way in which it treats files.

2.5.3.3 Linux Operating System

Linux is a free; UNIX work-alike designed for Intel processors on PC architecture machines. Linux is not UNIX, as UNIX is a copyrighted piece of software that demands license fees when any part of its source code is used. Linux was written from scratch to avoid license fees entirely, although the operation of the Linux operating system is based entirely on UNIX and it shares UNIX's command set.

In addition Linux has the following features :

- It is capable of multitasking.
- Has support for Netware clients and server.
- It multi-platform, that is it can run on any processor.
- Has memory protection between processes ensuring that a program cannot crash the entire system.

One of tha main weakness of Linux is lack of support for hardware making is to be difficult in setting up a machine with Linux. Fortunately support for Linux is growing every single day and more peripherals are being added to Linux's list of supported hardware. Besides that, Linux also suffers from non-standardized version issues, and each version has its own exclusive features.

Chapter 3:

Methodology & System Analysis

CHAPTER 3 : METHODOLOGY AND SYSTEM ANALYSIS

3.1 Methodology

The system development methodology is a method to create a system with a series of steps or operations or can be defined as system life cycle model. Every system development process model (see Figure 3.1) includes system requirements (user, needs, resource) as input and a finished product as output.

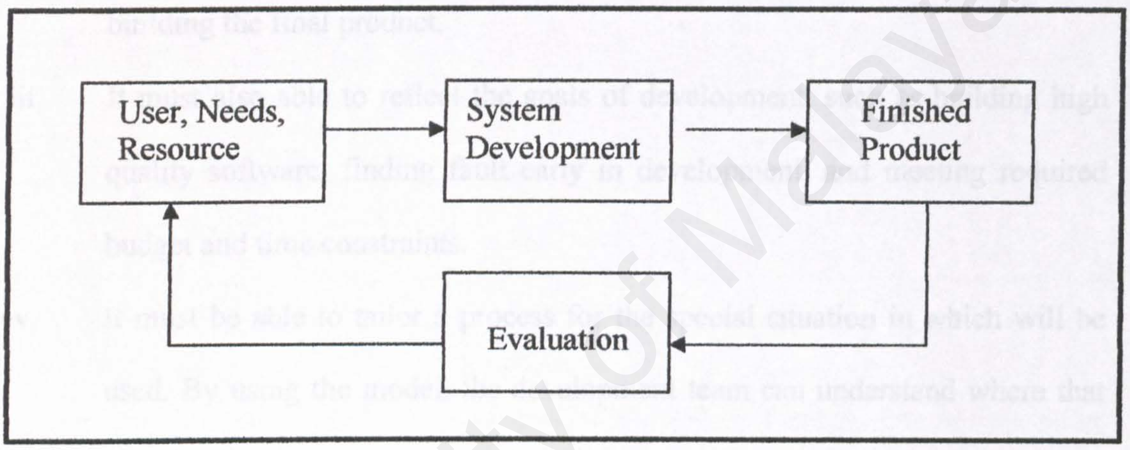


Figure 3.1 : System Development Process Model

There are several process models in system development:

i. Waterfall Model

ii. Waterfall – V Model

iii. Waterfall Model with prototyping

iv. Model System Development Life Cycle (SDLC)

v. Spiral Model

vi. Phased Development Model

vii. Transformation Model

viii. Operational Specification Model

Before any model has been decided to be used, it has to be very careful about why model is chosen. These are the requirement :

i. It must be able to form a common understanding of activities, resources and constraints involved in the software development.

ii. It must help the development team in find inconsistencies, redundancies, and omissions in the process and in its constituent parts. As these problems are noted and corrected, the process becomes more effective and focused on building the final product.

iii. It must also able to reflect the goals of development, such as building high quality software, finding fault early in development, and meeting required budget and time constraints.

iv. It must be able to tailor a process for the special situation in which will be used. By using the model, the development team can understand where that tailoring is occurred

3.1.1 Waterfall Model

In a waterfall model, the stage are depicted as cascading from one to another. As the figure implies, one development stage should be completed before the next begins. Thus, when all of the requirements are elicited, analyzed for completeness, consistency and documented in requirements document, system design activities will be carried out. The waterfall model presents a very high-level view of what goes on during development, and it suggests to developers the sequence of events they should expect to encounter.

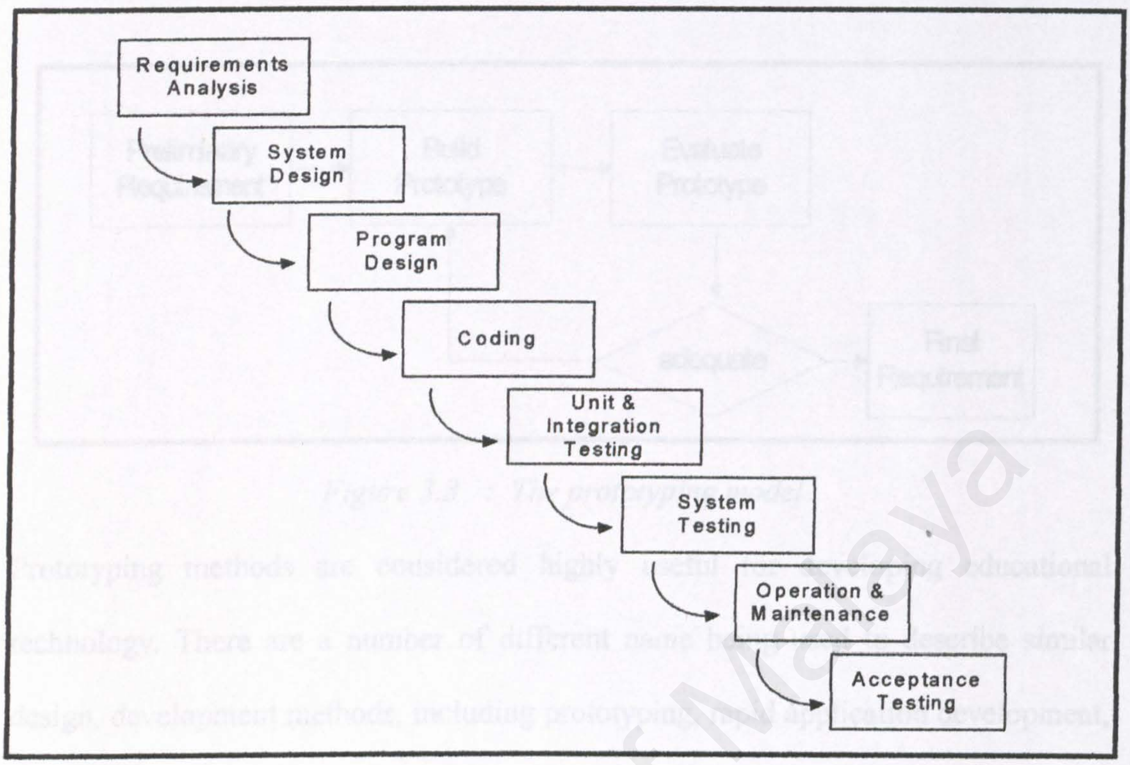


Figure 3.2 : The waterfall model

The waterfall model can be very useful in helping developers lay out what they need to do. Its simplicity makes it easy to explain to customers who are not familiar with software development; it makes explicit which intermediate products are necessary in order to begin the next stage of development. Many other, more complex models are really just embellishments of the waterfall, incorporating feedback loops and extra activities.

However, there are two major drawback concerning the waterfall model. Firstly, it shows how each major phase of development terminates in the production of come artifact (such as requirement, design or code) and this no insight into how each activity transforms one artifact to another, such as requirements to design. Thus, the model provides no guidance to managers and developers on how to handle changes to products and activities that likely to occur during development. Secondly, the model fails to treat the system as a problem-solving process.

3.1.2 Prototyping Model

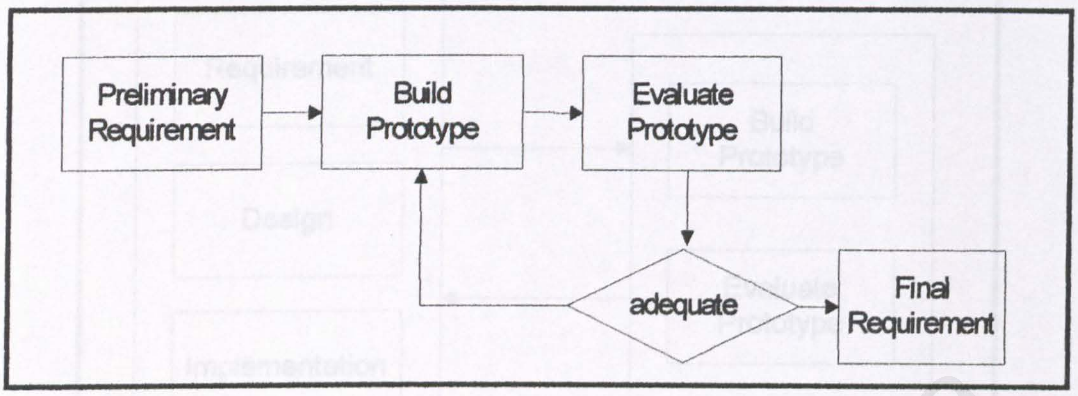


Figure 3.3 : The prototyping model

Prototyping methods are considered highly useful for developing educational technology. There are a number of different name being used to describe similar design, development methods, including prototyping, rapid application development, rapid prototyping and so on. There two main categories of prototyping technique, as outlined below.

a) Rapid Prototyping

Rapid prototyping is used to discover flaws in a design in a short amount of time. The initial design is tested and corrected then tested and corrected again and so on, until a certain level of satisfaction the end product. Others name for this technique include rapid application development. The emphasis is on quick, fast, interactive design.

b) Evolutionary Prototyping

Evolutionary prototyping or software prototyping can use rapid techniques, but the emphasis is more on creating a prototype in software, that will (not necessarily rapidly) form the basis of the final product. In a strict sense, once a satisfactory prototype has been created, the project continues on the more waterfall like method of development.

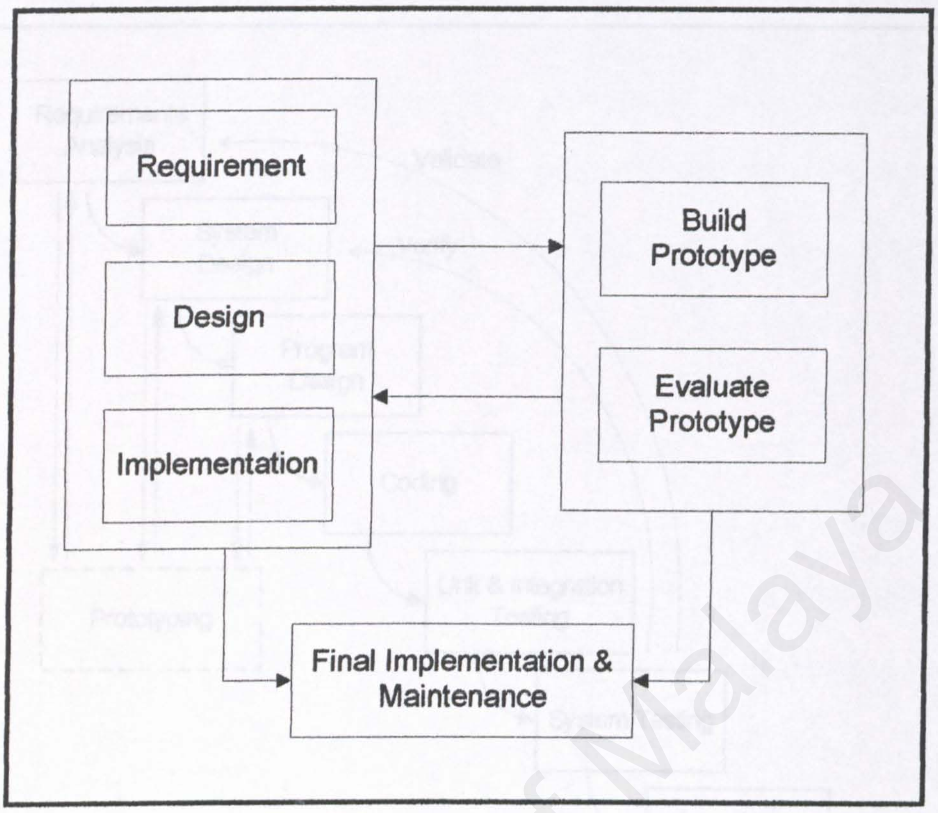


Figure 3.4 : The Evolutionary Prototyping

Prototyping techniques are very useful in situations where the user interface is of primary importance such as developing educational software. There are problems with prototype methods. At some point the prototyping has to stop, add the project continue. It is important the iterations be managed appropriately, and not continue on into actual development, where correcting mistakes is difficult and time consuming.

After a through research and analysis the writer has decided on the combination of **Waterfall and Prototype Methodology** to develop Cloth Simulation.

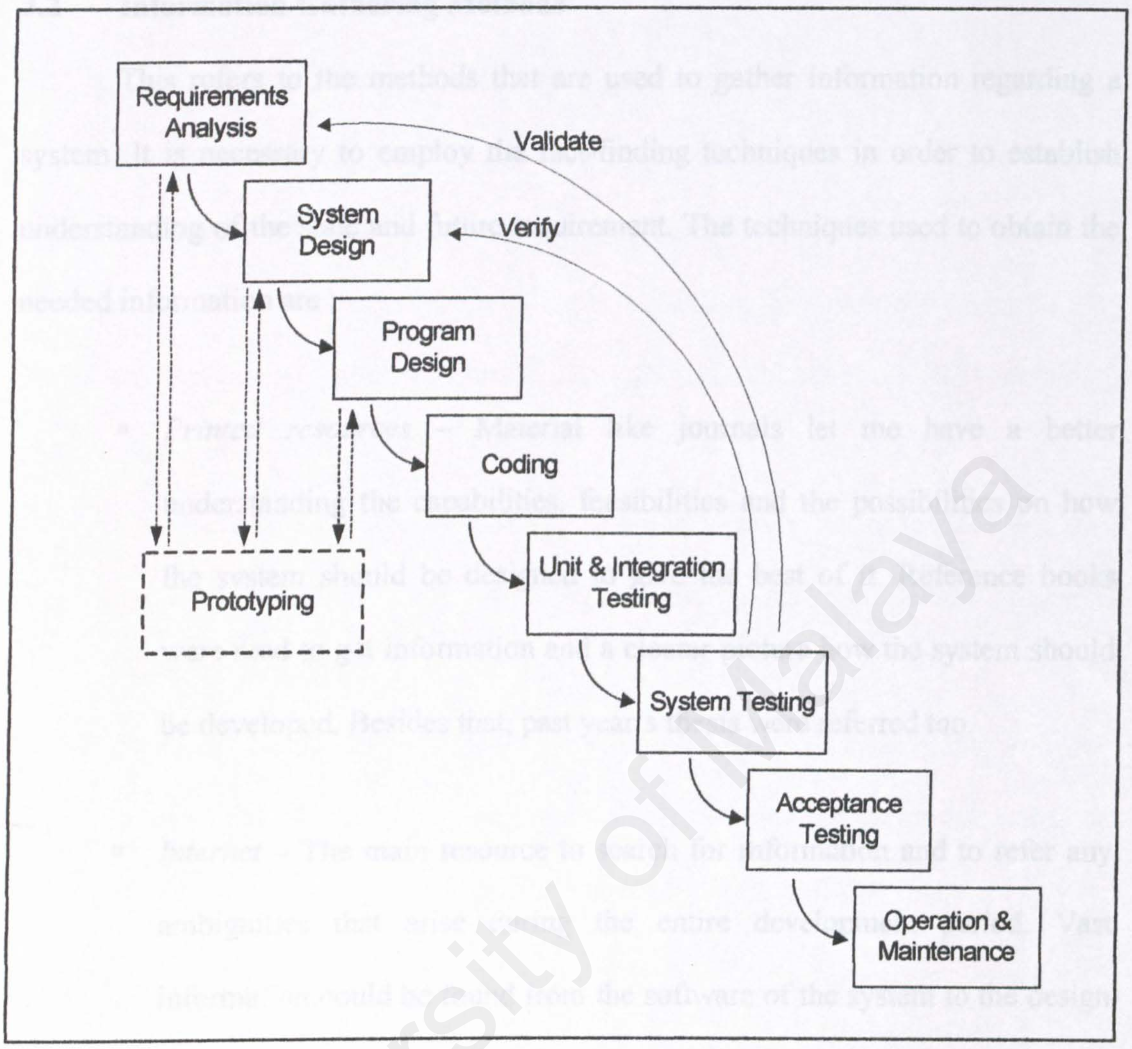


Figure 3.5 : The waterfall model with prototyping

3.2 Information Gathering Methods

This refers to the methods that are used to gather information regarding a system. It is necessary to employ the fact-finding techniques in order to establish understanding of the state and future requirement. The techniques used to obtain the needed information are :

- *Printed resources* – Material like journals let me have a better understanding the capabilities, feasibilities and the possibilities on how the system should be designed to give the best of it. Reference books were read to get information and a clearer picture how the system should be developed. Besides that, past year's thesis were referred too.

- *Internet* – The main resource to search for information and to refer any ambiguities that arise during the entire development period. Vast information could be found from the software of the system to the design of the system.

- *Supervisor* – Useful keyword and continuous advises are given for each section meeting. It is tremendously useful for further helpful survey, error-correctness and yet as a reminder when carrying out the whole system development process.

- *Past Research* – This research is my main reference in developing my system. Although there are different types that has already been built, it helps me to get some ideas on how to work on it, how to start and complete my report and how to start in developing my system.

3.3 Requirements Analysis

The analysis phase during developing a Cloth Simulation is one of the most important phases. In this phase, the entire package initial need such as identifying the objective, scope, functions, modules and others related information could be defined. Through the information's gathered, the outline for Cloth Simulation is prepared. There are two types of requirements. It can either be functional and non-functional requirement [Sommerwille, 2001].

i. ***Functional Requirements*** are services that are offered by the system, how the system reacts towards the input and the characteristic of the system at different situation.

ii. ***Non-functional Requirements*** *are limitations or constraints towards services* that is offered by the system. This includes limitation that exists on system developing process and time limitation

3.3.1 Functional Requirements

A functional requirement describes an interaction between the system and its environment. It also describes how the system should behave given certain stimuli. After analysis has been done, functional requirements that will be developed consists of a several modules.

i. *Main Module* : It's function is to allow user to make options on what do they want to learn in particular. There will be link to other pages that includes; gravity and wind affect module. Clicking on right button for the information that are desired will activate the links.

ii. *Exit Module* : The purpose of this module is to allow user to exit from system.

iii. *Gravity Module* : This module have two sub-module which user can see the cloth in the different situations. First, hanging the cloth on two fixed points and second, resting on a chair.

iv. *Wind Affect Module* : This module have two sub-module which user can see the cloth in the different situations. First, hanging the cloth on two fixed points and second, resting on a chair.

