REAL TIME PHYSICS-BASED AUGMENTED FITTING ROOM USING TIME-OF-FLIGHT CAMERAS

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Umut Gültepe

July, 2013

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Uğur Güdükbay (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Özgür Ulusoy

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Ahmet Oğuz Akyüz

Approved for the Graduate School of Engineering and Science:

Prof. Dr. Levent Onural
Director of the Graduate School

ABSTRACT

REAL TIME PHYSICS-BASED AUGMENTED FITTING ROOM USING TIME-OF-FLIGHT CAMERAS

Umut Gültepe
M.S. in Computer Engineering
Supervisor: Assoc. Prof. Dr. Uğur Güdükbay
July, 2013

This thesis proposes a framework for a real-time physically-based augmented cloth fitting environment. The required 3D meshes for the human avatar and apparels are modeled with specific constraints. The models are then animated in real-time using input from a user tracked by a depth sensor. A set of motion filters are introduced in order to improve the quality of the simulation. The physical effects such as inertia, external and forces and collision are imposed on the apparel meshes. The avatar and the apparels can be customized according to the user. The system runs in real-time on a high-end consumer PC with realistic rendering results.

Keywords: cloth simulation, computer vision, natural interaction, virtual fitting room, kinect, depth sensor.

ÖZET

UÇUŞ ZAMANI KAMERALARI KULLANAN GERÇEK ZAMANLI FİZİK TABANLI ARTTIRILMIŞ GİYİNME KABİNİ

Umut Gültepe
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Doç. Dr. Uğur Güdükbay
Temmuz, 2013

Bu tezde gerçek zamanlı fizik tabanlı bir artırılmış giyinme kabini ortamı için bir çalışma çerçevesi önerilmektedir. İnsan avatarı ve kıyafet için gerekli üç boyutlu modeller özel sınırlar çerçevesinde modellenmiştir. Bu modeller daha sonra bir derinlik alıcısı tarafından takip edilen bir kullanıcıdan alınan girdi ile gerçek zamanlı olarak hareket ettirilmektedir. Simülasyon kalitesini arttırmak amacı ile hareketler çeşitli filtrelerden geçirilmektedir. Eylemsizlik, dış kuvvetler ve çarpışma gibi dış etkenler kıyafet modeline uygulanmaktadır. Avatar ve kıyafet modelleri, kullanıcının boyutlarına göre özelleştirilebilir. Sistem üst kalite bir kişisel bilgisayar üzerinde gerçek zamanlı olarak gerçekçi görüntüler oluşturarak çalışmaktadır.

Anahtar sözcükler: kıyafet simülasyonu, bilgisayarla görü, doğal etkileşim, sanal giyinme kabini, kinect, derinlik sensörü.

Acknowledgement

I would like to express my sincere gratitude to my supervisor Assoc. Prof. Dr. Uğur Güdükbay, who guided and assisted me with his invaluable suggestions in all stages of this study. I also chose this area of study by inspiring from his deep knowledge over this subject.

I am very grateful to my jury members Prof. Dr. Özgür Ulusoy and Asst. Prof. Dr. Ahmet Oğuz Akyüz for reading and reviewing this thesis.

I would like to thank Computer Engineering Department of Bilkent University for providing me scholarship for my MS study. I also would like to thank the Scientific and Technical Research Council of Turkey (TÜBİTAK) and the Turkish Ministry of Industry and Technology for their financial support for this study and MS thesis.

to my mother, father and my brother...

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Computer graphics are being used in more and more areas today to help with visualization of data, such as in big data and crowd simulation. Human body animation and cloth simulation have been two significant subjects of the field for a while. Although there are cases where the simulation results are incredibly life-like, the task is still nothing trivial.

One of the most time-consuming stages of apparel shopping is the customer trying the apparels by putting them on, which is not even possible in online stores. With the advances in augmented reality technologies, virtual fitting rooms are slowly taking their places in both real and virtual stores [1, 2] to improve the quality of apparel trial experience while also making it faster. These frameworks utilize both humanoid and cloth animation features, hence they are limited by the bottlenecks in both fields. The demand for virtual dressing frameworks is increasing with the spread of online apparel commerce and the interactive advertisement platforms. There are various features of virtual fitting frameworks, where each has different priorities in different types of applications:

- The apparel can be displayed on a virtual avatar or on real photos or videos of the user. The former is used more in design stages, the latter more in online and in-store try-on frameworks. Avatars can be static or dynamic, animated with the motions of the user or with pre-recorded animations.

- A virtual fitting room framework can utilize an apparel image database which consists of pre-recorded two dimensional photos of apparels in various poses to render the apparel, or it can utilize a virtual three dimensional model. Former approach is considered to require more preprocessing and cause lapses between poses, however the rendered apparel looks realistic as it is a two dimensional photo. Latter approach enables rapid three dimensional apparel model generation and more realistic material and physical simulations.

- If the three dimensional approach is used, the apparel mesh can be processed with physical simulation or can be stagnant. The former requires more advanced frameworks and more powerful hardware, making them more suitable for desktop applications rather than online fitting rooms.

- A virtual fitting room framework can scale the apparel and meshes according to the active user. The scaling can be standardized where a fixed size among possible size options would be offered to the customer, or it can be detailed scaling similar to made-to-measure tailoring. The latter approach is more complex compared to the former, because a larger set of measurements are needed with higher precision.

On the low-end of the virtual fitting room spectrum, there are super positioned 2D images of the user and the apparel without any animation. Advanced virtual fitting rooms, on the other hand, show the apparel items either on the video of the user or on a virtual avatar, both scaled to reflect the user's body characteristics [3]. Physics-based garment simulation for a better fitting experience is included in the high end frameworks [2]. Our approach utilizes a three dimensional virtual avatar which is updated with user motions captured through a depth sensor. The apparels are rendered as three dimensional models and updated with physical simulation.

1.1 Our Approach

This study is aimed to develop a novel virtual fitting room framework that provides all the basic features expected from such an application, along with enhancements in various aspects for higher realism. These enhancements include motion filtering, customized user scaling, and physics engine. Motion filtering process starts with temporal averaging of joint positions in order to overcome the high noise of the depth sensor. However, temporal averaging does not prove to be sufficient because unnatural movements take place due to limited recognition capabilities and self-occlusion. Customized joint angle filters, along with bone splitting to let limbs twist in a more natural way and footskate correction filters are implemented.

The avatar utilizes a skeleton that conforms with the LOA 2 of H-ANIM 200x specification [4], although not all bones are used for animation because of the data received from the depth sensor. The skeleton and mesh are modeled in Blender [5], exported and used in binary format. The simulated apparel pieces are also modeled in Blender, although they are exported in Wavefront OBJ format in order to be parsed by the physics engine. They are binarized on-the-run to be used by the game engine.

The cloth pieces to be fitted on the user's avatar must first be scaled accordingly. To this end, a body measurement process is implemented, which starts with depth map smoothing, in order to reduce the noise. Afterwards, the filtered depth map is utilize along with filtered user joints to measure a set of parameters, which are used in conjunction to estimate the body height and width. These parameters are averaged over time to minimize the error.

The physics engine utilizes collision spheres and capsules to perform collision detection. The correct sphere radii and positions are determined during body measurements. The virtual avatar is aligned with a set of invisible spheres and capsules are aligned with joints and limbs, which are updated in real time and used in collision detection. Cloth particles are also affected by gravity and inertia.

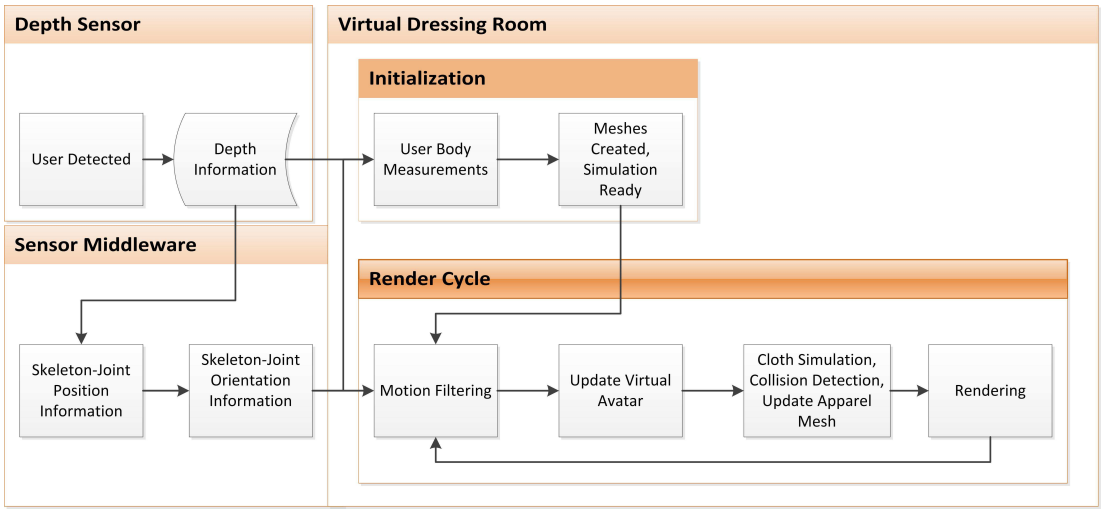


Figure 1.1: The overall virtual dressing framework

1.2 System Architecture

The framework operates in two distinct stages: *modeling* and *simulation*. The modeling stage consists of preparing the avatar and cloth meshes for the simulation. The base low-detail avatar meshes, a male and a female, are acquired from online sources [6, 7], rigged with a skeleton, and painted with materials and textures. The base apparel meshes are also acquired from online sources [8, 9, 10, 11, 12, 13]. They are aligned with the avatar meshes, painted, and their physical properties are specified. After the models are ready for simulation, they are exported in appropriate file formats to be loaded by the simulation stage.

The simulation stage starts after a user is identified. It is initialized by performing user body measurements and scaling the avatar and apparel meshes accordingly. The render cycle starts with fetching the user skeletal joint information from the depth sensor. The virtual avatar and the static apparel meshes are updated by applying the acquired joint orientations. The collision spheres that coincide with the skeleton joints are also updated during this process. The dynamic apparel meshes are updated with new positions and orientations and the collision data and the updated topology information are transferred from the physics environment to the virtual environment, followed by rendering. The

overall virtual dressing framework is shown in Figure 1.1.

1.3 Organization of the Thesis

The thesis is organized as follows. Chapter 2, we give related work on virtual fitting rooms and depth sensors. Chapter 3 describes the human and cloth modeling for virtual fitting room. Chapter 4 focuses on the animation techniques along with various optimizations for a realistic experience. Chapter 5 discusses the physics engine and cloth simulation process. Chapter 6 deals with the cloth resizing process for a customized fitting experience. The experiments, performance qualities and results are presented in 7. Chapter 8 outlines the thesis and shows the future direction of this research. Appendix A describes the game engine that provides the boilerplate functions for our framework. Appendix B gives information about the depth sensor and its user tracking capabilities. A user interaction feature, depth-based hand tracking is explained in Appendix C.

Chapter 2

Related Work

Virtual fitting frameworks are complex systems composed of many modules. The related works on the major components of the used framework are summarized in this chapter, along with a general history of the field. The overall framework can be divided into three major modules: *human body modeling and animation*, *motion capture systems* and *cloth modeling and simulation*. Although these topics are connected in many ways, their foundations are distinct and should be discussed separately, followed by the discussion of the work on virtual fitting rooms.

2.1 Human Body Modeling And Animation

Although the human and apparel modeling both have foundations in 3D mesh creation, they require different traits and qualities. Human body modeling utilizes disciplines such as rigging and skinning, whereas apparel modeling is mostly based on physics simulations.

2.1.1 Human Body Modeling

As the humans are mostly the main characters in virtual worlds, various methods and techniques exist for human body models and animations. Determining the suitable one depends on the requirements of the application. Real-time applications require a certain level of simplicity, as the simulation time cannot exceed the frame duration. Offline applications can utilize highly detailed models, looking much more realistic. The basis for both extremes however, is the same, which is a skeleton. The approach is starting with a connected set of rigid objects named as *bones*, continuing by adding layers of muscle, skin, hair and others, depending on the required quality level. This is a very common modeling technique used in computer graphics, which is called *layered modeling technique* [14]. The animation is achieved by rotating the bones, which is followed by upper layers. This technique also improves the reusability of the framework because the same animation sequence can be used for multiple body models with different detail levels utilizing the same base skeleton.

