JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Representation of sari using computer-generated patterns

Dissertation

zur Erlangung des akademischen Grades

Doktorin der Technischen Wissenschaften

Angefertigt am *Institut für Symbolisches Rechnen*

Betreuung:

Uni.-Doz. Dr. Sabine Stifter
O. Univ.-Prof. Dipl.-Math. Dr. Jochen Pfalzgraf

Eingereicht von:

M.C.M. Manasi Rahul Athale

Linz, Dezember 2004

Abstract

Cloth Modelling is contained in many applications: in e-commerce but in films too. A major aspect is put on *western clothes*, modelling of *eastern cloth* is not so much considered, especially in the literature. The reason may be that *eastern cloth* like *Indian Sari*, live much more from textile structure and draping the textile around a body.

In our thesis, we give an overview on geometric modelling methods and texture mapping with respect of cloth modelling and study their applicability with respect to the sari. In fact, none of the know methods is applicable without changes.

Sari is an Indian garment, worn by ladies. It is a rectangular, unstitched piece of fabric, woven using cotton, silk or other threads. Sari is usually five meters long in length and around one to one and half meters in width, with designed borders. Now a days, one can buy a sari on the world wide web. But its representation on the computer screen is the main obstacle in getting the idea of its texture and design pattern.

The kernel of the thesis is a generalization and adaptation of such methods, where we use a structuring of the textile shape into four major parts. Although a sari is one piece of textile, there are four parts distinguished by their patterns - twoKath, Padar and Ang -, which have to be modelled separately. By draping the textile around a body, i.e., by generating kinds of layers, the sari is worn and so a volumetric approach can model both, the structure of patterns and the draping around the body.

The proposed methods are implemented in Mesa, and examples of patterns for the sari, i.e., the four main parts are modelled and visualized. Pictures of results are shown in the thesis too.

Zusammenfassung

Die Modellierung von Bekleidung spielt in vielen Bereichen einen Rolle: im Bereich des E-Commerce aber auch in Trickfilmen. Ein Schwerpunkt wird hier auf „westliche Kleidung“ gelegt, die Modellierung von „östlicher Kleidung“ ist in der Literatur weniger betrachtet. Dies mag wohl darin liegen, dass bei „östlicher Kleidung“, zBsp. dem *indischen Sari*, sehr viel mehr Struktur im Muster und in der Drapierung liegt.

Der Sari ist ein Indianisches Gewand, getragen von Frauen. Es ist ein rechteckiges, ungenähtes Stück Stoff, und ist aus Baumwolle, Seide oder anderen Fäden gewebt. Ein Sari ist gewöhnlich fünf Meter lang und ca. 1 bis 1,5 Meter breit, mit gemusterten Rändern. Man kann zwar Saris übers Internet kaufen, aber die Darstellung ist meist ungenügend und gibt keinen guten Eindruck von der Qualität und dem Gewebe.

Wir geben in unserer Dissertation einen Überblick über Methoden des Geometrischen Modellierens und des Texture Mappings in Hinblick auf Modellierung von Bekleidung und studieren deren Anwendbarkeit auf das Bekleidungsstück „Sari“. Tatsächlich kann keine der Methoden unmittelbar angewendet werden.

Im Kern der Dissertation erweitern und adaptieren wir derartige Methoden, wobei ein Ansatz der Modellierung des textilen Stoffes in vier Teilbereiche - zwei Kath, Padar und Ang - verwendet werden. Obwohl ein Sari eine Stoffbahn ist, so sind in dieser Stoffbahn doch unterschiedliche Abschnitte durch unterschiedliche Musterungen erkennbar, die getrennt modeliert werden müssen, wobei hier gewisse sich wiederholende Strukturen zugrunde gelegt werden können. Durch die Drapierung der Stoffbahn um den Körper, d.h. ein „in-Falten-Legen“ des Stoffes, entsteht ein Kleidungsstück. Ein „volumetric-approach“ wurde gewählt, weil damit die Struktur des Musters und das geometrische Modell des Körpers gut vereint werden können.

Die vorgeschlagenen Methoden werden in Mesa implementiert und Beispiele für Muster für den sari, d.h. die vier Hauptteile, modelliert und simuliert. Bilder über die Ergebnisse runden die Dissertation ab.

Contents

List of Figures

Acknowledgements

I thank Prof. Sabine Stifter, my advisor, for all her support and guidance during my Ph.D. studies. I also thank Prof. Franz Winkler and Prof. Peter Paule for their help in academic matters. I thank Prod. David Breen, for his suggestions and encouragement. I thank Prof. Gerhard Kurka, for discussions and providing the study material.

Finally, I thank my lovely daughter, Sira, for being there patiently when I worked non-stop.

CHAPTER 1

Introduction

The world has become a *global village* because of the advent of the Internet. We can sit in one part of the world, say in Asia and use the Internet to communicate with a friend from other part of the world, say in Europe, in real time. We can buy almost everything, like books or electronic equipment over the Internet. Around three years ago, I wanted to buy a sari, an Indian garment worn by ladies about which I will talk later, on the Internet. I searched and found many on-line sari shops. I browsed their sari photographs and I was surprised with the quality of their representation on the computer screen. The web sites were very slow. Browsing through their products took a lot of time as every photograph of sari took a lot of time to load. Most of the web sites have photographs of saris, as a fabric, or as worn by someone. These photographs did not reveal any relevant information regarding the texture of the sari, its design patterns, or the details of the necessary parts.

When we want to buy a sari in a shop in person, we first see it folded, then the salesperson opens it and shows its important parts. We can touch the fabric of the sari and see if it is cotton or silk. In the light, we can see one layer or two layers to check the change in the colour. The appearance changes with the light. So when we buy a sari on-line we expect to see at least a few of these things. When we buy a book on-line, we get to read excerpt from the book. But when I browsed the on-line sari shops, I did not gather any such information from those photographs.

The web sites show the saris, sometimes as worn by the models or as hanging from a bar. One can see the colour of the sari but all the other details are not clear. Sometimes the photographs are not really aesthetic, like in Shalincraft India. Many such things regarding depicting a sari on the computer screen lingered in my mind for a long time.

My thesis advisor, Prof. Stifter introduced me to the texture mapping problems for my Ph. D. thesis. During the primary reading on the topic, I came across the problems in creating the realistic cloth textures. The jeans

trousers and t-shirts of the animated mouse in the film Stuart Little look realistic, while the clothing of Pocahontas, in the same named film, looks just like a patch of colours. What the Pocahontas character is wearing is a variation of a sari, but it does not look realistic. I realised that though computer graphics has progressed a lot in producing realistic clothing, not much effort has been made to show a realistic sari on the screen. So we decided to study the sari modelling and simulation and to generate it using the parts that are common to most of the saris. In short, the first task was to find the finite number of parts that will make any of the infinite number of possible saris on the screen, a symbolic computation problem.

The book by House and Breen [2000] proved to be the bible for my preliminary studies of the topic. I communicated with Prof. Breen via e-mail to discuss my particular problem with him. He has contributed extensively in the research of cloth modelling and simulation by suggesting a new method, namely a particle-based method. His suggestions about trying the clothing modelling software and hints about the difficulty level of the problem directed us to concentrate more on the representation of a sari and the study of the theoretical problems.

In the following chapter, we explain what is a sari.We also took liberty of explaining the Indian textile tradition as the variety of the fabric used for the saris is an important issue in creating the sari texture. We describe the problem of creating sari through the computer-generated patterns in detail.

We describe available clothing simulation software in the third chapter and our experiences with their installation and usage.

We take a brief look at Mesa, or OpenGL graphics programming language in the fourth chapter along with the texture mapping concept and how it is achieved in OpenGL. We use the term *OpenGL* for Mesa, as *Mesa* has the same structure like OpenGL and is authorised to follow the same syntax like OpenGL. So even if we discuss OpenGL here, we have actually used Mesa for our programming work.

In the fifth chapter, we give an overview of cloth modelling methods. Then we describe two approaches, one based on physically-based cloth simulation suggested by Baraff and Witkin and one based on particle-based approach suggested in House and Breen [2000]. We studied these methods from sari simulation point of view. We discuss here the most suitable one for simulation of a sari. As sari is worn in layers, cloth collision is a big issue. We explain collision detection algorithm.

We suggest an approach to generate a sari on the computer screen using four parts that are present in most of the saris. Though a sari is a continuous, non-stitched garment, we have to differentiate it into four symbolic parts, to make it representable on the computer screen. As usually there are many saris with the same four parts but with the

different colour we can save a lot of hard disk space with this approach. We explain this strategy in the sixth chapter. We are trying to achieve its full draping over a body.

In the last chapter, we list a few applications of our work, like simplicity of generation of the different saris will make it useful in the presentation of sari on an on-line shop web site.

CHAPTER 2

Sari

Animated figures, like characters in the animation films, need clothing. Most of the animated characters wear the skin-tight clothes like trousers and skirts. The sari, an Indian traditional wear for women, is quite different. It is worn around the body with some other under garments. So these kind of non-skin-tight clothing moves independently of the wearer. These depicts complicated movement with many wrinkles and creases. Thus, sari is itself a challenging thing to simulate.

For someone who is not very close to the Indian culture, it is very difficult to imagine what is a sari.Thanks to the world wide web, now there is a lot of information about the Indian textiles. Ghosh and Ghosh is a good source for the historical development of the Indian textile traditions. Also IndiCraft has a good collection of articles about the textile traditions in India and various clothing and printing styles from the different states of India. We reproduce some part from these sources, which is not technical but, is necessary for the better understanding of the problem.

2.1 History of Indian Textile Tradition

The origin of the Indian textiles can be traced to the Indus valley civilization. The people of this civilization used the homespun cotton for weaving their garments. Excavations at Harappa and Mohen-jo-Daro, have unearthed household items like needles made of bone and spindles made of wood, amply suggesting that homespun cotton was used to make garments. The fragments of woven cotton have also been found from these sites.

The first literary information about the textiles in India can be found in the Rigveda, which refers to weaving. The ancient Indian epics-Ramayan and Mahabharat also speak of a variety of fabrics of those times. The

Ramayan refers to the rich styles worn by the aristocracy on one hand and the simple clothes worn by the commoners and ascetics.

Ample evidence on the ancient textiles of India can also be obtained from the various sculptures belonging to Mauryan and Gupt age as well as from the ancient Buddhist scripts and murals in Ajantha caves.

India had numerous trade links with the outside world and the Indian textiles were popular in the ancient world. The Indian silk was popular in Rome in the early centuries of the Christian era. Hoards of fragments of cotton material originating from Gujarat have been found in the Egyptian tombs at Fostat, belonging to the fifth century A.D. Cotton textiles were also exported to China during the heyday of the silk route.

The silk fabrics from south India were exported to Indonesia during the thirteenth century. India also exported printed cotton fabrics or Chintz[1], to European countries and the Far East before the coming of the Europeans to India. The British East India Company also traded in the Indian cotton and silk fabrics, which included the famous Dacca muslin[2]. Muslin from Bengal, Bihar and Orissa were also popular abroad.

2.2 What is a sari?

The Indian sari (also written as saree, sadi) boasts of oldest existence in the sartorial world. It is more than 5000 years old. It is mentioned in the Vedas, written in 3000 B.C., the oldest existing (surviving) literature. Patterns of dress change throughout the world now and then but, the sari has survived. More than 75% of the woman population in India (now more than half a billion as per official estimate) wear versatile sari. We can certainly call this cloth versatile because it could be worn as shorts, trousers, flowing gown-like or convenient skirt-wise–all without a single stitch.

Sari (original–Chira in Sanskrit, cloth) is of varied length. From 5 yards[3] to 9.5 yards tied loosely, folded and pleated, it could be turned into working dress or party-wear with manual skill. The styles in wearing a sari vary from region to region. The Gujarati and Bengali style are different, and so are Mangalorean, Kannadiga, Kodava, Tamilian, Malayali, etc. The website Boulanger will be helpful to understand various ways to wear a sari.

[1]Chintz - Cotton cloth, printed with flowers and other devices, in a number of different colours, and often glazed.

[2]Muslin - A thin cotton, white, dyed, or printed. The name is also applied to coarser and heavier cotton goods; as, shirting and sheeting muslin.

[3]Yard - a unit of length equal to 3 feet; defined as 91.44 centimetres; originally taken to be the average length of a stride.

Thus, a sari is a rectangular piece of cloth, woven using cotton, silk or other threads, usually five meters long in length and around one to one and half meters in width. Please see Figure 2.1 to see a sari in its folded form. Although it is an un-tailored length of cloth, the fabric is highly structured and its design vocabulary very sophisticated. The main field of the sari is framed on three sides by a decorative frieze of flowering plants, figurative images or abstract symbols. Two of the borders define the edges of the length of the sari and the third comprises the end piece, which is a visible, broader, more complex version of the other two borders. This end piece is the part of the sari that is draped over the shoulder and left to hang over the back or front.Figure 2.2 shows saris worn in two styles. The red sari is worn in Gujarati style and the blue sari is worn in Maharashtrian style. This Maharashtrian style is very widely used and popular sari wearing style, please see Boulanger for further information on sari wearing styles.

Though, a sari is not a stitched fabric, for explaining it in simple terms, let us differentiate a sari into its main visual parts. So let us assume that a sari has four main parts: two Kath, Padar and Ang. A Kath is a border at the longer sides of the sari, woven using Jar. A Jar is a kind of brocade, usually golden or silver threads but sometimes it can be of the other colours matching to the fabric of the sari. Sometimes even real gold or silver filaments are also used in the Kath. The upper Kath is fixed in the petticoat and so it is not visible. A Padar is an ornamental border at the end of the sari, the end which is kept flowing from the shoulder. The inner Padar is wrapped around the waist under the layers of the sari, so is not visible and thus not so important. And Ang is the basic fabric of the sari, we can say it is the middle part of the sari, between the Kath and the Padar.

The Padar usually elaborates the theme found in the two borders and the actual field of the sari, a sort of repetition and amplification in the manner of the Indian musical mode, the raga. The raga has a set number of notes and these are intoned in a form of verbal mnemonics, before the song is actually sung. No new notes other than those in the introduction are used, but improvisation is allowed and results in endless permutations and combinations. This beautiful metaphor thus compares the two narrow borders to the introductory recital of the pure notes and the Padar to the song. The design, whether woven, embroidered, painted or block-printed, needs to maintain the proportion and balance between the Ang, two Kath and the Padar. The pattern creates its own rhythm. For instance, the scattering of spot weft gold dots increase in the Padar for a denser, richer pattern and gradually and softly decrease on the actual ground of the sari.

The traditions of the region dictates the pattern and content of the sari

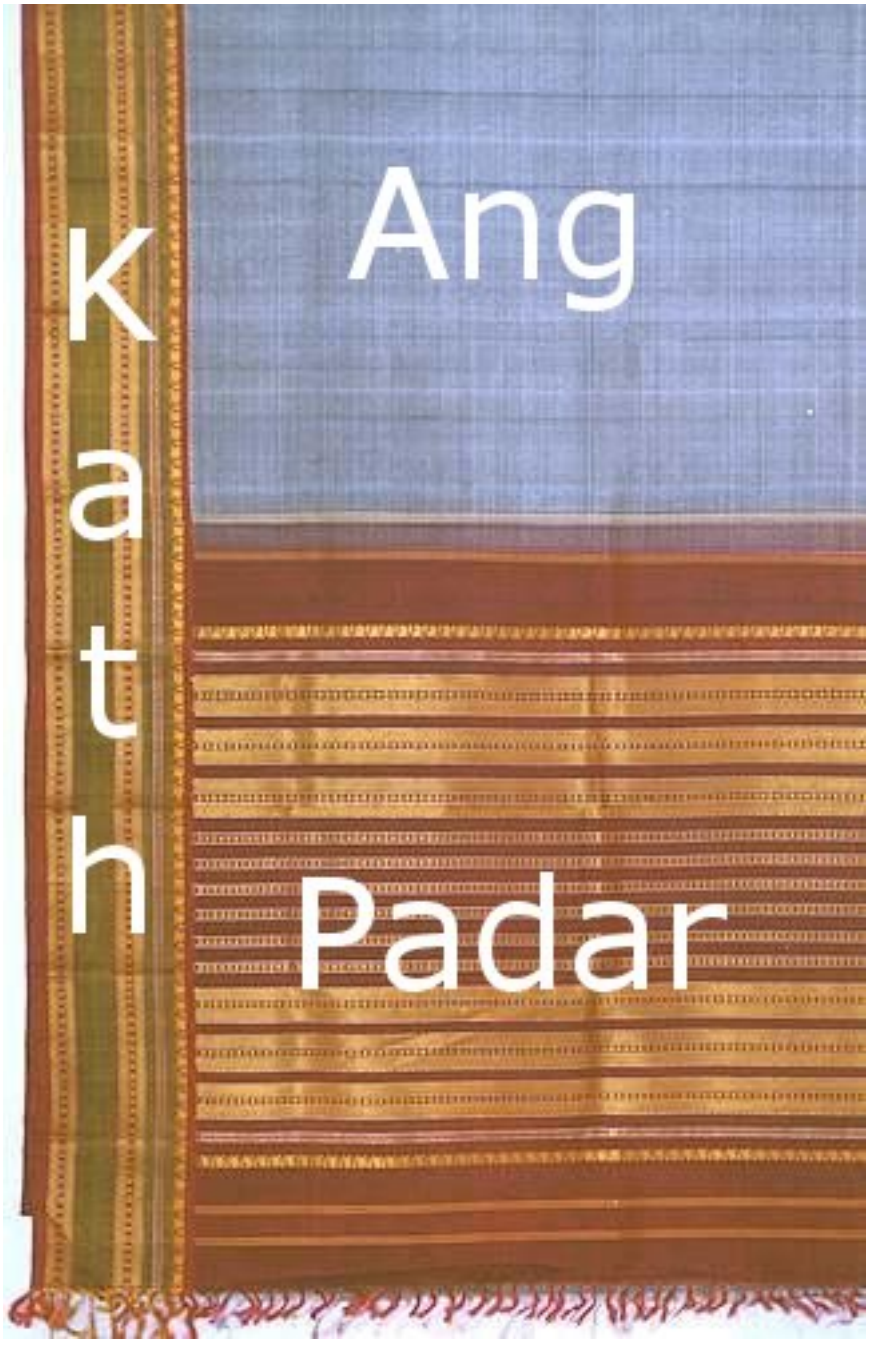


FIGURE 2.1: Sari in folded form

FIGURE 2.2: Sari worn in two different styles

where it is produced. Almost every district and sometimes even different villages have their own sari tradition which employs a complex language of symbols. But though characterized by geographical considerations, all the Indian symbolism, abstract or figurative, is rooted in the natural or physical world.

The fabric is always light enough not to interfere with the fluidity of the drape is another source of varied tactile delight—cottons, silks, cottons mixed with silk, chiffon and tissues are some of the preferred mediums. In the recent times, synthetic polyester has made inroads into the fashion world of the sari.