3.3.2 Non-functional Requirements

A non-functional requirements or constrain describes a restriction on the system that limits one choices for constructing a solution to the problem (Pfleeger, 1998). Non functional requirement to this project is described as below :

- *User friendly* – the usage of suitable and meaningful options and icon will help the user to use the system with more confidence, easy and save time. User are allowed to browse and use the site without any problem.
- *Usability* – The application system must be easy to use. Then can enhance and support rather than limit or restrict business processes.
- *Attractive and interactive interface* - With attractive interface, users will be able to enjoy surfing the website. With an interactive multimedia, the users are more comfortable and can have fan while visiting this website.
- *Reliability* – The reliability is to convince the user that this system will make the correct respond and provide error handling ability.
- *Efficiency* – efficiency is understood as the ability of a process procedure to be called or accessed unlimitedly to produce similar performance outcomes at an acceptable or credible speed [Sommerwille, 1995]. Even so, the efficiencies of usage of resources beside time are equally important.

3.4 System Requirement

In this phase developer will need to study and analyze the current system to gain an in-depth understanding if the system and assess its strength and weakness in meeting current and future requirements. Choosing hardware and software is very important in order for the developed system to be successful. Task to choose hardware and software needs to be done particularly to make sure it fulfill system requirement.

3.4.1 Hardware Requirement

As the networks technology grow, the aspects of openness and transparency of the software and hardware layer has reached a point that generally agreed upon. Thus, even though the system evolves both software and hardware, the architecture of the hardware architecture is nor critical. Below are the minimum requirement :

Main Machine	IBM Micro Computer or So Called PC
Hard Disk space	At least 10G
Memory	RAM 128MB
Others	Graphic card

Table 3.1 :   Hardware Requirement

3.4.2 Operating System Requirement

Operating system that referred here means that operating system that operate underlying the application or system being developing, instead of operating system used in the developing phase. Those operating system that fall into consideration domain with be evaluated from the perspective of end-users and developer, which they are available in the market, such as Microsoft Window 98, Microsoft Windows

NT Server 4.0, UNIX and LINUX. After the deep consideration Windows 2000 Professional is chosen. As the description going on the criteria will be explained.

Usability

Firstly, we consider the element of usability, which mean how easily the operating system can be used.

From the view of the installation of the operating system itself, user need a simple, straight forward, user friendly and wizard guided operating system installation. In survey done, we found that UNIX, LINUX and Server 2000 are more difficult to install and configure. Unlike Windows 2000, UNIX or LINUX is not an end-user-oriented operating system known for its user friendliness, the user need user have a certain level of IT knowledge fitness in order to get them installed well.

Technological Ability

Windows professional 2000 is built on Windows NT technology which highly supportive in developing and running a networked application. It provides a dynamic middleware to ease developer in developing the application. Furthermore, because its network technology supportive feature, most small networked based offices and plants are using it.

Most of the software, such as APIs, are developed exclusively for Microsoft Windows operating system, while they were originally developed for other system such as UNIX, but false for the reverse.

Maintenance

Maintenance is very critical in a Software Developing Life Cycle (SDLC), because of the strong background, the technical support is provided well by its vendor. Every detail documentation of Windows 2000 is documented well and freely available online. Unlike Windows 2000 professional, LINUX and UNIX are free and opened software, no specific party is responsible for the technical support. There is no standard version of LINUX and UNIX, thus, it would be a nightmare for the develops as searching for technical and maintenance information.

Cost

Among the entire platform, the pricing of Windows professional 2000 is not so expensive, even some may say some other operating system like UNIX and LINUX are well known as license free Operating System. Though, cost is only evaluate from the license paid. Besides the cost for buying the license this operating system, there is cost of setting up the operating system. For UNIX, which have cryptic user interfaces are hard to manage and give way to high administration cost. In this trade-off, Windows 2000 Professional still stand up to be cost-effective.

Reliability

From the experience of the developer and the information collected from the survey done, Windows 2000 professional is the most stable and reliable operating system among the product of Microsoft such as Windows 98, Windows NT 4.0 and Windows ME. Undeniable, UNIX and LINUX are very reliable as well, but in previous evaluation such as maintenance and cost-effective aspects, they has been removed from the consideration list.

3.4.3 Programming Language Requirement

3.4.3.1 C++

The developer choose C++ because in the first place, a compiled program will always be faster than an interpreted program. Think about high performance spreadsheet program with cell formulas and macros. Just-in-time compilation, it's necessary to compile the program every time programmers load it. That code will be as good as the optimized output from a C++ compiler. Execution speed is one factors; access to the operating system is another.

3.4.3.2 OpenGL

OpenGL routines simplify the development of graphics software—from rendering a simple geometric point, line, or filled polygon to the creation of the most complex lighted and texture-mapped NURBS curved surface. OpenGL gives software developers access to geometric and image primitives, display lists, modelling transformations, lighting and texturing, anti-aliasing, blending, and many other features.

Every conforming OpenGL implementation includes the full complement of OpenGL functions. The well-specified OpenGL standard has language bindings for C, C++, Ada, and Java. All licensed OpenGL implementations come from a single specification and language binding document and are required to pass a set of conformance tests. Applications utilizing OpenGL functions are easily portable across a wide array of platforms for maximized programmer productivity and shorter time-to-market.

All elements of the OpenGL state—even the contents of the texture memory and the frame buffer—can be obtained by an OpenGL application. OpenGL also supports visualization applications with 2D images treated as types of primitives that can be manipulated just like 3D geometric objects.

OpenGL supported on all UNIX® workstations, and shipped standard with every Windows 95/98/2000/NT and MacOS PC, no other graphics API operates on a wider range of hardware platforms and software environments. OpenGL runs on every major operating system including Mac OS, OS/2, UNIX, Windows 95/98, Windows 2000, Windows NT, Linux, OPENStep, and BeOS; it also works with every major windowing system, including Win32, MacOS, Presentation Manager, and X-Window System. OpenGL is callable from Ada, C, C++, Fortran, Python, Perl and Java and offers complete independence from network protocols and topologies.

3.4.4 Software Requirement

3.4.4.1 Microsoft Visual C++

Microsoft has built into its C++ development environment to enable developers to create very advanced applications for the Windows and NT platforms. When Microsoft's developers first came up with the idea behind Visual C++, they decided to take their world-class C++ compiler and create a development environment and set of tools that would enable developers to create Windows applications with a level of ease and speed that was unheard of among C++ development environment. Since that first version, Microsoft has continued to improve the tools that are a part of Visual C++ to make it even easier to cerate Windows applications.

3.5 Conclusion

The reviewing of the methodology being used, procedures that specifies the system requirements in detail, analysis of development tools will help in gaining the advantages and knowledge about the implementation of the proposed system. The following chapter will discuss about the design of the system, which includes the system architecture review, system interface design and other system components.

Chapter 4 : System Design

CHAPTER 4 : SYSTEM DESIGN

4.1 Overview

According to Webster, the process of design involves conceiving and planning out in the mind, and making a drawing, pattern, or sketch of. Design is the creative process of transforming the problem into solution and the description of the solution [Pelfre, 1998].

System design is the evolution of alternative solution and the specification of a detailed computer based solution. During this phase, the detail of how the system will meet the requirement identified during the requirement phase is described. Then the user requirement will be transformed into a working model. A working model is used as guidance to developer before developing the complete system.

4.2 Program Design

Program designing for development of system to be, using up-bottom approach design. These type of designing is described as a big system and decomposed to smaller parts [Kendall and Kendall, 1995].

Advantages of up-bottom design are :

i. It avoids development cycle from developing overall system at one time.

ii. It avoids developing cycle from misdirect from its purposes.

4.2.1 Module Design

Modular approach is needed in system designing when up-bottom approach is used. Modular approach is done by decomposed system into logic modules and manageable.

Advantages on using modular program are :

i. Easier to write and compile modules because it is able to stand-alone.

ii. Modules are easier to manage for modifying certain function and not the whole program.

iii. Easier to understand the characteristic of each module. Developer able to take a module and understand it's functions.

4.2.2 System Structured Chart

Structured chart are used during architectural design to document hierarchical structure, parameters and interconnections in a system. A structure chart differs from a flowchart in two ways : a structure chart has no decision boxes and the sequential ordering of tasks inherent in a flowchart can be suppressed in a structured chart [Fairley, 1985].

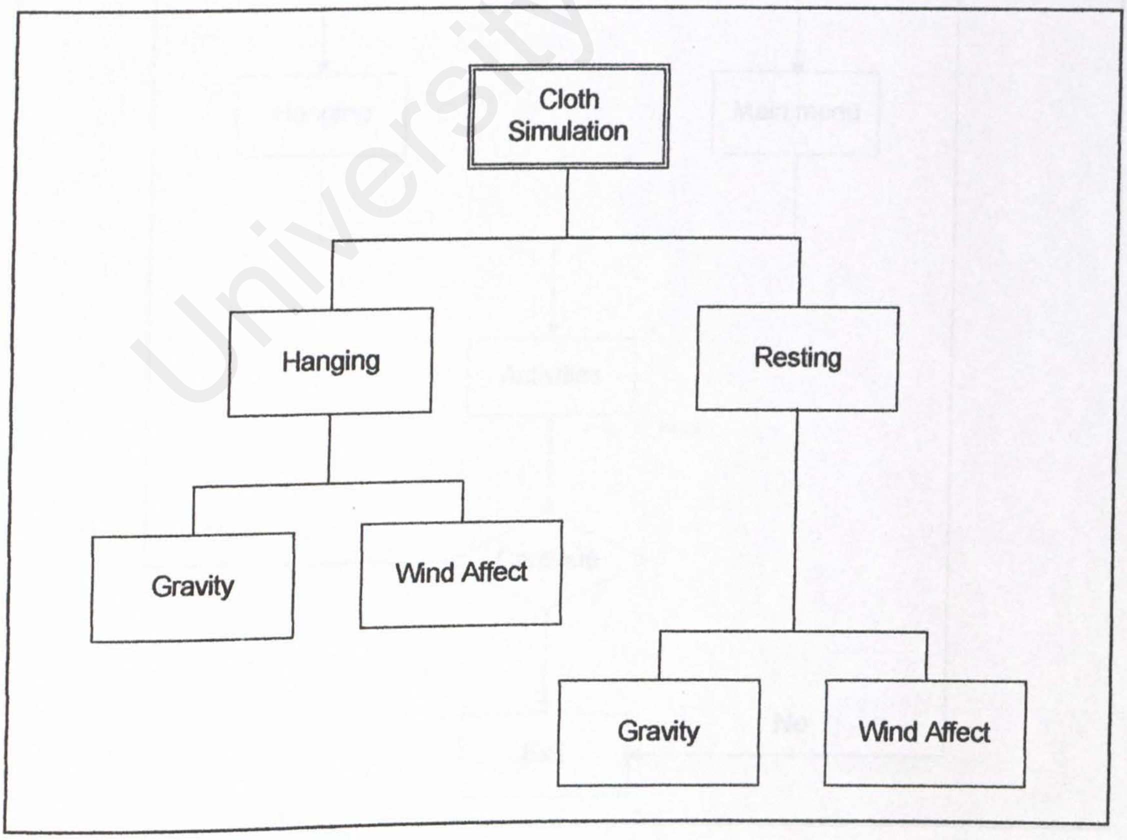


Figure 4.1 : Structured chart for Cloth Simulation

4.2.2 Process Flowcharts

Process flowcharts depicts the breaking of any process down into individual events or activities and display these in shorthand from showing the logical relationship between them. Process flowcharts provide better understanding of processes because it is a pre-requisite for effective development of the system.

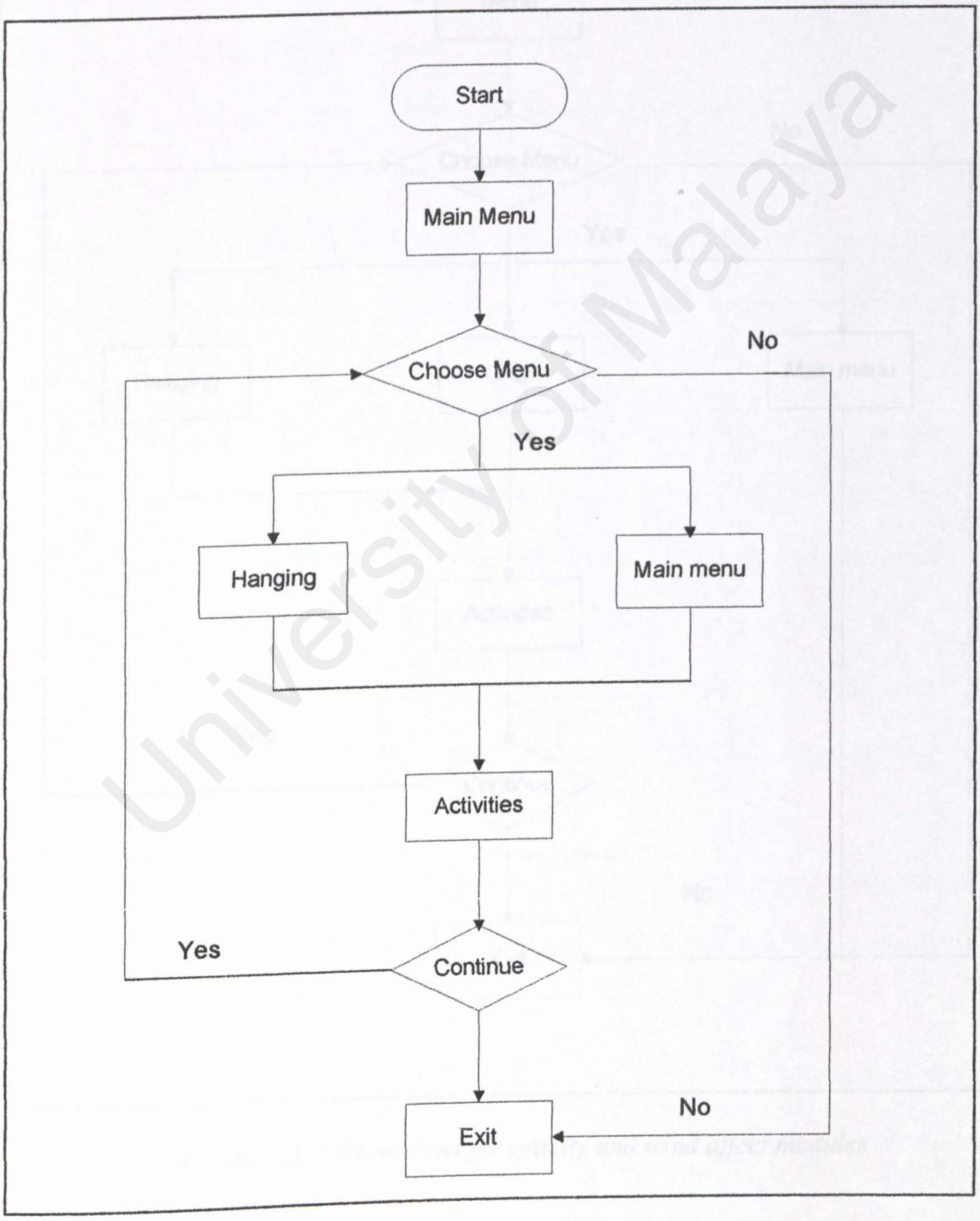


Figure 4.2 : Flow chart for main menu in Cloth Simulation

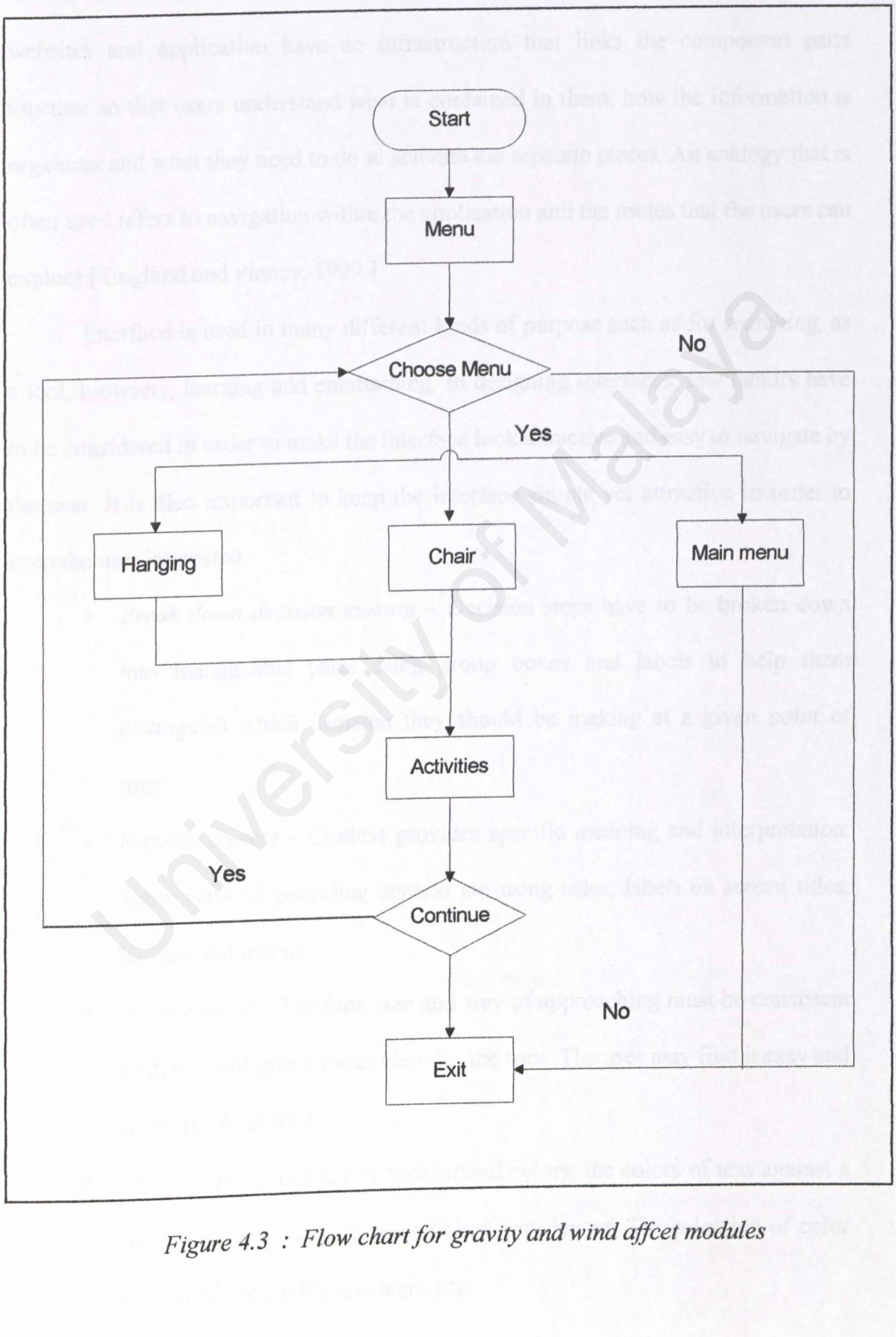


Figure 4.3 : Flow chart for gravity and wind affcet modules

4.3 User Interface Design

The interface is the infrastructure of the application. All online and offline websites and application have an infrastructure that links the component parts together so that users understand what is contained in them, how the information is organizes and what they need to do to activate the separate pieces. An analogy that is often used refers to navigation within the application and the routes that the users can explore [England and Finney, 1999].