The articulated skeleton consists of a hierarchical structure of joints and limbs to model a human-like skeleton. Joints are the points that act as the origin of the respective local coordinate space. The limbs are the rigid segments that connect the joints in the hierarchy. Rotation in the local coordinate systems defined by the joints cause the rigid limbs to be rotated, creating the motion. The complexity of the model can be determined by the number of joints and the degrees of freedom (DOFs). The DOF is the number of independent parameters that define the configuration of a joint; a joint can rotate and translate in at most three orthogonal directions, hence, the maximum DOF a joint can have is six. Having the maximum number of DOFs in a human body model might seem like a good way to improve the realism, however this also increases the complexity of the structure, resulting in more mathematical operations. As most human-body joints can only rotate, not translate, assigning six DOFs to every joint is redundant. Furthermore, angular and axis constraints with certain joints (such as knee or elbow) further simplify the model while making it more realistic.

In order to provide a common basis and a standard for modeling of 3D humans

with hierarchical skeletons, Humanoid Animation (H-Anim) specification was developed by Web3D Consortium [4]. Different levels of articulation are provided in X3D/VRML format, focusing specifically on humanoid objects rather than random articulated figures. H-Anim standard provides a common ground for applications to be classified depending on their humanoid animation complexity, mainly by the number of joints and DOFs.

2.1.2 Human Body Animation

Animating humanoid meshes is a complex and old subject of computer science, as there are many factors which contribute to the way humans move. The task gets even harder with the ability of the human eye to distinguish very minor unnatural motions. The first approaches on humanoid animation are stick figures, which have led to multi-layered high resolution meshes.

Stick figured animation dates back to 1970s, where the technology would limit the qualities of animation to one dimensional limbs [15]. With the advances in computer hardware, the details have improved and the complexities are increased. Surface models were the first improvement on top of the original stick figure animation. A surface or "skin", which envelopes the articulated skeleton is introduced to the model. The translation of the surface varies depending on how the vertices are assigned to the bones. The process of assigning vertices to a specific joint or set of joints is called skinning. The quality of the animation depends on both the complexity of the skeleton as well as the skinning quality and technique.

The initial approach to the animation of enveloped surface was assigning weights to the individual polygons. However, this approach resulted in broken surfaces almost every frame. The first solution to this problem was introduced by Komatsu [16], where a continuous deformation function is used with respect to the joints. Another study introduced a new skinning process by assigning vertices to joints instead of polygons [17]. This simple difference allowed a polygon to be assigned to multiple bones, preventing two polygons from separating as the

common vertices would not get ripped.

Although assigning vertices instead of polygons to bones improved the realism significantly, it produced artifacts in extreme rotations. Inspired by the true nature of human skin and deformation, the new solution introduced assigning a vertex to multiple joints. Called linear blend skinning, this technique further improved the quality of character animations. However, it still was not sufficient with certain parts of body such as forearm and elbow, where the bone positioning and configuration are more complex than a single series of connected bones. An example of this situation can be seen in Figure 4.3. A single bone cannot imitate the twisting motion enabled by two parallel bones. Various solutions to this problem has been proposed, the proposed solution in this study is described in Section 4.4.3.

A new deformation technique called double quaternion skinning overcomes these artifacts without introducing additional time complexity [18], even improving the performance. Quaternions, which are primarily used as a notation for rotations can also be used to define translations. Dual quaternions can be blended for rigid transformations and produce much more realistic results.

Another approach to fixing the extreme rotation situations was proposed by Kavan et al. [19], where the bones which could not produce realistic results would be split into child bones in runtime. This approach, although successfully correcting the otherwise present artifacts, required significantly more computational power and was not suitable for real-time applications.

2.2 Motion Capture Systems

2.2.1 Non-Optical Motion Capture Systems

Non-optical systems are based on mechanics, inertia [20] and magnetics [21]. Mechanical systems were the earliest examples of motion capture systems and their

cumbersome hardware requirements make them hard to use. Inertial systems utilize inertia sensors to determine the global orientation of the body part they are attached to, which can later be converted to local orientations in post processing. Magnetic systems perform the same operation, except they do so by measuring the magnetic field emitted by the magnetic markers. Non-optical systems tend to deliver more accurate results than optical motion capture systems, although their hardware requirements and hardness to use are two major drawbacks.

2.2.2 Optical Motion Capture Systems

Optical systems are based on recording the target in action by one or more cameras. The most common approaches in optical motion capture systems utilize some sort of markers placed on key positions of the target. Markers can be passive [22] or active [23], depending on whether they just reflect the light or they emit light themselves. Regardless, their task is identical in both cases, providing the camera with the positional information of key parts of the body.

With the advances in computer processing power and camera technology, markerless motion capture systems have emerged [24]. Theobalt et al. have contributed to the markerless human motion capture systems subject with multi view video sequences [25, 26, 27]. Their framework utilizes 8 or more cameras placed on a circle around the target. They construct a volumetric visual hull by getting the intersection of extruded 2D silhouettes - which is called shape from silhouette technique [28, 29]. Using a set of cameras rather than a single camera is a remedy to the self-occlusion problem. After the visual hull of the user is acquired, the hull is segmented into body parts by fitting a pre-defined body model. Accurate as they are, these applications require immense computing power and are not suitable for real time applications.

The emergence of the consumer-level depth sensors at affordable costs created a new possibility for markerless motion capturing [30]. As the depth sensors operate in real time and the visual hull is the input rather than a processing product, the computation time for skeletal position estimation is considerably lower,

enabling real-time motion transferring from actors to virtual avatars. However, depth sensors suffer from the same disadvantage as single-camera systems, self occlusion, although this problem can be solved partially by utilizing multiple depth sensors [31]. Furthermore, the IR and TOF depth sensing technologies are not evolved enough yet to provide high resolution depth frames and produce quite a bit of noise, hence reducing the output motion quality. This problem is analyzed in depth by Khoshelham and Elberink [32] and concluded that the standard deviation reaches two centimeters in a measuring distance of three meters. Matyunin et al. [33] attempted to improve the quality by filtering with additional information from the attached RGB camera.

2.3 Cloth Modeling and Simulation

A number of industries, mainly entertainment and apparel commerce, created a demand for rapid virtual apparel design and realistic simulation along with the advances in computer graphics. As every other physical phenomenon, garment simulation is no simple task to perform in virtual worlds. To be able to create realistic render results for entertainment industry while keeping the physical properties accurate requires very complex simulation frameworks and physics engines.

2.3.1 Cloth Design and Modeling

Garment design aims at creating meshes that look realistic. Furthermore, designing for virtual simulation environments require the garment to be suitable for physics simulation. The two approaches to creating 3D garment meshes are straightforward 3D design and combining 2D designed patches by "stitching". However, the main distinction between garment design suites and techniques comes from the desired outcome, whether it is a production sample or a virtual model for simulation.

2.3.1.1 Production-oriented Design Systems

The design suites explained in this section focus on product creation and industrial usage of the apparel design, rather than virtual simulation. As the goals of two systems are different, other design techniques are used for simulation purposes. However, the underlying 2D and 3D design principles are quite similar. The initial approaches of 2D garment design consisted of two parts where each part has several steps [34]:

1. Parametric design-based pattern generation
2. Pattern alteration-based on grading techniques

A widely used CAD suite with this approach is Gerbert Technologies' AccuMark solution [35]. A major drawback of 2D garment design systems is the necessity of a certain level of expertise with pattern design, as the 2D patches are not easy to visualize for an inexperienced person. 3D garment design is the solution to this problem as the apparel can be modeled on virtual human models directly and then separated into multiple parts for production. In order to create realistic apparels, the underlying 3D human model must be realistic itself; best way is to obtain the mesh from a 3D full body scan. Assyst-Bullmer's design software is an example of such CAD suites [36].

2.3.1.2 Simulation-oriented Design Systems

Simulation oriented design systems aim at creating garment meshes which are suitable for using in virtual environments, mainly in entertainment software such as video games, and recently, in virtual fitting rooms. The output of these suites must not be too complex to be simulated in real time, while being as realistic as possible.

One of the first physics-enabled 2D garment design systems was introduced by Yang et al. [37]. The suite included 2D design panels, deformable cloth modeling

and a human body model which could be used to simulate the designed apparel on. Chittaro and Corvaglio [38] aimed to develop a platform which would connect the textile industry with computer graphics and simulation world via defining an interchangeable format with VRML-based 3D meshes from 2D patterns. Turquin et al. [39] developed a sketch based interface for tailoring, dressing and simulating clothing pieces on virtual characters.

3D CAD systems suit simulation oriented design better than 2D by the nature of the general applications. The system developed by Bonte et al. [40] reverses the process defined in the previous examples, where the garment is designed in 3D and 2D patterns are generated from it. Garments are modeled on a mannequin to conform with the body characteristics of humans. The framework also includes a particle-based simulation feature. Cugini and Rizzi developed a framework [41] for the design of men apparel items with a 2D/3D hybrid approach using Autodesk Maya [42] for 3D modeling and simulation. A similar system was proposed by Durupınar and Güdükbay [43] where the 3D garments would be created from 2D patterns and simulated using a mass-spring system.

The GPU company NVIDIA's PhysX physics engine is one the best options for real-time physics simulation available [44]. An extension of this engine, APEX is a scalable dynamics framework with specialized physic based utilities such as destruction, particles, turbulence (fluids) and clothing [45]. The modeling extension of this framework is available as an extension for 3dsMax [46] and Maya [42] design suites. The extensions provide an interface which are aimed at creating an artist-oriented environment as possible, abstracting all the programming work. As the PhysX engine is optimized for NVIDIA GPUs, the easiness of the APEX along with its performance make it one of the best options for real-time clothing design-and-simulation framework.

With the advances in virtual try on systems, various frameworks for apparel design solely for the use with such environments emerged. The state of these frameworks are discussed in Section 2.4 along with virtual fitting room frameworks.

2.3.2 Garment Simulation

Garment simulation is mainly deforming an apparel mesh in a way which feels natural to the eye. It also includes collision detection and support for tearing behavior. The nature of the garment simulation depends on the used modeling approach. Two common garment modeling approaches are geometrical models [47] and physics-based models.

2.3.2.1 Geometric Garment Modeling

Geometric models do not take the physical properties such as stiffness and stretching into account. Instead, the apparel is modeled as a collection of cables and hyperbolic cosine curves, Weil added the stiffness factor as a distance constraint [47]. As the physical properties of clothes are omitted or not accurate, geometric models do not work well with dynamic models as well as they do with stationary renders [47].

2.3.2.2 Physical Garment Modeling

Physical approaches model the cloth as systems of springs-masses or particles, or as a continuous flexible material to be solved as a elasticity problem. The spring-mass system is first presented by Haumann and Parent [48], which converts each vertex into a point mass and converts each edge into a spring. The simulation is attained by solving the spring-mass equations. Further improvements on spring-mass systems include different sets of springs for orthogonal axes and distance constraints to achieve more garment-like simulations [49]. The NVIDIA PhysX engine utilizes an enhanced spring-mass system for its cloth solver [50].

Terzopoulos et al. [51] presented elastically deformable models based on continuum mechanics and differential geometry. Another continuum-based approach that sacrifices accuracy for performance by focusing on numerical solutions was

described by Baraff and Witkin [52]. Elasticity solutions rely on energy interactions between the particles and achieve a solution by minimizing the total energy stored within the whole mesh.

2.4 Virtual Fitting Rooms

Virtual fitting rooms have been a research subject for more than a decade. Protopsaltou et al. [53] developed an Internet-based approach for virtual fitting rooms, although it was not real time and required marker-based motion capture systems for animation. Zhang et al. [54] used a multi-camera system utilizing shell fitting space (SFS) [29] techniques to build a real time intelligent fitting room.

Advances in time-of-flight technology made depth sensors available at consumer-level prices with better performance. This prompted a wave of research based on depth sensors in various fields, such as rehabilitation [55], indoor modeling [56], and medicine [57]. Another topic that attracted significant attention from both researchers and companies is real-time virtual fitting rooms [58]. Giovanni et al. [59] developed a virtual try-on system utilizing a calibrated set of Kinect and high definition cameras, while comparing the two state-of-the-art depth sensing software development kits (SDKs)-OpenNI [60] and Kinect for Windows SDK [61]. While most frameworks utilize garment meshes with physics simulation [1, 2], another intriguing approach is using a pre-recorded apparel image database, from which the images are superpositioned onto the RGB video of the user [62, 63].