The sari takes final shape in visual terms only when it is draped on a person. The slightly off-centre fan of pleats in the front, the floating Padar with the intricate border thrown over the shoulder and the relatively smooth drape of the material at the back; the wound, pleated, tucked and coiled material give the proportions an aesthetic and intelligent rationality. To an unaccustomed onlooker, a draped sari seems an insecure affair, in danger of coming undone at the slightest movement. Actually, this apparently flimsy concoction is buttressed by a stout, distinctly unromantic, cotton petticoat. The top edges of the pleats are tucked into the waistband of this nether garment, thereby almost eliminating the risk of the sari coming adrift.

For an un-stitched length of material, the wearing of a sari entails a lot of preparation. Most saris have a fall made of cotton attached to the inside lower Kath, and the choli or blouse that teams up with the sari should match the ground colour of the sari, or at least echo one of the tints in the borders or motifs. The sari follows the shape of the body, yet conceals, it is often said, a hundred imperfections. It is true that not only is it one of the most graceful of garments, but also one of the kindest. This perhaps explains its perennial charm.

The success of the sari through the ages is attributable to its total simplicity and practical comfort, combined with the sense of luxury a woman experiences. Though men are intrigued by the demure, floor-length attire and tantalizing display of a bare midriff at the back, it is said that sari rarely fails to flatter a woman, making her feel fragile and feminine. It is an instant fashion, created by the hands of the wearer and subject to none of the vagaries and changes which plague the modern fashion scene.

Sari's price ranges from Rs. 250 to Rs. 50000, or Euro 5 to 1000 approximately and sometimes even more. This makes it accessible economically to almost all the people.

2.3 How to drape a sari?

As we have already explained there are many draping styles. Here we will explain step by step the most common method.

Step 1 To wear a sari, one needs a petticoat and a blouse. A petticoat is an undergarment worn under and prior the sari is draped around the body. Its height is from waist to the feet. It is of the same colour as theAnga of the sari so that it is not visible, directly or indirectly through the sari. A blouse is a top undergarment worn over the chest. Usually it is also of the same colour as of the Ang of the sari, but it could be matching to the Padar or contrast to the sari. It comes in various designs, sleeve's length is varying so as the neckline of the blouse. Also its length depends on the wearer's choice and liking. Usually it is tight fitting but one can found many variations of it too. So we can say that the blouse ends just above the waistline and the petticoat begins from the waistline.

Step 2 Just below the navel, upper end of the inner Kath of the sari is tucked into the petticoat and wrapped around it and brought where the wrap has begun. For a Maharashtrian draping style, this wrapping is done in anti-clockwise direction.

Step 3 This tucked-in end of the sari, is held in both hand and the pleats are made with left hand and right hand held the loose end of the sari, approximately four or five inches deep.

Step 4 About seven to ten pleats are made and hold them up together so that they fall straight and even.

Step 5 The pleats are tucked into the waist slightly to the left of the navel, just towards the left.

Step 6 The remaining sari is wrapped around the body, from left to right, and brought up from under the right arm and over the left shoulder and let it fall from backside about the level of the knees. This flowing part is the Padar of the sari.

Step 7 The Padar and pleats are fixed using a sari pin or a safety pin too.

Though, while reading its description of how to wear a sari, it looks like a complex process, it is very easy matter when we do the practical.

2.4 Problem Specification

The goal is to achieve the complete sari simulation. The whole problem is very difficult according to the expert's opinion. Therefore let us break this problem into small tasks.

First task is to divide this un-stitched long cloth in different parts so that we can represent the important parts with the details on the computer

screen. Also it should be more efficient in speed and memory than saving and showing the complete picture of a sari. This we have achieved by dividing a sari in four different parts as explained in the previous sections.

Every sari does not have all the parts. Sometimes even only Ang is present, of course then the design is not so interesting. But our program should cater all these possibilities. Also if we show the full length of the sari with exact proportions then Kath and Padar will be insignificant and one will not be able to see the details. Therefore in programming we have to somehow show these four parts in proportion where every part is significantly visible. This also helps us in generating different saris on the screen using different colours and texture mapping.

The next task is to simulate the draping of sari over the human body. To have a realistic sari simulation, we need a sari created using a cloth simulation software, which will serve the specific needs of the fabric of the sari. This involves the study of the cloth modelling methods. Also the task includes the survey of the available software and the previous work done in this direction.

The sari is wrapped over the body in layers. So collision of the cloth with itself, because of the layers, and collision of the cloth with the surface depicting human body is a crucial issue. Therefore the study of the collision detection algorithm is an important task. Finally, we need to find the suitable cloth modelling method for the generation of the realistic cloth, which also shows the minute differences of different fabrics used for the sari.

CHAPTER 3

Survey of Existing Systems

In this chapter, we describe the available software for cloth modelling and simulation.

One could try for freely available cloth simulators, but the main obstacle is there are not many. The only possibility is FreeCloth FreeCloth. This uses the method suggested by Baraff and Witkin Baraff and Witkin [1998] but matrices are solved using LU decomposition of a dense matrix, unlike to the Baraff and Witkin's modified conjugate method, which results in the computation of large cloth sizes impractical and also collision detection or adaptive time stepping is not been implemented. Hence there remains only one possibility to develop one cloth simulator as per the need.

This is a huge task in itself. We contacted Prof. Breen regarding our work and he warned us that it is huge task and will take enormous effort. According to him, sari is not yet modelled neither he knew any group working on this project. He suggested to use some kind of clothing modelling software from somewhere and then modify it for the sari simulation. FreeCloth is available for free use from the Internet but some of the libraries needed for it did not install in our institute and so we could not even try this. We tried to get hold of any other software but we did not find any working software of such type. Also commercial software systems like 3d Studio Max or Maya, don't come with the source code and they are very expensive. We found out that the only research group working on cloth modelling research is MIRALab in Switzerland. We could not communicate with this group and we found out that they usually do not share their software.

Our goal is to create computer-generated draping simulations of saris based on the type of material and design pattern. Since hardly anyone is aware of the sari patterns in computer graphics world, our main aim is to first introduce the concept and then improvise the simulation.

Unavailability of a good cloth modelling software forced us to study the cloth modelling methods from the point of view of sari and design a sari simulator.

3.1 Maya

The first and foremost software to simulate cloth is *Maya*. Maya is a trademark of *AliasWavefront*. An Academy Award winning software[1], Maya, is the most comprehensive software for producing 2D and 3D graphics. It is the most powerfully integrated software for 3D modelling, animation, effects, and rendering. It has customisable user interface and artist-friendly brush-based tools which make MayaŠs industry-standard 3D features easy to learn and easy to use. It adds to the quality and realism of 2D and 3D graphics. That's why it is widely used and popular among the film and video artists, game developers, visualization professionals, Web and print designers.

Maya family includes *Maya Unlimited*, *Maya Complete* and *Maya Personal Learning Edition*. The first two are for commercial users, and the last one is for non-commercial use and freely available from the web site Maya.

Maya can be used to produce unsurpassed character animation and create spectacular particle and dynamic effects, manage complex scenes and large data sets more efficiently. It can model breathtaking environments, move quickly back and forth between Maya and Adobe Photoshop, build and organize sophisticated shader networks faster. It uses mental ray more powerfully ever, add clothing, fur or long hair to the animated characters. It can also be to create and view interactive web pages for tutorials; scene and asset management.

This is especially useful to

- add 3D realism to 2D print projects and presentations such as advertisements, point of purchase displays and product logos,
- make web sites stand out from the crowd with dazzling visuals and animation,
- realize visually sophisticated artwork for books, magazines, posters, newsletters and other media,
- visualize models for industrial design, engineering and architecture, and then take them to the next level with 3D effects and animation,

[1]On March 1, 2003 the Academy of Motion Picture Arts and Sciences awarded AliasWavefront an Oscar for scientific and technical achievement for the development of Maya software.

- quickly add 3D treatments, animation and effects to existing work from other graphics programs, such as Adobe Photoshop and Illustrator,
- as well as the first choice for the creation of digital characters, animation and special effects for blockbuster movies and games.

Maya comes with three options:

Maya Unlimited is the ultimate version of Maya, and the choice of digital content creators who are looking to take their 3D projects to the most advanced level. Maya Unlimited provides artists and animators with industry leading innovations—Maya Cloth, Maya Fur, Maya Live and Maya Fluid Effects—for the creation of highly sophisticated digital content. It is available on Windows XP Professional, Windows 2000 Professional, IRIX and Linux except for Mac OS X.

Maya Complete integrates the worldŠs foremost animation, visual effects, modelling and advanced rendering technology into one complete workflow solution. Its development has been inspired by the film and video artists, computer game developers and design professionals who use it daily to create engaging digital imagery, animation and visual effects. It is available on Windows XP Professional, Windows 2000 Professional, MacOS X, IRIX and Linux.

Maya Personal Learning Edition is a special version of Maya software, which provides free access to Maya for non-commercial use. It allows 3D graphics and animation students, industry professionals, and those interested in breaking into the world of computer graphics (CG) an opportunity to explore all aspects of the award winning Maya Complete software in a non-commercial capacity. It is available for Mac OS X (10.2.4 or higher recommended), Windows 2000 Professional and Windows XP operating systems. It is not available for Windows 95, Windows 98, Windows ME, Linux or IRIX operating systems. Current version of Maya Personal Learning Edition is based on Maya 5.

Maya has tools such as marking menus and 3D manipulators and user friendly interface which help to speed up the workflow. It has tools for modelling a full suite of advanced Polygons, NURBS and Subdivision Surfaces. Its comprehensive range of keyframe, non-linear and advanced character animation editing tools help to create, animate, adapt and purpose the animation data and edit realistic digital characters. High-speed, dynamic interaction of hard and organic objects determined by physical rules help to create realistic visual effect. Brush-Based Technologies Maya

Artisan, Maya Paint Effects and 3D Paint offer a unique suite of integrated pressure sensitive brush tools for modelling, creating 2D and 3D effects, and painting on geometry and textures. A unified rendering workflow provides easy and consistent access to MayaŠs software, hardware, mental ray and vector renderers through a common interface. Maya API/SDK and MEL development resources allow to customize and extend Maya Šs capabilities via the renowned Maya embedded scripting language and a full Application ProgrammersŠ Interface.

Maya Unlimited 6 includes everything in Maya Complete. It has some additional tools which are useful for realistic computer animation. We list a few here.

Maya Fluid Effects, for the simulation and rendering of a huge variety of atmospheric, pyrotechnic, viscous liquid, and open ocean effects.

Maya Cloth, for simulating a wide variety of digital clothing and other fabric objects.

Maya Fur, for realistic styling and rendering of short hair and fur, with the Maya Artisan brush interface for painting fur attributes.

Maya Live , for creating original live-action footage with 3D elements rendered in Maya.

Maya Hair Tools, for the creation, styling and rendering of fully dynamic long hair on NURBS or polygon objects. It includes the ability to make any NURBS curve dynamic for use in advanced character rigging and effects.

Maya Personal Learning Edition has main toolsets of Maya Complete including modelling (NURBS, polygon, subdivision surface), animation, inverse kinematics, Maya Artisan, Maya Paint Effects, particles, dynamics and Maya's software and hardware renderers. The Maya Personal Learning Edition restricts users to non-commercial applications through the display of a watermark on images as well as through the use of a special non-commercial file format.

Thus, Maya Personal Learning Edition includes most of the functionality of Maya Complete with few differences. For example, is limited to using a single CPU. The commercial version of Maya takes advantage of multiple CPUs, resulting in faster performance in areas such as software rendering, IPR and Paint Effects. A watermark text image appears across all rendered images and in some Maya Personal Learning Edition panels. The watermark does not appear when working in wireframe mode. Rendering using mental ray, or vector renderer is not possible. The output cannot be saved as 16-bit rendered image formats. Camera's film fit offset

and film offset are limited to 0. Rendering is limited to a single CPU. Images from the software rendering output in Render View and batch mode, hardware rendering output in Render View and batch mode, Hardware render buffer Paint Effects canvas mode and scene mode, UV snapshot in UV texture Editor and 3D Paint Tool have been limited to 1024x768.

The standard Maya software files (.ma, .mb) cannot be used; it has different file format to save images, that is (.mp), however, the standard Maya software files can be imported. Particle disk caching is not supported. Text dump from the Blind Data Editor window is not supported. Exporting skin weight maps and character maps is not supported. The API developer's kit is not included with Maya Personal Learning Edition. It is not possible to load plug-ins from the commercial version of Maya or third party plug-ins. The scriptEditor - writeHistory option is not available. The following MEL commands are not available: system, fopen, popen, fwrite, fprint and cmdpipe. The script editor output section is limited to 75 lines of output. The script editor > Save Selected menu item has been removed. The background colour's value (in HSV space) of interactive window is limited to [0.3,0.9].

Since, Maya Personal Learning Edition is based on Maya Complete, therefore, features that are part of Maya Unlimited are not included. Hence, it does not have features like Maya Cloth.

The distinguished feature of Maya in which we are interested is *Maya Cloth*. It is the most accurate and fastest software solution for simulating a wide variety of digital clothing and other fabric objects.

It allows user to work with cloth objects created from modelled geometry or garments constructed from flat panels. It's intuitive workflow enables any 3D figure to be dressed and animated with automatic stitching, draping, and gathering of cloth panels into a perfectly fitting garment. It is possible to create realistic clothing including jackets with collars, vents and lapels, pants with cuffs and pockets. Also one can create loose or tight fitting clothing styles. Also it is possible to simulate fabric such as heavy cotton, stiff canvas and thick leather or even mix fabrics in one garment. Maya's clothing pinching solution addresses the difficult problem of cloth pinched between two objects such as under the arm. In Maya, controls such as button constraints, or cloth-to-cloth constraints allow for greater control of realistic clothing behaviour.

Using Maya, any cloth object can be animated including sails, skins, tents, drapery, bedding etc.. Real-world physical cloth characteristics can be reproduced using Maya Cloth at unsurpassed speed and accuracy. It is possible to animate multiple independent cloth systems with their own objects and forces and caching of data is supported for real-time playback.

Maya Cloth is fully(or totally) integrated with Maya Software. Thus, in Maya, clothing moves, folds, and gathers whenever the animated char-

acters move. Texturing and shading is easier using Maya Cloth. Cloth property and texture painting is done with Maya Artisan brush interface. In Maya, use of it's dynamic forces such as turbulence and air fields to create strong wind effects on a coat or to make rain particles splash off a flapping raincoat.

Maya Unlimited version is for around 7000 dollar and Maya Complete comes with price around 2000 dollar, for node-locked licenses. Node-locked License is locked to one workstation and cannot be used on other machines.

Since, Maya Unlimited is very expensive and its free version does not include the Maya Cloth, we could not use Maya.

3.2 SimCloth with 3D Studio MAX

SimCloth is a simple cloth plugin for *3D Studio MAX* capable of simulating different kinds of fabrics, as well as rigid bodies. SimCloth supports full collision and self-collision detection and response. Taking a source object it tries to modify it so that it looks as if its a soft or hard body or a piece of cloth having properties like material friction, internal tension, softness. The modification is based on collision objects that can be specified within the plugin itself.

It is freely available from the web site, SimCloth. Version 3 is also open source so that you can modify it to suit your needs.

3D Studio MAX software prices range between 200 to 1000 dollars, 3d Studio Max. Even though SimCloth is free, we need to buy 3D Studio MAX which is still very expensive so had to search for other cloth simulator which are freely available.

3.3 FreeCloth

FreeCloth is a free, open-source cloth simulation tool. It is intended to help with further research in cloth simulation, and to help in production of feature films or games using cloth. In its initial form, the simulation has been implemented using algorithms described in "Large Steps in Cloth Simulation" by Baraff and Witkin. Pritchard gives detailed information about the implementation work of FreeCloth. Till the FreeCloth, version 6, some features like collision detection are not incorporated but still it is better tool to begin with. It is available from the web site, FreeCloth. We tried to install and this version, that is, version 6, but its latest version is 7, available from the above web site although we did not yet try it. Since, its source code is available for improvisation, we were interested

to use it. We tried its installation and encountered some problem which we could not overcome. We enlist main problems here as they are quite a hindrance to the popularity of this software.

FreeCloth is implemented in C++. It depends upon external free libraries, that are, OpenGL, GLUT, GLUI, and libraries for liner algebra functionalities. Its version 5 needs Matrix Template Library, MTL, and version 6 needs BoostBoost and uBLAS, uBLAS, which in turn needs LAPACK, LAPACK. Till this version 6, it has been compiled on Linux and Windows, but it might be straightforward to port to other operating systems, provided that a good enough C++ compiler is available. Solid C++ engineering techniques have been applied to create a fairly robust and modern application. It depends upon external free libraries, that are, OpenGL, GLUT, GLUI, and libraries for liner algebra functionalities. Its version 5 needs Matrix Template Library, MTL, and version 6 needs BoostBoost and uBLAS, uBLAS, which in turn needs LAPACK, LAPACK. Till this version 6, it has been compiled on Linux and Windows, but it might be straightforward to port to other operating systems, provided that a good enough C++ compiler is available. Solid C++ engineering techniques have been applied to create a fairly robust and modern application.

uBLAS is a C++ template class library that provides *BLAS* level 1, 2, 3 functionality for dense, packed and sparse matrices. The design and implementation unify mathematical notation via operator overloading and efficient code generation via expression templates.

The *BLAS* (Basic Linear Algebra Subprograms) are high quality "building block" routines for performing basic vector and matrix operations. Level 1 BLAS do vector-vector operations, Level 2 BLAS do matrix-vector operations, and Level 3 BLAS do matrix-matrix operations. Because the BLAS are efficient, portable, and widely available, they're commonly used in the development of high quality linear algebra software, LINPACK and LAPACK for example.

LINPACK is a collection of Fortran subroutines that analyse and solve linear equations and linear least-squares problems. The package solves linear systems whose matrices are general, banded, symmetric indefinite, symmetric positive definite, triangular, and tri-diagonal square. In addition, the package computes the QR and singular value decompositions of rectangular matrices and applies them to least-squares problems. LINPACK uses column-oriented algorithms to increase efficiency by preserving locality of reference.

LINPACK was designed for supercomputers in use in the 1970s and early 1980s. LINPACK has been largely superseded by LAPACK, which has been designed to run efficiently on shared-memory, vector supercomputers.

LAPACK, or Linear Algebra PACKage, provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems. The associated matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur) are also provided, as are related computations such as reordering of the Schur factorizations and estimating condition numbers. Dense and banded matrices are handled, but not general sparse matrices. In all areas, similar functionality is provided for real and complex matrices, in both single and double precision.

The *MTL*, or Matrix Template Library, is a high-performance generic component library that provides comprehensive linear algebra functionality for a wide variety of matrix formats.

MTL uses a five-fold approach, consisting of generic functions, containers, iterators, adaptors, and function objects, all developed specifically for high performance numerical linear algebra. Within this framework, MTL provides generic algorithms corresponding to the mathematical operations that define linear algebra. Similarly, the containers, adaptors, and iterators are used to represent and to manipulate concrete linear algebra objects such as matrices and vectors.