Interface is used in many different kinds of purpose such as for searching, as a tool, browsers, learning and entertaining. In designing interfaces, few factors have to be considered in order to make the interface look attractive and easy to navigate by the user. It is also important to keep the interface simple yet attractive in order to keep the user interested.

- *Break down decision making* – Decision steps have to be broken down into manageable parts using group boxes and labels to help them distinguish which decision they should be making at a given point of time.
- *Provide context* – Context provides specific meaning and interpretation. Some ways of providing context are using titles, labels on screen titles, buttons and menus.
- *Be consistent* – The font, size and way of approaching must be consistent so that it will give a focus view for the user. The user may find it easy and comfortable as well.
- *Use of color* – The use of background colors, the colors of text against a colored background and color linked with layout. The selection of color need to be readability and legibility.

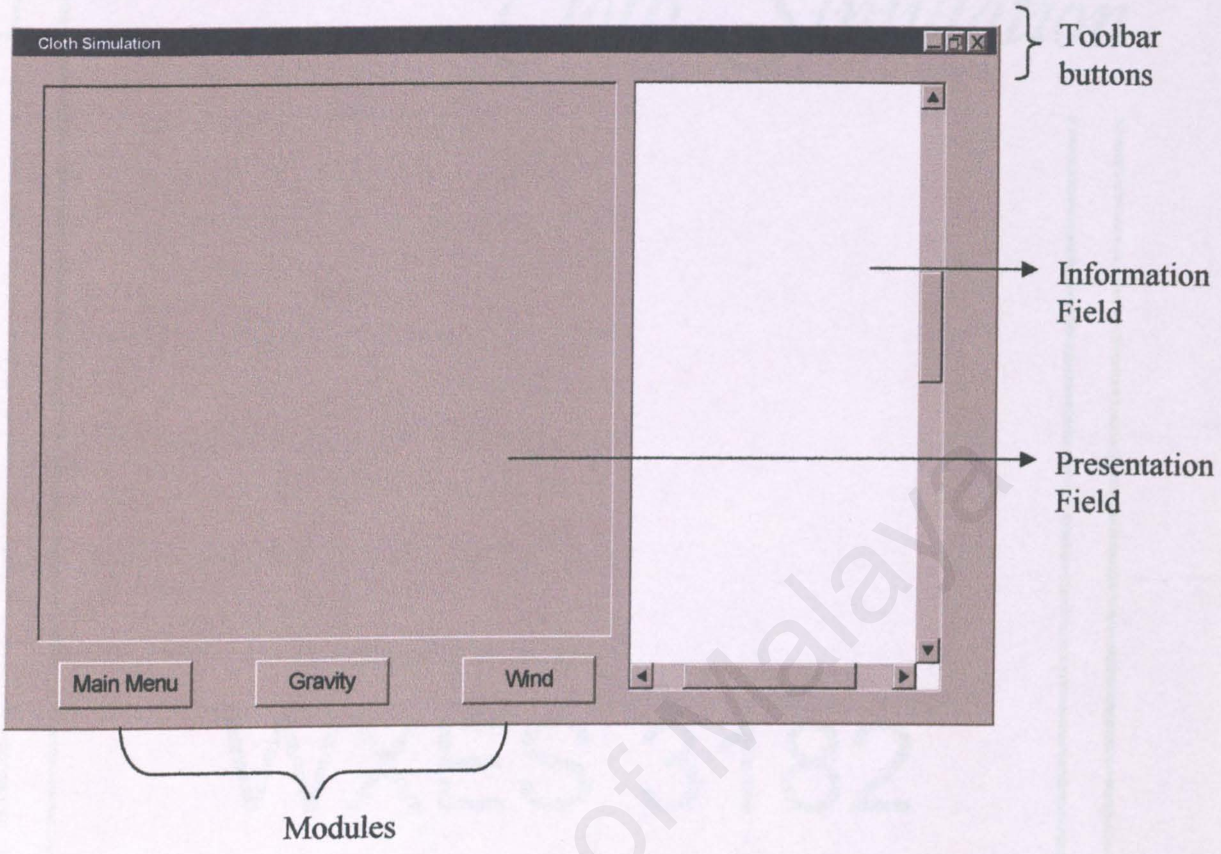


Figure 4.4 : Interface for Cloth Simulation

References :

Toolbar Buttons	- to control the windows size
Modules	- modules in the system
Presentation Field	- cloth in 3D will be present
Information Field	- information and calculation about the cloth can give a view the concept in this system

4.4 Conclusion

Functional and non-functional requirements found in the system analysis stage are turned into design specification. System design will model how operations routine look like logically, and determine the success of the application.

Chapter 5 : System Implementation

WXES 3182

Cloth Simulation

Chapter 5 : System Implementation

CHAPTER 5 : SYSTEM IMPLEMENTATION

5.1 Introduction

System implementation is a process that takes place after the design phase. It will convert the system requirement to the programming codes. This phase will describes how the initial and revised process design put into the real world. Besides that, this chapter will also explain the coding methods, techniques, important scripts involved in the development or implementation of Cloth Simulation as well as the functions or effect that are produced by these methods and scripts.

5.2 Development Environment / Platform

The review and the consideration of the development environment have been discussed in the previous chapters. In the implementation phase only basic minimum hardware and software requirements of a PC is needed, where they are :

5.2.1 Hardware Requirement

The hardware that I have been using for the development of the Cloth Simulation has the specifications as below :

- Pentium 133Mhz
- Memory of 32Mbytes
- Minimum of 10Mbytes hard disk space

  (excluding the development platform)
- Keyboard and mouse as input devices
- Basic input / output devices

5.2.2 Software Requirement

A number of software has being used to develop Cloth Simulation. The software environment needs to be set up are :

- Windows 2000 as the operating system of the machine
- Microsoft Visual C++ 6.0
- Microsoft Word 2002
- OpenGL

We will use **OpenGL Utility Toolkit (GLUT).** Its API includes the standard operations for window system and allows use of keyboard and mouse. In order to use glut with OpenGL programs , we need the following three files:

1. **glut.h**	in	c:\Program Files \ Micro Soft Visual Studio\VC98\include\gl
2 **glut32.dll**	in	c:\WINNT\System32
3. **glut32.lib**	in	c:\Program Files \ Micro Soft Visual Studio\VC98\lib

These files are available at 'http://www.xmission.com\~nate\glut.html'

After setting up these, we may proceed to the section of writing codes.

5.3 How cloth simulation work?

5.3.1 Mass-spring models of cloth

The simplest way to model a piece of cloth for dynamic simulation is with a *mass-spring system*. This is a special form of particle system which consists of a number of particles of (usually small) mass which are interconnected by (linear) springs.

The simplest type of spring, the linear spring connecting two particles i and j, is shown schematically in Figure 5.1. A linear spring has a certain *rest length* rij that it will attempt to maintain by exerting the necessary restoring forces on the two particle endpoints in case it is stretched or compressed to a different length. The force the spring exerts on one of those endpoints is dependent on its current length lij.

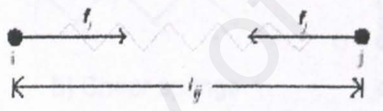


Figure 5.1: Forces a linear spring exerts on the two particles it connects

5.3.2 Surface topology

Just as a real piece of cloth can be created by weaving together fibers in a large number of ways, connecting particles in a particle system model for cloth can be done in at least as many ways. The two most common topologies used in cloth simulation are those where the cloth surface is modeled as a triangular or quadrilateral mesh of particles, see Figure 5.2.

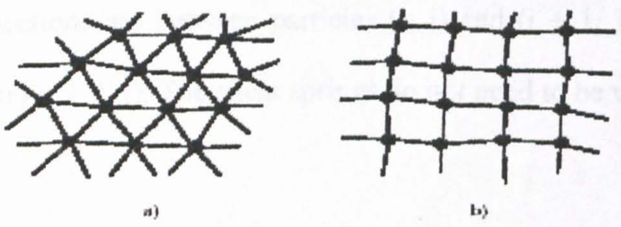


Figure 5.2: A cloth surface modeled with a triangular and quadrilateral mesh of particles.

The use of triangles has certain advantages, such as easier formulation of accurate physical behavior for each surface element and easier calculations for collision detection than compared to quadrilaterals. However, we chose to use a quadrilateral structure, since it captures the warp/weft structure of real cloth much better than when using triangles and so we get a bit of extra realism `for free', although accurate modeling of cloth's physical properties is not our primary concern.

5.3.3 Types of springs

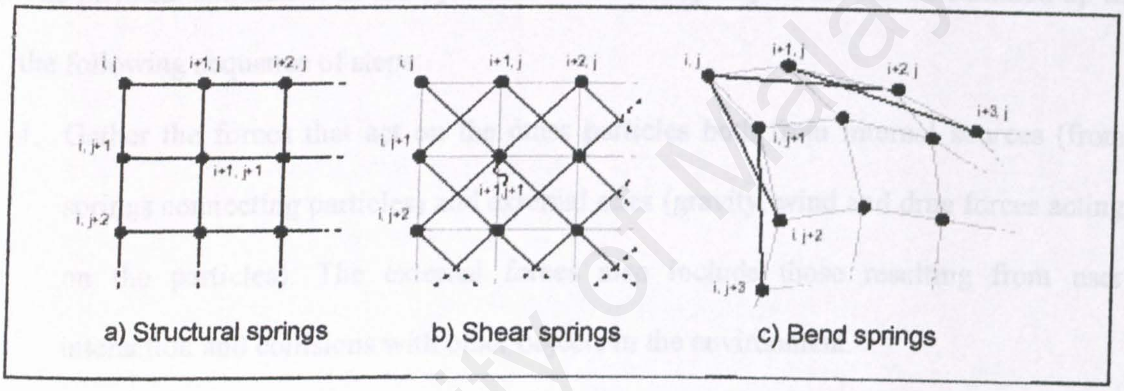


Figure 5.3: The three different types of springs used in the mass-spring model of cloth

The first type, called *structural springs*. That is, there are horizontal connections between particles (i; j) and (i + 1; j) and vertical connections between particles (i; j) and (i; j + 1).

The second type of springs, called *shear springs,* make in-plane shearing of the cloth more difficult. Shear springs connect particles in a diagonal way in each cloth quadrilateral. Connections are between particles (i; j) and (i + 1; j + 1) and between particles (i; j + 1) and (i + 1; j). The shear springs do not need to be very stiff in order to be effective.

The third and last type of springs used here are *bend springs*. These springs make bending the cloth harder. Although real cloth bends very easily, it still has a small resistance to this kind of deformation. These springs connect every other particle along the two directions of the rectangular cloth structure, connecting particles (i; j) and (i + 2; j) and particles (i; j) and (i; j + 2).

5.3.4 Evolving a mass-spring system

The physical simulation of the dynamics of a mass-spring model can be summed up as the following sequence of steps:

1. Gather the forces that act on the mass particles both from internal sources (from springs connecting particles) and external ones (gravity, wind and drag forces acting on the particles). The external forces also include those resulting from user interaction and collisions with other objects in the environment.
2. Calculate the particle accelerations resulting from these forces.
3. Update the particle velocities and positions using the found accelerations.

The details of *Step 1*, the gathering of the forces, is not that important at the moment. We will simply assume that when the first step is completed, each particle i will have received the accumulation of the forces acting on it, fi for a particle i.

Step 2 from the list, calculating a particle's acceleration from the forces acting on it, is easily done using Newton's second law, f = ma. The acceleration of a particle i is ai = fi=mi, where mi is the particle's mass. By repeating this for each particle, all the accelerations can be computed.

Step 3, updating positions and velocities, can be performed in a large variety of ways and the different methods of cloth simulation will each do this step differently. This area is also where the largest part of cloth simulation research is done.

5.3.5 Cloth Simulation Algorithm

4 passes :

- Each passes renders a single quad with a fragment program:

1. Perform integration (move particles)
2. Apply constraints:
   - Distance constraints between particles
   - Sphere collision constraint
3. Calculate normals from positions using partial differences
4. Render mesh

5.4 Overview of OpenGL API (Application Programmer's interface)

OpenGL is an application programmer's interface (API) that allows programmers to write graphics programs that access graphics hardware.

Main advantages are:

(i) OpenGL is close enough to hardware so those programs written with OpenGL run efficiently.

(ii) It is easy to learn and use.

(iii) Programs written using OpenGL are portable to any computer that supports the interface.

(iv) Implementations are available for most hardware and operating systems. Implementations range from pure software to pure hardware implementation

Disadvantages are:

(i) does not have input functions

(ii) does not have windowing functions.

5.4.1 Three views of OpenGL

(i) The programmer's view

Most application programs consist of three major elements:

(a) specifying geometric objects

(b) describing properties of these objects

(c) Defining how these objects should be viewed.

This gives a way of categorizing the OpenGL functions but it does not tell us how OpenGL works.

(ii) The OpenGL state machine

The inputs are description of geometric objects that are specified to function calls.

The machine produces the image.

The output is an image that we see on the display.

How the machine processes its inputs depends on its state. The state of the machine is set by OpenGL function calls.

(iii) The OpenGL Pipeline

Based on a pipeline model.

Primitives are generated in the application program and flow though the pipeline.

5.4.2 OpenGL functions:

OpenGL contains more than 200 functions for building application programs. We can group them by their functionality.

(i) Primitive functions

These are functions that define elements that can produce images on the screen.

(a) geometric primitives –such as polygons, lines etc

(b) bitmaps

(ii) Attribute functions

These control the appearance of functions such as color, line type, light sources etc

(iii) Input functions

These are not part of OpenGL. These are contained in the library called GLUT.

(iv) Viewing functions

Determines the properties of camera. Controls focal length. Wide angle, telephoto.

(v) Control functions

These allow us to start and terminate OpenGL programs and to turn on various OpenGL features.

OpenGL is not object oriented. We shall be using C language binding. This means no overloading and other object oriented features.

5.5 Coding and Coding Approach

Coding of a program can be considered as the building block of a system or application, it is written in a particular syntax to be a programming statement. An executable collection of programming statements that follows certain routine will be a program. Yet doing coding without a proper strategy and approach will be messy. At the same time, it will be hard to match the actual design. Thus, pseudo codes are necessary.

5.5.1 Pseudo Code

Pseudo code is an intermediate language between the actual design or blue print and the statements with correct syntax. The main advantages of writing pseudo codes are :

- Programming statement needs to be written in the correct syntax, whereas pseudo codes does not. So, a programmer can define the routines of the statement before searching for the specific codes , functions, classes and etc.

- Translating pseudo code is, in certain cases, easier than translating graphic data flow.
- A clear view of logic or semantic can be obtained. Since a programmer can easily got lost while cracking his / her head looking for the suitable functions or classes.
- Since it is a human language like statement, it can be a very good documentation of the coding.

A well written pseudo codes means the 50% of the coding is done.

5.6 Documentations

Documentations here means the record of programming process and routine. It also record down the techniques used in this phase. It focuses on the descriptions accompanying a collection of programs. It explains to a reader, most probably one of the stakeholders in developing the software. There are two type of documentations here; Internal documentation is descriptive material written directly within the code; whereas another documentation is external documentation. Which both of it are highly crucial especially working in a team. [Pfleeger, 1998]

5.6.1 Internal Documentation

Internal documentation contains information directed reader who is reading the source code of the programs. Thus, summary information is provided to identify program and describe its data structure, algorithms and control flow. For such documentation, there are a few methods to implement :

- **Header Comment Block**

  It is written at the beginning of each component in a set of comments that includes author name, matrix no., title and subtitle.

- **Other Program Comments**

  A header block acts as the introduction of the coding. An additional comment will enlighten reader as they go through the programmatic statement. This helps them to understand how the intentions described by the programmer at the header block, is implemented in term of code.

- **Meaningful Naming System**

  An effective or meaningful naming variables, operations or objects are already a kind of clear documentation by itself. It makes the statement so obvious that reader can spontaneously understand the statement by the very first sight.

5.6.2 External Documentation

External documentation is intended to be read by those who never look at the actual code. For example, an individual that reviews the external documentation when considering modification or enhancements. External documentations also explains things more detail as it might be within program comments. External documentation is part of the overall system documentation. The closest instance is this document you are reading right now. This is an overall documentation for the components developed. [Pfleeger, 1998]

A method need to be stressed here is continuous documenting, which is a demand that documentation is to be done parallel to the code itself. Programmers writes document in real time, as program has been coded; instead of recall the program routine after sometime upon finishing coding.

The external documentation prepared for this project are :

- **User manual**

User manual is a reference or guide for programmers. It will explain and describes how the component can be used in their application.

Please refer to Appendix A for more details.

- **Sample Coding**

The sample of coding that have developed and deploy in this system will be shown partly (not all in order to maintain intellectual property)

It is as references for the readers to know how OpenGL Coding looks like.

Please refer to Appendix B for more details.

Chapter 6 : System Testing

CHAPTER 6 : SYSTEM TESTING

6.1 Introduction

System testing is a verification and validation process. Verification and validation are sometimes confused but actually both are different activities. Testing in focused on finding faults and there are many ways we can make our testing effort more efficient and effective. It is an essential series of steps that helps assure the quality of the eventual system. A successful testing will uncover errors in the software and demonstrates that system functions appear to be working according to specification.

To develop and deliver robust systems, we need a high level of confidence that [Beizer, 1990] :

- Each component will behave correctly
- Collective behaviour is correct
- No incorrect collective behaviour will be produced

6.2 Types of Errors

One reason why quality assurance is needed is because computers are infamous for doing what uses tell them to do, not necessarily what users want them to do. To close this gap, the code must be free from errors of bugs that caused unexpected results, a process called debugging [Ali Bahrami, 1999].