A key purpose of both virtual and real fitting rooms is giving the customer the look and feel of a cloth with a specific size on the user's body, so the user can choose the appropriate size for him. Embedding the feature of matching cloth sizes with users requires capturing the users' body dimensions. More advanced frameworks even construct virtual avatars with input from only one depth sensor [64, 65]. On the other hand, despite these works provide higher detail avatars

and keener measurements, which might be more suitable for made-to-measure type of framework, the process requires too much time to work with a real-time 'fixed-size try-on' virtual fitting room application where simple body height and width measurements are sufficient. These applications require a faster approach along with a specialized garment design framework such as the works of Yasseen et al. [66] or Meng et al. [58]. There are also notable studies for made-to-measure technologies for online clothing stores [67], shape control techniques for automatic resizing of apparel products [68], modeling a 3D garment on a 3D human model by 2D sketches [69], and garment pattern design using 3D body scan data [70]. A recent study [71] shows that such applications are well-received by public and have potential commercial uses.

Chapter 3

Human and Cloth Modeling

The simulation software utilizes 3D models as the main displayed components. Since the quality of the models are a significant factor in the overall quality of the simulation, they must be prepared specifically for the purposes of the application, with attention to details. Two distinct sets of 3D meshes are required in general for the simulation:

- A cloth mesh to be processed by the embedded physics engine.
- A full body human mesh to act as an avatar for the simulated cloth mesh.

3.1 Human Avatar

After taking the complexity of both 3D CAD programs and the human body itself into consideration, designing a complete full human body mesh is deemed to be a laborious and inessential task. Through researching various digital sources, evaluating the quality and cost of various potential candidate meshes, a female [6] model and a male model [7] are elected, due to their realistic appearance, accurate proportions and fine details. For the purpose of improving the realism of the simulation, details are introduced to the base model with the Blender CAD software [5].

3.1.1 Rigging

Animating a 3D mesh requires skinning, which is moving the vertices with respect to a bone on a skeleton. Because the base models have no skinning or material properties, they require custom rigging and painting. For the purposes of this research a manual rigging and painting would be preferred, in order to be able to integrate the model easily into the software. The rigging is performed in Blender [5], where the base human skeleton is provided, detailed in H-ANIM2 level with a simplified spine (Figure 3.1).

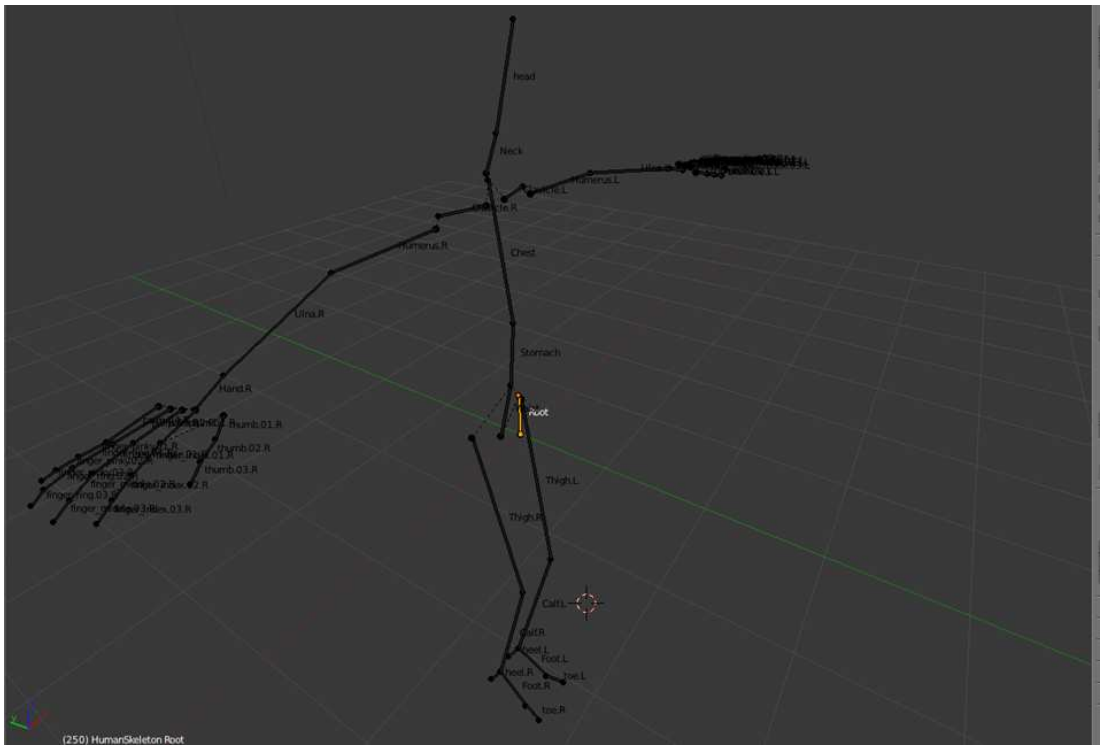


Figure 3.1: The rigging base skeleton.

Prior to rigging, the skeleton needs to be in perfect alignment with the mesh. After modifying the initial bone positions, the vertex groups are assigned to the bones with proper weights (Figure 3.2). After assigning every vertex to the appropriate bones, there are no cracked surfaces with motions which are natural for humans. In the end, the models are exported in OGRE .mesh format along with the skeleton it used.

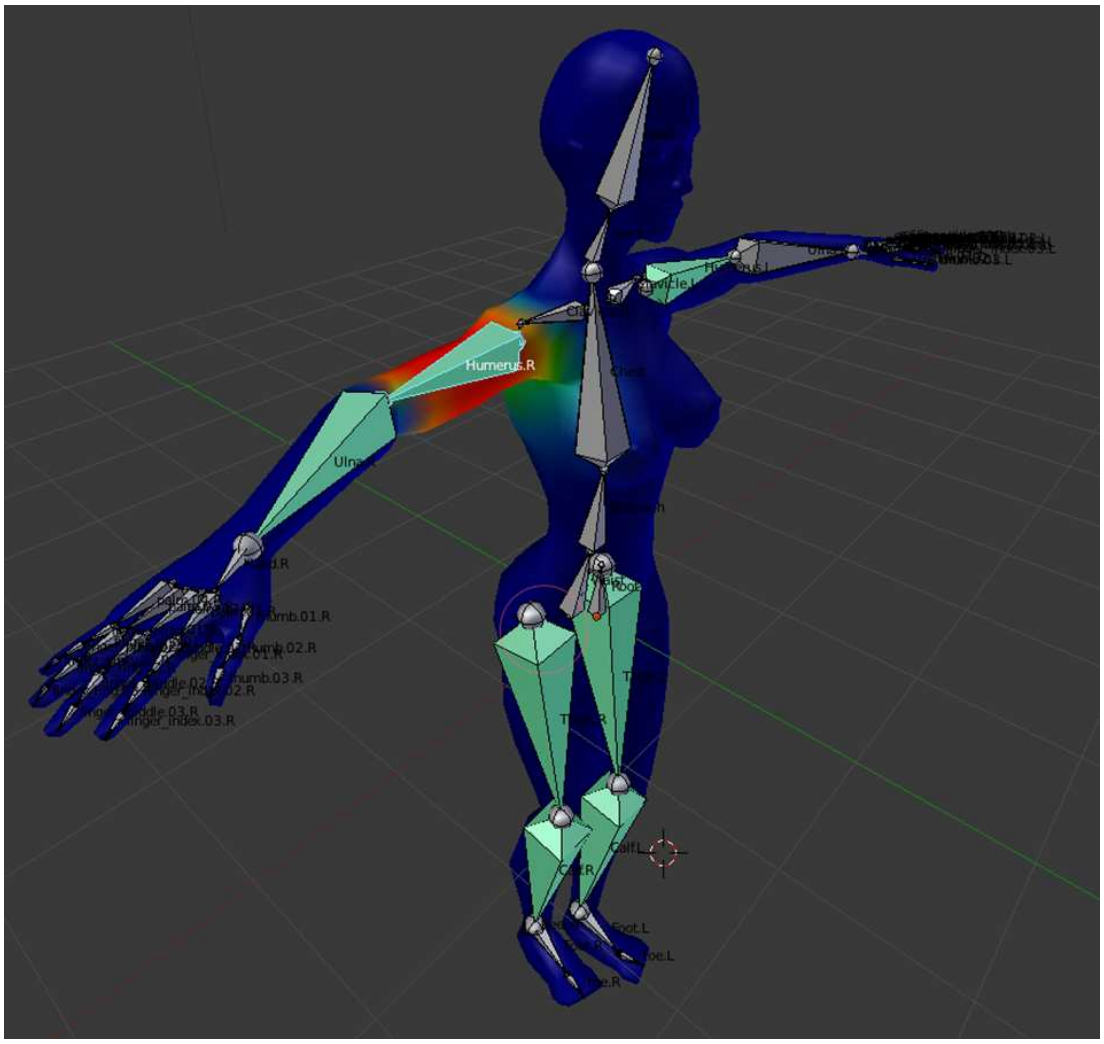


Figure 3.2: The vertex weights for the Humerus.R bone.

3.1.2 Material Properties

The base models come with no material properties. In order to improve the realism of the model, texturing/painting tasks for various parts of the body are performed. These parts include the general skin, hair, eyes and lips (Figure 3.3). Other additions include accessories such as hair sticks and earrings to achieve more realistic results.

Figure 3.3: Detailed appearance of the face.

3.2 Cloth Mesh

As one of the major pieces of this study is accurate cloth simulation, accurately modeled cloth meshes which are suitable for simulation for required. After a thorough research through a variety of available models, a set of base apparel meshes for both male and female models are selected [8, 9, 10, 11, 12, 13]. The models vary in detail and quality, all of them are optimized and enhanced for the framework.

3.2.1 Body Positioning and Splitting the Dress Mesh

For a successful animation and simulation of the dress on a human avatar, both meshes must be in proportion with each other and properly aligned. The required modifications to the meshes are performed in Blender(Figure 3.4). The initial approach is to include all the vertices of the clothing mesh in the simulation. However, after various attempts to simulate all vertices on the dress mesh, this approach failed to achieve a realistic looking result because of two reasons:

- Whole meshes contain too many vertices, the dress mesh for instance consists of 4088 vertices. The simulation Runs in real-time, however the topography of the mesh is not appropriate for the physics engine because of the very large number of vertices in the cloth. It shifted from a fabric structure to more of a jelly form, over stretching and tearing with its own weight.

- The friction necessary to keep the dress on the human avatar's shoulders is not sufficient enough to keep the dress on the avatar. The dress keeps sliding, stretching and acting unnaturally. The design of the used physics engine is not appropriate for this approach.

In order to overcome these problems, we split the dress mesh into two parts and utilize different animation techniques for each.

- The static part, which should be attached to the body and not affected by the wind, is modeled as a static mesh with skinning, like the human avatar. It is animated with the same transformations as the human avatar, matching its position and staying on the body perfectly.

- The dynamic part is the part to be simulated, affected by inertia, wind and other factors. Their attachment line is just above the waist of the human avatar, which is pinpointed through try-and-error experiments with other locations (Figure 3.4). Keeping the line very low resulted in both unnatural collisions and the cloth being too rigid. Keeping it too high brought out the problems in the first approach. With its current setting, the dress mesh is animated naturally on the virtual avatar.

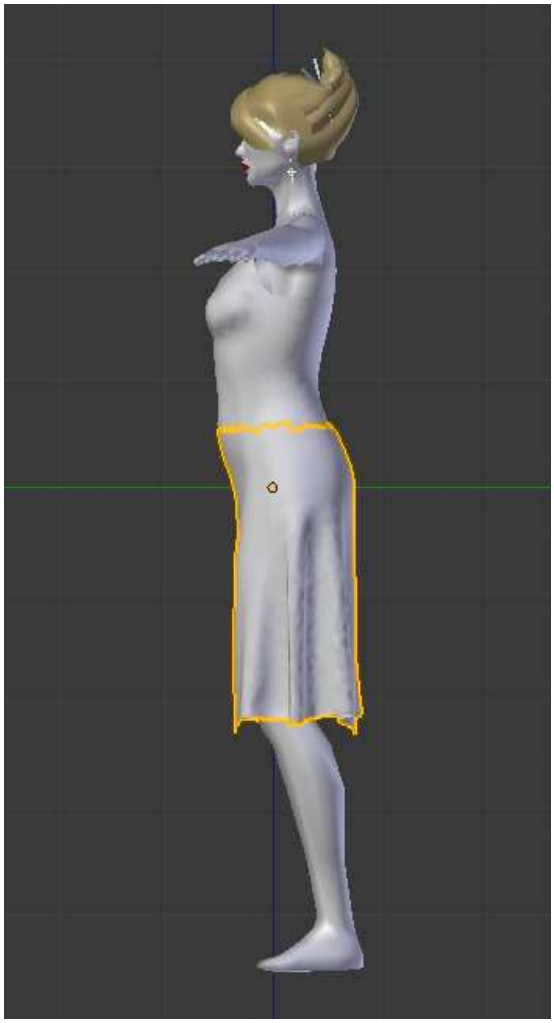


Figure 3.4: The dress, positioned on the body, along with the upper-part of the skeleton. In this shot, the dynamic part is highlighted with orange border.