To many scientific computing users, however, the advantages of an elegant programming interface are secondary to issues of performance. Generic programming is a powerful tool in this regard as well - performance tuning can itself be described in a generic fashion. These performance tuning abstractions are realized in a generic low-level library - the Basic Linear Algebra Instruction Set (BLAIS). Experimental results show that MTL with the BLAIS achieves performance that is as good as, or better than, vendor-tuned libraries. Thus, MTL demonstrates that the proper abstractions can be used to achieve high levels of performance, contrary to conventional wisdom. In addition, MTL requires orders of magnitude fewer lines of code for its implementation, with the concomitant savings in development and maintenance effort.

We tried to first install FreeCloth 5.0 and later 6.0 on Linux , but neither version worked. We needed to install other libraries, mentioned above, separately which did not install properly as some necessary files were missing. It took almost three weeks to download and try the installation of these libraries and FreeCloth itself. Since we could not find every file needed for the installation, we had to give up our efforts to use FreeCloth.

3.4 Mesa3d

The Mesa Mesa project was founded by Brian Paul. Mesa is a three dimensional graphics library with an API which is very similar to that of

OpenGL. To the extent that Mesa utilizes the OpenGL command syntax or state machine, it is being used with authorization from Silicon Graphics, Inc.(SGI). However, the author does not possess an OpenGL license from SGI, and makes no claim that Mesa is in any way a compatible replacement for OpenGL or associated with SGI. We here give a brief overview of Mesa and OpenGL as we used Mesa for our implementation work but since Mesa uses the syntax from OpenGL, we explain the working of OpenGL even though we did not exactly use it.

Mesa is an open-source implementation of the OpenGL specification. OpenGL is a programming library for writing interactive 3D applications. Mesa 5.x supports the OpenGL 1.4 specification. Mesa serves as the OpenGL core for the open-source XFree86/DRI OpenGL drivers. Hardware-accelerated OpenGL implementations are available for most popular operating systems today. Still, Mesa serves at least these purposes:

- Mesa is used as the core of the open-source XFree86/DRI hardware drivers.
- Mesa is quite portable and allows OpenGL to be used on systems that have no other OpenGL solution.
- Software rendering with Mesa serves as a reference for validating the hardware drivers.
- A software implementation of OpenGL is useful for experimentation, such as testing new rendering techniques.
- Mesa can render images with deep colour channels: 16-bit integer and 32-bit floating point colour channels are supported. This capability is only now appearing in hardware.
- Mesa's internal limits, maximum lights, clip planes, texture size, etc., can be changed for special needs, although hardware limits are hard to overcome.

Mesa serves as the OpenGL core for the open-source XFree86/DRI OpenGL drivers. There have been other hardware drivers for Mesa over the years, such as the 3Dfx Glide/Voodoo driver, an old S3 driver, etc., but the DRI drivers are the modern ones. We cannot upgrade the DRI installation to use a new Mesa release, as a copy of the Mesa source code lives inside the XFree86/DRI source tree and gets compiled into the individual DRI driver modules. If we try to install Mesa over an XFree86/DRI installation, we lose hardware rendering, because standalone Mesa's libGL.so is different than the XFree86 libGL.so.

To install Mesa on a Linux-based system, we can download it from the web site Mesa or use the distro CD, which most probably may have Mesa packages, like RPM or DEB . Unfortunately, the GNU autoconf/automake/libtool system does not work too well on non GNU/Linux systems. So Mesa uses a rather conventional Makefile system.

A GNU autoconf/automake system used to be included, but was discarded in Mesa 5.1 because it seldom worked on IRIX, Solaris, AIX, etc. For Compilation, once we get the hold of Mesa through CVS, we do this first: `cd Mesa chmod a+x bin/mklib` Just type make in the top-level directory, and we see a list of supported system configuration. Choose one from the list (such as linux-x86), and type: `make linux-x86`

We rebuild it for a different configuration by running make realclean before rebuilding. When compilation hi finished, we look in the top-level lib/ directory. We see a set of library files, like libGL, the main OpenGL library (i.e. Mesa), libGLU, the OpenGL Utility library, libglut, the GLUT library, libGLw, the Xt/Motif OpenGL drawing area widget library, libOSMesa, the OSMesa (Off-Screen) interface library.

Once we download or unpack the MesaDemos-x.y.z.tar.gz archive or obtained Mesa from CVS, the progs/ directory will contain a bunch of demonstration programs. Before running a demo, we may have to set an environment variable (such as `LD_LIBRARY_PATH on Linux` to indicate where the libraries are located.

The standard location for the OpenGL header files on Unix-type systems is in /usr/include/GL/. The standard location for the libraries is /usr/lib/. To install Mesa's headers and libraries, we run make install. Then we get the prompt to enter alternative directories for the headers and libraries.

We give here a brief procedure of installing Mesa on Linux environment, but it does not go so smooth as it is been described here. We almost spend fours weeks and eventually installed Mesa at least three times. Once we run some make command from above procedure, it gave some error messages. Then we checked logs and found what was missing. Then we go back to download and install that particular library and then again compile Mesa. It was quite a tedious work although it worked at the end and was a great help for our implementation work. The only problem about using Mesa is we cannot upgrade it to latest release and we would have to do all the same procedure we did for the version we installed, that is Mesa, 5.0. Mesa uses an even/odd version number scheme like the Linux kernel. for example, odd numbered versions designate new developmental releases and even numbered versions designate stable releases. So, using Mesa 5.0 has been proved a good choice and we did not encounter any problem during our implementation work.

3.5 OpenGL

OpenGL is a graphics rendering library, that is, it is a layer of abstraction between graphics hardware and an application program. It it an API to produce high-quality colour images from geometric and raster primitives. API is an Application Programming (or Procedural) Interface. Geometric primitives are vertex-based and are either 2D or 3D. Raster primitives are pixel-based (either bitmaps or pixmaps) and generally 2D. Texture mapping combines both raster and geometric primitives to create an image. OpenGL libraries are supported for use with X Window System and UNIX, Microsoft Windows, Microsoft Windows NT, and IBM OS/2. It it Window System and Operating System independent. It does not perform operations which are redundant with the window system: window management, event (mouse and keyboard) handling, and loading colour maps. We used the OpenGL Blue Book; OpenGL Red Book for studying the OpenGL concepts.

As a software interface for graphics hardware, OpenGL's main purpose is to render two- and three-dimensional objects into a frame buffer. These objects are described as sequences of vertices, which define geometric objects, or pixels, which define images. OpenGL performs several processing steps on this data to convert it to pixels to form the final desired image in the frame buffer.

Here, we present a global view of how OpenGL works;

- OpenGL Fundamentals explains basic OpenGL concepts, such as what a graphic primitive is and how OpenGL implements a client-server execution model.

- Basic OpenGL Operation gives a high level description of how OpenGL processes data and produces a corresponding image in the frame buffer.

3.5.1 OpenGL Fundamentals

This section explains some of the concepts inherent in OpenGL.

Primitives and Commands

OpenGL draws *primitives*, points, line segments, or polygons, subject to several selectable modes. We can control modes independently of each other; that is, setting one mode does not affect whether other modes are set, although many modes may interact to determine what eventually ends up in the frame buffer. Primitives are specified, modes are set, and

other OpenGL operations are described by issuing commands in the form of function calls.

Primitives are defined by a group of one or more *vertices*. A vertex defines a point, an endpoint of a line, or a corner of a polygon where two edges meet. Data consisting of vertex coordinates, colours, normals, texture coordinates, and edge flags is associated with a vertex, and each vertex and its associated data are processed independently, in order, and in the same way. The only exception to this rule is if the group of vertices must be clipped so that a particular primitive fits within a specified region; in this case, vertex data may be modified and new vertices created. The type of clipping depends on which primitive the group of vertices represents.

Commands are always processed in the order in which they are received, although there may be an indeterminate delay before a command takes effect. This means that each primitive is drawn completely before any subsequent command takes effect. It also means that state-querying commands return data that is consistent with complete execution of all previously issued OpenGL commands.

Procedural versus Descriptive

OpenGL provides direct control over the fundamental operations of two and three dimensional graphics. This includes specification of such parameters as transformation matrices, lighting equation coefficients, antialiasing methods, and pixel update operators. However, it does not provide a means for describing or modelling complex geometric objects. Thus, the issued OpenGL commands specify how a certain result should be produced, that is, what procedure should be followed, rather than what exactly that result should look like. That is, OpenGL is fundamentally procedural rather than descriptive. Because of this procedural nature, it helps to know how OpenGL works—the order in which it carries out its operations, for example—in order to fully understand how to use it.

Execution Model

OpenGL commands are interpreted using client-server model. An application, the client, issues commands, which are interpreted and processed by OpenGL, the server. The server may or may not operate on the same computer as the client. In this sense, OpenGL is network-transparent. A server can maintain several GL contexts, each of which is an encapsulated GL state. A client can connect to any one of these contexts. The required network protocol can be implemented by augmenting an already existing

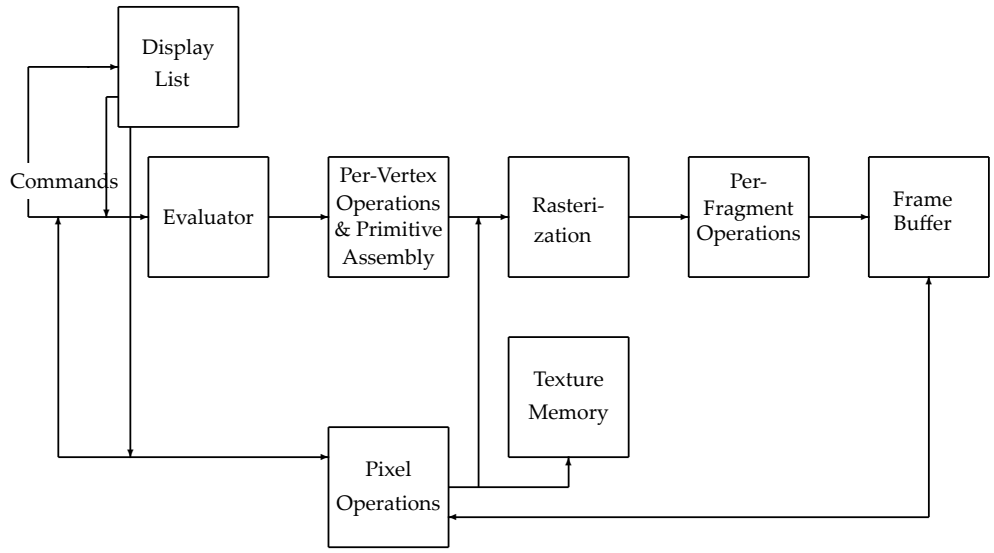


FIGURE 3.1: OpenGL Operations

protocol, such as that of the X Window System, or by using an independent protocol. No OpenGL commands are provided for obtaining user input.

The effects of OpenGL commands on the frame buffer are controlled by the window system that allocates frame buffer resources. The window system determines which portions of the frame buffer OpenGL may access at any given time and communicates to OpenGL how those portions are structured. Therefore, there are no OpenGL commands to configure the frame buffer or initialise OpenGL. Frame buffer configuration is done outside of OpenGL in conjunction with the window system; OpenGL initialization takes place when the window system allocates a window for OpenGL rendering.

Basic OpenGL Operation

The Figure 3.1 gives an abstract, high-level block diagram of how OpenGL processes data. In the diagram, commands enter from the left and proceed through what can be thought of as a processing pipeline. Some commands specify geometric objects to be drawn, and others control how the objects are handled during the various processing stages.

As shown by the first block in the diagram, rather than having all commands proceed immediately through the pipeline, we can choose to accumulate some of them in a display list for processing at a later time.

The *evaluator* stage of processing efficiently approximates curve and surface geometry by evaluating polynomial commands of input values. During the next stage, per-vertex operations and primitive assembly, OpenGL processes geometric primitives—points, line segments, and polygons, all of which are described by vertices. Vertices are transformed and lit, and primitives are clipped to the viewport in preparation for the next stage.

Rasterization produces a series of frame buffer addresses and associated values using a two-dimensional description of a point, line segment, or polygon. Each *fragment* so produced is fed into the last stage, *per-fragment operations*, which performs the final operations on the data before it's stored as pixels in the *frame buffer*. These operations include conditional updates to the frame buffer based on incoming and previously stored z-values, for z-buffering, and blending of incoming pixel colours with stored colours, as well as masking and other logical operations on pixel values.

Input data can be in the form of pixels rather than vertices. Such data, which might describe an image for use in texture mapping, skips the first stage of processing described above and instead is processed as pixels, in the *pixel operations* stage. The result of this stage is either stored as *texture memory*, for use in the rasterization stage, or rasterized and the resulting fragments merged into the frame buffer just as if they were generated from geometric data.

All elements of OpenGL state, including the contents of the texture memory and even of the frame buffer, can be obtained by an OpenGL application.

3.5.2 Overview of Commands and Routines

Many OpenGL commands pertain specifically to drawing objects such as points, lines, polygons, and bitmaps. Other commands control the way that some of this drawing occurs, such as those that enable antialiasing or texturing. Still other commands are specifically concerned with frame buffer manipulation. In this section, we briefly describe how all the OpenGL commands work together to create the OpenGL processing pipeline. Brief overviews are also given of the routines comprising the OpenGL Utility Library (GLU) and the OpenGL extensions to the X Window System (GLX).

OpenGL Processing Pipeline

Now that we have a general idea of how OpenGL works from previous section, let us take a closer look at the stages in which data is actually

processed and tie these stages to OpenGL commands. The following figure is a more detailed block diagram of the OpenGL processing pipeline.

For most of the pipeline, we can see three vertical arrows between the major stages. These arrows represent vertices and the two primary types of data that can be associated with vertices: colour values and texture coordinates. Also the vertices are assembled into primitives, then to fragments, and finally to pixels in the frame buffer. Here, we discuss this progression in more detail.

Many OpenGL commands are simple variations of each other, differing mostly in the data type of arguments; some commands differ in the number of related arguments and whether those arguments can be specified as a vector or whether they must be specified separately in a list. For example, if we use the glVertex2f() command, we need to supply x and y coordinates as 32-bit floating-point numbers; with glVertex3sv(), we must supply an array of three short (16-bit) integer values for x, y, and z. For simplicity, only the base name of the command is used in the discussion that follows, and an asterisk is included to indicate that there may be more to the actual command name than is being shown. For example, `glVertex*()` stands for all variations of the command we use to specify vertices.

Also the effect of an OpenGL command may vary depending on whether certain modes are enabled. For example, we need to enable lighting if the lighting-related commands are to have the desired effect of producing a properly lit object. To enable a particular mode, we use the `glEnable()` command and supply the appropriate constant to identify the mode, for example, `GL_LIGHTING`. Here we do not discuss specific modes, but we can refer to the reference page for `glEnable()` for a complete list of the modes that can be enabled, OpenGL Blue Book. Modes are disabled with `glDisable()`.

Vertices

We explain the OpenGL commands that perform per-vertex operations to the processing stages.

Input Data We must provide several types of input data to the OpenGL pipeline:

Vertices Used to describe the shape of the desired geometric object. To specify vertices, We use `glVertex*()` commands in conjunction with `glBegin()` and `glEnd()` to create a point, line, or polygon. We can also use `glRect*()` to describe an entire rectangle at once.

Edge flag By default, all edges of polygons are boundary edges. We use the `glEdgeFlag*()` command to explicitly set the edge flag.

Current raster position This is specified with `glRasterPos*()`. The current raster position is used to determine raster coordinates for pixel and bitmap drawing operations.

Current normal A normal vector associated with a particular vertex determines how a surface at that vertex is oriented in three-dimensional space. This affects how much light that particular vertex receives. We use `glNormal*()` to specify a normal vector.

Current colour The colour of a vertex, together with the lighting conditions, determine the final, lit colour. Colour is specified with `glColor*()` if in RGBA mode or with `glIndex*()` if in colour index mode.

Current texture coordinates Specified with `glTexCoord*()`, texture coordinates determine the location in a texture map that should be associated with a vertex of an object.

When `glVertex*()` is called, the resulting vertex inherits the current edge flag, normal, colour, and texture coordinates. Therefore, `glEdgeFlag*()`, `glNormal*()`, `glColor*()`, and `glTexCoord*()` must be called before `glVertex*()` if they are to affect the resulting vertex.

Matrix Transformations Vertices and normals are transformed by the modelview and projection matrices before they're used to produce an image in the frame buffer. We can use commands such as `glMatrixMode()`, `glMultMatrix()`, `glRotate()`, `glTranslate()`, and `glScale()` to compose the desired transformations, or we can directly specify matrices with `glLoadMatrix()` and `glLoadIdentity()`. Use `glPushMatrix()` and `glPopMatrix()` to save and restore modelview and projection matrices on their respective stacks.

Lighting and Colouring In addition to specifying colours and normal vectors, we may define the desired lighting conditions with `glLight*()` and `glLightModel*()`, and the desired material properties with `glMaterial*()`. Related commands we might use to control how lighting calculations are performed include `glShadeModel()`, `glFrontFace()`, and `glColorMaterial()`.

Generating Texture Coordinates Rather than explicitly supplying texture coordinates, we can have OpenGL generate them as a function of other vertex data. This is what the `glTexGen*()` command does. After the texture coordinates have been specified or generated, they are transformed by the texture matrix. This matrix is controlled with the same commands mentioned earlier for matrix transformations.

Primitive Assembly Once all these calculations have been performed, vertices are assembled into primitivesŮpoints, line segments, or polygons together with the relevant edge flag, colour, and texture information for each vertex.

Primitives

During the next stage of processing, primitives are converted to pixel fragments in several steps: primitives are clipped appropriately, whatever corresponding adjustments are necessary are made to the colour and texture data, and the relevant coordinates are transformed to window coordinates. Finally, rasterization converts the clipped primitives to pixel fragments.

Clipping Points, line segments, and polygons are handled slightly differently during clipping. Points are either retained in their original state, if they are inside the clip volume, or discarded, if they are outside. If portions of line segments or polygons are outside the clip volume, new vertices are generated at the clip points. For polygons, an entire edge may need to be constructed between such new vertices. For both line segments and polygons that are clipped, the edge flag, colour, and texture information is assigned to all new vertices.

Clipping actually happens in two steps:

Application-specific clipping Immediately after primitives are assembled, they are clipped in eye coordinates as necessary for any arbitrary clipping planes we have defined for our application with `glClipPlane()`. OpenGL requires support for at least six such application-specific clipping planes.

View volume clipping Next, primitives are transformed by the projection matrix (into clip coordinates) and clipped by the corresponding viewing volume. This matrix can be controlled by the previously mentioned matrix transformation commands but is most typically specified by `glFrustum()`or `glOrtho()`.

Transforming to Window Coordinates Before clip coordinates can be converted to window coordinates, they are normalized by dividing by the value of `w` to yield normalized device coordinates. After that, the viewport transformation applied to these normalized coordinates produces window coordinates. We control the viewport, which determines the area of the on-screen window that displays an image, with `glDepthRange()` and `glViewport()`.