Debugging is the process of finding out where something went wrong, and correcting the code to eliminate the errors of bugs that cause unexpected results. A software debugging system can provide tools to find errors in programs and corrects them. Below are a few types of error that encountered when the programs are run :

- *Language (syntax) errors* result from incorrect code. These are the most common type of error that usually occurs. It is also the most easiest type to detect, for most part we need no debugging tools to detect them. The very first time we run the program the system will report the existence of these errors. For example, type mismatch for parameter of function, especially there are a few types of character string data type or object (char*, char[], BSTR, wchar,_bstr..).

- *Run – time errors* occur and are detected as the program is running, when a statement attempts an operation that is impossible to carry out. This error happens when the operating system try to execute an operation that cannot be run under the system.

- *Logic errors* occur when code does bit perform the way programmers intended. The code might syntactically valid and run without performing any valid operations and yet produce incorrect results. Only by testing the code and analyzing the results can programmers verify that the code performs as intended. Logic errors also can produce run – time errors.

The test will not find everything, not they will cover at least the higher visibility system interaction bugs [Marick , 1995].

6.3 Testing Organization

Generally, testing are involves several stages. There are actually three stages of testing to by gone through. The three stages are unit testing, module testing and integration testing.

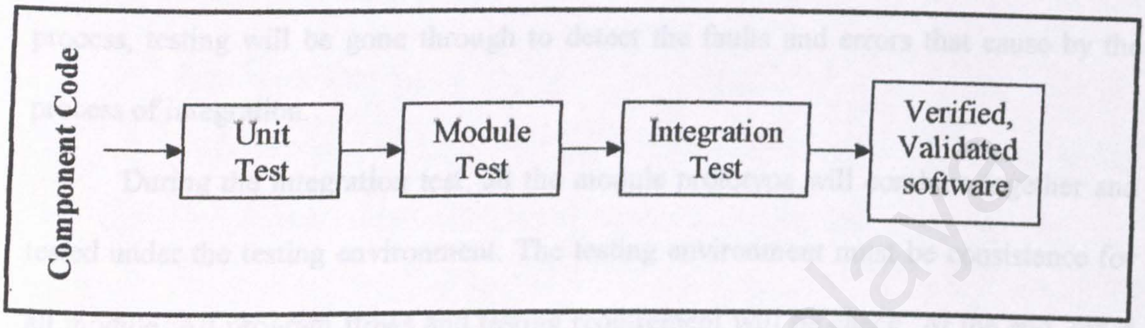


Figure 6.1 : Testing Steps

6.3.1 Unit Test

Unit test is very important to make sure the system can be correct without cause any side effect to the system. Unit test also make sure each of every sub module can be execute without error. Each program unit will be tested to make sure the correctness and able to run without error. Unit test is done under a controlled environment whenever possible.

6.3.2 Module Test

Module test will be apply when all module are done. This is to make sure all codes within a programming can function well and correctly when the code are integrated together. Before the modules are integrated, there are a few value cannot be manipulate correctly. That's why the correct values need to put in to run the testing process. After that, each module will be examine and if there is any error appears, the part of module are determine and unit test will go through again to detect the error.

6.3.3 Integration Test

Integration test is to test whether the whole system can be execute as a program. That is also to make sure all the module can be function with each others. When all the modules are meet the requirement, they will be integrated as a system. During the integration process, testing will be gone through to detect the faults and errors that cause by the process of integration.

During the integration test, all the module prototype will combine together and tested under the testing environment. The testing environment must be consistence for all module. All program flows and testing requirement will be check. At the end, other users will test the system to get the feedback and comment about the develop system.

6.4 Testing Strategies

The extent of testing system is controlled by many factors, such as the risks involved, limitations on resource and deadlines. In spite of these issues, we must deploy a testing strategies that does the best job of findings defects in a product within given constraints. There are many testing strategies, but most testing uses the combination of those, which are black box testing, white box testing, top down testing and bottom up testing. However no strategies truly can prove the correctness of a system; it can establish only its "acceptance", [Ali Bahrami, 1999].

- **Black Box Testing**

Is used to represent a system whose inside working are a bit available for inspection. The test item is treated as "black", since its logic is unknown.

- **White Box Testing**

Assumes that the specific logic is important and must be tested to guarantee the system's proper functioning. One form of white box testing is called path testing, as it make sure every path in the object is tested.

- **Top Down Testing**

Assumes that the main logic or object interactions and systems of the application need more testing than an individual object's method or supporting logic.

- **Bottom – Up Testing**

Starts with the details of the system and proceed to higher level by a progressive aggregation of detail until they collectively fit the requirement of the system.

6.5 Summary

Testing is a balance art of art, science and luck. It may seem that everything will fall into place without any preparation and a bug free product will be shipped. However in the real world, we must deploy a test plan for locating and removing bugs. [Ali Bahrami, 1999]

There are no ideal test plan and debugging, however, by selecting appropriate testing and test plan, we should be able to locate errors in the component and fix them by utilizing debugging tools.

In road maps of tests we had gone through, a few errors and bugs were encountered. Yet, all the bugs and errors were corrected in a short duration without affecting the whole timeline of entire project.

Chapter 7 : System Evaluation

CHAPTER 7 : SYSTEM EVALUATION

7.1 Introduction

Evaluation is a process that occurs continuously at all phases of the system development. Evaluation phase was to determine the extent to which the system's expected outcomes have been realized, and the prescriptive value of the process where extraneous factors were taken consideration.

In system evaluation we will discuss on problems encountered, solutions for encountered problems, user's evaluation, system constraints, future enhancements and experience gained in developing the project. Lastly we will conclude the quality for the system been developed.

7.2 Problems Encountered and Recommended Solutions

Along with the development stage of Cloth Simulation, a few problems were encountered. The following are the problem encountered :

7.2.1 Determining the Scope of System

Since there is no less experience for developer in developing a system, it was hard to determine to which extent to define the scope of the system so that it can be completed within the given time frame. The problem has been added by defining the work scope of project for the developer. A lot of time had been taken in discussing the work scope. By reference to a few software engineering reference books and articles, we had decided to develop the project using component based software development.

Following the right track, I had also engaged the iterative and waterfall and Prototype model for my development methodology. By selecting the ideal methodology, the process of development is run with a guideline, and work become much more easier.

7.2.2 Time Constraint

Time is always the most important resource of the project. I face the problem from the beginning of the project which a schedule with milestones is to be sketched out. Such tasks is challenging and sometimes hard to cope for a junior developer like me. I hardly estimate the amount of time that should be spent for particular job. For an inexperienced developer , it is very likely to over or under estimate the effort needed for certain tasks. The uncertainty can only be reduced by referring the masterpiece of our seniors, which are provided at the thesis document room.

Following schedule is the second challenge, some of the time space given may not be sufficient, and some delays happened. Extra time was used for certain process. A lot of adaptation needs to be done in order to tune the time line back onto the correct track.

The project uses a developing time that is a bit longer than a commercial product in the market, this is due to the effort that being put in developing the application. I am not a full time developer, thus concentrating on the developing work is a tough job. My time is also occupied with my degree education programs. However , I am still feeling relieved that the project is done in the given time.

7.2.3 Lack of Mastery in Technical & Programming Skill

In prior to developing the project, developer was only equipped with a basic programming knowledge of OpenGL, VC++ and VB, fundamental understanding of certain tools and a few concept of latest technologies. Though, what a developer need is of course more than that. Mastery of technical skills gives a direct affects in time usage upon the development process.

In selecting the idle tools and technologies, consultancy of experienced seniors play a very important role. I would need to find a suitable technology to suit my Cloth Simulation. Then information regarding the technologies need to be gathered thoroughly. Tools and technologies chosen must also suited developer's lack of experience in developing a system.

At first, the application was planned to be developed using VB.net. However, it is possible for the developer to have done it alone since this project is single developed. A team of two is the more suitable team to develop application such as this.

7.2.4 User Interface

User's interface is an important aspect in a system's development to ensure users can use the system easier. However, in this system, to develop the interface is the hardest part. To insert a button that leads the user to other module is a tough job. Plenty of time is need to call different libraries and procedures.

I have consulted more than one lecturer and seniors expert in OpenGL, and they have suggested to me, to use VB.NET. However, it is impossible to complete the task as time left is very limited major changes in source code have to made. As an alternative, I instead have used the Console Application to display the orders for users to use. Users have to use manually.

7.3 Strengths of Project

The implementation, testing and evaluation of the project has defined several strengths of this project which are worth mentioned here :

7.3.1 Interesting 3D Performance

Cloth Simulation is a 3D application. Its 3D modelling and environment are the main attractions that could inspired users with great imagination.

7.3.2 Object Interaction and 3D Environment

Users can interact with the system by only pressing button on the keyboard or by clicking the mouse. Repetition are allowed as many as the user wants in order to have a setter look at the graphic. Thrills of virtual environment can be felt by users while using Cloth Simulation.

7.3.3 Real Images

Images displayed will look like an actual one. It can interact with side effects such as gravity and wind.

7.4 Limitations of Project

Not all humans are perfect. I am not surprised if somebody said that my system is very weak. However, it still can fulfil the main objective of the system.

7.4.1 Interface

The interface is the weakness of my system. However, instructions will be displayed on the screen as user's references. Users could to alternately change the modules they intended to use manually.

7.5 Future Enhancement

Due to lack of time, there some ideas about the project that couldn't be completed or implemented. These ideas hopefully will be added into the Cloth Simulation in future.

- Self collisions of the cloth allowing the cloth to exhibit more complex deformations. Self collisions happen when the cloth penetrates parts of itself. Checking for self collision requires a more complex and efficient collision algorithm since we need to test a greater number of intersecting polygons and we need to handle multiple collisions at a time. Our model handles only simple collisions between two objects one at a time. In this simulation we will ignore self collisions with the cloth although it allows us to model more complex behavior such as crumpling of the cloth.

- The ability to compose primitives such as boxes to build more complex objects like tables, the table would have a surface and four legs each being boxes.

- Draping of more complex objects like people.

- The Modeling of friction, currently when the cloth vertex points collide with an object their velocity become zero

7.6 Knowledge and Experiences Gained

Knowledge and experiences were gained during each phase of the project development. A developer should not only gain knowledge and experience in technical aspects, but in aspect of managing a project as well. In the following section, knowledge and experiences gained will be discussed according to the phase throughout the project development.

7.6.1 Introduction Phase

Introduction phase is actually involving part like project objectives, project scope, project limitation and target user. The most challenging part of all is project scope defining. If there are any errors in defining the scope of the project , it is always time consuming. In some cases, the project has to be start from the beginning. A clear project scope will lower down the development risks, and be a good guideline for the following phases.

7.6.2 Literature Review

Literature review is the part the chapter is knowledge based. In the chapter, developer have done a lot of technical review, besides describing the overview of Cloth Simulation and analysing existing systems.

In analysis of existing systems, I have reviewed the architecture of each system and application such as the Syflex, Stitch Lite, ClothSim and others. It helped me a lot to understand the design of Cloth Simulation. Ideas collected can also be implemented in other application.

On the other hand, technical review has given me the chance to analyse current technologies. It includes further understanding of variety of programming languages, reusable component / object technologies, network technologies and strength of various operating systems.

Only with correct tools and matched technologies, the constraints in the project development could be minimized.

7.6.3 Methodology and System Analysis

This is the part where the knowledge of software engineering and software development management are actively used. One of the missions in this chapter is to select one combination of system developing methodology. It was a very good experience in analysing which methodology is idle and suits fairly with the system. A developer would have to consider the tools and techniques that may help us along the development line.

Knowledge of system's analysis is very practical here. By listing out the requirements of the application design, the consideration of the technologies used was carried out. Such experiences gained in this phase will surely help in the following system development and also system maintenance works.

7.6.4 System Design

System design is the evolution of alternative solution and the specification of a detailed computer based solution. During this phase, the detail of how the system will meet the requirement identified during the requirement phase is described. Then the user requirement will be transformed into a working model. A working model is used as guidance to developer before developing the complete system.

7.6.5 System Implementation

This is the part where technical knowledge counts. In this chapter , the main role Player were system analyst and programmer. System analyst will decide layout of the coding and programmer will deploy it into lines of codes. How good will this phase be will depend on the layout given out by the system analyst and how well the programmer codes. This can be evaluated by the tools mastered by the programmer. The programmer needs to explore the developing tools as well. As I'm using OpenGL in VC++, I would have to master OpenGL programming. In this phase of developing, I've learned and gained experiences from creating the project in an Integrated Environment. Every steps involved needs my technical knowledge.

7.6.6 System Testing

A test plan is developed to test every functional unit involved. A set of test is documented to give a route map of component testing.

7.8 Conclusion

Developing Cloth Simulation is a very novel and exciting experience. It offers practical experience in designing and developing a system in real life usage and working environment. Still more, I learned a lot and have the chance to apply my knowledge and theory gained from my classes at the FCSIT especially in programming language.

The development process was defined through a combination Waterfall and Prototype Model, which has made the system most expandable in terms of alternatives. I should say that I do enjoy the experience and learned a lot from mistakes and trouble I faced.

Lastly, all the efforts that I had put just for this projects, the adventures, ups and downs would be my source to be more high spirit for the sake of our field of computer science and information technology.

Cloth Simulation

- **Appendix A**
- **Appendix B**
- **References**

APPENDIX A

USER MANUAL

Installing GLUT

If you don't have it already, you will need to install GLUT before you can follow the rest of the directions.

Download the GLUT libraries and header files. There's no installer, so you will have to put the files manually in places where the compiler will find them. Assuming Developer Studio is in the standard places, put:

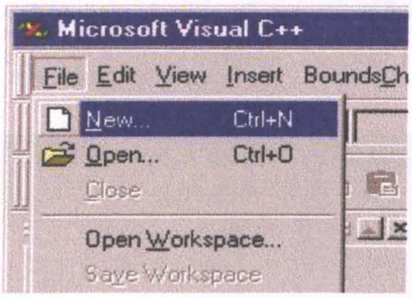
1. glut.h in C:\Program Files\Microsoft Visual Studio\VC98\include\
2. glut32.lib in c:\Program Files\Microsoft Visual Studio\VC98\lib
3. glut32.dll in c:\windows\system32 (or c:\WINNT\system32, if you're running Windows NT/2000)

You should only have to do this once.

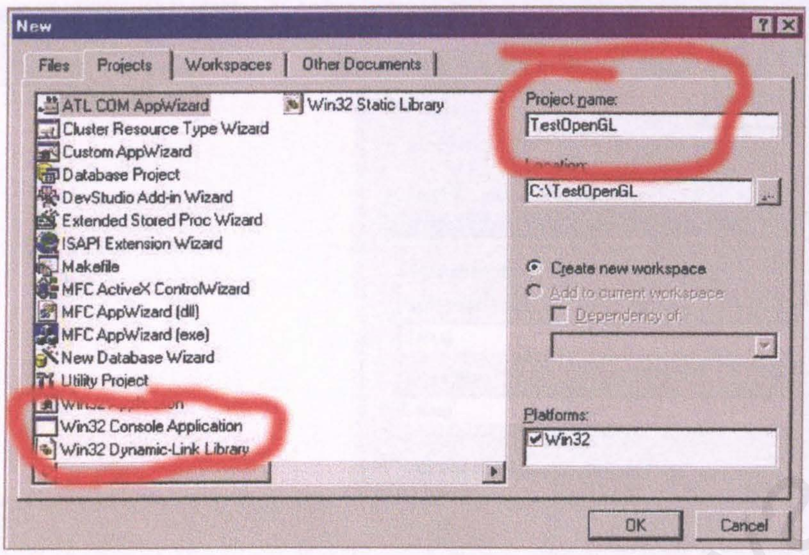
Writing an OpenGL program on Windows

This document will guide you through the steps needed to write an application that uses OpenGL on the Windows platform. We assume you're using Microsoft Visual Studio 6.0. If you're not, you should be able to adapt these instructions to suit your development environment.

1. Start Visual Studio.
2. Create a new project by selecting "New" from the "File" menu.



3. Select "Win32 Console Application" from the list of choices, and give your project a name.



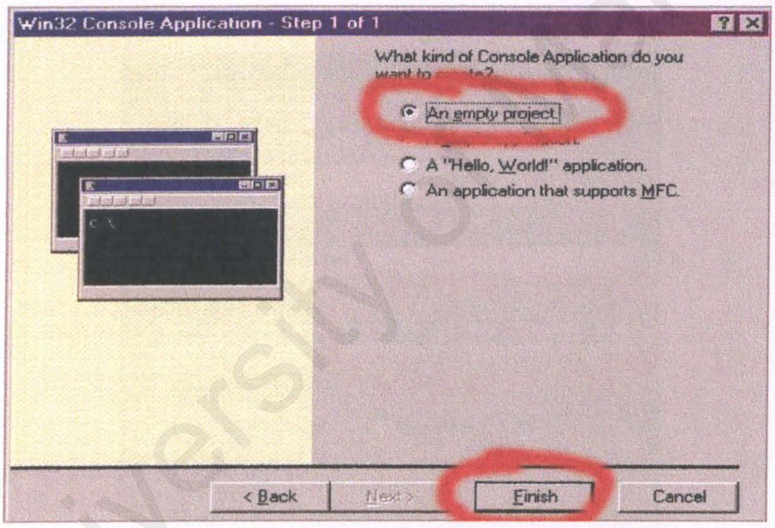
Project name: TestOpenGL

Location: C:\TestOpenGL

[✓ Create new workspace]

Platforms: [✓ Win32]

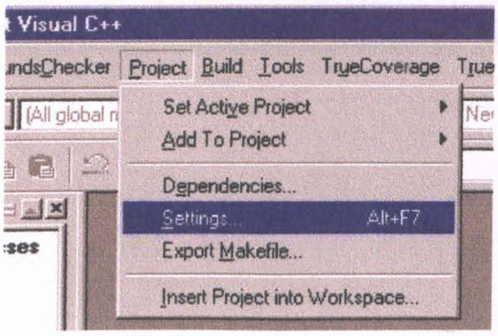
4. Create an empty project.