Chapter 4

Animation

The animation in the system consists of moving the virtual avatar and the cloth mesh in a realistic manner, with the addition of physical simulaton of the apparel meshes. This chapter focuses on the animation of human avatar and the static cloth meshes, the physical animation is explained in Chapter 5.

The rigged human body mesh ready to be animated is exported from Blender in Ogre XML format. The XML files are converted to binary skeleton and mesh files by the OgreXML serializer and imported natively into the software. Prior to animation, the bones are set up to be updated by the data from depth sensor. Afterwards, they are updated with the orientation data.

4.1 Initialization

Animatable meshes are loaded and maintained via the custom *SkeletalMesh* class, which is the middleware between Ogre skeletal animating system and the input from Kinect sensor. An instance of the *SkeletalMesh* class is initialized with a mesh file. After the mesh is loaded with the skeleton data, the bone list is iterated through, given the initial orientation uniquely for each animatable bone with the Kinect sensor.

The animatable bones are set to be manually controlled and set to their initial orientations. The lower part of the body inherits orientations as they are filtered for footskating and use different orientation mechanisms. The upper part of the body is rotated in global space. The bones are set to their initial state, to be reset at every frame and updated with new orientations. The non-animated bones are left to be automatically controlled in order to keep them aligned with their parent bones.

4.2 Animation

Every frame, the orientation information from the bones are extracted in quaternion form, along with the confidence of the sensor in that orientation. If the confidence is less than 0.5 over 1, the bones are left as they were in the previous frame to avoid unnatural movements. The root bone is translated in local space at the end to simulate the translation above rotation. This technique is used with the static parts of the apparel meshes as well. The linear weighted skin blending [72] is used in order to simulate deformation on the characters skin. The effects of four bones with different weights are combined linearly to change the positions and normal of vertices.

Algorithm 1 is executed during the pre-render cycle. The bone orientations are set to be ready for animation, deformation and rendering. At every render cycle, update Skeleton function is called, which automatically fetches the coordinates from the Kinect and sets up the skeleton for vertex blending.

Algorithm 2 is executed prior to rendering, to update the vertices of the skin. The vertex blending function is where the deformation actually occurs. Every vertex is assigned to at most four different bones. In order to speed up the deformation process, the transformation matrices for the corresponding matrices are collapsed into one. Collapsing is simply weighted addition of the four weighting matrices for a vertex.

After four weighting matrices for each of four vertices are collapsed into four

Algorithm 1: Bone transformation algorithm

```
1  function transformBone(bone)
       Input: A bone and the corresponding orientation matrix from Kinect
       Output: The same bone with updated orientation
2      q_I = initial orientation of bone
3      q_N = relative orientation
4      q_K = 3 × 3 Orientation matrix from Kinect
5      if kinect_confidence > 0.5 then
6          q_Q = toQuaternion(q_K)
7          q_N = toLocalSpace(q_Q)
8          q = q_N × q_I
9          bone.orientation = q.normalise()
10 if user is new then
11     p_torso.initialize()                    /* Initialize torso position */
12
13 foreach bone do
14     if bone_orientation is new then
15         transformBone(bone)
16         skeleton.needsUpdate()
```

matrices, the following matrix operations are performed. Four vertices are processed together in order to fully utilize the parallel matrix multiplication features. Let us take the weighted matrix for vertex i:

$$M_i = \begin{bmatrix} m^i_{00} & m^i_{01} & m^i_{02} & m^i_{03} \\ m^i_{10} & m^i_{11} & m^i_{12} & m^i_{13} \\ m^i_{20} & m^i_{21} & m^i_{22} & m^i_{23} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

It should be noted that this matrix is a linear combination of four linear transformation matrices from four weighting bones; hence, the 3^{rd} row is by default [0 0 0 1]. We also have the initial positions for the four vertices as $[p^i_x\ p^i_y\ p^i_z\ 1]$.3^{rd} value, which is one is not included in the vertex buffer and it will not be taken into account with the calculations. In order to generate an efficient SIMD process, we perform 4×3 dot product calculations in each instruction. Dot products are commanded in the machine language, which makes it more

efficient compared to higher level matrix calculations. To compute the simulated x coordinates for four vertices, we combine the first rows of all collapsed matrices into a 4×4 matrix and transpose it:

$$M_T = \begin{bmatrix} m_{00}^0 & m_{00}^1 & m_{00}^2 & m_{00}^3 \\ m_{01}^0 & m_{01}^1 & m_{01}^2 & m_{01}^3 \\ m_{02}^0 & m_{02}^1 & m_{02}^2 & m_{02}^3 \\ m_{03}^0 & m_{03}^1 & m_{03}^2 & m_{03}^3 \end{bmatrix} \tag{4.2}$$

We construct a position matrix of size 4×3 with the positions of all vertices:

$$P_T = \begin{bmatrix} p_x^0 & p_x^1 & p_x^2 & p_x^3 \\ p_y^0 & p_y^1 & p_y^2 & p_y^3 \\ p_z^0 & p_z^1 & p_z^2 & p_z^3 \end{bmatrix} \tag{4.3}$$

Next, the matrices M_T and P_T are multiplied in a row-by-row fashion and summed together to calculate the x-coordinate displacements of four vertices:

$$d_x = M_{T0} \times P_{T0} + M_{T1} \times P_{T1} M_{T2} \times P_{T2} + M_{T3} \tag{4.4}$$

The result vector d_x is a 4×1 vector, which has the post-blended vertex x-coordinates: $[P_x^0\ P_x^1\ P_x^2\ P_x^3]$. The same procedure is applied to the normal of a vertex. Process continues with the next set of four vertices.

4.3 Interaction Between the Body and Cloth

The movements of the user are passed on the pieces of cloth separately. Non simulated parts of the clothes are the ones which do not get separated from the body most of the time. Most parts of our clothes usually stick to the body and experience the same deformation as our skin. In order to increase performance and get better results, detailed simulation on these parts are not run, instead

they are deformed the same as the remained of human body. For instance, in the full-body dress mesh, the part above the waist has a skeleton similar to the body and the information from the Kinect is passed on to this portion as well. It is treated as a part of the actual avatar.

The movements of the body are transferred into the simulated section of the cloth through transformation, collision and inertia. The fixed vertices are transformed to match the remainder of the cloth. The transformation is done on the rendering level, the physx world experiences no difference in transformation manner. This process keeps the cloth aligned with the rest of the visible world. The colliding body is updated and collided with the cloth. The colliding body consists of 16 spheres and the capsules connecting the spheres. The details of the colliding body is explained in Section 3.1. This process keeps the cloth out of the avatar's way. The Inertia of transformations is passed onto the simulated cloth, increasing the realism. The passed on inertia comes from the rotation and the transformation of the root bone of the human skeleton. With this process, although there is no actual transformation in the physics world, the resulting inertia effects are visible on the cloth itself.

4.4 Motion Filtering

Application of the raw data from the sensor causes unnatural movements due to the noise in the sensor input, self-occlusions of the body and inadequate IK solvers. In order to present a more realistic avatar animation, a series of filters and constraints are applied to the sensor data.

4.4.1 Position Filtering

The most severe disruption of the self-occlusion problem takes place in the joint position acquisition. There is no possible way of acquiring the correct position of a limb when the sensor has no vision of it. However, the way humans move

their limbs under normal conditions follow certain principles and trends, which can be used to estimate the locations of occluded body parts. The nature of these motions, demonstrating traits similar to seasonal behavior, makes them suitable for applying a variety of filters [73]. The framework utilizes the Holt-Winters double exponential smoothing [74, 75] as it comes with the middleware, easy to use and delivers good quality results with acceptable latency for the purposes of this application.

4.4.2 Rotation Filtering and Constraints

The joint orientations are acquired from the sensor middleware directly, however the data is not smooth. Although the middleware enforces certain constraints (such as allowing only pitch rotations on ulna), there are often significant gaps in the estimated angles that produce unnatural tremor-like movements on the avatar. Furthermore, there is no filtering of unnatural rotations that take place when an occluded body part is estimated to be in a wrong location.

The inferior quality of the orientation data is improved in two stages: applying another set of constraints on the joint data based on the natural limits of human bones, followed by an asymptotic smoothing of the joint angles to prevent the effect of gaps in the angles (see Figure 4.1).

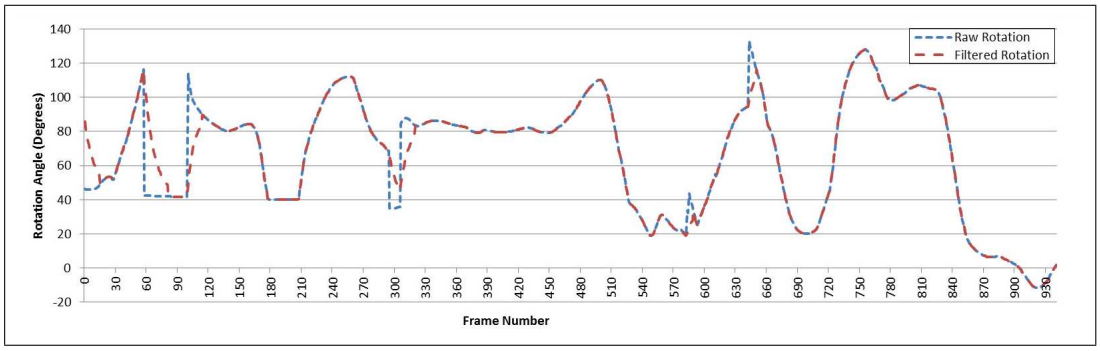


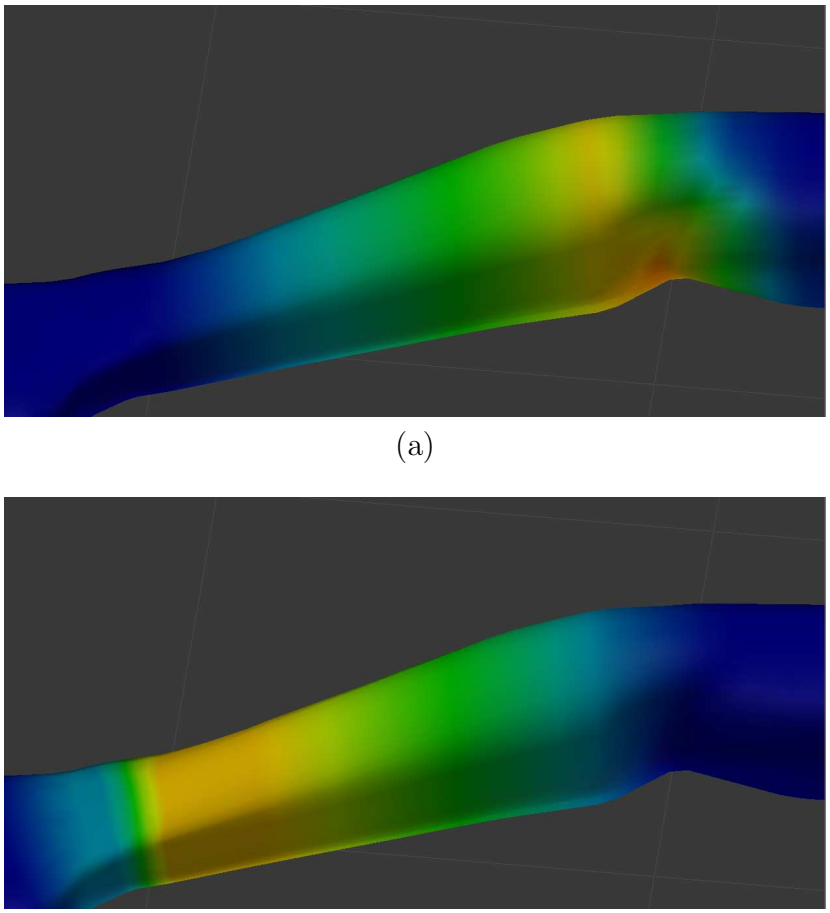
Figure 4.1: The row and filtered samples for right the humerus roll angle.

4.4.3 Bone Splitting

The lower sections of human limbs contain two parallel bones allowing the twisting rotation on the hands and feet. The configurations of bones allow all portions of the lower limbs follow the bones in pitch and roll rotations, although the effect of yaw rotation decreases as it gets closer to the mid-section joint (elbow or knee). This effect is not possible to achieve with a single bone fore-arm representation, as specified in the highest level of detail in the H-ANIM standard [4], using unified weights (same for all types of rotations, transformations and scaling) and linear skinning. On the other hand, applying a different set of weights for each possible transformation, rotation or scaling requires additional space and time, which can be considered redundant as it is not going to be used in most parts of the skeleton and surface mesh; thus, it is not implemented in most of the popular rendering engines. This problem is addressed by Kavan et al. [19], proposing a method of introducing additional blending bones to simulate non-linear skinning. However, this approach is not suitable for a real-time application with previously unknown motions.