Rasterization Rasterization is the process by which a primitive is converted to a two-dimensional image. Each point of this image contains such information as colour, depth, and texture data. Together, a point and its associated information are called a fragment. The current raster position (as specified with `glRasterPos*()`) is used in various ways during this stage for pixel drawing and bitmaps. As discussed below, different issues arise when rasterizing the three different types of primitives; in addition, pixel rectangles and bitmaps need to be rasterized.

Primitives

We control how primitives are rasterized with commands that allow we to choose dimensions and stipple patterns: `glPointSize()`, `glLineWidth()`, `glLineStipple()`, and `glPolygonStipple()`. Additionally, we can control how the front and back faces of polygons are rasterized with `glCullFace()`, `glFrontFace()`, and `glPolygonMode()`.

Pixels

Several commands control pixel storage and transfer modes. The command `glPixelStore*()` controls the encoding of pixels in client memory, and `glPixelTransfer*()` and `glPixelMap*()` control how pixels are processed before being placed in the frame buffer. A pixel rectangle is specified with `glDrawPixels()`; its rasterization is controlled with `glPixelZoom()`.

Bitmaps

Bitmaps are rectangles of zeros and ones specifying a particular pattern of fragments to be produced. Each of these fragments has the same associated data. A bitmap is specified using `glBitmap()`.

Texture Memory

Texturing maps a portion of a specified texture image onto each primitive when texturing is enabled. This mapping is accomplished by using the colour of the texture image at the location indicated by a fragment's texture coordinates to modify the fragment's RGBA colour. A texture image is specified using glTexImage2D() or glTexImage1D(). The commands `glTexParameter*()` and `glTexEnv*()` control how texture values are interpreted and applied to a fragment.

Fog

We can have OpenGL blend a fog colour with a rasterized fragment's post-texturing colour using a blending factor that depends on the distance between the eyepoint and the fragment. Use `glFog*()` to specify the fog colour and blending factor.

Fragments

OpenGL allows a fragment produced by rasterization to modify the corresponding pixel in the frame buffer only if it passes a series of tests. If it does pass, the fragment's data can be used directly to replace the existing frame buffer values, or it can be combined with existing data in the frame buffer, depending on the state of certain modes.

Pixel Ownership Test The first test is to determine whether the pixel in the frame buffer corresponding to a particular fragment is owned by the current OpenGL context. If so, the fragment proceeds to the next test. If not, the window system determines whether the fragment is discarded or whether any further fragment operations will be performed with that fragment. This test allows the window system to control OpenGL's behaviour when, for example, an OpenGL window is obscured.

Scissor Test With the `glScissor()` command, we can specify an arbitrary screen-aligned rectangle outside of which fragments will be discarded.

Alpha Test The alpha test, which is performed only in RGBA mode, discards a fragment depending on the outcome of a comparison between the fragment's alpha value and a constant reference value. The comparison command and reference value are specified with `glAlphaFunc()`.

Stencil Test The stencil test conditionally discards a fragment based on the outcome of a comparison between the value in the stencil buffer and a reference value. The command `glStencilFunc()` specifies the comparison command and the reference value. Whether the fragment passes or fails the stencil test, the value in the stencil buffer is modified according to the instructions specified with `glStencilOp()`.

Depth Buffer Test The depth buffer test discards a fragment if a depth comparison fails; `glDepthFunc()` specifies the comparison command. The result of the depth comparison also affects the stencil buffer update value if stenciling is enabled.

Blending Blending combines a fragment's R, G, B, and A values with those stored in the frame buffer at the corresponding location. The blending, which is performed only in RGBA mode, depends on the alpha value of the fragment and that of the corresponding currently stored pixel; it might also depend on the RGB values. We control blending with `glBlendFunc()`, which allows we to indicate the source and destination blending factors.

Dithering If dithering is enabled, a dithering algorithm is applied to the fragment's colour or colour index value. This algorithm depends only on the fragment's value and its x and y window coordinates.

Logical Operations Finally, a logical operation can be applied between the fragment and the value stored at the corresponding location in the frame buffer; the result replaces the current frame buffer value. We choose the desired logical operation with `glLogicOp()`. Logical operations are performed only on colour indices, never on RGBA values.

Pixels

During the previous stage of the OpenGL pipeline, fragments are converted to pixels in the frame buffer. The frame buffer is actually organized into a set of logical buffers—the colour, depth, stencil, and accumulation buffers. The colour buffer itself consists of a front left, front right, back left, back right, and some number of auxiliary buffers. We can issue commands to control these buffers, and we can directly read or copy pixels from them.

Frame Buffer Operations We can select into which buffer colour values are written with `glDrawBuffer()`. In addition, four different commands are used to mask the writing of bits to each of the logical frame buffers after all per-fragment operations have been performed: `glIndexMask()`, `glColorMask()`, `glDepthMask()`, and `glStencilMask()`. The operation of the accumulation buffer is controlled with `glAccum()`. Finally, `glClear()` sets every pixel in a specified subset of the buffers to the value specified with `glClearColor()`, `glClearIndex()`, `glClearDepth()`, `glClearStencil()`, or `glClearAccum()`.

Reading or Copying Pixels We can read pixels from the frame buffer into memory, encode them in various ways, and store the encoded result in memory with `glReadPixels()`. In addition, we can copy a rectangle of

pixel values from one region of the frame buffer to another with `glCopyPixels()`. The command `glReadBuffer()` controls from which colour buffer the pixels are read or copied.

Additional OpenGL Commands

Here, we briefly describe special groups of commands that were not explicitly shown as part of OpenGL's processing pipeline. These commands accomplish such diverse tasks as evaluating polynomials, using display lists, and obtaining the values of OpenGL state variables.

Using Evaluators

OpenGL's evaluator commands allow we to use a polynomial mapping to produce vertices, normals, texture coordinates, and colours. These calculated values are then passed on to the pipeline as if they had been directly specified. The evaluator facility is also the basis for the NURBS (Non-Uniform Rational B-Spline) commands, which allow us to define curves and surfaces.

The first step involved in using evaluators is to define the appropriate one- or two-dimensional polynomial mapping using `glMap*()`. The domain values for this map can then be specified and evaluated in one of two ways:

By defining a series of evenly spaced domain values to be mapped using `glMapGrid*()` and then evaluating a rectangular subset of that grid with `EvalMesh*()`. A single point of the grid can be evaluated using `glEvalPoint*()`.

By explicitly specifying a desired domain value as an argument to `glEvalCoord*()`, which evaluates the maps at that value.

Performing Selection and Feedback

Selection, feedback, and rendering are mutually exclusive modes of operation. Rendering is the normal, default mode during which fragments are produced by rasterization; in selection and feedback modes, no fragments are produced and therefore no frame buffer modification occurs. In selection mode, we can determine which primitives would be drawn into some region of a window; in feedback mode, information about primitives that would be rasterized is fed back to the application. We select among these three modes with glRenderMode().

Selection Selection works by returning the current contents of the name stack, which is an array of integer-valued names. We assign the names and build the name stack within the model

Feedback In feedback mode, each primitive that would be rasterized generates a block of values that is copied into the feedback array. We supply this array with glFeedbackBuffer(), which must be called before OpenGL is put into feedback mode. Each block of values begins with a code indicating the primitive type, followed by values that describe the primitive's vertices and associated data. Entries are also written for bitmaps and pixel rectangles. Values are not guaranteed to be written into the feedback array until glRenderMode() is called to take OpenGL out of feedback mode. We can use glPassThrough() to supply a marker that's returned in feedback mode as if it were a primitive.

Using Display Lists

A display list is simply a group of OpenGL commands that has been stored for subsequent execution. The glNewList() command begins the creation of a display list, and glEndList() ends it. With few exceptions, OpenGL commands called between glNewList() and glEndList() are appended to the display list, and optionally executed as well. (The reference page for glNewList() lists the commands that can't be stored and executed from within a display list.) To trigger the execution of a list or set of lists, use glCallList() or glCallLists() and supply the identifying number of a particular list or lists. We can manage the indices used to identify display lists with glGenLists(), glListBase(), and glIsList(). Finally, we can delete a set of display lists with glDeleteLists().

Managing Modes and Execution

The effect of many OpenGL commands depends on whether a particular mode is in effect. We use glEnable() and glDisable() to set such modes and glIsEnabled() to determine whether a particular mode is set.

We can control the execution of previously issued OpenGL commands with glFinish(), which forces all such commands to complete, or glFlush(), which ensures that all such commands will be completed in a finite time.

A particular implementation of OpenGL may allow certain behaviours to be controlled with hints, by using the glHint() command. Possible behaviours are the quality of colour and texture coordinate interpolation, the accuracy of fog calculations, and the sampling quality of antialiased points, lines, or polygons.

Obtaining State Information

OpenGL maintains numerous state variables that affect the behaviour of many commands. Some of these variables have specialized query commands:

```
glGetLight()
glGetMaterial()
glGetClipPlane()
glGetPolygonStipple()
glGetTexEnv()
glGetTexGen()
glGetTexImage()
glGetTexLevelParameter()
glGetTexParameter()
glGetMap()
glGetPixelMap()
```

The value of the other state variables can be obtained with `glGetBooleanv()`, `glGetDoublev()`, `glGetFloatv()`, or `glGetIntegerv()`, as appropriate. The reference page for `glGet*()` explains how to use these commands. Other query commands we might want to use are `glGetError()`, `glGetString()`, and `glIsEnabled()`. Finally, we can save and restore sets of state variables with `glPushAttrib()` and `glPopAttrib()`.

OpenGL Utility Library

The OpenGL Utility Library (GLU) contains several groups of commands that complement the core OpenGL interface by providing support for auxiliary features. Since these utility routines make use of core OpenGL commands, any OpenGL implementation is guaranteed to support the utility routines. Note that the prefix for Utility Library routines is glu rather than gl.

Manipulating Images for Use in Texturing

GLU provides image scaling and automatic mipmapping routines to simplify the specification of texture images. The routine `gluScaleImage()` scales a specified image to an accepted texture size; the resulting image can then be passed to OpenGL as a texture. The automatic mipmapping routines gluBuild1DMipmaps() and gluBuild2DMipmaps() create mipmapped texture images from a specified image and pass them to `glTexImage1D()` and `glTexImage2D()`, respectively.

Transforming Coordinates

Several commonly used matrix transformation routines are provided. We can set up a two-dimensional orthographic viewing region with gluOrtho2D(), a perspective viewing volume using `gluPerspective()`, or a viewing volume that's centred on a specified eyepoint with `gluLookAt()`. Each of these routines creates the desired matrix and applies it to the current matrix using `glMultMatrix()`.

The `gluPickMatrix()` routine simplifies selection by creating a matrix that restricts drawing to a small region of the viewport. If we render the scene again in selection mode after this matrix has been applied, all objects that would be drawn near the cursor will be selected and information about them stored in the selection buffer.

If we need to determine where in the window an object is being drawn, use `gluProject()`, which converts specified coordinates from object coordinates to window coordinates; `gluUnProject()` performs the inverse conversion.

Polygon Tessellation

The polygon tessellation routines triangulate a concave polygon with one or more contours. To use this GLU feature, first create a tessellation object with `gluNewTess()`, and define callback routines that will be used to process the triangles generated by the tessellator (with `gluTessCallBack()`). Then use `gluBeginPolygon()`, `gluTessVertex()`, `gluNextContour()`, and `gluEndPolygon()` to specify the concave polygon to be tessellated. Unneeded tessellation objects can be destroyed with `gluDeleteTess()`.

Rendering Spheres, Cylinders, and Disks

We can render spheres, cylinders, and disks using the GLU quadric routines. To do this, we create a quadric object with `gluNewQuadric()`. To destroy this object when we are done with it, we can use `gluDeleteQuadric()`, then specify the desired rendering style, as listed below, with the appropriate routine:

Whether surface normals should be generated, and if so, whether there should be one normal per vertex, or one normal per face: `gluQuadricNormals()`,

Whether texture coordinates should be generated: `gluQuadricTexture()`,

Which side of the quadric should be considered the outside and which the inside: `gluQuadricOrientation()`,

Whether the quadric should be drawn as a set of polygons, lines, or points: `gluQuadricDrawStyle()`,

After we have specified the rendering style, simply invoke the rendering routine for the desired type of quadric object: gluSphere(), `gluCylinder()`, `gluDisk()`, or `gluPartialDisk()`. If an error occurs during rendering, the error-handling routine we have specified with `gluQuadricCallBack()` is invoked.

NURBS Curves and Surfaces

NURBS (Non-Uniform Rational B-Spline) curves and surfaces are converted to OpenGL evaluators by the routines described in this section. We can create and delete a NURBS object with `gluNewNurbsRenderer()` and `gluDeleteNurbsRenderer()`, and establish an error-handling routine with `gluNurbsCallback()`.

We specify the desired curves and surfaces with different sets of routines like `gluBeginCurve()`, `gluNurbsCurve()`, and `gluEndCurve()` for curves or `gluBeginSurface()`, `gluNurbsSurface()`, and `gluEndSurface()` for surfaces. We can also specify a trimming region, which defines a subset of the NURBS surface domain to be evaluated, thereby allowing we to create surfaces that have smooth boundaries or that contain holes. The trimming routines are `gluBeginTrim()`, `gluPwlCurve()`, `gluNurbsCurve()`, and `gluEndTrim()`.

As with quadric objects, we can control how NURBS curves and surfaces are rendered:

Whether a curve or surface should be discarded if its control polyhedron lies outside the current viewport,

What the maximum length should be (in pixels) of edges of polygons used to render curves and surfaces,

Whether the projection matrix, modelview matrix, and viewport should be taken from the OpenGL server or whether we will supply them explicitly with `gluLoadSamplingMatrices()`.

Use `gluNurbsProperty()` to set these properties, or use the default values. We can query a NURBS object about its rendering style with `gluGetNurbsProperty()`.

Handling Errors

The routine `gluErrorString()` is provided for retrieving an error string that corresponds to an OpenGL or GLU error code. The currently defined OpenGL error codes are described in the `glGetError()` reference

page. The GLU error codes are listed in the `gluErrorString()`, `gluTessCallback()`, `gluQuadricCallback()`, and `gluNurbsCallback()` reference pages. Errors generated by GLX routines are listed in the relevant reference pages for those routines.

3.6 GIMP

Introduction to the GIMP

We needed a software to create the textures for sari. We searched the Internet to find a suitable software for our need. There are many like CorelDraw CorelDraw but we wanted some free software. There are many software available for this like ImageMagic ImageMagic, but we found that GIMP is better than others in terms of simplicity and its variety of features to manipulate the image.

GIMP GIMP is an acronym for GNU Image Manipulation Program. It is a freely distributed program for such tasks as photo retouching, image composition and image authoring.

It has many capabilities. It can be used as a simple paint program, an expert quality photo retouching program, an online batch processing system, a mass production image renderer, an image format converter, etc..

GIMP is expandable and extensible. It is designed to be augmented with plug-ins and extensions to do just about anything. The advanced scripting interface allows everything from the simplest task to the most complex image manipulation procedures to be easily scripted.

GIMP is written and developed under X11 on UNIX platforms. But basically the same code also runs on MS Windows and Mac OS X.

Features and Capabilities

Here, we list main features of GIMP and what we can achieve with it.

Painting GIMP has full suite of painting tools including Brush, Pencil, Airbrush, Clone, etc. It has sub-pixel sampling for all paint tools for high quality anti-aliasing. It is extremely powerful gradient editor and blend tool. It supports custom brushes and patterns.

System It has tile based memory management, so the image size is limited only by available disk space. It can virtually unlimited number of images open at one time.

Advanced Manipulation It gives full alpha channel support. Its Layers and channels, help to create an image in modular way, thus making

it easier to edit it endlessly, without having to disturb previous stage of created image. It allows to Undo/Redo multiple times but it is limited only by disk-space.

Editable text layers It has transformation tools including rotate, scale, shear and flip, selection tools including rectangle, ellipse, free, fuzzy and intelligent, and advanced path tool doing bezier and polygonal selections. It has transformable paths, transformable selections and it allows to quickmask to paint a selection.

Extensible GIMP has a procedural database for calling internal GIMP functions from external programs as in Script-fu, and has advanced scripting capabilities like scheme, Python, or Perl. It has plug-ins which allow for the easy addition of new file formats and new effect filters. GIMP already has over 100 plug-ins available.

Animation GIMP can load and save animations in a convenient frame-as-layer format. It supports MNG, Frame Navigator (in GAP, the GIMP Animation Package), Onion Skin (in GAP, the GIMP Animation Package), or Bluebox (in GAP, the GIMP Animation Package) .

File Handling File formats supported by GIMP are bmp, gif, jpeg, mng, pcx, pdf, png, ps, psd, svg, tiff, tga, xpm, and many others. It can load, display, convert, and save to many file formats. It supports SVG path import/export.

Installation of GIMP on linux as well as on any Windows environment is quite easy. It took around one hour to install and setup it. We used this software to create the textures of Ang, Kath and Padar.

CHAPTER 4

Texture Mapping

Texture is a physical attribute that characterizes all surfaces. It gives information about the roughness and composition of the surface. It also gives information about size, shape and density of the surface of the object. To generate an image of such object showing all the details is an intriguing problem in computer graphics world as the texture is a difficult visual attribute to synthesize.

In the quest for more realistic imagery, one of the most frequent criticisms of early, synthesized raster images was the extreme smoothness of surfaces, they showed no texture, bumps, scratches, dirt, or fingerprints. Reality demands complexity, or at least the appearance of complexity. Texture mapping is a relatively efficient means to create the appearance of complexity without the tedium of modelling and rendering every three-dimensional detail of a surface Heckbert [1986].

Texture mapping is a technique, which increases the visual complexity of a scene without increasing its geometric complexity. The idea is that the rendering system maps an image onto simple scene geometry to make objects look much more realistic than the underlying geometry.

The study of texture mapping is valuable because its methods are applicable throughout the computer graphics and image processing. Geometric mappings are relevant to the modelling of parametric surfaces in CAD and to general two-dimensional image distortions for image restoration and artistic uses. The study of texture filtering leads into the development of space variant filters, which are useful for image processing, artistic effects, depth-of-field simulation, and motion blur Heckbert [1986].

In 1974, Catmull introduced texture mapping and since then a great deal of work has been devoted towards the improvement of the quality of the generated images and the reduction of the computational cost. In general, the problem of texture mapping can be stated as follows: given an arbitrarily curved surface and a texture domain, a (at least locally

inevitable) mapping between these two domains. The texture domain is one-dimensional, two-dimensional or three-dimensional and may include colour, roughness or transparency entries Azariadis and Aspragathos [2000].

When mapping an image onto an object, the colour of the object at each pixel is modified by a corresponding colour from the image. In general, obtaining this colour from the image conceptually requires several steps Heckbert [1989]. The image is normally stored as a sampled array, so a continuous image must first be reconstructed from the samples. Next, the image must first be warped to match any distortion in the projected object being displayed. The warped image is then filtered to remove high frequency components that would lead to aliasing in the final step: sampling again to obtain the desired colour to apply to the pixel being textured Haeberli and Segal [1993].