Win32 Console Application - Step 1 of 1
A simple application.
A "Hello, World!" application.
An application that supports MFC.
< Back
Next >
Finish
Cancel

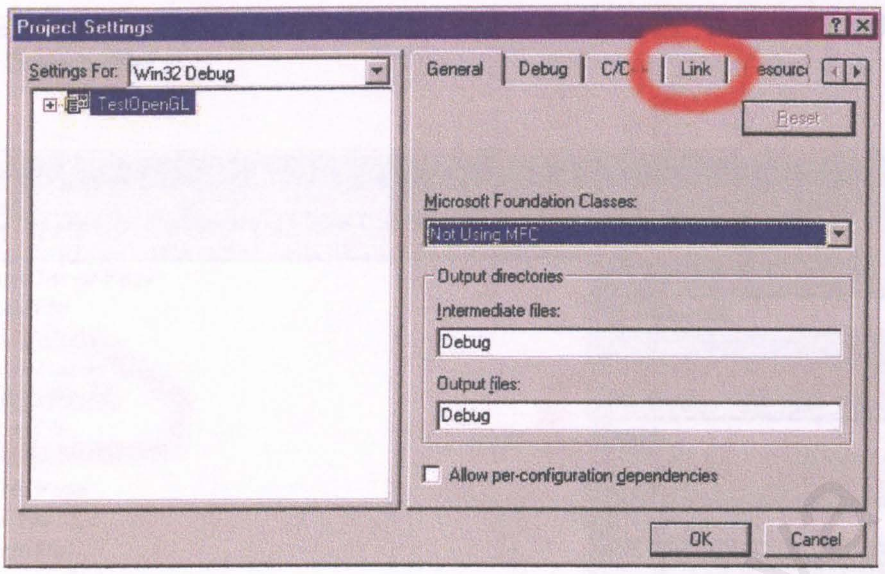
What kind of Console Application do you want to create?

[✓ An empty project.]

5. From the "Project" menu, choose "Settings...".



6. Choose the "Link" tab.



Settings For: Win32 Debug

TestOpenGL

Microsoft Foundation Classes: Not Using MFC

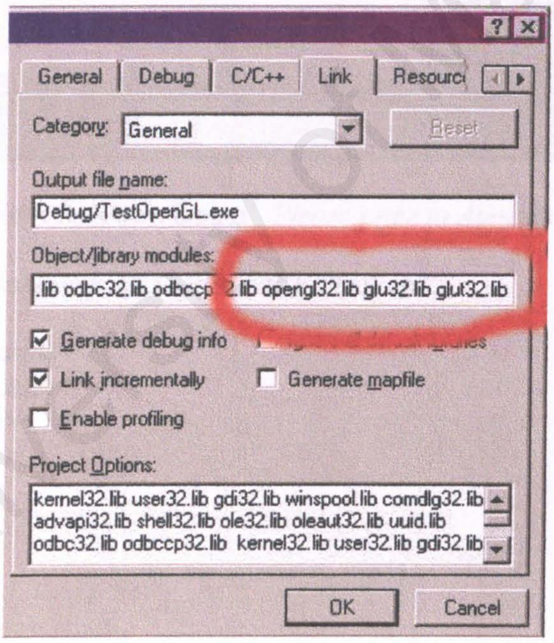
Output directories

Intermediate files: Debug

Output files: Debug

☐ Allow per-configuration dependencies

7. Add "opengl32.lib glu32.lib glut32.lib" to the end of the list of libraries.



Category: General

Output file name: Debug/TestOpenGL.exe

Object/library modules: .lib odbc32.lib odbccp32.lib opengl32.lib glu32.lib glut32.lib

☑ Generate debug info

☑ Link incrementally ☐ Generate mapfile

☐ Enable profiling

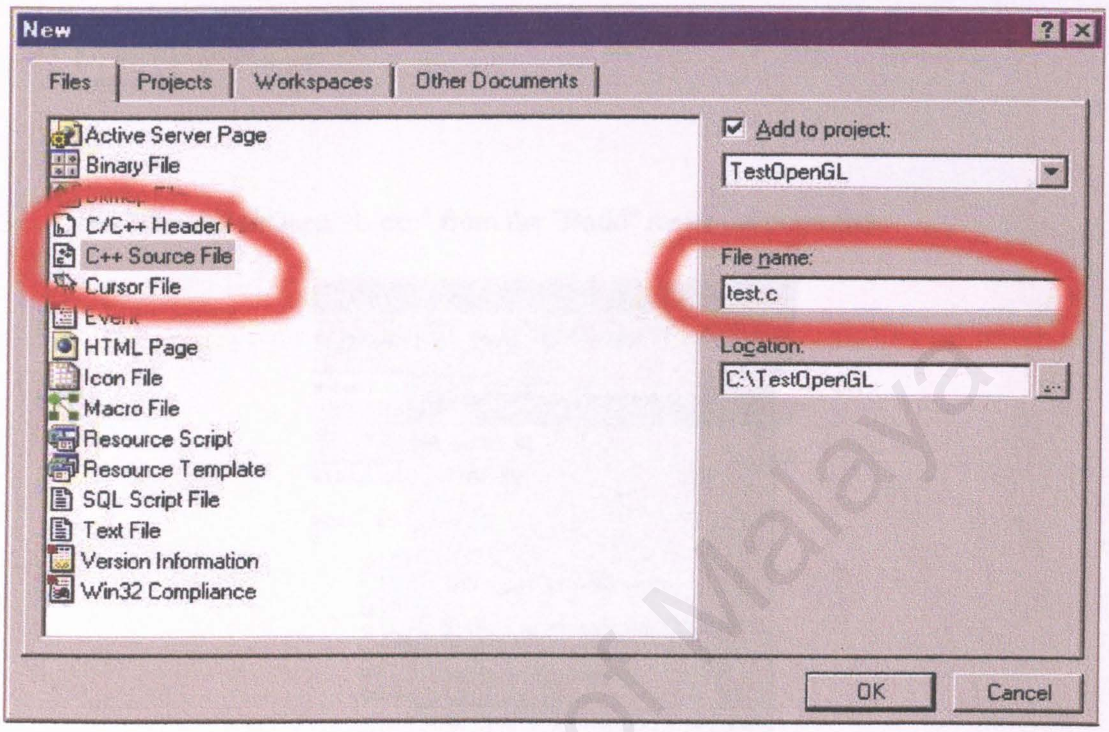
Project Options:

```
kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib
odbc32.lib odbccp32.lib  kernel32.lib user32.lib gdi32.lib
```

8. Click "OK".

Your project is now ready to use OpenGL, the GLU library, and the GLUT library. To test your setup, try the following:

1. create a new C file called "test.c" by choosing "New" from the "File" menu and picking "C++ Source File".



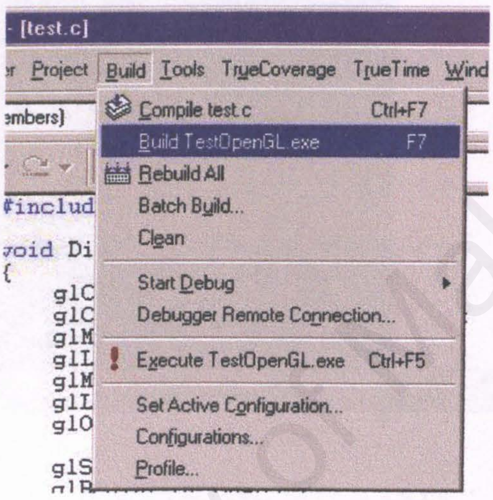
2. Type source code for 'Cloth Simulation'. *Please refer to Appendix B for more details :*

   > B1 to hanging situation and B2 for resting situation.

```
#include <windows.h>
#include <GL/glut.h>
#include <math.h>
#include <stdio.h>
#include <string>

        :
        :

int main(int argc, char** argv)

{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
glutInitWindowSize(700, 700);
glutCreateWindow("Hanging");
init();
```

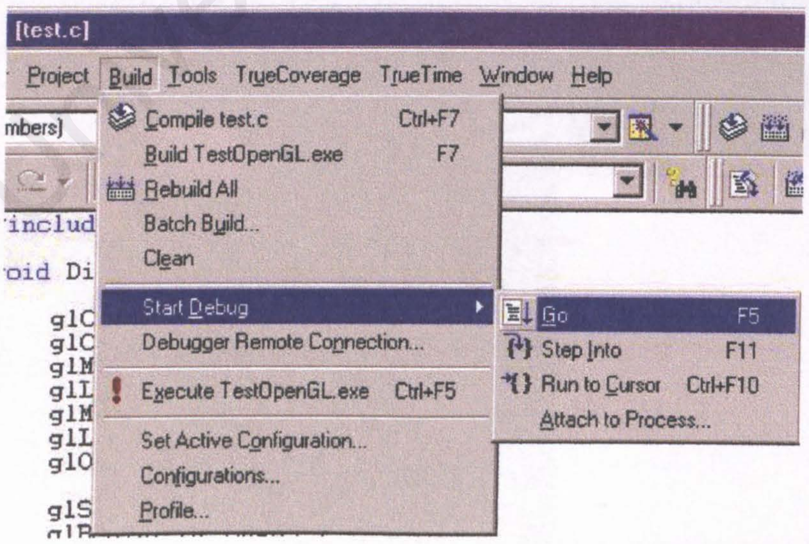
```
        //Set up control functions
        glutIdleFunc(idle);
        glutDisplayFunc(display);
        glutReshapeFunc(reshape);
        glutKeyboardFunc(keyboard);
        glutMotionFunc(mousemove);
        glutMouseFunc(mousedown);
        myHelp();
        glutMainLoop();
        return 0;
}
```

3. Select "Build TestOpenGL.exe" from the "Build" menu (or press F7)



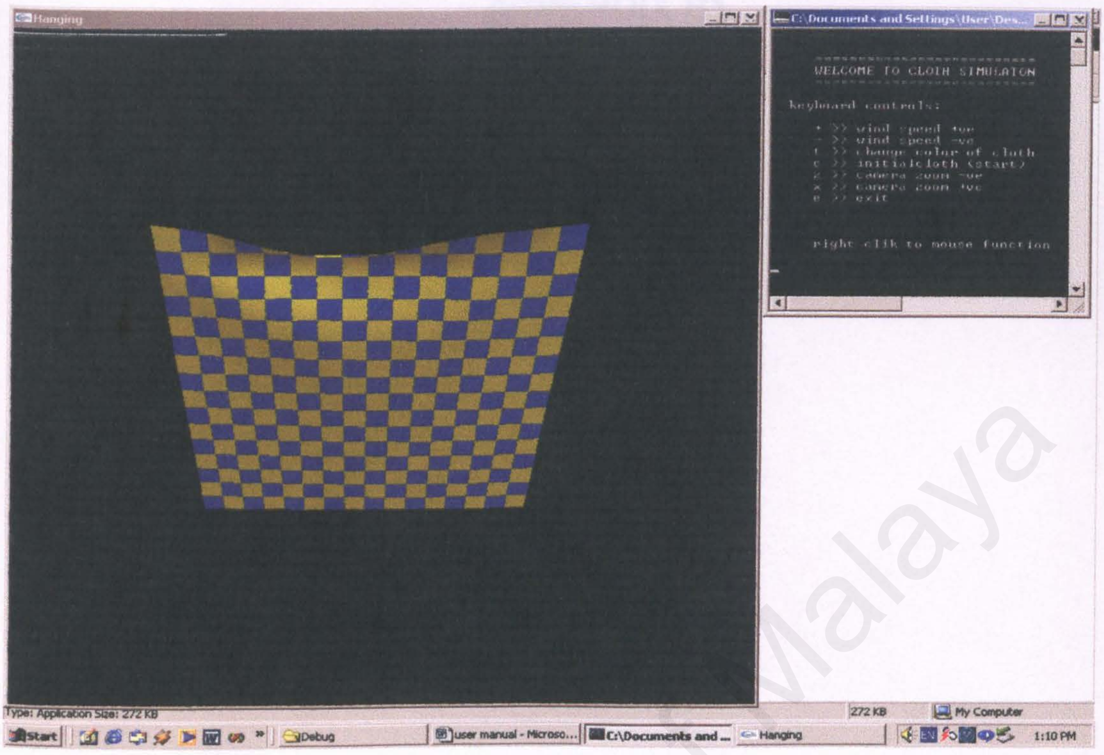
- [test.c]
Project Build Tools TrueCoverage TrueTime Wind
Compile test.c Ctrl+F7
Build TestOpenGL.exe F7
Rebuild All
Batch Build...
Clean
Start Debug
Debugger Remote Connection...
Execute TestOpenGL.exe Ctrl+F5
Set Active Configuration...
Configurations...
Profile...

4. Select "Go" from the "Debug" submenu of the "Build" menu (or prss F5)



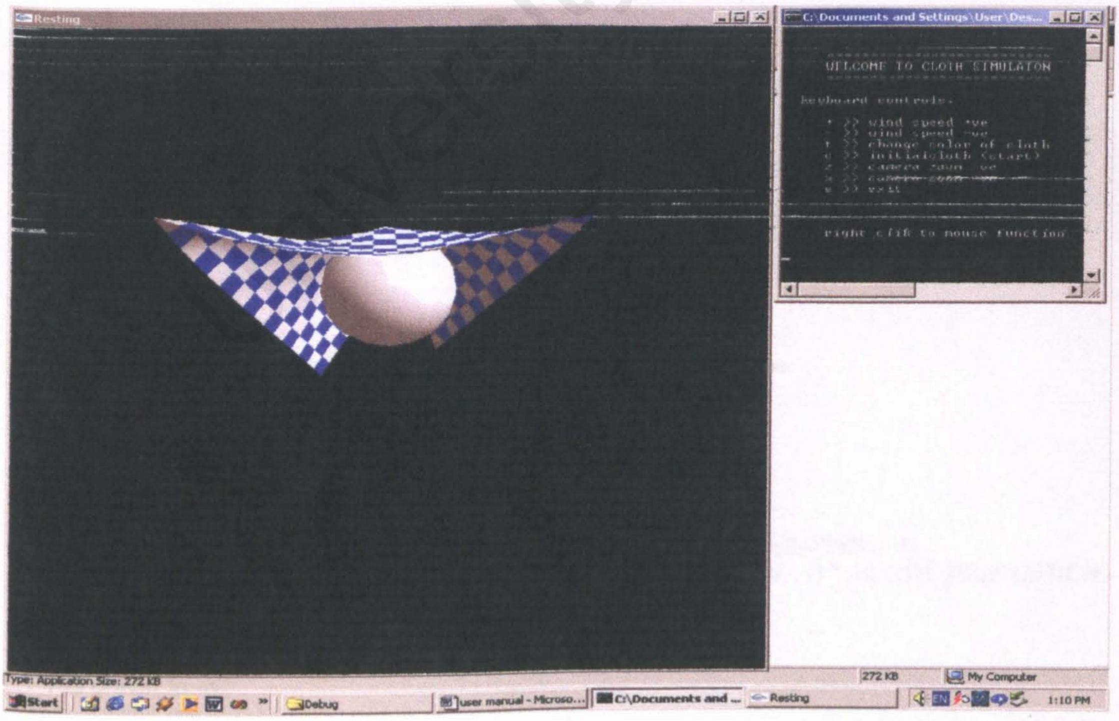
5. You should see a window that looks like the following :

>>> FOR HANGING SITUATION



```
C:\Documents and Settings\User\Des...

    ===========================
    WELCOME TO CLOTH SIMULATON
    ===========================

keyboard controls:

    + >> wind speed +ve
    - >> wind speed -ve
    t >> change color of cloth
    c >> initialcloth (start)
    z >> camera zoom -ve
    x >> camera zoom +ve
    e >> exit


    right clik to mouse function
```

>>>>> FOR RESTING SITUATION



```
C:\Documents and Settings\User\Des...

    ===========================
    WELCOME TO CLOTH SIMULATON
    ===========================

keyboard controls:

    + >> wind speed +ve
    - >> wind speed -ve
    t >> change color of cloth
    c >> initialcloth (start)
    z >> camera zoom -ve
    x >> camera zoom +ve
    e >> exit


    right clik to mouse function
```