Since the only problematic bones for this particular case are the upper limbs, this problem is solved using a novel approach by introducing an additional bone connected in series for the upper limbs. The ulna bones are split halfway and the lower sections are labeled as ulna-extension. The vertex weights in the corresponding sections are divided linearly among two sections, as seen in Figure 4.2.

During runtime, the filtered local rotation of the upper limb bones are separated into two distinct rotations, one containing the yaw and the other containing pitch and roll rotations, which are applied to the extension bone and the original bone, respectively. With proper weights, the rotation of the users arm is transferred to the virtual avatar naturally without introducing any artifacts. As seen in Figure 4.3, the vertices on the forearm twist in a more natural way resembling the real motion.



(a)

(b)

Figure 4.2: The vertex weights for (a) the upper ulna and (b) the lower ulna bone (weight increases from blue to yellow).

(a)

(b)

Figure 4.3: Comparison of a -90° yaw rotation on the forearm with: (a) single and (b) double-boned skinning.

4.5 Handling the Foot Skating Problem

The virtual model is animated by changing bone orientations and root bone position. If the orientations and root position are applied to the bones respectively, the feet of the character appear to be floating on the ground that deteriorates the realism. This problem, known as footskating, is not limited to depth sensor applications. It shows up in motion-capture-based animation systems and solved in a reasonable way with approaches, such as Kovar et al. [76] and Ikemoto et al. [77]. However, these methods rely on some sort of preprocessing or supervised learning, which is not suitable for our system, as the motion is captured and applied im real time. Hence, a solution that does not require a training process is needed for such applications. Mankyu and Choi [78] propose a similar method that does not require training for real-time depth sensor applications. Their approach is adopted into our framework to overcome the footskating problem. However, it requires additional filters and constraints in order to be usable in our framework. It is assumed that one foot is always on the ground; otherwise, artifacts may occur, for example, if the user attempts jumping. The footskating handling algorithm consists of two parts: *constraint checking and adaptation* (cf. Algorithm 3) and *joint angle determination and application* (cf. Algorithm 4). The whole algorithm consists of the following four steps.

1. Determine which foot is constrained by checking speed and location thresholds. The thresholds are changed adaptively in order to compensate for different sensor and user placements.

2. Place the constrained foot at its last recorded position and solve the inverse kinematics problem to determine the orientations of the hip and knee joints. Solving an inverse kinematic problem is no trivial task. There are various numerical and analytical methods for inverse kinematic problems and many implementations focusing on each. We use the IKAN library by University of Pennsylvania [79], which uses the approach described in [80].

3. Check if the foot coincides with its intended position after the inverse kinematic orientations are applied to the bones. If there is a positional mismatch, relocate the root joint to complete alignment.

4. Smooth the joint orientation difference in order to avoid jumps from frame to frame. This is especially important when the constraint switches, as the source of the orientations are different in two cases and they do not always overlap. Smoothing parameters are optimized using trial and error; they are not viable for adaptive nature. Smoothing also helps to overcome the self-occlusion and data-noisiness.

Joint smoothing consists of interpolating between frames, especially when there is a significant change in orientations, which would normally cause a non-continuous animation. Old joint orientations are always recorded in the global coordinate system and updated everyframe. After we get the new orientation from the new frame, we calculate the quaternion that would rotate the old orientation to the new one:

$$Q_d = Q_{new} \times Q_{old}^{-1} \tag{4.5}$$

When we compute the delta quaternion, we get its axis-angle representation. Because we are interested in rotating the joint partially, we build up a new quaternion with the same axis and a fraction of the same angle. The fraction varies with whether the constraint is switched recently.

$$\begin{aligned} Q_d &= Quaternion((x, y, z), \alpha) \\ Q_d^* &= Quaternion((x, y, z), \alpha/k) \end{aligned} \tag{4.6}$$

The fractional rotations are then applied to the joints to get a smoother movement. Other implicit operations include coordinate system transformations, as we are working with three different coordinate systems: Global-Render Coordinate System, Local-Render Coordinate System, and IKAN Coordinate System.

Algorithm 2: Mesh update algorithm called at every frame

```
1  function prepareBlendMatrices(mesh)
2      foreach bone do
3          bone.applyScale()
4          bone.applyTransform()
5          bone.applyOrientation()
6      i = 0
7      foreach bone do
8          m[i] = bone.getTransformationMatrix 4 × 4
9          i++
           /* Index map contains the bone pointers for every vertex
       */
10
11     mapIndex=mesh.getIndexMap() i = 0
                 /* Blend matrices are pointers to the individual bone
       matrices */
12
13     foreach indexSet in mapIndex do
14         m_b[i] = indexSet[i] + m i++
15     return m_b
16 function vertexBlend(m_b)
17     pos=*mesh.positions                /* Pointer to position matrix */
18
19     norm=*mesh.normals                   /* Pointer to normal matrix */
20
21     b_i =*mesh.blendIndices          /* Pointer to blend index matrix */
22
23     b_w =*mesh.blendWeights        /* Pointer to blend weight matrix */
24
25     foreach 4 vertices in pos do
26         foreach vertex in 4 vertices do
27             m[1, 2, 3, 4] = m_b[b_i[vertex]]              /* Weighting Bones */
28
29             m_c[j] = collapseMatrix(m, b_w[vertex])(i)
30         pos[4vertex] = m_c × pos[4vertex]
           norm[4vertex] = m_c × norm[4vertex]
31 if skeleton needs update then
32     m_b = prepareBlendMatrices(skeleton.mesh)
33     vertexBlend(m_b)
```

Algorithm 3: Constrained foot determination

```
if isFirstFrame then
    if p^y_left < p^y_right then
        constrained = left
        y_threshold = p^y_left
    else
        constrained = right
        y_threshold = p^y_right
if constrainedisleft then
    /* Check if constraints still hold                              */
    if p^y_left > y_threshold and v_left > v_threshold then
        /* Foot is not constrained anymore, check other foot        */
        if p^y_right > y_threshold and v_right > v_threshold then
            /* Neither foot are constrained                         */
            updateThresholds()    /* Thresholds should be updated */
            restart
        else        /* Right foot is constrained, feet switched     */
            constrained=right
            recordRightFootPosition()
            constraintSwitched = True
else
    /* Check if constraints still hold                              */
    if p^y_right > y_threshold and v_right > v_threshold then
        /* Foot is not constrained anymore, check other foot        */
        if p^y_left > y_threshold and v_left > v_threshold then
            /* Neither foot are constrained                         */
            updateThresholds()    /* Thresholds should be updated */
            restart
        else        /* Left foot is constrained, feet switched      */
            constrained = left
            recordRightFootPosition()
            constraintSwitched = True
```

Algorithm 4: Foot skating filtering

checkFootConstraints() `/* Check which foot is constrained */`
$p^{foot}_{constrained} = getRecordedFootPosition()$
$target_{ik} = p^{foot}_{constrained} - p^{hip}_{constrained}$
`/* Get the rotation quaternions for constrained hip and knee */`
$q_{hip}, q_{knee} = solveIkan(constrained,\ target_{ik})$
$hip_{constrained}.rotate(q_{hip})$
$foot_{constrained}.rotate(q_{hip})$
$p^{foot}_{new} = calculateForwardKinematics()$
`/* Calculate the root displacement vector */`
$d_{root} = p^{foot}_{constrained}$ - p^{foot}_{new}
$root.translate(\mathrm{d}_{root})$
`/* Proceed with joint smoothing */`

Chapter 5

Cloth Simulation

Rendering physically accurate apparel meshes is the goal of the physics component of the framework. Because the construction of a real-time high quality physics engine that supports collision detection is a study subject on its own, The Nvidia PhysX engine is utilized as the physics solver [50] in the simulation software. The quality of the simulation depends on a number of different steps; the model preparation, physics environment initialization and the real-time simulation including numerical solution of the spring mass sytems and collision handling.

5.1 Model Setup

The mesh to be simulated is the dynamic part of the modeled dresses that contains vertices in the order of thousands. It is aligned precisely with the static part and the human body to be simulated realistically. The PhysX framework requires vertex and topology information delivered separately and manually, instead of a binary file or an automatic parser. For this purpose, a custom Wavefront object file (OBJ) parser is implemented to acquire the information exported from a modeling suite, and feed the data into the physics engine.

Other than the vertex and topology information, an inverse weight property needs to be specified for every vertex. This information could be embedded into the wavefront object as a second set of texture coordinates. On the other hand, this method adds too much data to the file, which brings extra computational overhead. Therefore, the vertex weight information is embedded into the model by specifying the material properties separately for vertex groups with different weights. The vertices that have a non-zero inverse weight have the suffix "Free" attached to their material properties. The vertices are selected manually and their material properties are attached. The current simulation required only two sets of weights, which can be seen in Figure 5.1. After the weight information is implemented, the model is exported as a Wavefront object file (OBJ) along with the material file (MTL) and parsed into the software.

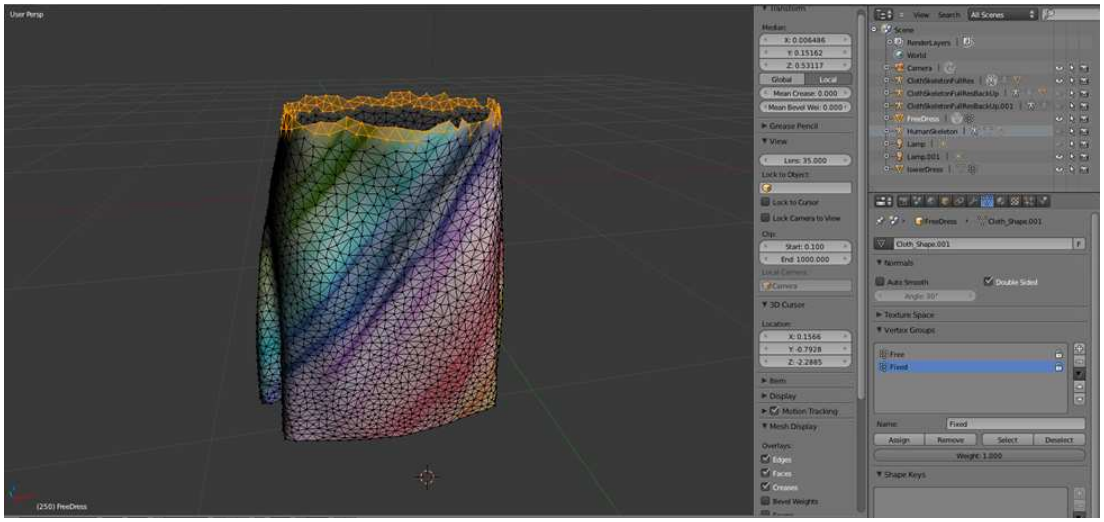


Figure 5.1: The fixed vertices of the cloth.

5.2 The Initialization

Following the parsing of the model from the object file, it is stored within the memory to be used in different frames. The object file is converted into the native mesh format for the rendering engine to be loaded and rendered. The process includes feeding the vertex, topology and material information into the

OGRE framework as submeshes, to be combined as a mesh. Prior to creating the simulated cloth, the PhysX engine needs to be initialized:

1. The foundation and SDK objects are created.
2. In order to implement the GPU solver for the cloth and collision, the PhysX engine needs to bind with the CUDA context to deliver the GPU tasks. The CUDA Context manager object is created and given to the SDK object.
3. Afterwards, the virtual floor and the environment are created with the gravity specified as 9.8 m/s^2.

After the initialization, the cloth is loaded with specific stiffness, stretch, damping, friction, inertia, bending, and collision parameters. These parameters are decided through numerous try-and-error experiments in order to acquire the most realistic simulation results. After creating the environment and the virtual cloth, the collision spheres are created, which is explained in Section 5.5.

5.3 The Animation

At each frame, the following steps are performed:

1. The passed time since the last frame is added to the counter.
2. The Kinect sensor is checked for new data. If the data is not new, next frame is called.
3. If there is an active user with the Kinect, the sensor middleware pointer is sent to the human body and the upper cloth *SkeletalMesh* objects to be updated. If there are none, the bones are reset to their initial state.
4. The upper body mesh and the human body mesh are updated. The details of the update process can be found in Section 4.2. This function returns the translation of the root node.

5. The returned vector from step 4 is used to translate the dynamic cloth handle. After the translation, the orientation is updated with the root bone orientation.