There are several methods in use for texture mapping and several different approaches have been suggested to achieve high quality in image synthesis. In the next section we describe them as they got developed in years.

Though one strives to add reality to images produced on the screen, it is not that simple. We would like to quote from Firebaugh, "Visual forms - lines, colours, proportions, etc. - are just as capable of articulation, that is, of complex combinations, as words but the laws that govern this sort of articulation are altogether different from the laws of syntax that govern language . . .. They do not present their constituents successively, but simultaneously, so the relations determining a visual structure are grasped in one act of vision." This is the motivation and also inspiration to all researchers in computer graphics field for generating realistic images.

Texture mapping is an important technique for improving the reality of objects rendered on computer. Using texture mapping techniques, the appearance of detail can be added to an object without increasing the amount of geometry needed to model the object. We give here a brief overview of texture mapping basics and some of the methods. Please see Athale and Stifter [2001] for further details.

4.1 What is texture mapping?

Every object in nature has a specific size, shape, colour, density and texture. To generate its image on the computer, one has to consider each of the above characteristics. The characteristics like size and shape of the object are easy to manipulate on the screen than its texture. For example, a stone or a piece of wood has different consistency at each point. It is not smooth at every point, or have definite structure.

A *texture* means a detailed pattern that is repeated many times to tile the plane, or more generally, a multi-dimensional image that is mapped to a multi-dimensional space. *Texture mapping* means the mapping of a function onto a surface in three-dimensional space Heckbert [1986]. The domain of the function can be one, two, or three-dimensional space. This can be represented by an array or a mathematical function. One-dimensional texture can simulate rock layer two-dimensional texture can represent waves, and three-dimensional texture can represent wood. In other words, *texture mapping* is the process of shaping a texture image and applying it to the surface of a geometric primitive.

4.2 Texture Mapping Methods

The basic idea of texture mapping methods is that the rendering system maps an image onto simple scene geometry to make objects look much more realistic than the underlying geometry. We describe here some of the basic methods for texture mapping Bitter [1996].

(Flat) Texture Mapping: Mapping the texture onto the scene objects is done at the end of the rendering process. Once it is determined which pixels are covered by the object on the screen each square Srepresenting one of those pixels is back-projected onto the three dimensional object. Then, in a second transformation, the area on the texture image A corresponding to S is found. The colour of S is assigned to be some average of all pixels in A. In order to find this average a number of re-sampling filters can be used.

Bump Mapping: Flat textures just modify the surface colours, but not the light reflection patterns. Bump maps have for every pixel also a vector or an index to a vector stored, which is used to perturb the surface normal at the corresponding location of the textured object. In the shading stage of the rendering pipeline Phong shading has to be used. Thus for each screen pixel the illumination equation is evaluated, each time with a new normal slightly modified due to the bump map. The effect is that rough surfaces look a lot more realistic most of the time. Along the edges of an object it is still visible that the underlying geometry is actually flat.

Reflection Mapping: This technique requires two passes through the rendering pipeline. First the viewpoint is transformed to the virtual projection centre inside the object O to be textured. Now the scene of all other objects is rendered and the resulting image is stored as texture image of O. In the second pass the scene is rendered with the original viewpoint and the Surface of O will now show the reflection of its environment.

Displacement Mapping: This method is very similar to the bump mapping. Just in this case the vectors represent perturbations of the

normal vectors and the actual surface. So now even the edges of curved objects can appear a lot more complex than the geometric representation. The price for this added realism is the need to apply the texture map in object space before visible surface determination.

MIP mapping (Multo In Parvo): This method stores texture maps of multiple resolutions efficiently. This Store RGB components in three quarters of the MIP map and four times smaller averaged versions of the RGB components in the remaining quarter. It is then repeated recursively until the remaining quarter is only one pixel. Computing the colour for a screen pixel of a texture mapped object is done by:

- finding the area on the texture map affecting the pixel,
- selecting the two resolutions of the MIP map in which four pixel squares cover just a little more or less than the corresponding screen pixel,
- tri-linearly interpolate the MIP map pixel values to get the screen pixel colour.

The researchers have developed various methods for texture mapping since the introduction of the texture mapping methods in 1974 by Catmull. This is done using sophisticated hardware as well using software. For hardware, consider Fang and Chen [2000]; Gelb, Malzbender, and Wu [2000, 2001]. We here concentrate only on texture mapping methods using software.

4.3 Applications of Texture Mapping Methods

Texture Mapping methods are used in variety of applications to improve the quality of rendered images. We here give some of the widely known applications in brief.

In basic texture mapping, an image is applied to a polygon (or some other surface facet) by assigning texture coordinates to the polygon's vertices. These coordinates index a texture image, and are interpolated across the polygon to determine, at each of the polygon's pixels, a texture image value. The result is that some portion of the texture image is mapped onto the polygon when the polygon is viewed on the screen. Typical two-dimensional images in this application are images of bricks or a road surface (in this case the texture image is often repeated across a polygon); a three-dimensional image might represent a block of marble from which objects could be "sculpted".

Projective Texture Mapping: A generalization of this technique projects a texture onto surfaces as if the texture were a projected slide or film. In

this case, the texture coordinates at a vertex are computed as the result of the projection rather than being assigned fixed values. This technique may be used to simulate spotlights as well as the re projection of a photograph of an object back onto that object's geometry.

Projective textures are also useful for simulating shadows. In this case, an image is constructed that represents distances from a light source to surface points nearest the light source. This image can be computed by performing Z-buffering Mayr [2000-01] from the lights point of view and then obtaining the resulting Z-buffer. When the scene is viewed from the eye-point, the distance from the light source to each point on a surface is computed and compared to the corresponding value stored in the texture image. If the values are (nearly) equal, then the point is not in shadow otherwise it is in shadow. This technique should not use MIP-mapping, because filtering must be applied after the shadow comparison is performed.

Image Warping: Image warping Heckbert [1989] may be implemented with texture mapping by defining a correspondence between a uniform polygonal mesh (representing the original image) and a warped mesh (representing the warped image). The warp may be affine (to generate rotations, translations, shearing, and zooms). The points of the warped mesh are assigned to the corresponding texture coordinates of the uniform mesh, and the mesh is texture mapped with the original image. This technique allows interactive image warping which could be easily controlled. The technique can also be used for panning across a large texture image by using a mesh that indexes only a portion of the entire image.

Transparency Mapping: Texture mapping may be used to lay transparent or semi-transparent objects over a scene by representing transparency values in the texture image as well as colour values. This technique is useful for simulating clouds and trees for example, by drawing appropriately textured polygons over a background. The effect is that the background shows through around the edges of the clouds or branches of the trees. Texture map filtering applied to the transparency and colour values automatically leads to soft boundaries between the clouds or trees and the background.

Surface Trimming: This technique is similar to transparency Mapping. This may be used to cut holes out of polygons or perform domain space trimming on curved surfaces. An image of the domain space trimmed regions is generated. As the surface is rendered, its domain space coordinates are used to reference this image. The value stored in the image determines whether the corresponding point on the surface is trimmed or not.

CHAPTER 5

Cloth Modelling

We all think we know what cloth means since we are constantly in contact with it as clothing and in our homes. Because cloth is so much a part of our daily lives we take it for granted—our minds do not question this wonderful substance and the myriad of guises it comes in. It is really very deceptive, simple word but it conveys a lot of information. It is something we can better understand simply using our senses. We do not need any complex theory to understand its structure. we simply have to see the cloth and touch it. We know the saying, "A Picture is worth a thousand words." Same way, feeling the touch of the cloth will tell us things which will need many words to explain.

Figure 5.1 gives a hint of the complex underlying structure of cloth and how much this structure can vary between fabrics. How is cloth formed or made? What gives one kind of cloth a particular set of characteristics not found in another? These questions may not occur to the average person, but when one models and simulates cloth for computer graphics, answers to these questions are of great importance. As one obtains a deeper knowledge of cloth, a whole new world opens up that has many surprises and questions in it.

To the fashion or textile designer, cloth has many nuances in addition to its basic function of covering a body or any object like a table or a chair. Its importance to the fashion designer cannot be overestimated since so often it is said that the inspiration for a garment comes from the cloth. Designers often develop a clothing design in silhouette, much like an architect starts with a set of floor plans. but even then, nature of the cloth used is of prime importance in determining the cutting and shaping of the garment. It was the late, great fashion designer Balenciaga who is credited with saying that "One most never annoy a fabric. Fabrics have their own life and breath like human beings."

Textile or cloth comes in a variety of types or categories, based on how it is constructed. The two most common types for use in apparel are

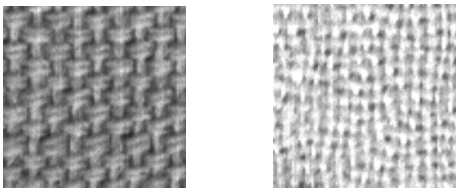


FIGURE 5.1: Magnified cloth samples

woven and *knits*. Based on a number of factors, woven cloth can appear hard or supple or any degree in between. It can be shiny or dull, textured or flat, rough or smooth, compact or open. A cloth can have many layers and still be one fabric. It can have a puckered effect with deep shadows. It can hold a shape so rigid that it can stand by itself or be so drapery that a garment have a distinct woven pattern or a printed pattern or both. It can demonstrate a combination of many qualities or express a quiet, simple idea.

What gives a cloth all of these possibilities? It is a combination of factors that includes the yarns that make up the cloth and how these yarns are interlaced or woven in a pattern called a *weave structure*. When a fabric is being designed, decisions are made as to how firm or loose the fabric will be. The final cloth has a certain *look* and a *hand*. The *look* of the fabric is its appearance to the eye—what we see when we enter a store and are drawn to a certain garment or bolt of fabric. The term *hand* pertains to how the fabric feels as we pick it up and lightly crush it in our hand. "Is it crisp, soft, or limp?" is one of the more common questions a fabric user will ask.

The look and hand do not always tell the same story; for example, sometimes the look on the printed page of a catalogue can be flowing, whereas in the hand the fabric is somewhat brittle. There are also fabrics that look open and airy and yet in hand are extremely heavy. An illustrator depicting a fabric tries to capture not only the look of a fabric but also ideally some indication or hand.

Cloth is woven from *yarns* or *threads*. The words yarn and thread are often used interchangeably, and popular definitions are that yarn is heavier than thread and thread is finer than yarn. To be more precise, yarn is a group of loose fibres or filaments that have been spun or twisted together to form a continuous strand. *Thread* starts out as yarn but is usually twisted with another yarn to give it strength for a specific purpose.

Cloth is woven on a device called a *loom*. The woven cloth is composed of two sets of yarns. On the loom, one set of yarns is placed on the loom and is called the *warp*. During weaving, the warp is held under tension and its yarns are spaced parallel to each. The second set of threads is

FIGURE 5.2: Woven and knitted textiles and detail of a knit

horizontally interlaced into the warp threads. The horizontal threads are called the *weft* or *filling* (sometimes the *woof*, a term that has become archaic today). The final look and hand of the woven fabric is an integration of the characteristics of the warp yarns and the weft yarns, and how they are interlaced. Warp yarns by themselves do not totally control the end product. They need the weft and a weave structure to complete the picture. Thus, the textile designer must make a number of choices, all of which will affect the final cloth.

Knit fabric are complex and has more general structure than woven fabric. Knits are very important to designers and textile industry, being used whenever comfort and insulation are required. Unlike woven textiles, which consist of interlaced weft and warp yarns, knits are constructed by the interleaving of loops. The difference between these fabric types are seen in two samples at the top of the Figure 5.2. The more magnified knit sample at the bottom of the Figure 5.2 clearly shows the interlocking loops inherent to the structure of the knits. Various combinations of these loops can produce an almost infinite variety of designs. Knitted fabrics also drape differently from woven textiles, allowing for the creation of endless new looks. These design considerations, along with the wide variety of knitting modules available, make knits vital to the textile industry, and thus deserving of study. Secondly, the problems specific to knitted materials are much more complex than for woven materials, so they are, therefore, a more general model of textiles. Finally, the micro-structure of knitted fabrics has rather pronounced three dimensional patterns.

What are the differences in a more physical sense between knitted and woven materials? From the representative samples of knitted fabric in the upper right figure Figure 5.2, one can see that for knits we have to deal with a highly anisotropic material. The mass distribution on the

textile varies greatly over different areas of the surface. Moreover, the interleaving loops have an inherent tendency to slip and interact with each other while the fabric is worn. with woven textiles, this type of behaviour is much less evident.

5.1 Cloth from a computer graphics point of view

The name of the fabric usually forms an instant picture in textile designer's mind of the type of the fabric, how it was constructed, and its inherent qualities based on the construction. This is the ideal to strive for in computer animation systems dealing with woven textiles, that this type of information would be embodied in the system. Shapes are easy to get on the computer screen but in life it is the fabrics that will ultimately decide the shapes and volumes of garments.

An animator should be aware of modern textiles that create new problems and also new possibilities for the animator. The current age of techno-textiles means that even for experienced textile designers nothing remains as it started out. Not only are finishing techniques making new and elaborate fabric surfaces, but the use of new yarns is a new learning experience for the textile designer. For example, who would have thought a few years ago that stainless steel would become a new *fibre* that creates a surface unique to itself.

As we said in the beginning, an acquaintanceship with fabric beyond reading is necessary. Simply looking at swatches on the Internet does not help, we have to experience of running our hand over the surface of a velvet or a tussah silk—or picking it up and crushing it in our hands. This experience leads us to understand the cloth scientifically. After this brief overview of cloth, we turn to the modelling of cloth, how textile engineers thought about it and how computer graphics community viewed it. Here, we referred House and Breen [2000] to summarise different cloth modelling methods approaches and how the research in cloth modelling area evolved. We also referred the tutorial on cloth modelling methods, Karthikeyan and Ranganathan, which is a good introduction for the beginners in this field.

5.2 Survey of cloth modelling methods

The research for cloth modelling began around 1930s. In the beginning only the textile engineers were working in this field but around 1980s even computer graphics community started taking interest in it. While the tex-

tile engineers were interested in mechanical properties of the cloth, the computer graphics community was interested in using cloth structures for the computer-generated images and animation. The goals of these two groups were different, and therefore they each focus on different aspects of the same problem. The members of the textile community look at woven cloth as an engineering material. The perspective has led them to measure and model cloth from a mechanical, engineering point of view, and they spent much time on developing devices that measure conventional material properties in cloth. Their modelling efforts were focused on developing models that attempted to explain and predict the highly non-linear behaviour of cloth. Another significant area of textile mechanics research focuses on modelling the micro-mechanical relationships occurring at thread crossings. Around late 1980s, textile community began to apply this fundamental understanding of cloth mechanical behaviour to the problem of the predicting the large-scale shape and structure of cloth objects. By this time, the computer graphics community, motivated by the desire to produce images of ever-increasing complexity had already begun to develop simple models that could produce geometric structures that resembled cloth. The goal of this work was to develop models that reproduce the look of the cloth, within an efficient computational framework. Developers in the computer graphics community were generally interested in creating the simplest model possible that will produce results that appear realistic or acceptable to the average observer. Thus, producing physically accurate and predictive models had never been their goal. They simply wanted their pictures *look right*.

Cloth modelling research as a whole may be placed into three broad categories: modelling the geometric-mechanical structures occurring at yarn crossing, modelling the mechanics of cloth with continuous elastic sheets and rods, and modelling the macroscopic geometric features of cloth. Peirce presented the first yarn-level structural model of cloth. Over the years his model was expanded and enhanced with the addition of new geometric features and force calculations. His basic approach inspired other low-level structural models. Much work has been focused on modelling cloth with continuous elastic structures. This approach attempted to apply elasticity theory to the problem of predicting the deformation of cloth. Another modelling strategy took a more geometrical approach. One example involves modelling the macroscopic geometric features of cloth and relating these features to external forces and conditions. Most recently the computer graphics community has focused on simulating the behaviour of a piece of a cloth, as well as a complete set of clothing, as it interacts with its environment. In following sections, we will review the approaches taken to model the cloth by textile engineers and computer

graphics community.

5.3 Contributions of the textile community

5.3.1 Peirce model

F.T. Peirce was the first pioneer in cloth modelling research. In the mid-1930s, he developed and analysed a basic modelling cell of fabric geometry, dealing with the geometric relationships among yarns at a yarn crossing. The model consisted of two cross-section constrained by a third yarn segment running perpendicular to the cross-sections. The modelling cell would be used to analyse fabric yarn crossings in both the warp and weft direction. Given that the Peirce model was strictly based on the geometric relationships, it could only be applied to a limited set of problems. The model was most useful for determining the weaveability of a particular fabric structure. This involved analysing the *jammed* condition of a fabric, or in other words, the state of maximum yarn packing. This, a fabric can be designed to have a particular yarn density and the Peirce model can help to determine if the density is geometrically possible.

5.3.2 Strain energy methods

Strain energy methods attempted to model the parameters and structures of cloth by creating and minimizing equations that define the strain energy in a fabric. Strain energy methods fall into two categories, low-level structural models and high-level continuum models. They seem to have been developed in response to the complexity and intractability of approaches, based on Peirce model and improved later. The low-level structural models were developed once it was realized that force-based analysis of more realistic forms of the Peirce model was impractical. The continuum based strain energy models were explored once the limitations of applying the conventional theory of elastic plates and shells to fabric mechanics was understood.

Low-level structural models

In the late 1970s, S. De Jong and R. Postle presented an elegant general theory of the elastic behaviour of fabrics that is based on modelling the shape of deforming yarns. They assumed that yarns have simple elastic deformation properties and that any fabric structure consisting of these yarns comes to equilibrium in a minimum strain energy configuration. By focusing on the strain energy of just the yarns, their approach became independent of overall yarn structure, freeing it from special case analysis. They broke down the total strain energy of a yarn into four

components, bending, torsion, lateral compression, and longitudinal tension, based on the curvature, twist, and extension of the yarn geometry. A unit modelling cell was considered that not only contains the yarns but also the constraints on the yarns and other external potentials (e.g., gravity). The total energy of the complete system was then minimized by applying methods from optimal control theory. Lagrange multipliers were introduced for the constraint equations, a Hamiltonian for the system was defined, and the minimization process for the constrained yarn energy was then transformed into a minimization of an unconstrained system. They applied their model to the problem of predicting the load-extension, yarn-decrimping, and bending rigidity properties of several materials and produced reasonable results as compared to actual fabrics.