APPENDIX B1

```
// Author     : Nor Azlina Mohamad Salleh
// Matrix No  : WEK 000266
// Title      : Cloth Simulation
// Subtitile  : Hanging Situation

#include <windows.h>
#include <GL/glut.h>
#include <math.h>
#include <stdio.h>
#include <string>
#include <iostream>
#include <fstream>
#include <cstdlib>

using namespace std;

#include "fonts.h"
#define CLOTH_SIZE 5.0f
#define CONSTRAINT_UPDATE 3
#define TEXTURE_SIZE 16
#define CLOTH_RESOLUTION 16
#define TIME_STEP 0.01f
#define TIME_FACTOR TIME_STEP*TIME_STEP

/*Constraint structure*/
struct CONSTRAINT
{
        int particle1, particle2;
        float restlength;
};

/*Texture object*/
GLuint texName;
/*Texture data array*/
GLubyte texture[3 * TEXTURE_SIZE * TEXTURE_SIZE];
/*Number of particles in the cloth*/
const int numparticles = CLOTH_RESOLUTION * CLOTH_RESOLUTION;
/*Array of forces on the particles in the cloth*/
float forces[numparticles * 3];
/*Array of normals for particles in the cloth*/
float normals[numparticles * 3];
/*First particle array for cloth*/
float cloth1[numparticles * 4];
/*Second particle array for cloth*/
float cloth2[numparticles * 4];
/*Pointer to current cloth array*/
float *currentcloth;
/*Pointer to previous cloth array*/
float *previouscloth;
/*Number of constraints on cloth particle system*/
const int numconstraints = 2 * CLOTH_RESOLUTION * (CLOTH_RESOLUTION - 1)
                                   + (CLOTH_RESOLUTION - 1) * (CLOTH_RESOLUTION - 1);
/*Array of constraints on the cloth particle system*/
CONSTRAINT constraints[numconstraints];

/*The gravity force*/
float gravityforce = -9.8f;
```

```
  /*Variables for controlling the wind*/
  float winddirection[3] = {0.0f, 0.0f, -1.0f};
  float windspeed = 0.0f;
  float windvector[3] = {0.0f, 0.0f, 0.0f};
  /*Lighting variables*/
  float lightpos[4] = {1.0f, 1.0f, 1.0f, 0.0f};
  float lightambient[4] = {0.2f, 0.2f, 0.2f, 1.0f};
  float lightdiffuse[4] = {1.0f, 1.0f, 1.0f, 1.0f};
  float material[4] = {1.0f, 1.0f, 1.0f, 1.0f};

  /*The elapsed time of the program*/
  int currenttime = 0;
  /*How far into the simulation the program is*/
  int simulationtime = 0;

  /*Keeps track of where the mouse was last*/
  int lastx = 0, lasty = 0;

  /*Camera control variables*/
  float rotation = 0;
  float elevation = 1.0f;
  float cameradistance = 8.0f;
  float camerapos[3] = {0.0f, 0.0f, 0.0f};

  /*Controls what mouse position changes update*/
  int controlmode = 0;

  /*Stores the current texture used for the flag*/
  int currenttexture = 0;

  void myHelp()
  {
          cout << endl << endl << "     ";
          cout <<"=========================="<< endl << "     ";
          cout <<"WELCOME TO CLOTH SIMULATON "<< endl << "     ";
          cout <<"==========================" << endl << endl << "  ";
          cout << "keyboard controls:" << endl << endl << "     ";
          cout << "+ >> wind speed +ve" << endl << "     ";
          cout << "- >> wind speed -ve" << endl << "     ";
          cout << "t >> change color of cloth" << endl << "     ";
          cout << "c >> initialcloth (start)" << endl << "     ";
          cout << "z >> camera zoom -ve" << endl << "     ";
          cout << "x >> camera zoom +ve" << endl << "     ";
          cout << "e >> exit" << endl << endl << endl << endl <<"     ";
          cout << "right clik to mouse function" << endl << endl;

}

void generatetexture(void)
{
        int i, j;

        /*Depending on which texture is desired, generate different
        coloured checkerboard textures*/
        if(currenttexture == 0)
        {
                for(i = 0; i < TEXTURE_SIZE; i++)
                {
                        for(j = 0; j < TEXTURE_SIZE; j++)
                        {
                                texture[3 * (i * TEXTURE_SIZE + j)] = 255 - ((i + j) % 2) * 255;
                                texture[3 * (i * TEXTURE_SIZE + j) + 1] = 255 - ((i + j) % 2) * 255;
                                texture[3 * (i * TEXTURE_SIZE + j) + 2] = ((i + j) % 2) * 255;
                        }
                }
        }
        else if(currenttexture == 1)
        {
                for(i = 0; i < TEXTURE_SIZE; i++)
                {
                        for(j = 0; j < TEXTURE_SIZE; j++)
                        {
                                texture[3 * (i * TEXTURE_SIZE + j)] = ((i + j) % 2) * 255;
                                texture[3 * (i * TEXTURE_SIZE + j) + 1] = 255 - ((i + j) % 2) * 255;
                                texture[3 * (i * TEXTURE_SIZE + j) + 2] = 0;
                        }
                }
        }
        else if(currenttexture == 2)
        {
                for(i = 0; i < TEXTURE_SIZE; i++)
                {
                        for(j = 0; j < TEXTURE_SIZE; j++)
                        {
                                texture[3 * (i * TEXTURE_SIZE + j)] = 50 + ((i + j) % 2) * 205;
                                texture[3 * (i * TEXTURE_SIZE + j) + 1] = 50 + ((i + j) % 2) * 205;
                                texture[3 * (i * TEXTURE_SIZE + j) + 2] = 50 + ((i + j) % 2) * 205;
                        }
                }
        }
        else if(currenttexture == 3)
        {
                for(i = 0; i < TEXTURE_SIZE; i++)
                {
                        for(j = 0; j < TEXTURE_SIZE; j++)
                        {
                                texture[3 * (i * TEXTURE_SIZE + j)] = 255 - ((i + j) % 2) * 255;
                                texture[3 * (i * TEXTURE_SIZE + j) + 1] = 255 - ((i + j) % 2) * 255;
                                texture[3 * (i * TEXTURE_SIZE + j) + 2] = 255;
                        }
                }
        }

        /*Load the texture into OpenGL*/
        glBindTexture(GL_TEXTURE_2D, texName);
        glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
                                        GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                                        GL_NEAREST);
        gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGB, TEXTURE_SIZE, TEXTURE_SIZE, GL_RGB,
GL_UNSIGNED_BYTE, texture);
}

void accumulateforces(void)
{
        int i;
        float windforce;

        //Determine the wind force vector
        windvector[0] = winddirection[0] * windspeed;
        windvector[1] = winddirection[1] * windspeed;
        windvector[2] = winddirection[2] * windspeed;
```

```
		for(i = 0; i < numparticles; i++)
		{
			//Initialize forces
			forces[3 * i] = 0.0f;
			forces[3 * i + 1] = 0.0f;
			forces[3 * i + 2] = 0.0f;

			//Gravity force
			forces[3 * i + 1] += gravityforce;

			//Wind force
			windforce = windvector[0] * normals[3 * i]
					+ windvector[1] * normals[3 * i + 1]
					+ windvector[2] * normals[3 * i + 2];
			forces[3 * i] += normals[3 * i] * windforce;
			forces[3 * i + 1] += normals[3 * i + 1] * windforce;
			forces[3 * i + 2] += normals[3 * i + 2] * windforce;

			//Dampening force
			windforce = (previouscloth[4 * i] - currentcloth[4 * i]) * normals[3 * i]
				+ (previouscloth[4 * i + 1] - currentcloth[4 * i + 1]) * normals[3 * i + 1]
				+ (previouscloth[4 * i + 2] - currentcloth[4 * i + 2]) * normals[3 * i + 2];
			forces[3 * i] += 80.0f * normals[3 * i] * windforce;
			forces[3 * i + 1] += 80.0f * normals[3 * i + 1] * windforce;
			forces[3 * i + 2] += 80.0f * normals[3 * i + 2] * windforce;
		}
	}

	void verlet(void)
	{
		int i;
		float *tempfloatptr;

		//Update each particle via verlet integration
		for(i = 0; i < numparticles; i++)
		{
			previouscloth[4 * i] = 2.0 * currentcloth[4 * i]
					- previouscloth[4 * i]
					+ forces[3 * i] * currentcloth[4 * i + 3]* TIME_FACTOR;
			previouscloth[4 * i + 1] = 2.0 * currentcloth[4 * i + 1]
					- previouscloth[4 * i + 1]
					+ forces[3 * i + 1] * currentcloth[4 * i + 3] * TIME_FACTOR;
			previouscloth[4 * i + 2] = 2.0 * currentcloth[4 * i + 2]
					- previouscloth[4 * i + 2]
					+ forces[3 * i + 2] * currentcloth[4 * i + 3] * TIME_FACTOR;
		}

		//Update the current and previous cloth pointers
		tempfloatptr = previouscloth;
		previouscloth = currentcloth;
		currentcloth = tempfloatptr;
}

void satisfyconstraints(void)
{
		int i;
		float deltalength;
		float delta[3];
		float diff;
		float invmass1, invmass2;
		//Satisfy stick constraints within cloth
		for(i = 0; i < numconstraints; i++)
		{
			delta[0] = currentcloth[4 * constraints[i].particle1]
						- currentcloth[4 * constraints[i].particle2];
			delta[1] = currentcloth[4 * constraints[i].particle1 + 1]
						- currentcloth[4 * constraints[i].particle2 + 1];
			delta[2] = currentcloth[4 * constraints[i].particle1 + 2]
						- currentcloth[4 * constraints[i].particle2 + 2];

			invmass1 = currentcloth[4 * constraints[i].particle1 + 3];
			invmass2 = currentcloth[4 * constraints[i].particle2 + 3];

			deltalength = (float)sqrt(delta[0] * delta[0] + delta[1] * delta[1] + delta[2] * delta[2]);

			diff = (deltalength - constraints[i].restlength) / (deltalength * (invmass1 + invmass2));

			currentcloth[4 * constraints[i].particle1] -= invmass1 * delta[0] * diff;
			currentcloth[4 * constraints[i].particle1 + 1] -= invmass1 * delta[1] * diff;
			currentcloth[4 * constraints[i].particle1 + 2] -= invmass1 * delta[2] * diff;

			currentcloth[4 * constraints[i].particle2] += invmass2 * delta[0] * diff;
			currentcloth[4 * constraints[i].particle2 + 1] += invmass2 * delta[1] * diff;
			currentcloth[4 * constraints[i].particle2 + 2] += invmass2 * delta[2] * diff;
		}
}

void calculatenormals(void)
{
		int i, j;
		float vectorlength;
		float trianglenormal[3];
		float v1[3], v2[3];

		//Set all normals to 0
		for(i = 0; i < numparticles; i++)
		{
			normals[3 * i] = 0.0f;
			normals[3 * i + 1] = 0.0f;
			normals[3 * i + 2] = 0.0f;
		}

		//Calculate the normal of each triangle
		//in the mesh and factor it into all the vertices
		//that are a part of it
		for(i = 0; i < CLOTH_RESOLUTION - 1; i++)
		{
			for(j = 0; j < CLOTH_RESOLUTION - 1; j++)
			{
				v1[0] = currentcloth[4 * (i * CLOTH_RESOLUTION + j)]
							- currentcloth[4 * ((i + 1) * CLOTH_RESOLUTION + j)];
				v1[1] = currentcloth[4 * (i * CLOTH_RESOLUTION + j) + 1]
							- currentcloth[4 * ((i + 1) * CLOTH_RESOLUTION + j) + 1];
				v1[2] = currentcloth[4 * (i * CLOTH_RESOLUTION + j) + 2]
							- currentcloth[4 * ((i + 1) * CLOTH_RESOLUTION + j) + 2];

				v2[0] = currentcloth[4 * (i * CLOTH_RESOLUTION + j + 1)]
							- currentcloth[4 * (i * CLOTH_RESOLUTION + j)];
				v2[1] = currentcloth[4 * (i * CLOTH_RESOLUTION + j + 1) + 1]
							- currentcloth[4 * (i * CLOTH_RESOLUTION + j) + 1];
				v2[2] = currentcloth[4 * (i * CLOTH_RESOLUTION + j + 1) + 2]
							- currentcloth[4 * (i * CLOTH_RESOLUTION + j) + 2]
```

```
                    trianglenormal[0] = v1[1] * v2[2] - v1[2] * v2[1];
                    trianglenormal[1] = v1[2] * v2[0] - v1[0] * v2[2];
                    trianglenormal[2] = v1[0] * v2[1] - v1[1] * v2[0];

                    normals[3 * (i * CLOTH_RESOLUTION + j)] += trianglenormal[0];
                    normals[3 * (i * CLOTH_RESOLUTION + j) + 1] += trianglenormal[1];
                    normals[3 * (i * CLOTH_RESOLUTION + j) + 2] += trianglenormal[2];

                    normals[3 * (i * CLOTH_RESOLUTION + j + 1)] += trianglenormal[0];
                    normals[3 * (i * CLOTH_RESOLUTION + j + 1) + 1] += trianglenormal[1];
                    normals[3 * (i * CLOTH_RESOLUTION + j + 1) + 2] += trianglenormal[2];

                    normals[3 * ((i + 1) * CLOTH_RESOLUTION + j)] += trianglenormal[0];
                    normals[3 * ((i + 1) * CLOTH_RESOLUTION + j) + 1] += trianglenormal[1];
                    normals[3 * ((i + 1) * CLOTH_RESOLUTION + j) + 2] += trianglenormal[2];

                    v1[0] = currentcloth[4 * ((i + 1) * CLOTH_RESOLUTION + j + 1)]
                                    - currentcloth[4 * (i * CLOTH_RESOLUTION + j + 1)];
                    v1[1] = currentcloth[4 * ((i + 1) * CLOTH_RESOLUTION + j + 1) + 1]
                                    - currentcloth[4 * (i * CLOTH_RESOLUTION + j + 1) + 1];
                    v1[2] = currentcloth[4 * ((i + 1) * CLOTH_RESOLUTION + j + 1) + 2]
                                    - currentcloth[4 * (i * CLOTH_RESOLUTION + j + 1) + 2];

                    v2[0] = currentcloth[4 * ((i + 1) * CLOTH_RESOLUTION + j)]
                                    - currentcloth[4 * ((i + 1) * CLOTH_RESOLUTION + j + 1)];
                    v2[1] = currentcloth[4 * ((i + 1) * CLOTH_RESOLUTION + j) + 1]
                                    - currentcloth[4 * ((i + 1) * CLOTH_RESOLUTION + j + 1) + 1];
                    v2[2] = currentcloth[4 * ((i + 1) * CLOTH_RESOLUTION + j) + 2]
                                    - currentcloth[4 * ((i + 1) * CLOTH_RESOLUTION + j + 1) + 2];

                    trianglenormal[0] = v1[1] * v2[2] - v1[2] * v2[1];
                    trianglenormal[1] = v1[2] * v2[0] - v1[0] * v2[2];
                    trianglenormal[2] = v1[0] * v2[1] - v1[1] * v2[0];

                    normals[3 * ((i + 1) * CLOTH_RESOLUTION + j + 1)] += trianglenormal[0];
                    normals[3 * ((i + 1) * CLOTH_RESOLUTION + j + 1) + 1] += trianglenormal[1];
                    normals[3 * ((i + 1) * CLOTH_RESOLUTION + j + 1) + 2] += trianglenormal[2];

                    normals[3 * (i * CLOTH_RESOLUTION + j + 1)] += trianglenormal[0];
                    normals[3 * (i * CLOTH_RESOLUTION + j + 1) + 1] += trianglenormal[1];
                    normals[3 * (i * CLOTH_RESOLUTION + j + 1) + 2] += trianglenormal[2];

                    normals[3 * ((i + 1) * CLOTH_RESOLUTION + j)] += trianglenormal[0];
                    normals[3 * ((i + 1) * CLOTH_RESOLUTION + j) + 1] += trianglenormal[1];
                    normals[3 * ((i + 1) * CLOTH_RESOLUTION + j) + 2] += trianglenormal[2];
            }
    }

    //Normalize vertex normals
    for(i = 0; i < numparticles; i++)
    {
            vectorlength = 1.0f / (float)sqrt(normals[3 * i] * normals[3 * i]
                                            + normals[3 * i + 1] * normals[3 * i + 1]
                                            + normals[3 * i + 2] * normals[3 * i + 2]);

            normals[3 * i] *= vectorlength;
            normals[3 * i + 1] *= vectorlength;
            normals[3 * i + 2] *= vectorlength;
    }
}


void updatecloth(void)
```

```
{
        int i;

        //Accumulate forces on the particles
        accumulateforces();
        //Do verlet integration
        verlet();
        //Attempt to satisfy all the constraints on the system
        for(i = 0; i < CONSTRAINT_UPDATE; i++)
        satisfyconstraints();
        //Calculate the triangle normals
        calculatenormals();
}

void initializecloth(int setuptype)
{
        int i, j, currentconstraint = 0;
        float restlength;
        float diagonallength;

        //Initialize cloth position and point weights
        if(setuptype == 0)
        {
                for(i = 0; i < CLOTH_RESOLUTION; i++)
                {
                        for(j = 0; j < CLOTH_RESOLUTION; j++)
                        {
                                cloth1[4 * (CLOTH_RESOLUTION * i + j)] = -CLOTH_SIZE / 2.0f + j *
CLOTH_SIZE / CLOTH_RESOLUTION;
                                cloth1[4 * (CLOTH_RESOLUTION * i + j) + 1] = 2.0f;
                                cloth1[4 * (CLOTH_RESOLUTION * i + j) + 2] = -i * CLOTH_SIZE /
CLOTH_RESOLUTION;

                                cloth1[4 * (CLOTH_RESOLUTION * i + j) + 3] = 1.0f;
                        }
                }
        }

        //Initialize 2 corner points to infinite weight
        cloth1[3] = 0.0f;
        cloth1[4 * (CLOTH_RESOLUTION - 1) + 3] = 0.0f;

        for(i = 0; i < 4 * CLOTH_RESOLUTION * CLOTH_RESOLUTION; i++)
                cloth2[i] = cloth1[i];

        //Set up pointers for the previous frame and the current frame for the cloth
        currentcloth = cloth1;
        previouscloth = cloth2;

        restlength = CLOTH_SIZE / CLOTH_RESOLUTION;

        //Constraints in the horizontal direction along the cloth
        for(i = 0; i < CLOTH_RESOLUTION; i++)
        {
                for(j = 0; j < CLOTH_RESOLUTION - 1; j++)
                {
                        constraints[currentconstraint].particle1 = i * CLOTH_RESOLUTION + j;
                        constraints[currentconstraint].particle2 = i * CLOTH_RESOLUTION + j + 1;
                        constraints[currentconstraint].restlength = restlength;
                        currentconstraint++;
                }
        }
        //Constraints in the vertical direction along the cloth
        for(i = 0; i < CLOTH_RESOLUTION - 1; i++)
        {
                for(j = 0; j < CLOTH_RESOLUTION; j++)
                {
                        constraints[currentconstraint].particle1 = i * CLOTH_RESOLUTION + j;
                        constraints[currentconstraint].particle2 = (i + 1) * CLOTH_RESOLUTION + j;
                        constraints[currentconstraint].restlength = restlength;
                        currentconstraint++;
                }
        }

        //A single diagonal constraint across every square in the cloth grid
        diagonallength = (float) sqrt(2.0f * restlength * restlength);
        for(i = 0; i < CLOTH_RESOLUTION - 1; i++)
        {
                for(j = 0; j < CLOTH_RESOLUTION - 1; j++)
                {
                        constraints[currentconstraint].particle1 = i * CLOTH_RESOLUTION + j;
                        constraints[currentconstraint].particle2 = (i + 1) * CLOTH_RESOLUTION + j + 1;
                        constraints[currentconstraint].restlength = diagonallength;
                        currentconstraint++;
                }
        }
}

void drawcloth(void)
{
        int i, j;

        //Draw a series of triangle strips for the cloth
        for(i = 0; i < CLOTH_RESOLUTION - 1; i++)
        {
                glBegin(GL_TRIANGLE_STRIP);
                for(j = 0; j < CLOTH_RESOLUTION; j++)
                {
                        glNormal3fv(&(normals[3 * ((i + 1) * CLOTH_RESOLUTION + j)]));
                        glTexCoord2f(((float)j) / (CLOTH_RESOLUTION - 1), ((float)(i + 1)) /
(CLOTH_RESOLUTION - 1));
                        glVertex3fv(&(currentcloth[4 * ((i + 1) * CLOTH_RESOLUTION + j)]));
                        glNormal3fv(&(normals[3 * (i * CLOTH_RESOLUTION + j)]));
                        glTexCoord2f(((float)j) / (CLOTH_RESOLUTION - 1), ((float)i) /
(CLOTH_RESOLUTION - 1));
                        glVertex3fv(&(currentcloth[4 * (i * CLOTH_RESOLUTION + j)]));
                }
                glEnd();
        }
}

void menu(int selection)
{
        controlmode = selection;
}

void idle(void)
{
        currenttime = glutGet(GLUT_ELAPSED_TIME);

        //Update the simulation to match the current time
        while(simulationtime < currenttime)
        {
                updatecloth();
                simulationtime += TIME_STEP * 1000;
        }
        glutPostRedisplay();
```

```
    }

    void init(void)
    {
            initializecloth(0);

            glShadeModel(GL_SMOOTH);
            glDisable(GL_CULL_FACE);
            glEnable(GL_DEPTH_TEST);

            //Setup texturing
            glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
            glGenTextures(1, &texName);
            generatetexture();
            glEnable(GL_TEXTURE_2D);

            //Setup lighting
            glLightfv(GL_LIGHT0, GL_AMBIENT, lightambient);
            glLightfv(GL_LIGHT0, GL_DIFFUSE, lightdiffuse);
            glLightfv(GL_LIGHT0, GL_POSITION, lightpos);
            glEnable(GL_LIGHTING);
            glEnable(GL_LIGHT0);
            glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, 1);
            glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, material);
   }

   void display(void)
   {
          //Set the camera position
          camerapos[0] = sin(rotation) * sin(elevation) * cameradistance;
          camerapos[1] = cos(elevation) * cameradistance;
          camerapos[2] = cos(rotation) * sin(elevation) * cameradistance;

          //Set up for rendering
          glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
          glLoadIdentity();
          gluLookAt(camerapos[0], camerapos[1], camerapos[2],
                                   0.0f, 0.0f, 0.0f,
                                   0.0f, 1.0f, 0.0f);

          //Draw the cloth
          glColor3f(1.0f, 1.0f, 1.0f);
          drawcloth();

          glDisable(GL_TEXTURE_2D);

          //Draw the wind direction
          glColor3f(0.0f, 1.0f, 4.0f);
          glDisable(GL_LIGHTING);
          glBegin(GL_LINES);
          glVertex3fv(windvector);
          glVertex3f(0.0f, 0.0f, 0.0f);
          glEnd();
          glEnable(GL_LIGHTING);
          glColor3f(1.0f, 1.0f, 1.0f);

          glEnable(GL_TEXTURE_2D);

          glutSwapBuffers();
}
void reshape(int w, int h)
{
         glViewport(0, 0, (GLsizei) w, (GLsizei) h);
         glMatrixMode(GL_PROJECTION);
```

```
            glLoadIdentity();
            gluPerspective(60.0, (GLfloat) w/(GLfloat) h, 0.1, 100.0);
            glMatrixMode(GL_MODELVIEW);
    }

    void mousemove(int x, int y)
    {
            double modelviewmatrix[16], projectionmatrix[16];
            int viewport[4];
            double projectedpoint[3];
            float u;
            float veclength;

            //Move camera
            if(controlmode == 0)
            {
                    rotation += (x - lastx) / 100.0f;
                    elevation += (y - lasty) / 100.0f;

                    if(elevation < 0.1f)
                            elevation = 0.1f;
                    if(elevation > 3.0f)
                            elevation = 3.0f;
            }
            //Change wind direction
            else if(controlmode == 2)
            {
                    glLoadIdentity();
                    gluLookAt(camerapos[0], camerapos[1], camerapos[2],
                                    0.0f, -0.5f, 0.0f,
                                    0.0f, 1.0f, 0.0f);
                    glGetDoublev(GL_MODELVIEW_MATRIX, modelviewmatrix);
                    glGetDoublev(GL_PROJECTION_MATRIX, projectionmatrix);
                    glGetIntegerv(GL_VIEWPORT, viewport);
                    gluUnProject(x, (viewport[3] - y), 1.0, modelviewmatrix, projectionmatrix, viewport,
                                    &(projectedpoint[0]), &(projectedpoint[1]), &(projectedpoint[2]));

                    //Calculate wind direction
                    u = camerapos[1] / (camerapos[1] - projectedpoint[1]);
                    winddirection[0] = camerapos[0] + u * (projectedpoint[0] - camerapos[0]);
                    winddirection[2] = camerapos[2] + u * (projectedpoint[2] - camerapos[2]);

                    //Normalize wind direction
                    veclength = (float) sqrt(winddirection[0] * winddirection[0]
                                                    + winddirection[2] * winddirection[2]);
                    winddirection[0] /= veclength;
                    winddirection[2] /= veclength;
            }
            lastx = x;
            lasty = y;
            idle();
}

void mousedown(int button, int state, int x, int y)
{
        lastx = x;
        lasty = y;
}

void keyboard (unsigned char key, int x, int y)
{
        switch (key)
        {
                case '+':
                case '=':
                            windspeed += 1.0f;
                            break;
                case '-':
                            windspeed -= 1.0f;
                            break;
                case 'c':
                            initializecloth(0);
                            break;
                case 'z':
                            cameradistance += 0.5f;
                            break;
                case 'x':
                            cameradistance -= 0.5f;
                            if(cameradistance < 4.0f)
                                        cameradistance = 4.0f;
                            break;
                case 't':
                            currenttexture++;
                            currenttexture %= 4;
                            generatetexture();
                            break;
                case 'e':
                            exit(-1);
                            break;
                default:
                            break;
        }
}

int main(int argc, char** argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(700, 700);
        glutCreateWindow("Hanging");
        init();

        //Setup menu
        glutCreateMenu(menu);
        glutAddMenuEntry("Mouse changes view direction", 0);
        glutAddMenuEntry("Mouse changes wind direction", 2);
        glutAttachMenu(GLUT_RIGHT_BUTTON);

        //Set up control functions
        glutIdleFunc(idle);
        glutDisplayFunc(display);
        glutReshapeFunc(reshape);
        glutKeyboardFunc(keyboard);
        glutMotionFunc(mousemove);
        glutMouseFunc(mousedown);
        myHelp();
        glutMainLoop();
        return 0;
}
```