6. The colliding human capsules are updated with the human body bone orientation. The orientation and the position of the dynamic cloth handle is synchronized with the virtual cloth. This input automatically introduces the inertia effect on the cloth.

7. At the end, the cloth is simulated as follows:

   (a) PhysX is ordered to start the simulation on the GPU. The details of the simulation algorithm are explained in Section 5.4.

   (b) The vertex data is read from the output, and the vectors of the object file are updated, except the fixed mesh vertices. The reason for omitting the fixed vertices is to avoid unnatural bends and cracks on the fixed vertices.

   (c) The updated object file vectors are transformed into the native mesh buffers, to be rendered.

The cloth simulation algorithm is based on the position-based dynamics, introduced by Müller et al. [81]. The main approach is to calculate the position of the particles from their previous positions and applying constraints on mutual distance and angle. The collision is also calculated as a force and applied to both particles. The approach is stable and efficient for real time applications. The dynamics are stable as long as the constraint solvers converge. When this criteria is not met, rendering anomalies occur, such as a vertex being pulled too far away from the cloth. The solution is parallelized by fibers, as Single Instruction Multiple Data (SIMD) process. With CUDA support, however, this is even parallelized more with Single Instruction Multiple Thread (SIMT) processes, where each thread works on one fiber.

5.4 Numerical Solution

The position of a particle in the next time interval is acquired by performing explicit Euler integration over δt, where the velocity and the force are assumed to be constant during the interval. The pseudo code of the integration procedure is given in Algorithm 5.

Algorithm 5: Position-based dynamics solver

```
1  foreach vertices i do
2      initialize x_i = x_i^0, v_i = v_i^0, w_i = 1/m_i
3  while true do
4      foreach vertices i do
5          v_i ← v_i + Δt w_i f_ext(x_i)
6      dampVelocities(v_1, ..., v_N)
7      foreach vertices i do
8          p_i ← x_i + Δt v_i
9      foreach vertices i do
10         generateCollisionConstraints(x_i → p_i)
11     while solverIterates do
12         projectConstraints(C_1, ..., C_{M+M_coll}, p_1, ..., p_N)
13     foreach vertices i do
14         v_i ← (p_i − x_i)/Δt
15         x_i ← p_i
16     velocityUpdate(v_1, ..., v_N)
```

Algorithm 5 predicts the next position and velocity (lines 1-2), performs the corrections by solving the constraints (lines 3-12), updates the position and velocities accordingly (lines 13-15), adds damping to velocities (line 16). The key issue in the simulation is the position correction due to the constraints. Each vertex is moved either towards or away from each other, the distance is scaled by the inverse mass of the vertices. If a vertex position needs to be fixed, the inverse weight should be set to zero

5.4.1 Constraints, Fibers and Sets

In order to simulate the cloth, all constraints should be solved, which is achieved through linearizing the non-linear problem. This linearization results in a sparse matrix problem. Although the problem is solvable in real-time, performance is increased further by pivoting vertical and horizontal constraints, solving separately. The vertical and horizontal constraints are divided in the input sense by fibers and sets. Fibers represent independent sets of connected constraints and sets are non-overlapping fibers, which are solved in parallel. A fiber is either a horizontal, vertical or a diagonal line, and the set is the collection of parallel lines. These fibers and sets are generated for both shear and stretch constraints by the PhysX mesh cooker, which auto-generates the fibers and sets from a given mesh topology.

5.4.2 Set Solvers

There are two possible solvers to apply on fiber block (set) which come with PhysX:

- The Gauss-Seidel solver continues along the fiber after completing a constraint solution and updating the results. This solver is easy to tweak and use, has low-cost for iteration, however the convergence factor is low due to sequential update, which results in a stretchy cloth.
- Semi-implicit solver factorizes the tri-diagonal system with LDLT and solves the overall system. This method preserves momentum since it is not sequential and converges ten times more than Gauss-Seidel solver. However, the matrices can be ill conditioned, which requires special treatment, iteration cost is higher and tweaking is more difficult.

In order to get the best performance from these solvers, they are applied on different sets, taking advantage of parallel implementation. The Gauss-Seidel solver is used for horizontal stretch fibers and the shear fibers, which do not

experience too much stretching, taking advantage of its low cost. A semi-implicit solver is applied only to the vertical stretch fibers, resulting in both a fast and robust solution.

5.5 Collision Handling

Preventing the apparel meshes intersecting with the virtual avatar and itself is attained with collision detection and prevention. Cloth vertices can collide with pre-defined spheres, capsules or planes. In order to keep the program stable and running at real-time, it is important to find the balance between collision detail and performance.

The female body bone information is extracted from the input skeletal mesh class, and the colliding capsules are generated automatically, although the radii of the bones are either specified manually or input through the user measurement process 7. The collision information is specified in two arrays, one being the positions and radii of the spheres and the second defining pairs of spheres which form capsules. A total of 28 capsules and spheres are used in the whole collision model, which simulates all the movable bones of the human body (see Figure 5.2) plus a few additional bones for regions which are not close enough to a bone. The collision data is prepared before the cloth creation, and given as a parameter.

Other than the defined collision spheres and capsules, the cloth naturally collides with the floor actor of the PhysX environment as well. This, however, introduces a problem: the models from Blender are exported as root joint coinciding with the origin. This is required for successful animation. However, in the PhysX environment, the floor is created automatically at (0,0,0), and the initial position of the lower part of the apparel is penetrates the floor, which produces unrealistic visuals. In order to overcome this issue, the lower cloth vertices are introduced into the software with modified y coordinates, in order to keep them above ground. In the rendering process, the lower part of the apparel is attached to a scene node with negative y offset equal to the boost in the vertices, which

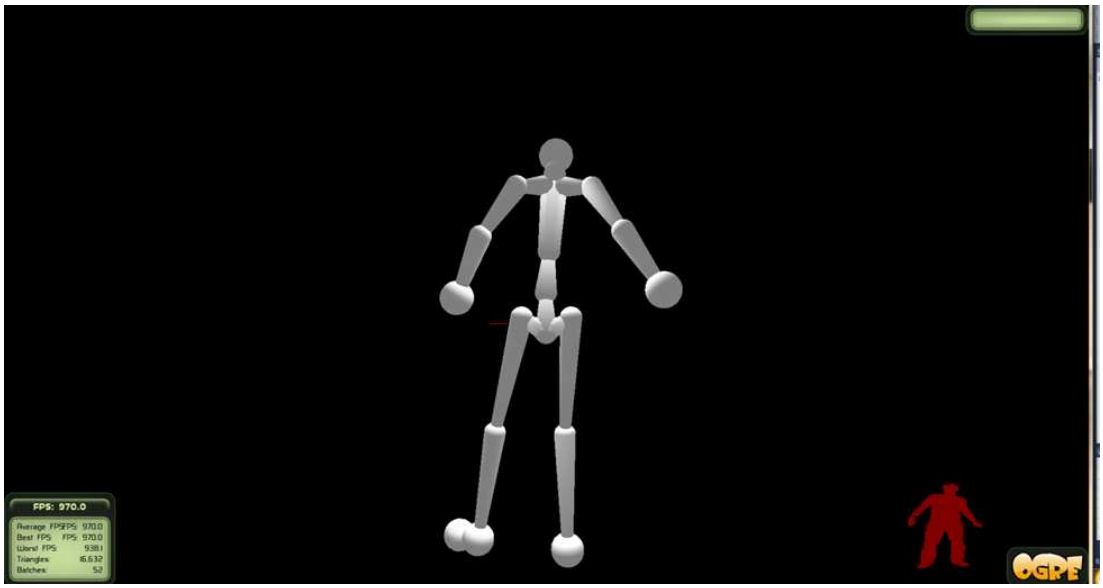


Figure 5.2: Character formed with collision spheres and capsules.

places the lower part of the apparel exactly where it should be. Every frame, the collision sphere positions are updated with the data from the updated body skeleton, prior to the cloth simulation.

PhysX provides two options for collision detection: *discrete and* continuous collision. The framework works with the latter due to more robust results, although it takes twice as long to do the calculation. The solution is performed for the trajectory of the capsule or sphere and the particle for frame interval. This approach is especially robust in fast motion, which is important because the motion is created in real time. The required solution is a sixth degree polynomial, which is approximated with a quadratic equation. The solution is performed on the GPU, which increases the performance greatly, allowing for frame rates of 600+ frames per second (fps). The cloth is discretized as a triangle mesh, and because the collision is only detected on vertices, the density must be high enough in order to avoid penetration [50], [82].

Chapter 6

Cloth Resizing

For a realistic fitting experience, another requirement is acquiring a set of simulation parameters from a human test-subject for a pre-modeled apparel mesh, which is to be displayed on a virtual avatar reflecting the body characteristics of the aforementioned subject. The set of parameters for the simulation includes the body height and width, and the radii for the collision spheres, which have their centers coinciding with the joints of the virtual avatar's skeleton. The body width and height are then utilized to estimate the body size of the user, collision spheres are used in the dressing room simulation to detect and resolve collision with the cloth particles.

As opposed to the studies focusing on acquiring high-detail avatars of subjects for made-to measure applications, this study focuses on the speed of the overall algorithm for real-time purposes and acquire just enough measurements for a 'fixed-size try-on' system. The cloth resizing algorithm consists of three stages:

1. Improve the raw depth map from Kinect by filtering.
2. Fit the contour on the user blob and perform measurements. Compare with known human body proportions to acquire required scaling parameters.
3. Perform the same measurements over a time frame in order to smooth the results. Scale the avatar along with the cloth mesh prior to simulation.

Distance of Point	1m	3m	5m
Error in measurement	0.5cm	1.5cm	4cm

Table 6.1: Kinect depth accuracy.

6.1 Depth Map Optimization

At 30 frames per second (fps), Kinect provides a depth map and a user map, both at 640 × 480 resolution. The depth map contains the distances of pixels from the sensor in millimeters. The depth measurement of the Kinect sensor is not very accurate compared to high-end 3D depth systems such as laser scanners. The error in the depth value increases quadratically with the distance. The error values for different distances are shown in Table 6.1 (please see [83] for details of the accuracy and resolution of Kinect depth data).

For the purposes of this study, the Kinect sensor needs to be able to see the whole human body, which requires at least 3m away from the sensor for a person with 1.7m height, resulting in an erroneous depth map. In order to acquire a better depth map, the following operations are performed: Let us take the depth map D as a 640 × 480 matrix. Initially, the user pixels are extracted by a pixel-by-pixel comparison with the user map. The user map is another acquired map from the sensor, with the same size as depth map. The value of a pixel is set to a non-zero value if the pixel belongs to a recognized user. In this case, we are only interested in one user; D_1 represents the depth pixels of the user and U_1 is the bit map of the user. Besides, the non-user pixels must be filled with the mean value of the user pixels in order to perform Gaussian filtering on the image.

$$D_1 = (D - (D \times U_1)) \times \frac{1}{n} \times \sum_{i=0}^{n}((D \times U_1)_i + d \times U_1) \tag{6.1}$$

Next, we perform Gaussian filtering on the user's depth map, to normalize and improve the quality of the depth map. The size and sigma parameters of the Gaussian filter are selected through try-and-error experimentations in order

to maximize the performance and the quality of the results.

$$D_G = D_1 * G, \tag{6.2}$$

where '$*$' is the convolution operator. The Gaussian filtering completes the enhancement of the input depth map. After these operations, we have a normalized and filtered depth map, which also has better planar values (x and y) since the holes due to depth stream will be filled.

Algorithm 6: Depth map optimization algorithm

Input: Raw Depth Stream From Kinect
Output: Depth Stream With Patched Holes and Gaussian Optimization

```
1  depth_sum = 0
2  n_user = 0
3  for i from 0 to d_width do
4      for j from 0 to d_height do
5          if U(i, j) then
6              depth_sum = depth_sum + D(i, j)
7              n_user += 1
8  depth_average = depth_sum / n_user
9  for i from 0 to d_width do
10     for j from 0 to d_height do
11         if not U(i, j) then
12             D(i, j) = depth_average
13 for i from 0 to d_width do
14     for j from 0 to d_height do
15         if U(i, j) then
16             D(i, j) = D(i − m : i + m, j − n : j + m) * Gaussian(m, n, e)
17 return D
```

6.2 Parameter Measurement

In parameter measurement, there are two objectives to be handled: To determine the optimal sizes of collision spheres for cloth simulation and the required scaling

parameters for the cloth to optimally fit the user. It is important that these algorithms do not take more than a thirtieth seconds on a high-end consumer computer in order to keep the real time experience smooth, since there is an averaging over time is involved.