High-level continuum models

In the late 1980s, J. Amirbayat and J.W.S. Heartle proposed an energy-based method for modelling the large scale deformations of a thin flexible sheet. They highlighted several limitations that arose while applying conventional elasticity-based techniques to modelling. They stated that thin sheet theory was only a collection of special-case analyses derived for specific, simple three dimensional geometries, implying that it was not well suited to modelling the arbitrary and complex geometrise of buckling cloth. Furthermore, more elasticity-based techniques were developed for small strains and small displacements, which could not be assumed for most cloth structure. Finally, in cloth there was no direct connection between its in-plane and out-of-plane mechanical properties, in stark contrast to the assumptions made for continuous sheet and films. Given these limitations, and the assumption that the cloth can be modelled as an isotropic Hookean material, Amirabayat and Heartle developed a strain energy model of a cloth sheet. The strain energy function consisted of five components: the energy due to external normal or frictional forces, the bending energy integrated over single and double curvature zones, the membrane strain energy, the gravitational potential energy, and additional energy terms reflecting the aerodynamic, electrostatic or other forces. They applied their strain energy model to the study of a three-fold *crow's foot* buckling element which they suggested was a fundamental element of more complex buckling and folding configuration. This buckling element was broken in two zones. The central zone was a dome of double curvature, a unique type of deformation found in cloth. Radiating out from this dome were three regions of single curvature. They developed an experimental apparatus that reproduced the *crow's foot* element. It was used to test several sheet materials and to compare their final equilibrium configuration with those generated by the model. They claimed that the measurements produced from their experiment device supported and

validated the basic principles of their strain energy approach.

5.3.3 Elasticity-based methods

Another significant area of research in fabric modelling has been the application of the theory of elasticity, continuum solid mechanics and finite element techniques to modelling the mechanical properties of cloth. In 1963, W.F. Kilby applied elasticity theory to modelling of woven fabrics. He developed planer stress-strain relationships for a simple trellis using a conventional elasticity-based analysis. He assumed that fabric can be modelled with a rectilinear trellis in which the elements are pivoted together at their intersection points, but do not pass under and over one another. For small strains in the plane, he went on to show that the stress-strain relationships are identical with those for an anisotropic elastic lamina that does not display Poisson effect. From his equations he predicted the behaviour of the Young's modulus as a function of angle across a woven cloth's surface. He showed that his theoretical model compiles with a set of experimental data. He admitted though that, in general, the mechanical behaviour of cloth was non-linear and hysteric, but he showed that for small strains the planer behaviour of woven fabric was essentially linear and elastic.

5.4 Contributions of the computer graphics community

In 1980s, computer graphics community became interested in cloth modelling. They focused mainly on the problem of simulating the complex shapes and deformations of fabric and clothing in three dimensions. Their research can be divided into geometric and physically based approaches.

5.4.1 Geometric approaches

In 1986, J. Weil defined a geometric approach that approximates the folds in a constrained piece of square cloth. His approach used a two-step process to model a rectangular cloth structure hanging from several constraint points. The cloth structure was modelled topologically as a two dimensional grid of three dimensional geometric points. The first step recursively connected constraint points, which held up the cloth, with catenary curves as an initial approximation. The curves have the form

$$y = \frac{a}{2}(e^{x/a} + e^{-x/a}) = a\cosh\left(\frac{x}{a}\right)$$

The grid point lying between the constraint points were placed on the three dimensional catenary curves. When two curves cross, but do not intersect at the same point in space, the lower curve is eliminated. New constraint points and catenary curves are then added until all points on the catenary curves lie within the convex hull of the constraint points. The second pass used a relaxation technique to enforce distance constraints between all grid points in order to create smooth cloth-like folds in the rectangular grid. Weil modelled with a second-order distance constraint. He also included a rendering technique where the cloth surface was subsequently modelled as a collection of cylinders.

5.4.2 Physically based approaches

Mass and spring models

In 1998, D.R. Haumann and R.E. Parent simulated simple cloth-like objects within their behavioural test bed. They were interested in determining how complex global phenomena may be synthesized from simple local interaction rules acting over a large collection of interconnected actors. Towards that end, they developed an object-oriented environment with a library of simple physically based actors. Their actors included a point mass, environmental forces, a spring connecting two point masses, a hinge that connects the two triangles formed by four point masses, and aerodynamic drag and wind actors. Polygonal models consisting of triangles can be easily covered into a collection of interconnected actors, by converting each vertex into a point mass, each edge into a spring, and each set of adjacent faces into a hinge.Given initial conditions and environmental forces, animations of complex physically based motion can be calculated by having the actors respond to the resultant forces and torques based on Newton's laws of motion. They dis not claim that their approach accurately models woven cloth, but merely some kind of deformable surface. Nevertheless, using their test bed they created several animations including a flag weaving and curtains blowing in a breeze. A few years later J.A. Thingvold and E. Cohen presented extension to Haumann and Parent's work, by applying similar behavioural actors, not to the vertices of a polygonal model, but to the control points of a B-spline surface. They also developed techniques that allow for subdivision of the surface and the physically based actors in areas of high curvature. Their method produced cloth-like surfaces, but it also suffered from the same deficiencies as Haumann and Parent's when it came to modelling the mechanical behaviour of woven cloth.

In 1995, Provot described a mass and spring system, similar to Haumann and Parent's, which included spring configuration meant specifi-

cally to model cloth. The mass particles, arranged in a rectilinear grid are connected with the three types of springs.These are

1 structural springs that connect nearest-neighbour particles along thread lines,

2 shear springs that connect nearest-neighbour along diagonal, and

3 flexicon springs that connect a particle with its second neighbour along thread lines.

Provot calculated the dynamic behaviour of the springs in order to simulate a cloth hanging from two points, and a weaving flag. A heuristic method, reminiscent of a method developed by D.H. House and D.E. Breen was presented which first takes a step in time based on the dynamics of the system, then adjusts the positions of the particles that have violated local distance constraints. This work is give in the article Provot. The enforcement of distance constraints eliminated the unacceptable elongation generally found near fixed model points in elastic model. Removing the elongations made these kinds of models less stretchy and more cloth-like.

Elasticity-based models

In 1986, C.R. Feynman simulated some of the mechanical properties of cloth defining a set of energy functions over a two dimensional grid of three dimensional points. The total energy of his cloth model contained tensile strain, bending and gravity terms. He minimized the energy of the grid with a stochastic technique, and a multigrid method. Feynman assumed that cloth is a continuous flexible material and derived his energy functions from the theory of elastic shells. His energy functions were only based on the distance between points and a simple measure of curvature. The strain energy was defined as

$$E_s = \frac{E}{1-\nu^2}(u_{xx}^2 - u_{yy}^2) + \frac{2\nu E}{1-\nu^2}u_{xx}u_{yy}$$

where E is Young's modulus, ν is Poisson's ratio and u_{ii} is the strain. The energy of bending was defined as

$$E_b(S) = \int_0^{v_{\max}} \int_0^{u_{\max}} c_1\kappa^2 du dv$$

where κ is the principal curvature of the surface and c_1 is a mechanical stiffness parameter. Because his functions were based on the behaviour of a deforming membrane, they could not maintain tight distance constraints between adjacent points, and yield a very stretchy cloth. Under normal modes, a real draping cloth does not stretch significantly. Feynman's approach did not take into consideration the shearing behaviour of

cloth, as well as the self intersection with arbitrary solid geometric methods. Since he assumed that cloth is a membrane, he also introduced a questionable energy of buckling that attempted to model cloth's differing behaviour under compression and extension.

In 1998, D. Baraff and A. Witkin described a simple cloth continuum model that was motivated more by numerical computing issues than a desire for mechanical accuracy. They presented a computational framework for producing clothing simulations based on an implicit numerical integration method. Their approach produced outstanding animation results while requiring significantly less CPU time. In order to achieve these short computation times, they formulated a simplified cloth model that was based on geometric conditions on a triangular mesh and was straightforward to evaluate. The internal strain energy of the cloth model was defined as a function of a geometric vector function $C(x)$, which was used to maintain soft constraints on inter-vertex distance, shearing within triangles, and bending along triangle edges. The condition was defined to be zero in its minimal state. They presented several exceptional animations of cloth draping over a cylinder, and virtual actors wearing clothing while walking and dancing.

Particle models

In 1992, D.H. House and D.E. Breen developed a non-continuum particle model that explicitly represents the micro-mechanical structure of the cloth via and interacting particle system. Their model was based on the observation that cloth is best described as a mechanism of interacting mechanical parts rather than a continuous substance, and derived its macro-scale dynamic properties from the micro-mechanical interaction between threads. Crossing point of warp and weft threads are represented by particles. These particles interact with adjacent particles and the environment through mechanical connections represented by energy functions. A stochastic gradient descent technique was used to relax the cloth particles toward a final equilibrium position, producing fabric drape. In 1994, they showed how this model can be used to reproduce the drape of a scientific materials accurately, but the model produced only draped configuration without attempting to model cloth motion, and its original implementation was slow and inefficient.

In following sections we elaborate the models suggested by Baraff and Witkin, and the model suggested by Breen and House in detail.

5.5 Continuum model by Baraff and Witkin for rapid dynamic simulation

Although specific details vary (underlying representations, numerical solution methods, collision detection and constraint methods, etc.), there is a deep commonality amongst all the approaches: physically-based cloth simulation is formulated as a time-varying partial differential equation which, after discretisation, is numerically solved as an ordinary differential equation

$$\ddot{x} = M^{-1}(-\frac{\partial E}{\partial x} + F) \tag{5.1}$$

In this equation, the vector x and diagonal matrix M represent the geometric state and mass distribution of the cloth, E–a scalar function of x–yields the cloth's internal energy, and F (a function of x and $\dot{x}$. $\dot{x}$ is the differential of x with respect to time t, and in $\dot{x}$ the dot gives the number of times x is differential.)

The simulator models cloth as a triangular mesh of particles. Given a mesh of n particles, the position in world-space of the ith particle is $x_i \in \mathbb{R}^3$. The geometric state of all the particles is simply $x_i \in \mathbb{R}^{3n}$. A force $f \in \mathbb{R}^{3n}$ acting on the cloth exerts a force f_i on the ith the particle. We capture the rest state of cloth by assigning each particle an unchanging coordinate (u_i, v_i) in the plane.

The most critical forces in the system are the internal cloth forces which impart much of the cloth's characteristic behaviour. Breen et. al. describe the use of the Kawabata system of measurement for realistic determination of the in-plane shearing and out-of-plane bending forces in cloth. These two forces are called as shear and bend forces. Shear force is formulated on a per triangle basis, while the bend force is formulated on a per edge basis—between pairs of adjacent triangles. Stretch force, which is the strongest internal force, resists in-plane stretching or compression, and is also formulated per triangle.

Complementing the above three internal forces are three damping forces. These are used to subdue oscillations due to internal forces. The damping forces do not dissipate energy due to the other modes of the cloth. Additional forces include air-drag, gravity, and user-generated mouse-forces (for interactive simulations). Cloth/cloth contacts generate strong repulsive linear-spring forces between cloth particles.

Combining all forces into a net force vector f, the acceleration $\ddot{x}_i$ of the ith particle is simply $\ddot{x}_i = \frac{f_i}{m_i}$, where m_i is the ith particle's mass and in $\ddot{x}_i$ the dots give the number of times x_i is differential of x_i, here, wrt t. The mass m_i is determined by summing one third the mass of all triangles containing the ith particle. (A triangle's mass is the product

of the cloth s density and the triangle s fixed area in the uv coordinate system.) Defining the diagonal mass matrix $M \in \mathbb{R}^{3n\times 3n}$ by $diag(M) = (m_1, m_1, m_1, m_2, m_2, m_2, \ldots m_n, m_n, m_n)$, we can write that

$$\ddot{x} = M^{-1} f(x, \dot{x}) \tag{5.2}$$

and we can calculate $\ddot{x}$ iteratively using implicit integration method.

Implicit Integration

Given the known position $x(t_0)$ and velocity $\dot{x}(t_0)$ of the system at time t_0, our goal is to determine a new position $x(t_0 + h)$ and velocity $\dot{x}(t_0 + h)$ at time $t_0 + h$. This requires solution of non-linear equation and its input are $x(t_0)$, $v(t) = \dot{x}(t_0)$, $f(t_0)$, $\frac{\partial f}{\partial x}$, $\frac{\partial f}{\partial v}$. Essentially, we have to keep track of x and v, find f and derivatives $\frac{\partial f}{\partial x}$, $\frac{\partial f}{\partial v}$ and apply implicit integration. This gives unbanded sparse linear system which is then solved using conjugate gradient iterative method.

Forces

We define internal behaviour by formulating a vector condition $C(x)$ which we want to be zero, and then by defining the associated energy as

$$E_c = (k/2)C(x)^T C(x) \tag{5.3}$$

where k is stiffness constant. Thus, force is then defined in terms of energy

$$f_i = -\frac{\partial E_c}{\partial x_i} = -k\frac{\partial C(x)}{\partial x_i} C(x) \tag{5.4}$$

We can also calculate the second derivative of f. $\frac{\partial f}{\partial v}$ is zero.

We describe here the main dominating forces in brief.

Stretch - We imagine a continuous mapping function $w(u, v)$ which maps from rest space (u, v) to 3D world space. Derivatives of w represent stretch. Let $w_u = \frac{\partial w}{\partial u}$, $w_v = \frac{\partial w}{\partial v}$. Magnitude of w_u describes stretch/compression in u direction. If $\|w_u\| = 1$, then there is no stretch. We can approximate $w(u_v)$ as linear function over each triangle. So, w_u and w_v are constant over triangle. We can construct condition $C(x)$ for stretch energy

$$C(x) = a \begin{pmatrix} \|w_u(x)\| - b_u \\ \|w_v(x)\| - b_v \end{pmatrix} \tag{5.5}$$

Vector condition $C(x)$ is zero when no stretch occurs. a is area of triangle (u, v) space. $b_u = b_v = 1$.

Shear - We can assume low stretch ($\|w_u\|, \|w_v\| \approx 1$). We can assume low shear (small angle approximately) and construct another condition

$$C(x) = aw_x(x)^T w_v(x) \tag{5.6}$$

This is essentially just inner product of world-space u and v axes.

Bend - Given two adjacent triangles, we calculate the angle θ between them and construct another condition

$$C(x) = \theta \tag{5.7}$$

Assuming negligible stretch, we can formulate this as linear equation of particle positions.

Damping - We define a damping force from $C(x)$

$$d = -k_d(\frac{\partial C(x)}{\partial x})\dot{C}(x) \tag{5.8}$$

This is similar to the definition of regular force

$$f = -k_s(\frac{\partial C(x)}{\partial x})C(x) \tag{5.9}$$

We can calculate $\frac{\partial d}{\partial x}$ and $\frac{\partial d}{\partial v}$

Thus, we have got condition $C(x)$ for stretch and shear of each triangle, and condition $C(x)$ for bend of each edge. We calculate the derivatives to get forces and then apply implicit integration.

Constraints

We can also impose constraints on individual cloth particles. The constraints wither automatically determined by the user such as geometric attachment constraints on a particle or contact constraints generated by the system between a solid object and a particle. At any given step of the simulation, a cloth particle is either completely unconstrained (though subject to forces), or the particle may be constrained in either one, two or three dimensions. If the particle is constrained in all three directions, then we are explicitly setting the particle's velocity (at the next step). If the constraint is in two or one dimensions, we are constraining the particle's velocity along either two or one mutually orthogonal axes.

Mass Modification

A dynamic simulation usually requires knowledge of the *inverse* mass of objects. In the case of a single particle, we write $\ddot{x}_i = \frac{1}{m_i} f_i$ to describe a particle's acceleration. When inverse mass is used, it becomes trivial to enforce constraints by altering the mass.

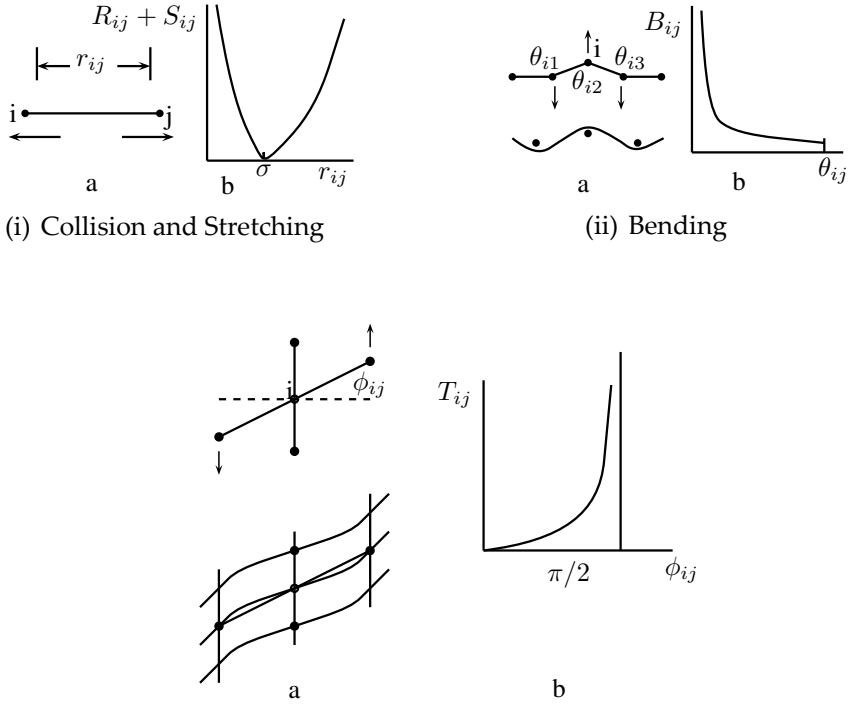
This describes the physically-based cloth simulation method given by Baraff and Witkin and its basics like forces and constraints acting on the particles of the cloth. We here did not elaborate the full paper nor we describe the modified conjugate gradient method or Collisions or adaptive time stepping from the original paper Baraff and Witkin [1998].

5.6 Particle based approach by Breen and House

The particle-based approach to cloth modelling was first applied to the problem of computing the static drape. A piece of cloth is modelled as a two-dimensional array of particles conceptually representing the crossing points of warm and weft yarns in a plain weave. The various inter-crossing strain energies are represented with energy functions parametrised by simple geometric relationships among particles. These energy functions account for the four basic mechanical interactions of yarn collision, yarn stretching, out-of-plane bending, and trellising (in-plane bending) that are show in Figure 5.3. The model does not consider twisting strain, although it could be easily extended to include this. The strain energy for crossing particle i is given by

$$U_i = U_{repel_i} + U_{stretch_i} + U_{bend_i} + U_{trellis_i}.$$

U_{repel_i} is an artificial energy of repulsion that effectively keeps every other particle at a minimum distance, providing some measure of yarn collision detection, and helping prevent cloth self-interaction. The other three terms represent true strain energies. $U_{stretch_i}$ captures the energy of tensile strain between each particle and its four-connected neighbours. U_{bend_i} is the energy due to yarns bending out of the local plane of the cloth, and $U_{trellis_i}$ is the energy due to bending around a yarn crossing in the plane. Repelling and stretching are functions only of interparticle distance r_{ij} (Figure 5.3Ia), whereas bending and trellising are functions of various angular relationships among segments joining particles (Figure 5.3(IIa and IIIa). Trellising occurs when yarns are held fast at a crossing and bend to create an "S-curve" in the local plane of the cloth, and will be seen macroscopically as shearing.