APPENDIX B2

```
// Author      : Nor Azlina Mohamad Salleh
// Matrix No  : WEK 000266
// Title         : Cloth Simulation
// Subtitile    : Resting Situation

#include <windows.h>
#include <GL/glut.h>
#include <math.h>
#include <stdio.h>
#include <string>
#include <iostream>
#include <fstream>
#include <cstdlib>

using namespace std;

#include "fonts.h"

/*Constraint structure*/
struct CONSTRAINT
{
        int particle1, particle2;
        float restlength;
};

#define CLOTH_SIZE 5.0f
#define CONSTRAINT_UPDATE 3
#define TEXTURE_SIZE 16
#define CLOTH_RESOLUTION 16
#define TIME_STEP 0.01f
#define TIME_FACTOR TIME_STEP*TIME_STEP

/*Texture object*/
GLuint texName;
/*Texture data array*/
GLubyte texture[3 * TEXTURE_SIZE * TEXTURE_SIZE];
/*Number of particles in the cloth*/
const int numparticles = CLOTH_RESOLUTION * CLOTH_RESOLUTION;
/*Array of forces on the particles in the cloth*/
float forces[numparticles * 3];
/*Array of normals for particles in the cloth*/
float normals[numparticles * 3];
/*First particle array for cloth*/
float cloth1[numparticles * 4];
/*Second particle array for cloth*/
float cloth2[numparticles * 4];
/*Pointer to current cloth array*/
float *currentcloth;
/*Pointer to previous cloth array*/
float *previouscloth;
/*Number of constraints on cloth particle system*/
const int numconstraints = 2 * CLOTH_RESOLUTION * (CLOTH_RESOLUTION - 1)
                                        + (CLOTH_RESOLUTION - 1) * (CLOTH_RESOLUTION - 1);
/*Array of constraints on the cloth particle system*/
CONSTRAINT constraints[numconstraints];

/*Position and radius of the sphere*/
float sphere[4] = {0.0f, 0.0f, -2.0f, 1.0f};
```

```
/*The gravity force*/
float gravityforce = -9.8f;

/*The elapsed time of the program*/
int currenttime = 0;

/*Variables for controlling the wind*/
float winddirection[3] = {0.0f, 0.0f, -1.0f};
float windspeed = 0.0f;
float windvector[3] = {0.0f, 0.0f, 0.0f};

/*Lighting variables*/
float lightpos[4] = {1.0f, 1.0f, 1.0f, 0.0f};
float lightambient[4] = {0.2f, 0.2f, 0.2f, 1.0f};
float lightdiffuse[4] = {1.0f, 1.0f, 1.0f, 1.0f};
float material[4] = {1.0f, 1.0f, 1.0f, 1.0f};

/*How far into the simulation the program is*/
int simulationtime = 0;

/*Keeps track of where the mouse was last*/
int lastx = 0, lasty = 0;

/*Camera control variables*/
float rotation = 0;
float elevation = 1.0f;
float cameradistance =8.0f;
float camerapos[3] = {0.0f, 0.0f, 0.0f};

/*Controls what mouse position changes update*/
int controlmode = 0;

/*Stores the current texture used for the flag*/
int currenttexture = 0;

void myHelp()
{
        cout << endl << endl << "     ";
        cout <<"==========================" << endl << "     ";
        cout <<"WELCOME TO CLOTH SIMULATON " << endl << "     ";
        cout <<"=========================="<< endl << endl << "  ";
        cout << "keyboard controls:"  << endl << endl << "     ";
        cout << "+ >> wind speed +ve" << endl << "    ";
        cout << "- >> wind speed -ve" << endl << "     ";
        cout << "t >> change color of cloth" << endl << "    ";
        cout << "c >> initialcloth (start)" << endl << "    ";
        cout << "z >> camera zoom -ve" << endl << "     ";
        cout << "x >> camera zoom +ve" << endl << "     ";
        cout << "e >> exit" << endl << endl << endl << endl <<"      ";
        cout << "right clik to mouse function" << endl << endl;
}

void generatetexture(void)
{
        int i, j;

        /*Depending on which texture is desired, generate different
        coloured checkerboard textures*/
        if(currenttexture == 0)
        {
                for(i = 0; i < TEXTURE_SIZE; i++)
                {
                        for(j = 0; j < TEXTURE_SIZE; j++)
                        {
                                texture[3 * (i * TEXTURE_SIZE + j)] = 255-((i + j) % 2) * 255;
```

```
				texture[3 * (i * TEXTURE_SIZE + j) + 1] = 255 - ((i + j) % 2) * 255;
				texture[3 * (i * TEXTURE_SIZE + j) + 2] = 255;
			}
		}
	}
	else if(currenttexture == 1)
	{
		for(i = 0; i < TEXTURE_SIZE; i++)
		{
			for(j = 0; j < TEXTURE_SIZE; j++)
			{
				texture[3 * (i * TEXTURE_SIZE + j)] = 255 - ((i + j) % 2) * 255;
				texture[3 * (i * TEXTURE_SIZE + j) + 1] = 255 - ((i + j) % 2) * 255;
				texture[3 * (i * TEXTURE_SIZE + j) + 2] = ((i + j) % 2) * 255;
			}
		}
	}
	else if(currenttexture == 2)
	{
		for(i = 0; i < TEXTURE_SIZE; i++)
		{
			for(j = 0; j < TEXTURE_SIZE; j++)
			{
				texture[3 * (i * TEXTURE_SIZE + j)] = 50 + ((i + j) % 2) * 205;
				texture[3 * (i * TEXTURE_SIZE + j) + 1] = 50 + ((i + j) % 2) * 205;
				texture[3 * (i * TEXTURE_SIZE + j) + 2] = 50 + ((i + j) % 2) * 205;
			}
		}
	}
	else if(currenttexture == 3)
	{
		for(i = 0; i < TEXTURE_SIZE; i++)
		{
			for(j = 0; j < TEXTURE_SIZE; j++)
			{
				texture[3 * (i * TEXTURE_SIZE + j)] = ((i + j) % 2) * 255;
				texture[3 * (i * TEXTURE_SIZE + j) + 1] = 255 - ((i + j) % 2) * 255;
				texture[3 * (i * TEXTURE_SIZE + j) + 2] = 0;
			}
		}
	}
	/*Load the texture into OpenGL*/
	glBindTexture(GL_TEXTURE_2D, texName);
	glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
	gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGB, TEXTURE_SIZE, TEXTURE_SIZE, GL_RGB,
GL_UNSIGNED_BYTE, texture);
}

void accumulateforces(void)
{
	int i;
	float windforce;

	//Determine the wind force vector
	windvector[0] = winddirection[0] * windspeed;
	windvector[1] = winddirection[1] * windspeed;
	windvector[2] = winddirection[2] * windspeed;
	for(i = 0; i < numparticles; i++)
	{
		//Initialize forces
		forces[3 * i] = 0.0f;
```

```
                    forces[3 * i + 1] = 0.0f;
                    forces[3 * i + 2] = 0.0f;

                    //Gravity force
                    forces[3 * i + 1] += gravityforce;

                    //Wind force
                    windforce = windvector[0] * normals[3 * i] + windvector[1] * normals[3 * i + 1]
                                        + windvector[2] * normals[3 * i + 2];
                    forces[3 * i] += normals[3 * i] * windforce;
                    forces[3 * i + 1] += normals[3 * i + 1] * windforce;
                    forces[3 * i + 2] += normals[3 * i + 2] * windforce;

                    //Dampening force
                    windforce = (previouscloth[4 * i] - currentcloth[4 * i]) * normals[3 * i]
                              + (previouscloth[4 * i + 1] - currentcloth[4 * i + 1]) * normals[3 * i + 1]
                              + (previouscloth[4 * i + 2] - currentcloth[4 * i + 2]) * normals[3 * i + 2];
                    forces[3 * i] += 80.0f * normals[3 * i] * windforce;
                    forces[3 * i + 1] += 80.0f * normals[3 * i + 1] * windforce;
                    forces[3 * i + 2] += 80.0f * normals[3 * i + 2] * windforce;
            }
    }

    void verlet(void)
    {
            int i;
            float *tempfloatptr;

            //Update each particle via verlet integration
            for(i = 0; i < numparticles; i++)
            {
                    previouscloth[4 * i] = 2.0 * currentcloth[4 * i] -  previouscloth[4 * i]
                                        + forces[3 * i] * currentcloth[4 * i + 3] * TIME_FACTOR;
                    previouscloth[4 * i + 1] = 2.0 * currentcloth[4 * i + 1] -  previouscloth[4 * i + 1]
                                        + forces[3 * i + 1] * currentcloth[4 * i + 3] * TIME_FACTOR;
                    previouscloth[4 * i + 2] = 2.0 * currentcloth[4 * i + 2] - previouscloth[4 * i + 2]
                                        + forces[3 * i + 2] * currentcloth[4 * i + 3] * TIME_FACTOR;
            }

            //Update the current and previous cloth pointers
            tempfloatptr = previouscloth;
            previouscloth = currentcloth;
            currentcloth = tempfloatptr;
}

void satisfyconstraints(void)
{
            int i;
            float sphereradius;
            float deltalength;
            float delta[3];
            float diff;
            float invmass1, invmass2;

            sphereradius = sphere[3] + 0.05;

            //Satisfy stick constraints within cloth
            for(i = 0; i < numconstraints; i++)
            {
                    delta[0] = currentcloth[4 * constraints[i].particle1]
                                        - currentcloth[4 * constraints[i].particle2];
                    delta[1] = currentcloth[4 * constraints[i].particle1 + 1]
                                        - currentcloth[4 * constraints[i].particle2 + 1];
                    delta[2] = currentcloth[4 * constraints[i].particle1 + 2]
                                        - currentcloth[4 * constraints[i].particle2 + 2];
				invmass1 = currentcloth[4 * constraints[i].particle1 + 3];
				invmass2 = currentcloth[4 * constraints[i].particle2 + 3];
				deltalength = (float)sqrt(delta[0] * delta[0] + delta[1] * delta[1] + delta[2] * delta[2]);
				diff = (deltalength - constraints[i].restlength) / (deltalength * (invmass1 + invmass2));

				currentcloth[4 * constraints[i].particle1] -= invmass1 * delta[0] * diff;
				currentcloth[4 * constraints[i].particle1 + 1] -= invmass1 * delta[1] * diff;
				currentcloth[4 * constraints[i].particle1 + 2] -= invmass1 * delta[2] * diff;

				currentcloth[4 * constraints[i].particle2] += invmass2 * delta[0] * diff;
				currentcloth[4 * constraints[i].particle2 + 1] += invmass2 * delta[1] * diff;
				currentcloth[4 * constraints[i].particle2 + 2] += invmass2 * delta[2] * diff;
		}

		//Satisfy constraint that cloth points can't enter the sphere
		for(i = 0; i < numparticles; i++)
		{
			delta[0] = currentcloth[4 * i] - sphere[0];
			delta[1] = currentcloth[4 * i + 1] - sphere[1];
			delta[2] = currentcloth[4 * i + 2] - sphere[2];

			deltalength = delta[0] * delta[0] + delta[1] * delta[1] + delta[2] * delta[2];

			//If the particle is in the sphere, move it to outside the sphere
			if(deltalength < sphereradius * sphereradius)
			{
				deltalength = (float)sqrt(deltalength);
				currentcloth[4 * i] += delta[0] * (sphereradius - deltalength) / sphereradius;
				currentcloth[4 * i + 1] += delta[1] * (sphereradius - deltalength) / sphereradius;
				currentcloth[4 * i + 2] += delta[2] * (sphereradius - deltalength) / sphereradius;
			}
		}
}

void calculatenormals(void)
{
	int i, j;
	float vectorlength;
	float trianglenormal[3];
	float v1[3], v2[3];

	//Set all normals to 0
	for(i = 0; i < numparticles; i++)
	{
		normals[3 * i] = 0.0f;
		normals[3 * i + 1] = 0.0f;
		normals[3 * i + 2] = 0.0f;
	}

	//Calculate the normal of each triangle
	//in the mesh and factor it into all the vertices
	//that are a part of it
	for(i = 0; i < CLOTH_RESOLUTION - 1; i++)
	{
		for(j = 0; j < CLOTH_RESOLUTION - 1; j++)
		{
			v1[0] = currentcloth[4 * (i * CLOTH_RESOLUTION + j)]
						- currentcloth[4 * ((i + 1) * CLOTH_RESOLUTION + j)];
			v1[1] = currentcloth[4 * (i * CLOTH_RESOLUTION + j) + 1]
						- currentcloth[4 * ((i + 1) * CLOTH_RESOLUTION + j) + 1];
			v1[2] = currentcloth[4 * (i * CLOTH_RESOLUTION + j) + 2]
						- currentcloth[4 * ((i + 1) * CLOTH_RESOLUTION + j) + 2];
			v2[0] = currentcloth[4 * (i * CLOTH_RESOLUTION + j + 1)]
						- currentcloth[4 * (i * CLOTH_RESOLUTION + j)];
```

```
                        v2[1] = currentcloth[4 * (i * CLOTH_RESOLUTION + j + 1) + 1]
                                        - currentcloth[4 * (i * CLOTH_RESOLUTION + j) + 1];
                        v2[2] = currentcloth[4 * (i * CLOTH_RESOLUTION + j + 1) + 2]
                                        - currentcloth[4 * (i * CLOTH_RESOLUTION + j) + 2];

                        trianglenormal[0] = v1[1] * v2[2] - v1[2] * v2[1];
                        trianglenormal[1] = v1[2] * v2[0] - v1[0] * v2[2];
                        trianglenormal[2] = v1[0] * v2[1] - v1[1] * v2[0];

                        normals[3 * (i * CLOTH_RESOLUTION + j)] += trianglenormal[0];
                        normals[3 * (i * CLOTH_RESOLUTION + j) + 1] += trianglenormal[1];
                        normals[3 * (i * CLOTH_RESOLUTION + j) + 2] += trianglenormal[2];

                        normals[3 * (i * CLOTH_RESOLUTION + j + 1)] += trianglenormal[0];
                        normals[3 * (i * CLOTH_RESOLUTION + j + 1) + 1] += trianglenormal[1];
                        normals[3 * (i * CLOTH_RESOLUTION + j + 1) + 2] += trianglenormal[2];

                        normals[3 * ((i + 1) * CLOTH_RESOLUTION + j)] += trianglenormal[0];
                        normals[3 * ((i + 1) * CLOTH_RESOLUTION + j) + 1] += trianglenormal[1];
                        normals[3 * ((i + 1) * CLOTH_RESOLUTION + j) + 2] += trianglenormal[2];

                        v1[0] = currentcloth[4 * ((i + 1) * CLOTH_RESOLUTION + j + 1)]
                                        - currentcloth[4 * (i * CLOTH_RESOLUTION + j + 1)];
                        v1[1] = currentcloth[4 * ((i + 1) * CLOTH_RESOLUTION + j + 1) + 1]
                                        - currentcloth[4 * (i * CLOTH_RESOLUTION + j + 1) + 1];
                        v1[2] = currentcloth[4 * ((i + 1) * CLOTH_RESOLUTION + j + 1) + 2]
                                        - currentcloth[4 * (i * CLOTH_RESOLUTION + j + 1) + 2];

                        v2[0] = currentcloth[4 * ((i + 1) * CLOTH_RESOLUTION + j)]
                                        - currentcloth[4 * ((i + 1) * CLOTH_RESOLUTION + j + 1)];
                        v2[1] = currentcloth[4 * ((i + 1) * CLOTH_RESOLUTION + j) + 1]
                                - currentcloth[4 * ((i + 1) * CLOTH_RESOLUTION + j + 1) + 1];
                        v2[2] = currentcloth[4 * ((i + 1) * CLOTH_RESOLUTION + j) + 2]
                                - currentcloth[4 * ((i + 1) * CLOTH_RESOLUTION + j + 1) + 2];

                        trianglenormal[0] = v1[1] * v2[2] - v1[2] * v2[1];
                        trianglenormal[1] = v1[2] * v2[0] - v1[0] * v2[2];
                        trianglenormal[2] = v1[0] * v2[1] - v1[1] * v2[0];

                        normals[3 * ((i + 1) * CLOTH_RESOLUTION + j + 1)] += trianglenormal[0];
                        normals[3 * ((i + 1) * CLOTH_RESOLUTION + j + 1) + 1] += trianglenormal[1];
                        normals[3 * ((i + 1) * CLOTH_RESOLUTION + j + 1) + 2] += trianglenormal[2];

                        normals[3 * (i * CLOTH_RESOLUTION + j + 1)] += trianglenormal[0];
                        normals[3 * (i * CLOTH_RESOLUTION + j + 1) + 1] += trianglenormal[1];
                        normals[3 * (i * CLOTH_RESOLUTION + j + 1) + 2] += trianglenormal[2];

                        normals[3 * ((i + 1) * CLOTH_RESOLUTION + j)] += trianglenormal[0];
                        normals[3 * ((i + 1) * CLOTH_RESOLUTION + j) + 1] += trianglenormal[1];
                        normals[3 * ((i + 1) * CLOTH_RESOLUTION + j) + 2] += trianglenormal[2];

                }
        }


        //Normalize vertex normals
        for(i = 0; i < numparticles; i++)
        {
                vectorlength = 1.0f / (float)sqrt(normals[3 * i] * normals[3 * i]+ normals[3 * i + 1] *
                                        normals[3 * i + 1]+ normals[3 * i + 2] * normals[3 * i + 2]);
                normals[3 * i] *= vectorlength;
                normals[3 * i + 1] *= vectorlength;
                normals[3 * i + 2] *= vectorlength;
        }
}
```

```
void updatecloth(void)
{
        int i;

        //Accumulate forces on the particles
        accumulateforces();
        //Do verlet integration
        verlet();
        //Attempt to satisfy all the constraints on the system
        for(i = 0; i < CONSTRAINT_UPDATE; i++)
                satisfyconstraints();
        //Calculate the triangle normals
        calculatenormals();
}


void initializecloth(int setuptype)
{
        int i, j, currentconstraint = 0;
        float restlength;
        float diagonallength;

        //Initialize cloth position and point weights
        if(setuptype == 0)
        {
                for(i = 0; i < CLOTH_RESOLUTION; i++)
                {
                        for(j = 0; j < CLOTH_RESOLUTION; j++)
                        {
                                cloth1[4 * (CLOTH_RESOLUTION * i + j)] = -CLOTH_SIZE / 2.0f + j *
CLOTH_SIZE / CLOTH_RESOLUTION;
                                cloth1[4 * (CLOTH_RESOLUTION * i + j) + 1] = 2.0f;
                                cloth1[4 * (CLOTH_RESOLUTION * i + j) + 2] = -i * CLOTH_SIZE /
CLOTH_RESOLUTION;
                                cloth1[4 * (CLOTH_RESOLUTION * i + j) + 3] = 1.0f;
                        }
                }
        }

        //Initialize second cloth array to match first
        for(i = 0; i < 4 * numparticles; i++)
                cloth2[i] = cloth1[i];

        //Initialize 2 corner points to infinite weight
        cloth1[3] = 0.0f;
        cloth1[4 * (CLOTH_RESOLUTION - 1) + 3] = 0.0f;

        for(i = 0; i < 4 * CLOTH_RESOLUTION * CLOTH_RESOLUTION; i++)
                cloth2[i] = cloth1[i];

        //Set up pointers for the previous frame and the current frame for the cloth
        currentcloth = cloth1;
        previouscloth = cloth2;

        restlength = CLOTH_SIZE / CLOTH_RESOLUTION;

        //Constraints in the horizontal direction along the cloth
        for(i = 0; i < CLOTH_RESOLUTION; i++)
        {
                for(j = 0; j < CLOTH_RESOLUTION - 1; j++)
                {
                        constraints[currentconstraint].particle1 = i * CLOTH_RESOLUTION + j;
                        constraints[currentconstraint].particle2 = i * CLOTH_RESOLUTION + j + 1;
                        constraints[currentconstraint].restlength = restlength;
                        currentconstraint++;
                }
        }
        //Constraints in the vertical direction along the cloth
        for(i = 0; i < CLOTH_RESOLUTION - 1; i++)
        {
                for(j = 0; j < CLOTH_RESOLUTION; j++)
                {
                        constraints[currentconstraint].particle1 = i * CLOTH_RESOLUTION + j;
                        constraints[currentconstraint].particle2 = (i + 1) * CLOTH_RESOLUTION + j;
                        constraints[currentconstraint].restlength = restlength;
                        currentconstraint++;
                }
        }

        //A single diagonal constraint across every square in the cloth grid
        diagonallength = (float) sqrt(2.0f * restlength * restlength);
        for(i = 0; i < CLOTH_RESOLUTION - 1; i++)
        {
                for(j = 0; j < CLOTH_RESOLUTION - 1; j++)
                {
                        constraints[currentconstraint].particle1 = i * CLOTH_RESOLUTION + j;
                        constraints[currentconstraint].particle2 = (i + 1) * CLOTH_RESOLUTION + j + 1;
                        constraints[currentconstraint].restlength = diagonallength;
                        currentconstraint++;
                }
        }
}

void drawcloth(void)
{
        int i, j;

        //Draw a series of triangle strips for the cloth
        for(i = 0; i < CLOTH_RESOLUTION - 1; i++)
        {
                glBegin(GL_TRIANGLE_STRIP);
                for(j = 0; j < CLOTH_RESOLUTION; j++)
                {
                        glNormal3fv(&(normals[3 * ((i + 1) * CLOTH_RESOLUTION + j)]));
                        glTexCoord2f(((float)j) / (CLOTH_RESOLUTION - 1), ((float)(i + 1)) /
(CLOTH_RESOLUTION - 1));
                        glVertex3fv(&(currentcloth[4 * ((i + 1) * CLOTH_RESOLUTION + j)]));
                        glNormal3fv(&(normals[3 * (i * CLOTH_RESOLUTION + j)]));
                        glTexCoord2f(((float)j) / (CLOTH_RESOLUTION - 1), ((float)i) /
(CLOTH_RESOLUTION - 1));
                        glVertex3fv(&(currentcloth[4 * (i * CLOTH_RESOLUTION + j)]));
                }
                glEnd();
        }
}

void menu(int selection)
{
        controlmode = selection;
}

void idle(void)
{
        currenttime = glutGet(GLUT_ELAPSED_TIME);

        //Update the simulation to match the current time
        while(simulationtime < currenttime)
        {
                updatecloth();
                simulationtime += TIME_STEP * 1000;
            }
            glutPostRedisplay();
    }
    void init(void)
    {
            initializecloth(0);

            glShadeModel(GL_SMOOTH);
            glDisable(GL_CULL_FACE);
            glEnable(GL_DEPTH_TEST);

            //Setup texturing
            glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
            glGenTextures(1, &texName);
            generatetexture();
            glEnable(GL_TEXTURE_2D);


            //Setup lighting
            glLightfv(GL_LIGHT0, GL_AMBIENT, lightambient);
            glLightfv(GL_LIGHT0, GL_DIFFUSE, lightdiffuse);
            glLightfv(GL_LIGHT0, GL_POSITION, lightpos);
            glEnable(GL_LIGHTING);
            glEnable(GL_LIGHT0);
            glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, 1);
            glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, material);
}

void display(void)
{
        //Set the camera position
        camerapos[0] = sin(rotation) * sin(elevation) * cameradistance;
        camerapos[1] = cos(elevation) * cameradistance;
        camerapos[2] = cos(rotation) * sin(elevation) * cameradistance;

        //Set up for rendering
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();
        gluLookAt(camerapos[0], camerapos[1], camerapos[2],
                                0.0f, 0.0f, 0.0f,0.0f, 1.0f, 0.0f);

        //Draw the cloth
        glColor3f(1.0f, 1.0f, 1.0f);
        drawcloth();

        glDisable(GL_TEXTURE_2D);

        //Draw the wind direction
        glColor3f(0.0f, 1.0f, 0.0f);
        glDisable(GL_LIGHTING);
        glBegin(GL_LINES);
        glVertex3fv(windvector);
        glVertex3f(0.0f, 0.0f, 0.0f);
        glEnd();
        glEnable(GL_LIGHTING);
        glColor3f(1.0f, 1.0f, 1.0f);

        //Draw the sphere
        glTranslatef(sphere[0], sphere[1], sphere[2]);
        glutSolidSphere(sphere[3], 16, 16);
```

```
            glEnable(GL_TEXTURE_2D);

            glutSwapBuffers();
}

void reshape(int w, int h)
{
            glViewport(0, 0, (GLsizei) w, (GLsizei) h);
            glMatrixMode(GL_PROJECTION);
            glLoadIdentity();
            gluPerspective(60.0, (GLfloat) w/(GLfloat) h, 0.1, 100.0);
            glMatrixMode(GL_MODELVIEW);
}

void mousemove(int x, int y)
{
            double modelviewmatrix[16], projectionmatrix[16];
            int viewport[4];
            double projectedpoint[3];
            float u;
            float veclength;

            //Move sphere
            if(controlmode == 1)
            {
                        glLoadIdentity();
                        gluLookAt(camerapos[0], camerapos[1], camerapos[2],
                                                0.0f, -0.5f, 0.0f,
                                                0.0f, 1.0f, 0.0f);
                        glGetDoublev(GL_MODELVIEW_MATRIX, modelviewmatrix);
                        glGetDoublev(GL_PROJECTION_MATRIX, projectionmatrix);
                        glGetIntegerv(GL_VIEWPORT, viewport);

                        gluUnProject(x, (viewport[3] - y), 1.0, modelviewmatrix, projectionmatrix, viewport,
                                                &(projectedpoint[0]), &(projectedpoint[1]), &(projectedpoint[2]));

                        u = camerapos[1] / (camerapos[1] - projectedpoint[1]);

                        sphere[0] = camerapos[0] + u * (projectedpoint[0] - camerapos[0]);
                        sphere[2] = camerapos[2] + u * (projectedpoint[2] - camerapos[2]);
            }
            //Move camera
            else if(controlmode == 0)
            {
                        rotation += (x - lastx) / 100.0f;
                        elevation += (y - lasty) / 100.0f;

                        if(elevation < 0.1f)
                                    elevation = 0.1f;
                        if(elevation > 3.0f)
                                    elevation = 3.0f;

                        camerapos[0] = sin(rotation) * sin(elevation) * cameradistance;
                        camerapos[1] = cos(elevation) * cameradistance;
                        camerapos[2] = cos(rotation) * sin(elevation) * cameradistance;
            }
            //Change wind direction
            else if(controlmode == 2)
            {
                        glLoadIdentity();
                        gluLookAt(camerapos[0], camerapos[1], camerapos[2],
                                                0.0f, -0.5f, 0.0f,
                                                0.0f, 1.0f, 0.0f);
                        glGetDoublev(GL_MODELVIEW_MATRIX, modelviewmatrix);
                        glGetDoublev(GL_PROJECTION_MATRIX, projectionmatrix);
```

```
                glGetIntegerv(GL_VIEWPORT, viewport);

                gluUnProject(x, (viewport[3] - y), 1.0, modelviewmatrix, projectionmatrix, viewport,
                             &(projectedpoint[0]), &(projectedpoint[1]), &(projectedpoint[2]));

                //Calculate wind direction
                u = camerapos[1] / (camerapos[1] - projectedpoint[1]);
                winddirection[0] = camerapos[0] + u * (projectedpoint[0] - camerapos[0]);
                winddirection[2] = camerapos[2] + u * (projectedpoint[2] - camerapos[2]);

                //Normalize wind direction
                veclength = (float) sqrt(winddirection[0] * winddirection[0]
                                         + winddirection[2] * winddirection[2]);
                winddirection[0] /= veclength;
                winddirection[2] /= veclength;
        }
        lastx = x;
        lasty = y;
        idle();
}

void mousedown(int button, int state, int x, int y)
{
        lastx = x;
        lasty = y;
}

void keyboard (unsigned char key, int x, int y)
{
        switch (key)
        {
                case '+':
                case '=':
                        windspeed += 1.0f;
                        break;
                case '-':
                        windspeed -= 1.0f;
                        break;
                case 'c':
                        initializecloth(0);
                        break;
                case 'z':
                        cameradistance += 0.5f;
                        break;
                case 'x':
                        cameradistance -= 0.5f;
                        if(cameradistance < 4.0f)
                                cameradistance = 4.0f;
                        break;
                case 't':
                        currenttexture++;
                        currenttexture %= 4;
                        generatetexture();
                        break;
                case 'e':
                        exit(-1);
                        break;

                default:
                        break;
        }
}
```

```
int main(int argc, char** argv)
{
            glutInit(&argc, argv);
            glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
            glutInitWindowSize(700, 700);
            glutCreateWindow("Resting");
            init();

            //Setup menu
            glutCreateMenu(menu);
            glutAddMenuEntry("Mouse changes view direction", 0);
            glutAddMenuEntry("Mouse changes sphere position", 1);
            glutAddMenuEntry("Mouse changes wind direction", 2);
            glutAttachMenu(GLUT_RIGHT_BUTTON);

            //Set up control functions
            glutIdleFunc(idle);
            glutDisplayFunc(display);
            glutReshapeFunc(reshape);
            glutKeyboardFunc(keyboard);
            glutMotionFunc(mousemove);
            glutMouseFunc(mousedown);
            myHelp();
            glutMainLoop();


            return 0;
}
```