The first step is fitting spheres in various locations in the optimized body map. These fitted spheres provide the radii for the collision spheres, which are used to simulate the cloth. Locations of the machine-provided user joints are where spheres are located. The pseudo-code of the sphere fitting is given in Algorithm 7. It performs the following steps:

1. Take vector J_i, which represents the coordinates of the i^{th} joint. First, initialize the radius of the sphere by the difference of z-dimension with the overlaying point in the depth map.

$$r_i^z = J_i^z - D^z(J_i^x, J_i^y) \tag{6.3}$$

2. Repeat the same process for the x and y dimensions in both negative and positive directions. Take the bigger radius. If there are no points on either side, set it to zero.

$$r_i^{x,y} = max(\| \pm J_i^{x,y} \mp D^{x,y}(J_i^{y,x}, J_i^z)\|) \tag{6.4}$$

3. The radius of the sphere is equal to the minimum of these three values.

$$r_i = min(r_i^{x,y,z}) \tag{6.5}$$

6.3 Human Body Parameters

The next step will be acquiring the optimal scaling parameters for the cloth. The width and height of the subject is important for determining the proper actual size for the cloth. However, a straightforward estimation of the body height and shoulder width is prone to errors due to the noise and quality of the

Algorithm 7: Sphere fitting algorithm

Input: Optimized depth stream from Kinect
Output: Collision sphere radii for each joint

foreach *joint* **do**
  $p = pos_{J_m}$
  $r_z = \sqrt{P_z^2 - D_z(P_x, P_y)^2}$ **for** i **from** P_x **to** 0 **do**
    if $D(i, P_y)$ **equals** P_z **then**
      $r_x^- = i$
      break
  for i **from** P_x **to** $\boldsymbol{depth_{width}}$ **do**
    if $D(i, P_y)$ **equals** P_z **then**
      $r_x^+ = i$
      break
  for j **from** P_y **to** 0 **do**
    if $D(P_x, j)$ **equals** P_z **then**
      $r_y^- = j$
      break
  for j **from** P_y **to** $\boldsymbol{depth_{height}}$ **do**
    if $D(P_x, j)$ **equals** P_z **then**
      $r_y^+ = j$
      break
  $r_m = min(r_z, r_x^-, r_x^+, r_y^-, r_y^+)$
return $(r_0, r_1 \dots r_n)$

incoming depth map. In order to minimize this error, an upscaled version of the subject's body is measured, to be used in width-height estimation later. We use the human body proportions specified in [84], which are shown in Table 6.2. In these proportions, the unit width and height are taken as the width and height of the head. The measurement source column in the table represents the source for the estimation of the respective parameter:

- The joint location represents that the measurement will take the input subject joint locations as the reference.
- The depth map represents the measurement will instead perform measurements based on the pixel distribution in the filtered subject depth map.

Distance	**Width**	**Height**	**Measure Source**
Head	1w (1)	1h (2)	Depth Map+Joint Location
Body Height	-	7 (3)	Depth Map
Hip Height	-	4 (4)	Joint Location
Elbow-Fingertip	-	2 (5)	Depth Map+Joint Location
Wrist to Fingertip	-	1 (6)	Depth Map+Joint Location
Shoulder Width	3 (7)	-	Depth Map+Joint Location
Hip Width	- (8)	-	Depth Map
Torso Height	-	- (9)	Joint Location

Table 6.2: Human body proportions. Numbers in parenthesis represent the lines on Figure 6.1.

Type of Cloth	**Primary Height Proportions**	**Primary Width Proportions**
Trousers	Hip Height	Hip Width
Long Sleeves	Body Height	Elbow-Fingertip Height, Shoulder Width
Short Sleeves-Sleeveless	Torso Height	Shoulder Width

Table 6.3: Primary proportions for different cloth types.

They are often used together for better performance. Please note that some of these parameters are not standard to be used as relative references, such as the hip width. These parameters do not effect others in the estimation process and vice versa.

Along with the ratios, the actual size in meters in height and width will be measured and recorded as well because the cloth needs to be scaled according to the user. In the proposed approach, the whole cloth is scaled as a whole, with different parameters for three dimensions. This approach qualifies as the best option for the purposes of this application, as most shops do not offer extensive customization. Because different types of cloth focus on different portions of the body, the human body proportions from different areas should not affect the scaling parameters in the same way. The main body parameters for different types of clothes are listed in Table 3. The pseudo-code of the parameter estimation is

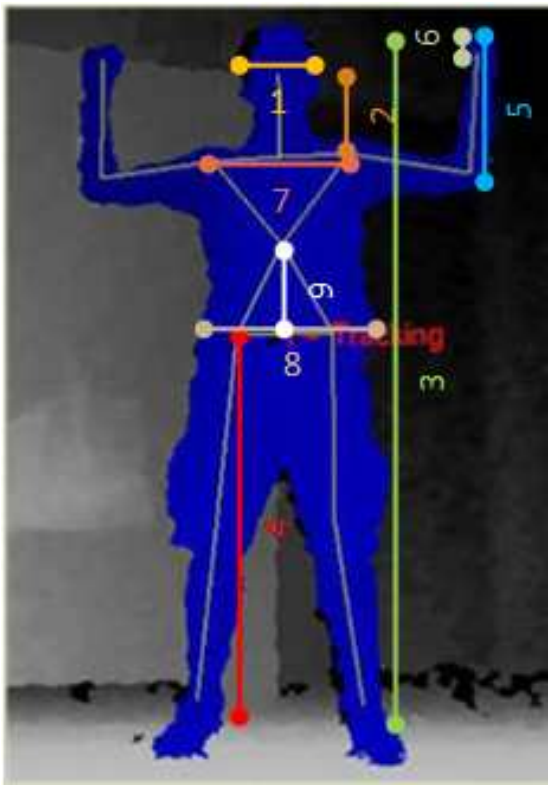


Figure 6.1: Proportions of the body.

given in Algorithm 7. It performs the following steps:

1. For a particular cloth, take the primary measured proportion as P_i^0. This will be the measured dimension of the corresponding proportion. This process will be repeated for width (W) and height (H).

2. With all other measured proportions, calculate the estimated value of P_i as P_i^j. Here, R denotes the ratio from Table 6.2.

$$(W, H)_i^j = (W, H)_j \times R_i^j, \quad j = 1, \ldots, N \tag{6.6}$$

3. Find the optimized value of the main width parameter as the average:

$$(W, H)_i = \frac{1}{(n+1)} \times \sum_{j=0}^{n} (W, H)_i^j \tag{6.7}$$

4. After finding the optimized value of the main parameter in meters, it can be used to scale the virtual cloth by calculating the ratio.

Algorithm 8: Cloth resizing algorithm

$t_{proportion}$ = import(Table 6.2)
$t_{primary}$ = import(Table 6.3)
$ct = cloth_{type}$
$width_{main} = t_{proportion}.width(ct)$
$width_{sum} = 0$
$count_{effector} = 0$
foreach *width* **in** $t_{proportion}$ **do**
    $w_i = measure(p_i)$
    $w_i^j = w \times t_{proportion}.ratio(p_i, parameter_{main})$
    $width_{sum} = width_{sum} + w_i^j$
    $count_{effector}++$
$width_{weighted} = \frac{width_{sum}}{count_{effector}}$
$x_s = \frac{width_{weighted}}{width_{cloth}}$
$height_{main} = t_{proportion}.height(ct)$
$height_{sum} = 0$
$count_{effector} = 0$
foreach *height* **in** $t_{proportion}$ **do**
    $h_i = measure(p_i)$
    $h_i^j = h \times t_{proportion}.\ ratio(p_i, parameter_{main})$
    $height_{sum} = height_{sum} + h_i^j$
    $count_{effector}++$
$height_{weighted} = \frac{height_{sum}}{count_{effector}}$
$y_s = \frac{height_{weighted}}{height_{cloth}}$
return (x_s, y_s)

6.4 Temporal Optimization and Scaling

After performing Step 2 of the cloth resizing algorithm (cf. Algorithm 7), we acquire the following usable parameters:

- collision sphere radii, (r_i) and
- cloth and avatar scaling parameters (x_s, y_s).

By now, the required body dimensions and collision sphere parameters for a realistic simulation are acquired. Yet, the measurements are performed on a

filtered version of a depth sensor with high error rates. In order to overcome the noise and overall depth-sense faults, the prior measurements are repeated for the duration of one second, which corresponds to 30 frames of input depth map. A considerably different approach here would be to employ the temporal averaging on the depth map instead of the measured parameters. However it is observed that the results suffer due to the motions of the subject because most subjects fail to keep their exact form for one second. To overcome these problems, temporal averaging that takes the mean of the specified parameters for the frames in one second is used. This step finalizes the parameters and delivers the required parameters for simulation environment synthesis. The averaging process includes summing up all the information from previous time frames and dividing the result buy the number of samples. After the optimized parameters are acquired, the cloth and body meshes are scaled accordingly. The mesh parameters for three axis will be as $(x_s, y_s, avg(x_s, x_y))$, because there is not enough information on z-dimension to make accurate calculations. Therefore the value will be according to the scaling factors for x and y dimensions.

Algorithm 9: Temporal averaging

Input: Raw depth stream from Kinect
Output: Depth stream with patched holes and Gaussian optimization

```
1 s = 2 × 30 Array for x and y scaling parameters for 30 frames
2 r = 16 × 30 Array for joint radii for 30 frames
3 for i from 0 to 30 frames do
4     r[i]=fitSpheres()
5     s[i]=optimizeScaleParameters()
6 r_final=avg(r)
7 s_final=avg(s)
```

Chapter 7

Experiments

Experiments are performed on a high-end PC with Intel i7-2600 and NVIDIA GeForce GTX560Ti. The frame rate of the application varies, depending on the complexity of the apparel mesh and whether the avatar is female or male. If the apparel mesh contains a large number of vertices, the physics simulation takes more time and the frame rate drops. The accessories of the female mesh such as the hair and earrings are very highly detailed and cause a considerable drop in the frame rate. The minimum acceptable frame rate is determined by the simulation rate of the physics environment. Simulation rate of the physics environment determines the timestep in the numerical solution of apparel simulation and collision solutions. Higher simulation rates produce better physical results, however they also increase the required computational power, decreasing the overall frame rate. If the simulation rate is specified higher than 150 frames per second(fps), the overall frame rate drops below 150 fps in the complicated models, nullifying the benefit of higher number of iterations of the physics solver. Through try-and-error experimentation, 120 fps is found to be the optimal physical simulation rate, producing good quality physical results with sufficiently high frame rates. Two sets of frame rates across the simulation can be seen in Figure 7.1. Notice the common pattern in both of the simulations. The first drop corresponds to the start of the animation of the avatar with the input from depth sensor. Until that point, the avatar is static, the motion filters do not run and the simulation runs

at a higher speed. Approximately 600 fps is lost due to the motion filtering and skinning. The first cliff in the graph corresponds to the point where the physics simulation is stopped. The second cliff starts when the input from the depth sensor is stopped and the simulation is stagnant.

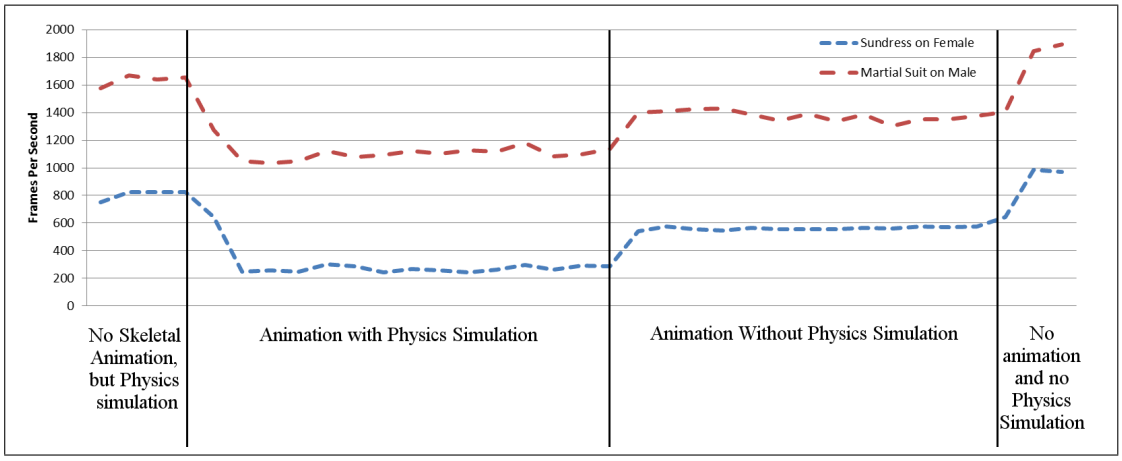


Figure 7.1: The frame rates for two different apparel meshes.