(i) Collision and Stretching

(ii) Bending

(iii) Trellising

FIGURE 5.3: Cloth model energy functions

The function U_{repel_i} prevents collision and self-intersection. so it is calculated by summing over all particles, as given by

$$U_{repel_i} = \sum_{j \neq i} R(r_{ij}),$$

In this simulation algorithm, a spatial enumeration is maintained so that the summation needs to be done only near to the particle i. An energy well is produced by directly coupling each particle with the stretching function S only to its four-connected neighbours, as given by

$$U_{stretch_i} = \sum_{j \in N_i} S(r_{ij}),$$

where N_i is the set of particle i's four connected neighbours.

A unit of the bending energy $B$is defined as shown in Figure 5.3(IIb) as a function of the angle formed by three particles along a weft or warp "thread line", as shown in Figure 5.3(IIa). The complete bending energy is

$$U_{bend_i} = \sum_{j \in M_i} B(\theta_{ij}),$$

where M_i is the set of six angles θ_{ij} formed by the segments connecting particle i and its eight nearest horizontal and vertical neighbours. This definition is used so that the spatial derivatives of bending energy reflect the total change in bending energy due to a change in position of particle i. The redundancy in this formulation is taken care by proper scaling in later stage of the algorithm. The phenomenon of trellising energy is diagrammed in Figure 5.3 and a corresponding unit of the trellising energy T is shown in Figure 5.3. Two segments are formed by connecting the two pairs of neighbouring particles surrounding a central particle. An equilibrium crossing angle of 90°is assumed, but one could model slippage by allowing this angle to change, over the course of a simulation, as a function of load. the trellis angle ϕ is then defined as the angle formed as one of the line segments moves away from this equilibrium. The complete function for the energy of trellising is

$$U_{trellis_i} = \sum_{j \in K_i} T(\phi_{ij}),$$

where K_i is the set of four trellising angles ϕ_{ij} formed around the four-connected neighbours of particle i. As with bending, this redundant formulation was chosen so that the change in total energy with change in particle's position is completely accounted for locally.

This simulation of this particle-based model takes place in three- phase process operating over a series of small discrete time steps.

The first phase for a single time step calculates the dynamics of each particle as if it were falling under gravity in a viscous medium and accounts for collisions between particles and surrounding geometry.

The second phase performs as energy minimization to enforce interparticle constraints. A stochastic element of the energy minimization algorithm serves both to avoid local minima and to perturb the particle grid, producing a more natural asymptotic final configuration.

The third phase corrects the velocity of each particle to account for movements of particles during the second phase.

Use of the energy functions have made this particle-based cloth model robust as these energy functions met reasonable boundary conditions, reaching minima at the right places, and having asymptotes at the right places.

It is assumed that the yarns in the fabric do not stretch significantly when a cloth is simply draping under its own weight. Therefore, the combined stretching and repelling energy function $R + S$ shown in figure

Figure 5.3(Ib) provides a steep energy well that acts to constrain each particle tightly to the normal distance σ from each of its four-connected neighbours. The energy functions

$$R(r_{ij}) = \begin{cases} C_0[(\sigma - r_{ij})^5 / r_{ij}] & r_{ij} \leq \sigma \\ 0 & r_{ij} > \sigma, \end{cases}$$

and

$$S(r_{ij}) = \begin{cases} 0 & r_{ij} \leq \sigma \\ C_0[((r_{ij} - \sigma)/\sigma)^5] & r_{ij} > \sigma, \end{cases}$$

where C_0 is a scale parameter, provide better results.

The bending energy function should be at a minimum with the cloth completely flat (i.e., when $\theta = \pi$) and become arbitrarily large when the cloth bends entirely back on itself (i.e., when $\theta = 0$. The function

$$B(\theta_{ij} = C_1 \tan(\pi - \theta_{ij}/2,$$

where C_i is a scale parameter, meets these conditions. Likewise, the trellising energy function should be at a minimum when crossing yarns are perpendicular to each other (i.e. when $\phi = \pi/2$). The function

$$T(\phi_{ij}) = C_2 tan(\phi_{ij}),$$

where C_2 is a scale parameter, meets these conditions.

CHAPTER 6

Representing sari using computer-generated patterns

We have discussed the problem of representation of sari in second chapter. In this chapter, we explain our method to divide the sari symbolically into fours parts and then use these parts to simulate it on the computer screen. We then discuss the cloth modelling method suitable for sari simulation. We also discuss the collision detection problem, which needs special attention from the sari simulation point of view. The books Mortenson [1985, 1999] were useful for understanding the mathematical concepts in cloth modelling methods. Also the books Foley and van Dam [1984]; Foley et al. [1994] were useful to understand the Computer Graphics concepts and the lecture notes Cok [2000] were very useful to design a system to represent the sari on computer screen. The lectures notes Stifter [2001, 2002] were useful for devising a strategy for our work.

6.1 Symbolic representation of sari

To model a sari, we should know how a sari is created? What kind of structure the fabric of the sari has? How its design patterns are created? How this affects the appearance and behaviour, like at the creases and folds, or when it is draped over a body? How can the variety of draping styles of the sari be simulated on computer screen? There are many such questions and so we break the main sari simulation problems in small parts. Let us first concentrate on how a sari is created.

A sari is a rectangular, un-stitched piece of fabric, woven using cotton, silk or other threads, usually five meters long in length and around one to one and half meters in width, usually having designed length-wise borders and one breadth-wise designed end of the sari. Figure 6.1 shows a plain sari hanging from a cloth-line. Here, we can see all symbolic parts

FIGURE 6.1: Sari with all its symbolic parts

FIGURE 6.2: Banarasi Sari

of sari. This sari is made up of *Crape* material, taken from the web site Ananda.

See the difference between texture in the above sari and the Paithani shown in the Figure 6.2.

We want to see sari as plain sheet of the fabric too. Picture of a full-length, that is, at least five meters long sari will be uninteresting as it will not reveal any detailed information. So we have to decide how a sari can represented so that it will show all the necessary details. Figure 6.3 shows sari in its rest position, that is at a flat angle. Figure 6.4 shows the Padar of the sari, in more detailed form. Figure 6.5 shows the Kath of the sari. Note that this sari has very small and plain Kath.

In second chapter, we explained a sari, using its symbolic parts. Though, it is an un-stitched fabric, for the sake of clarification, we differentiated it in its four visual parts, symbolically. A sari follows a definite pattern of designs. Irrespective of its material, whether it cotton or silk or made up of using any synthetic threads, or weaving styles, or any traditional design patterns, every sari has definite pattern of designs. In all the variety of the saris, there is a fixed set of patterns, let us call them parameters. We identify the parameters, common to all saris, and which

FIGURE 6.3: Sari

parameters which change with the sari.

A sari consists of Ang. This is common parameter in every sari. This is sort of basic fabric of the sari. This is present in every sari and the texture of the threads used for weaving and the colour(s) define the plain sari. Here all the other visual parts, like Kath or Padar are absent. The Ang can be plain or it can have any design, created while weaving.

A sari can have lower Kath only, or it can have both. A sari will not have upper Kath if it does not have lower Kath. The saris like Paithani have upper Kath but since this upper Kath is tucked inside the petticoat and is not visible when the sari is worn, many saris do not have the upper Kath. It is more for sake of completion of the design.

Most beautiful and prominently visible part of the sari is its Padar. A sari can have only Padar or it can appear with the lower Kath, or with both of them.

We created similar textures of the symbolic parts of the above sari, using *GIMP*, version 2.0. Then we fixed the proportion of these parts of the sari. We then used texture mapping to generate a sari with all possibilities of combination of Ang, Kath and Padar. Figure 6.6 shows the texture of the Ang of the sari, Figure 6.7 shows the Padar and Figure 6.8

FIGURE 6.4: Padar of the sari

shows the Kath of the sari. We used these textures and generated a sari on a computer screen, please see Figure 6.9.

The pictures of the saris used here are from our personal collection, and also from the web sites Ananda; Banarasi Saree; IndiCraft; Sari Safari; Tourism of India.

6.2 Problems related to sari simulation

We discussed different approaches for cloth modelling in previous chapter. Here, let us discuss the cloth modelling methods particularly for modelling of the sari.

When we take a sari in our hands or wear it, it feels different. The same sari with different folding looks different, or even it looks different when worn by different persons. There is no rule how it should look, actually, this is true with any kind of clothing but it is especially true with saris as sari comes in same size and shape for any person unlike the stitched garments which are prepared according to the body measurements of the wearer. So if it looks right to our eyes then it is right. Thus, if the

FIGURE 6.5: Kath of the sari

behaviour of the sari we modelled is same or almost same as of the real one, then we have right model. Now the questions arise like what we exactly wish. We sometimes need the sari draped over a body. Sometimes we might need only main parts of the sari like Padar, Kath and Ang. We might need just to see if the sari is hung from some point then how its creases will look like. In short, sometimes, we will need the exact model of the sari,

In cloth modelling, the phenomenon under consideration are fundamentally chaotic. Each time one puts on a shirt or drapes a tablecloth, many of its details look different. Given this fact, the computer graphics dictum "if it looks right it is right" seems somehow to be a very powerful one. How can an engineer know if a model of cloth is correct? Our answer from computer graphics would be to visualize its performance and compare results with those obtained with real cloth. The criteria for comparison, at least at the first level of analysis, can and should be visual. does it drape like the real material? Does it move like the real material? Then at some fundamental level, the model is likely to be a correct description of the material. The point of view seems first to be largely antithetical to the point of view required by an engineer, responsible for

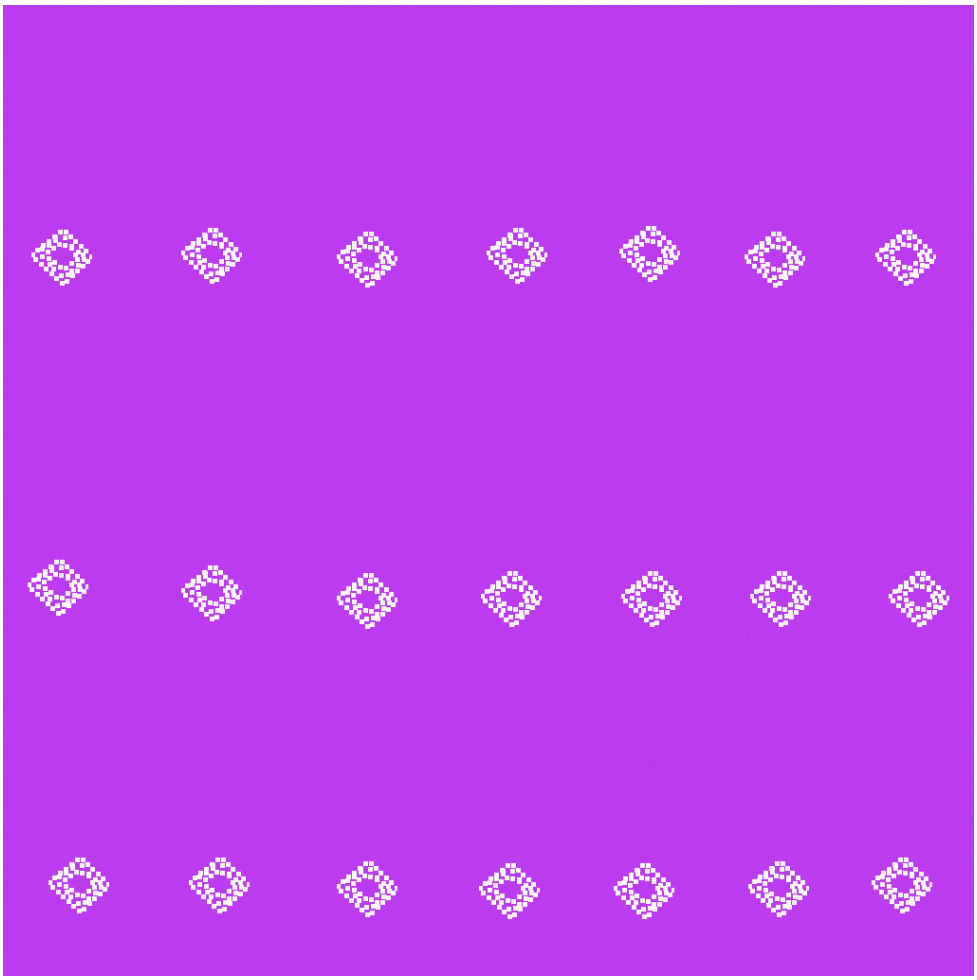


FIGURE 6.6: Ang of the sari.

design. However, it does not preclude deeper investigations but simply supplements it with an approach that taps directly into the immense power of human brain and visual system to extract patterns from visual phenomena.

To simulate a sari, we first have to decide a suitable model. We give a brief overview of models suggested so far since the beginning of this research field, with detailed description of the continuum model suggested by Baraff and Witkin [1998] and the particle model suggest by House and Breen [2000]. The first model, suggested by Baraff and Witkin [1998], is best suitable for the dynamic simulation of cloth, with compromising the mechanical accuracy. This model is helpful mainly when the numerical computing issues are more important then the accurate simulation of the cloth. The second model, suggested by Baraff and Witkin [1998], is non-continuum particle model for cloth drape that explicitly represents the

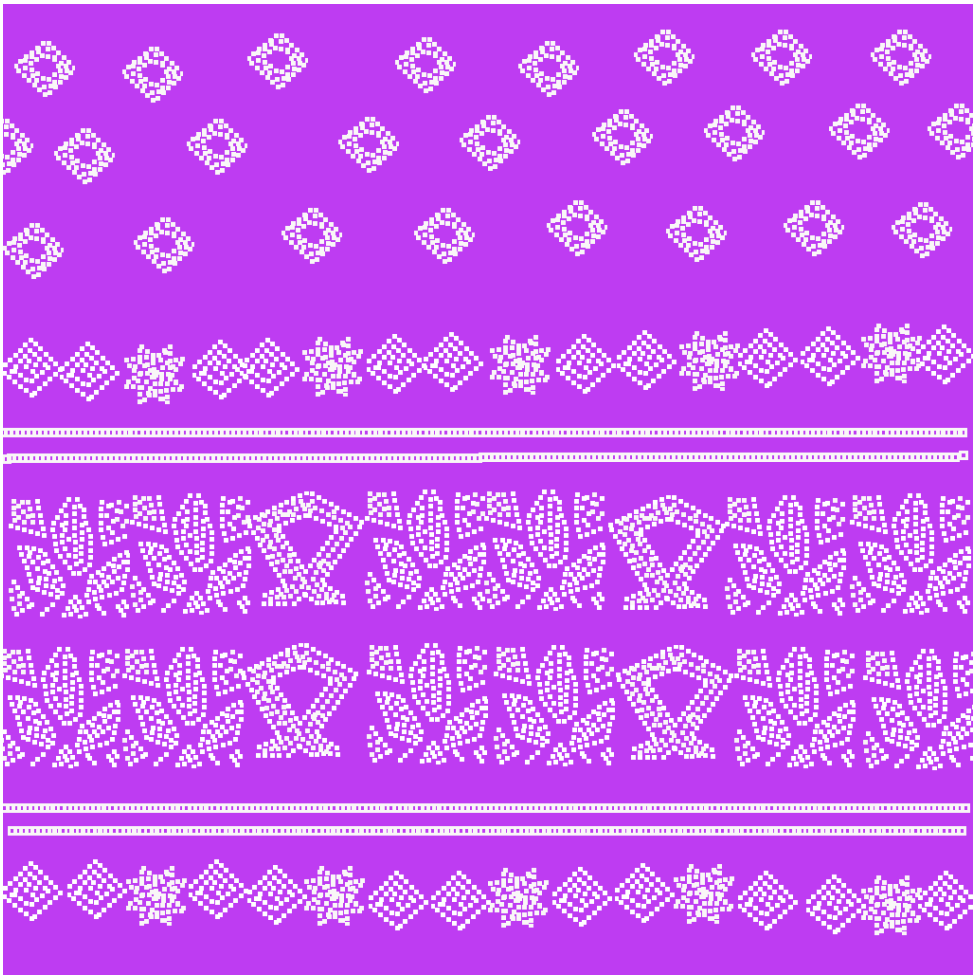


FIGURE 6.7: Padar of the sari.

FIGURE 6.8: Kath of the sari.

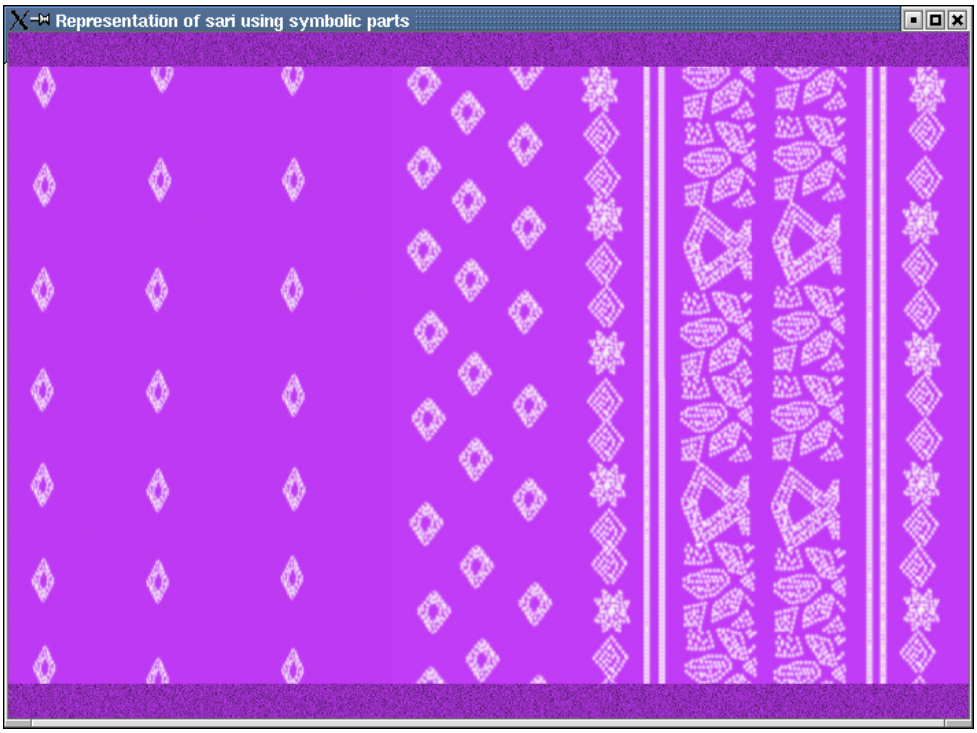


FIGURE 6.9: Sari generated using its symbolic parts.

micro-mechanical structure of the cloth via an interacting particle system. The particle model can be used to reproduce the drape of a specific material accurately, although it can produce only the draped configuration and not the cloth motion. The later variations of this method are faster and more efficient than the original version but this method is not that fast enough. For sari, we need a model which can produce a scientifically accurate simulation of cloth, that is in our interest, sari, in real-time, or nearest to it.