REFERENCES

1. Ali Bahrami (1999), **Object Oriented System Development,** (1st Ed.), Singapore : Irwin McGraw-Hill.

2. Baraff, David and Andrew Witkin, **Large Steps in Cloth Simulation,** *Proceedings of SIGGRAPH 1998***,** ACM SIGGRAPH, pp. 43-54.

3. Eberhardt B, Weber A and Strasser W. **A fast, flexible, particle-system model for cloth-draping**. *IEEE Computer Graphics and Applications* 1996; **16**:52-59.

4. Jeff Lander, **Devil in the Blue-Faceted Dress: Real-Time Cloth Animation.** Journal of Game Developer, March 27, 2000.

5. Kenneth E. Kendall and Julie E. Kendall (1999), **System Analysis and Design,** (4th Ed.), United States of America, Prentice Hall International, Inc.

6. M. Kass, **An Introduction To Physically Based Modeling**, chapter Introduction to Continuum Dynamics for Computer Graphics. SIGGRAPH Course Notes, ACM SIGGRAPH, 1995.

7. Pascal Volino and Nadia Magnenat Thalmann, **Interactive Cloth Simulation: Problems and Solutions,** Journal of *MIRALab,* 1998.

8. Pfleeger, Shari Lawrence (2001), **Software engineering: Theory and practice,** (2nd Ed.), United States of America, Prentice Hall International, Inc. Mc GrawHill.

9. Sommerville, I. (2001), **Software Engineering** (6th Ed.), Addison Wesley.

10. T. Vassilev, B. Spanlang and Y. Chrysanthou, **Fast Cloth Animation on Walking Avatars,** EUROGRAPHICS, Vol. 20, No. 3, 2001.

11. Tania H. Gottschalk (1996), **Computer in Distance Education,** University of Idaho, Engineering Outreach.

12. Vivek Kwatra**. Cloth Simulation : Numerical Methods for PDEs**. College of Computing.

13. F.S.Hill, Jr (2001), **Computer Graphics Using OpenGL,** (2nd Ed.), United States of America, Prentice Hall International.

14. McReynolds, Tom and David Blythe, **Programming with OpenGL : Advanced Rendering,** (Course Nodes) SIGGRAPH' 97.

15. http://www.cgchannel.com

16. http://www.digimation.com

17. http://web.tiscali.it/no-redirect-tiscali/hiforce/ClotSim/indexeng.htm

18. http://www.qru.com/studio/tech/aug0398/cghistory.htm