Saris are of various materials, like cotton, silk, or even synthetic threads. They are either woven or knitted, depending upon the type and the tradition of that sari, for example, Paithani, is woven with silk threads and Jar while the printed cotton saris are knitted with variety of different coloured threads, sometimes even synthetic threads are also used. Thus, to generate a sari, accurate visualization of the underlying yarn is very important. This holds true for the woven fabrics but it is especially true for knitted fabrics, where the individual yarns and the yarn-loops contribute significantly to the overall appearance. Without considering the fibres and the structure of the material, realistic-looking close-up images cannot be generated. Thus, the modelling of the fine and highly detailed structures of yarn and its fibres is a very challenging task.

Thus, to select a suitable model for sari simulation comes to selecting a suitable model for lengthy woven or knitted fabric.

6.2.1 A volumetric appearance model proposed by Meissner, Eberhardt and Strasser

The requirements for woven as well as knitted fabrics are very different, hence we have to consider the constraints and the target application of the fabrics carefully. Here, we describe *a volumetric appearance model* suggested by *Meissner, Eberhardt* and *Strasser*, House and Breen [2000]. While woven fabrics are usual made using a regular weave structure, knitted fabrics can consist of simple up to very complex knitting patterns. The more complex the patterns are, the more effort has to be spent to model the fabric correctly. Both the complexity of the fabric and the required level of detail of the yarn determine the necessary computations. Depending on the level of the detail required, it might be possible to use a simple yarn representation like a tube, or it might be necessary to go to a more detailed volumetric representation.

Saris, may woven or knitted, follow a repetitive structure of patterns. For example, traditional saris, like Paithani or Banarasi Sari are well known for their traditional designs of peacock or objects like Koyari. Or the printed saris have block prints, where the blocks are printed linearly on the basic fabric of the sari, using same or different colours. Here, we discuss why and how the volumetric appearance model will be suitable for the sari simulation. We use the terminology used by the authors, that is, we stick to the woven fabrics and knitted fabrics.

Simple knitted fabric patterns Simple knitted fabric patterns are in principle made up of only two types of loops, R-loops, or plain stitch, and L-loop, or reverse stitch, and therefore have a highly repetitive structure. In order to generate a realistic appearance of such knitwear, it is necessary to model the detailed structure. Simple texture mapping could be used but it is limited to a surface representation, while knitwear has a three dimensional structure that influences its appearance. Furthermore, the very fine details of the yarns, including hairiness, have volumetric properties that can only be partially captured using two dimensional textures. To achieve an accurate appearance, *Meissner, Eberhardt* and *Strasser*, House and Breen [2000], used volumetric models of the yarns that take advantage of repetitiveness by generating only a small portion of the cloth, that is, a module, which then can be patched together repeatedly. This will be especially useful for the sari, as we can find the repetitive pattern of the

designs in Kath and Padar and then patch them together, along the length and width of the fabric of the sari, as per the sari's type.

Unlike woven textiles, which consist of interlacing weft and warp yarns, knits are constructed by interleaving of loops. The differences between these fabric types are clearly seen in the two samples in New combinations of these loops can produce an almost infinite variety of designs. Knitted fabrics also drape differently from woven textiles, allowing for the creation of endless new looks. These design considerations, along with the wide variety of knitting modules available, make knits vital to the textile industry. Also, the specific problems for knitted materials are much more complex than for woven materials and are, therefore, a more general model of textiles. Finally, the micro-structure of knitted fabrics is a rather pronounced three dimensional micro-shape.

To begin with, we first have to construct a pattern of the knitted fabric, row by row, with each row consisting of a series of consecutive loops. To construct these loops, for each loop of the previous row a new loop on the current row is built by pulling the knitting yarn through the loop of the previously knitted row. This is either done from front-to-back, or L-loop, or back-to-front, or R-loop, as shown in Note that the type of loop is view dependent definition, the front view of an R-loop is the back view of an L-loop and vice-versa.

The knit pattern are given by the geometry of the thread structure and an abstract representation. This abstract representation tells us how R-loops and L-loops are combined together in the knit pattern. The basic elements needed for to model the simple patterns of knitted fabrics are fairly small while the more general knitting patterns are specified by an arbitrary arrangement of R-loops and L-loops, and there are many more knitting patterns that consist of more than two types of loops.

Due to the repetitive structure of patterns, a fabric can be subdivided into basic elements that repeat across the fabric. To describe any pattern, one basic element has to be generated for each kind of loop, these elements are then patched together to generate a larger representation. Thus, modelling a knitted fabric pattern is reduced to modelling the structure of knitting yarns within the basic elements. In a subsequent step, the knitted fabric pattern can then be synthesized from these basic elements by taking into account various boundary conditions that must be fulfilled.

The location of the knitting yarn within a basic element is define by four components C_1, C_2, C_3, C_4 of a parametrised $3D$ skeleton curve $C(t)$ The four pieces together define a single knitting loop and are parametrised as indicated in the A curve-length parametrisation is done such that the following conditions hold:

$$C(t_i) = p_i, t_0 = 0, t_1 = t_2, t_3 = 0.5, t_4 = t_5, t_6 = 1.0, \text{ with } t_i \leq t_{i+1}. \quad (6.1)$$

The parametrisation of the skeleton curve affects the properties of the yarn structure, for example, twisting behaviour, and must be chosen to ensure continuous transitions of the yarn and its micro-structure between adjacent basic elements. The skeleton curve C has various symmetry characteristics: C_1 is basically a rotated version of c_2 and c_3 is a rotated version of C_4. Reflecting C_2 through a plane parallel to the yz-plane produces C_3. The same holds for C_1 and C_4. Furthermore, the skeleton curve of a R-loop is the same as the reflection (through xy-plane) of the skeleton curve of an L-loop. The skeleton curve C determines the location of the thread course.

Modelling of the yarn Micro-structure Knitting yarn typically consists of a large number of thin fibres. A fibre may be of different materials like wool, cotton, silk, or nylon and it has a much greater length than thickness.

For realistic representation of knitted fabrics, an approach is described by emphasizing their rendering in detail. The yarn micro-structure is modelled as 3D-data to allow a close-up inspection of the model. Other cloth shading models found in the literature are based on 2D texture mapping to create a textile look. Typically this technique is based on scanning real textile materials and applying the scanned images as texture maps.

Knitting yarn has a fine micro-structure, where single fibres are not perceivable by themselves, but provide an important contribution to the overall visual impression of the yarn structure. Rendering primitives are volume densities with anisotropic lighting behaviour. Instead of an explicit representation of individual yarns, volume data sets are used to represent collections of fibres simultaneously. Rendering time is therefore independent from the geometric complexity of the yarn structure.

A yarn has a complex micro-structure due to the large number of thin constituent fibres. As the diameter of a single fibre is merely micrometers in width, it has a very low opacity and is not perceivable individually. Only the collection of fibres and their spatial arrangement determines the visual impression of a yarn. A geometric model with the representation of each single fibre was considered to be too costly and unnecessarily detailed for the generation of the visual appearance the yarn structure. Additionally, severe aliasing problems must be expected when using a geometric model for representing the micro-structure of knitting yarn.

In light of these considerations, the yarn is modelled as a volume data set. A density value thereby reflects the frequency of fibres in a certain region of a basic element which makes up a knitted fabric. The volume data of a basic element is generated by specifying a 2D cross-section of a

knitting yarn. This cross-section is swept along the skeleton curve, and one or more rotations produce the twisted shape of the yarn. Density values correspond to fibre frequency, i.e., high density values correspond to locations with a large number of fibres. A yarn cross-section usually consists of one or more circular regions of high density where most of the fibres are located. Density values drop off at the boundary of the circular regions due to the fact that fibres are less numerous in these regions. Scattered spots of high density correspond to fibres or bundles of fibres which are detached from the main strand of the yarn. They are essential for the fleecy and soft appearance of a knitting yarn.

These yarn cross-sections are swept and rotated along the earlier determined polygonal lines of the threads. By blending together these cross-sections the final volume of the knitting pattern is obtained. For rendering, one may use direct volume visualization.

6.2.2 Volumetric approach for sari simulation

We saw that sari is knitted or woven fabric, made up of from variety of materials and worn around the body in layers. To achieve the realistic sari simulation we need very realistic and high-quality images of sari. We suggest that the volumetric approach described above will be best suitable for it. Since, saris are designed with repetition of some pattern and their distinct looks are because of the yarns used for the weaving, or knitting, the volumetric approach will be best suited for the sari simulation.

We used the polygonal surfaces and texture mapping methods to generate the sari but this approach has limitations. This texture mapping method does not incorporate the underlying detailed structure of fibres. This is mainly because of the interaction of light being strongly dependent on yarn and fibres that absorb light depending on the density distribution of the fibres. Using polygon texture-mapping and local illumination properties (per triangle based), this could only be simulated in a very limited way, since material properties vary from yarn to yarn depending on the material used and the applied knitting pattern. In contrast, a volumetric approach reveals the three-dimensional structure of knitting, especially if one looks at a knitted fabric at a very flat angle. This will in fact help to depict the sari better in its static form, that is before it is worn.

We have described the generation of volumetric models from basic elements of single yarn-loops forming a regular knitting pattern.This approach can be extended with complex patterns with enriched designed saris, like Paithani.Extending the above concept to more complex patterns can be accomplished with a closer look at the basic elements. More complicated knitting patterns are a combination of basic elements and usually consist of more than one or two basic yarn loops extending over several

rows. An example of such a larger basic elements is a *cable*, where yarn loops cross each other and, hence, only the entire cable can be used as a basic element, which can then again be repetitively fit together if necessary. The 3D curve of the yarns for such a basic module needs to be known before the technique described here can be applied. Thus, to generate a sari using its symbolic parts, as explained in previous section, we can use this volumetric approach provided we have the detailed description of the underlying structure of the yarns and fibres of the fabric of the sari.

6.2.3 Collision Detection

In the second chapter, we described the steps to drape a sari. The sari wearing consists of draping it over the body, in a respective style, in several layers. From simulation point of view, it causes a bottleneck problem, that is of collision—collision of the fabric with the three dimensional surface resembling the human body and the self-intersection, that is collision of the cloth particles from one layer of fabric to another layer of the fabric. Because of layers, self-collision of cloth particles is a very important issue. The collision with other surface than the cloth itself, is the body around which the sari is draped. Cloth collision and detection is very important issue for cloth modelling and simulation research area. House and Breen [2000] gives a detailed summary of available methods for this.

The problem of collision detection is quite difficult due to the complex configuration that limp cloth fabric can fall into, making the issue of self collision one that cannot be ignored for any practical animation application. In 1995, Volino, Courchesne and Magnenat Thalman presented an algorithm that deals efficiently with this and other collision detection and response issues. Self intersection calculation is made tractable through the use of a hierarchical algorithm utilizing surface curvature. Calculation times resulting from the use of this algorithm are roughly proportional to the number of colliding elements and independent of the total number of elements in the surface. Collision consistency, which is the problem of keeping all the elements of a complex surface on the appropriate side of another complex surface during collision response, is handled with heuristics and a history mechanism.

CHAPTER 7

Applications

Sari simulation have many interesting applications. We will list them here. First is in e-Commerce. The web shops like Ananda can use the our programs like and their sari patterns and generate the saris on screen. For example, in kolkata saris, the user can choose from various coloured Ang for sari and the Kath and Padar and see how it looks like together. Keeping photos of each such combination will take lot of space and also the web pages will need more time to load. In our system, one can choose from available options and then select his choice of sari. Also he can see how it will look like when it is worn, or when it is hanging from a cloth-line and showing folds and creases showing colour changes at plaits and Padar. Here, we have shown only one sari but we can create many textures and provide them to the user and see how he likes it.

Animated figures, like characters in animation films, need clothing. Most of the animated characters wear the skin-tight clothes like shirts or skirts. The sari, Indian traditional wear for women is quiet different. It is worn around the body with some other under garments. So these kind of non-skin-tight clothing moves independently of the wearer. These depicts complicated movement with many wrinkles and creases. Indian culture getting popular worldwide and more animation films are created based on those stories, we will need the Indian clothing for them. Thus, sari simulation will indeed help to attract worldwide viewers to Indian animation films.

Computer games is another good candidate which will help from the sari simulation research. The strategic games, like based on stories like Harry Potter, need long robes flowing over the body. The more realistic looks surely add to joy of playing the game.

Textile industry is the most important field which will benefit from sari simulation results. India has a great tradition of various saris. Every state, even different towns have different designs of saris and different draping styles. Also the new development in textile industries give rise

to new varieties, for example, the new printing technology allows to have same design as any Bandhani sari will have without its creases and on any synthetic material. The more traditional and luxurious saris like Paithani or Baluchary saris are a national treasure.

Bibliography

3d Studio Max. 3D graphics and animation software. http://www.3dmax.com.

Ananda. An exclusive boutique of saris in Kolkata, West Bengal, India. http://www.anandacal.com/products/products.htm/.

Manasi Athale and Sabine Stifter. *Survey Report on Texture Mapping*. Technical Report 01-21, Research Institute for Symbolic Computation, Johannes Kepler University, Linz, Austria, 2001. The report is electronically available as ftp://ftp.risc.uni-linz.ac.at/pub/techreports/2001/01-21.ps.gz.

P. N. Azariadis and N. A. Aspragathos. On using planar developments to perform texture mapping on arbitrarily curved surfaces. *Computer and Graphics*, 24: 539–554, 2000.

Banarasi Saree. Online shop of banarasi saris. http://www.banarasisaree.com/.

D. Baraff and A. Witkin. Large Steps in Cloth Simulation. In *SIGGRAPH'98, Computer Graphics Proceedings*, Annual Conference Series, pages 43–54. 1998. The article is available electronically as http://www-2.cs.cmu.edu/~baraff/papers/sig98.pdf.

Ingmar Bitter. Texture Mapping. Seminar on 3D Graphics Hardware, Experimental Computer Systems Lab, Stony Brook University, USA, 1996. The article is electronically available as http://www.ecsl.cs.sunysb.edu/cse659/tm.html.

Boost. Web site providing free peer-reviewed portable C++ source libraries. http://www.boost.org/.

Chantal Boulanger. http://www.devi.net/shakti/sari/. This website provides a valuable information on sari wearing styles.

K. Cok. Developing Efficient Graphics Software: The Yin and Yang of Graphics. SIGGRAPH 2000 Course Notes, 2000.

CorelDraw. An image manipulation program. http://www.corel.com/.

S. Fang and H. Chen. Hardware Accelerated Voxelization. *Computer and Graphics*, 24: 433–442, 2000.

M. Firebaugh. Computer Graphics. Lecture Notes, CSci 320. The notes are available electronically as http://www.uwp.edu/academic/computer.science/Faculty/firebaugh.www/LectIndex320.html.

J. D. Foley and A. van Dam. *Fundamentals of Interactive Computer Graphics*. Addison Wesley, 1984.

J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes, and R. L. Phillips. *Introduction to Computer Graphics*. Addison Wesley, 1994.

FreeCloth. A free, open-source cloth simulation tool. http://freecloth.enigmati.ca.

D. Gelb, T. Malzbender, and K. Wu. Polynomial Texture Maps. Hewlett Packard Laboratories, April 2000.

D. Gelb, T. Malzbender, and K. Wu. Light-Dependent Texture Mapping. Hewlett Packard Laboratories, April 2001.

G. K. Ghosh and Shukla Ghosh. https://www.vedamsbooks.com/11322.htm/.

GIMP. GNU Image Manipulation Program. http://www.gimp.org/.

P. Haeberli and M. Segal. Texture Mapping as a Fundamental Drawing Primitive, June 1993. The article is electronically available as http://www.sgi.com/misc/grafica/texmap/. Also part of the Graphica Obscura web site http://www.sgi.com/misc/grafica/.

P. Heckbert. Survey of Texture Mapping. *IEEE Computer Graphics and Applications*, November 1986. The article is electronically available from the author's web site http://www-2.cs.cmu.edu/~ph/.

P. Heckbert. *Fundamentals of texture mapping and image warping*. Master's thesis, University of California, Berkeley, 1989. Available electronically as http://www-2.cs.cmu.edu/~ph/texfund/texfund.pdf.

D. H. House and D. E. Breen, editors. *Cloth Modeling and Animation*. A. K. Peters, Ltd., 2000.

ImageMagic. A free image manipulation software. http://www.imagemagic.org/.

IndiCraft. Exclusive India Handicrafts Store. http://www.india-crafts.com/textile_products/index.html.

P. S. Karthikeyan and P. S. Ranganathan. Tutorial on Cloth Modelling. The article is electronically available as http://www.geocities.com/SiliconValley/Heights/5445/cloth.html.

LAPACK. Linear Algebra PACKage. http://www.netlib.org/lapack/.

Maya. A 2D and 3D graphics and animation software. http://www.aliaswavefront.com/maya/.

H. Mayr. Virtual Environments: Design, Modeling, Visualization and Simulation. Lecture Notes, Johannes Kepler Universität, 2000-01.

Mesa. A 3D Graphics Library. http://www.mesa3d.org/.

MIRALab. http://miralabwww.unige.ch.

M. E. Mortenson. *Geomwtric Modeling*. John Wiley & Sons Inc., second edition, 1985.

M. E. Mortenson. *Mathematics for Computer Graphics Applications*. Industrial Press, Inc., second edition, 1999.

MTL. Matrix Template Library. http://www.osl.iu.edu/research/mtl/.

OpenGL Blue Book. OpenGL Reference Manual. Reading, MA: Addison-Wesley Developers Press, 1996. The book is available electronically as http://www.parallab.uib.no/SGI_bookshelves/SGI_Developer/books/OpenGL_RM/sgi_html/bk02.html.

OpenGL Red Book. OpenGL Programming Guide. First printing, January 1997. The book is available electronically as http://www.parallab.uib.no/SGI_bookshelves/SGI_Developer/books/OpenGL_PG/sgi_html/ch01.html.

D. Pritchard. Implementing Baraff & Witkin's Cloth Simulation. The article is electronically available as http://freecloth.enigmati.ca/docs/report-chaps/.

X. Provot. Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior. The article is electronically available as http://graphics.stanford.edu/courses/cs468-02-winter/Papers/Rigidcloth.pdf.

Sari Safari. Online sari shop. http://www.sarisafari.com/.

Shalincraft India. Online sari shop. http://www.shalincraft-india.com/subhome/saree.html/.

SimCloth. A simple cloth plugin for 3D Studio MAX. http://www.chaosgroup.com/software/software.html.

Sabine Stifter. Geometric Foundations for Symbolic computations. Lecture Notes, Research Institute for Simbolic Computation, Johannes Kepler Universität, 2001.

Sabine Stifter. Geometric Modelling. Lecture Notes, Research Institute for Simbolic Computation, Johannes Kepler Universität, 2002.

Tourism of India. http://www.tourismofindia.com/exi/sari.htm/.

uBLAS. C++ template class library that provides BLAS (Basic Linear Algebra Subprograms) level 1, 2, 3 functionality for dense, packed and sparse matrices. http://www.genesys-e.org/ublas/.