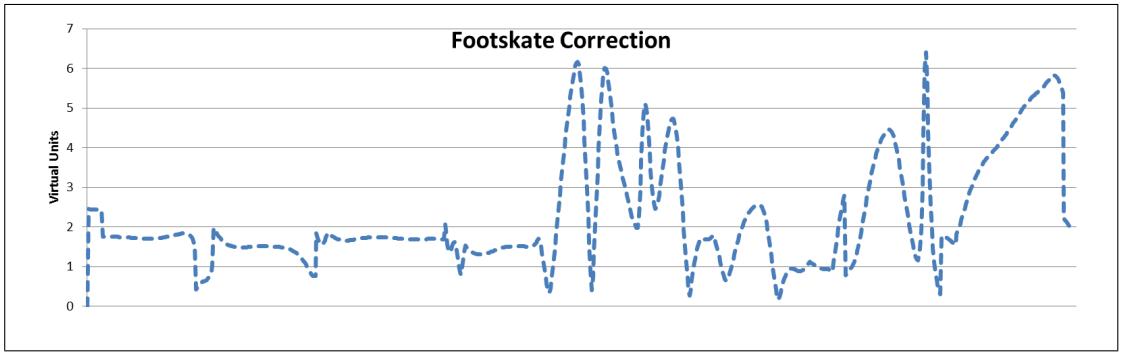


Figure 7.2: The corrected displacement of feet. The local minima correspond to constrained foot changes and subsequent position smoothing. The zig zag regions correspond to time intervals where the user performing body yaw motion where the foot are considerably sliding.

Aside from the overall motion smoothing, the main motion filters are dominant over the arms and the legs. The effects of motion filtering and smoothing on the arm can be seen in Figures 4.3 and 4.1, respectively. The filters on the legs focus on solving the foot skating problem. The corrected displacement values can be

	Shoulder Width (cm)				**Body Height (cm)**			
Subject	Real	Estimated	Error (%)	Deviation	Real	Estimated	Error (%)	Deviation
1	43	43.6	1.0	7.8	172	170.8	0.6	2.5
2	46	44.3	3.0	8.5	176	172.1	2.0	1.4
3	51	48.8	4.0	12.1	176	174.0	1.0	6.6
4	48	50.0	4.0	2.5	188	191.0	1.5	7.1
5	44	42.6	3.0	4.2	178	175.8	1.2	3.1

Table 7.1: Performance figures for five different subjects.

seen in Figure 7.2. The virtual unit is the unit step in the virtual world. For reference, the virtual avatars' height is 55 virtual units. Hence, a virtual unit corresponds close to 3.5 cm.

The measurement process to identify the body parameters starts after a subject is recognized, identified and calibrated for tracking. Following the parameter estimation, the virtual avatar and cloth are created and the simulation starts. The total time for the measurement of body parameters and resizing the cloth does not exceed 1.005 seconds. Considering the time for acquiring 30 frames of input from the depth sensor is 1.0 second, it is safe to say that the measurement algorithm does not introduce any delays for a real time application because it needs to run only once at the beginning of the simulation. The body sizes are estimated with error rates less than 4%, which is sufficient for the realism of the simulation and determining the appropriate apparel size. Table 7.1 presents the measurements of the body dimensions, as well as the errors and standard deviations of the measurements in 30 frames for three subjects. The results are compared with the results from [59] and [85], which are also real-time body size estimation techniques using Kinect depth sensor for virtual try-on applications, in Table 7.2.

For the collision spheres, the quality of the results can be assessed by the smoothness of the collision simulation, as seen in Figure 7.3. Throughout the

	Error Average (%)	Error Deviation (%)	Duration	Estimation
Our Approach	2.13	1.20	1.00	Fixed
Giovanni et al. [59]	4.00	3.00	1.00	None
Samejima et al. [85]	6.72	4.68	n/a	PCA

Table 7.2: Performance comparison with other state-of-the-art approaches regarding height measurements.

simulation, unnatural intersections between the cloth and the avatar never take place, while the cloth appears to rest on the skin naturally, without space between the two meshes. Figure 7.4 shows examples of three garments on a model with different postures generated with our implementation. The six apparel meshes, three for female avatar and three for male avatar can be seen in Figure 7.5.

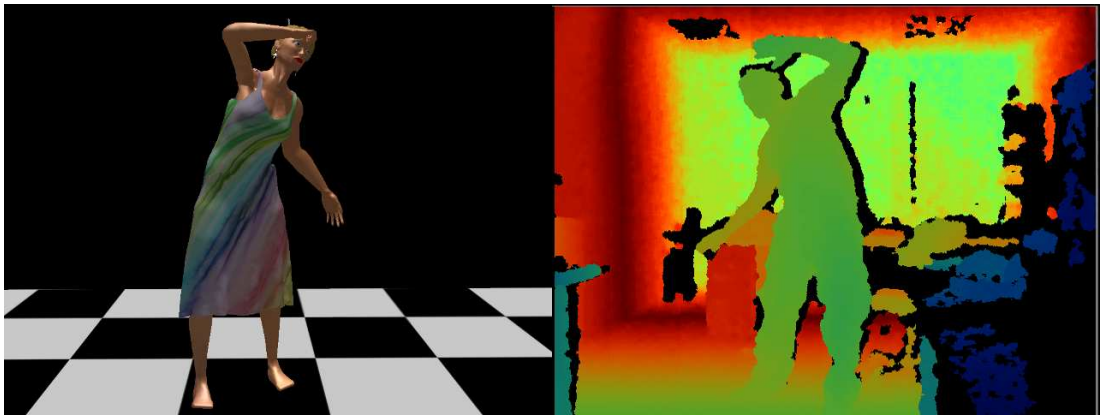
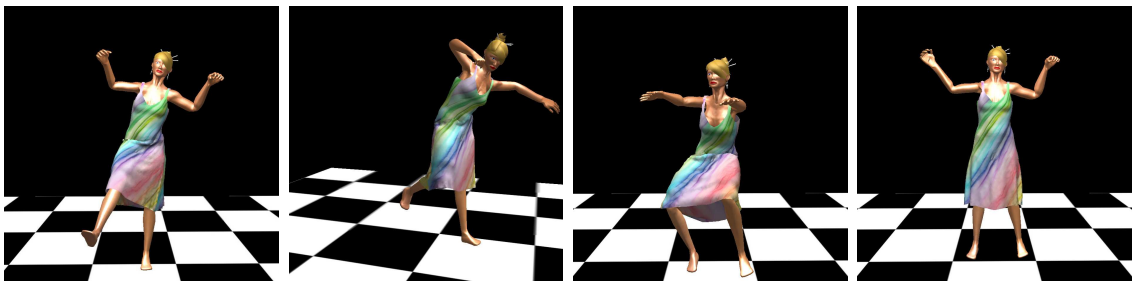
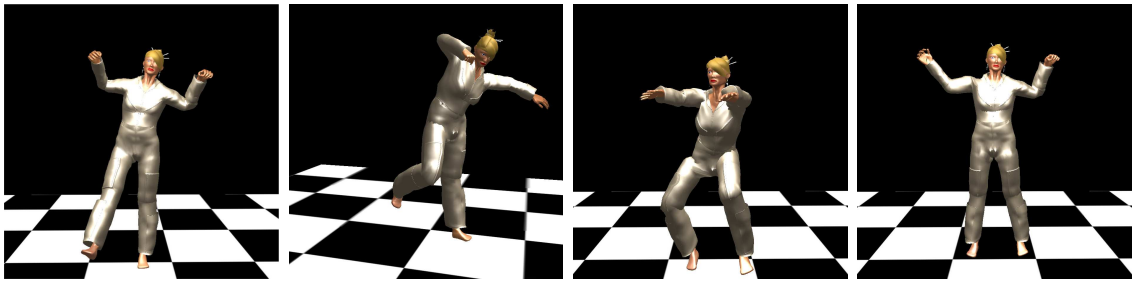


Figure 7.3: An example depth map data and the corresponding posture of the subject with a virtual cloth on it.



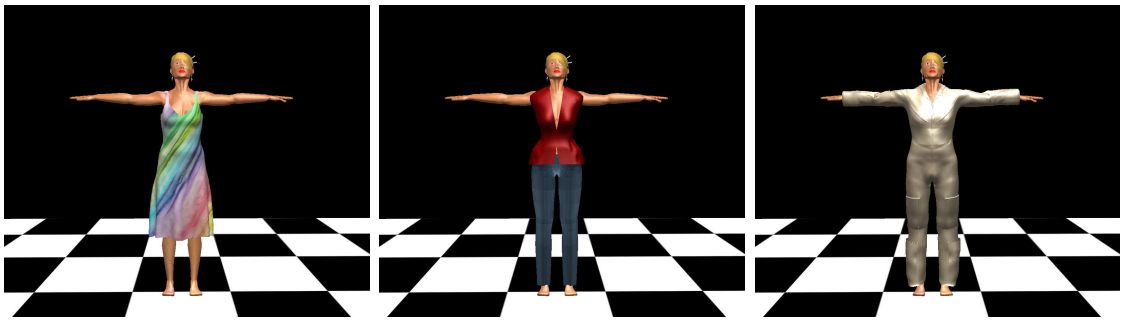
(a)

(b)



(c)

Figure 7.4: Examples of different garments on a model with different postures: (a) sun dress, (b) jeans and vest, and (c) flight suit.



(a)

(b)

Figure 7.5: The designed apparel meshes for the male and female avatars.

Chapter 8

Conclusions and Future Work

We present a real-time physics-based virtual fitting room framework. The simulation software simulates of a set of apparels on customized virtual avatars, including the effects of wind, inertia, gravity and collision on the apparel pieces. In addition, the framework includes a set of animation filters such as joint angle constraints, motion smoothing and foot-skate correction. The total framework consists of two distinct stages:

- The design stage where the avatar mesh and the apparel meshes are designed.
- The simulation stage where the apparel meshes are simulated on a virtual avatar which imitates the motions of the active user in real-time.

The framework utilizes a set of external third party software packages including design suites and libraries. The Blender software [5] is used as the design suite, with a set of specific design principles for the use of meshes within the simulation system. The simulation software is built on top of the Open-Source 3D Rendering Engine [86], which serves as the boilerplate including window management, peripheral input handling, low-level rendering and skeletal animation. The physics simulations including garment animation and collision detection are handled using the NVIDIA Physx Engine [44]. Kinect for Windows SDK [61] is used

as the middleware for acquiring depth maps and user skeletal information. For computer-vision related functions, Open Computer Vision project (OpenCV) [87] is included. The IKAN library by Pennsylvenia University is used as the inverse kinematics solver [79]. The motions filters, user measurements functions, modeling approaches and the bundling framework are in the scope of this thesis study.

Modeling of the human meshes enables convenient creation and modification of the virtual avatars used in the simulation, including construction, skinning and painting. The avatar meshes also serve as the modeling mannequin of the apparel meshes, which must be aligned accordingly prior to being exported for simulation. Apparel meshes are first modeled as a whole, to be split into a dynamic part and a static part later. The vertices of the dynamic part are subsequently labeled as fixed or free, determining the effect of external forces during the simulation.

The motion filters in the software are a substantial part of the overall system, as the input quality from the depth sensors is quite low compared to high-end motion capture systems. The correction in the upper limbs of the body employs angle constraints and bone splitting, in order to prevent the manifestation of unnatural orientations, especially in the lower part of the upper limbs. The lower limbs motion filter focus on the solution of foot skating problem, that is the sliding movement of the feet across the floor. Foot skate solution first constraints one of the feet to be stationary, followed by inverse kinematic solution to determine the orientations of the hip and knee joints.

User measurement feature provides accurate dimensions of the current user, enabling the avatar to be customized accordingly in real-time. The process takes advantage of both the depth map and extracted skeletal joint coordinates to estimate the various dimensions in the human body. Using a set of different parameters (such as torso height and head height) to estimate the target parameter(body height) rather than a single measurement of the target parameter reduces the error. The final step is temporal averaging, for purpose of overcoming the possible temporal noises and errors.

The simulation system is a complex software, because of many bundled libraries and the implemented custom features. There are a total of three different

coordinate systems used in general, for rendering, physics simulation and inverse kinematics solution. The skeletal animation is achieved through a custom class which acts as the connector between the rendering engine and the depth sensor middleware. In addition to the motion filtering and user measurement features, a custom hand tracking feature is implemented, however not embedded within the system.

8.1 Future Work

The modeling of the apparel meshes are currently done manually, with custom placement, separation and labeling, which makes the creation of new apparel models extremely expensive. On the other hand, all the steps in the process is automizable, hence a suite for rapid apparel mesh designing must be implemented before considering a commercial application for the framework.

The most prominent problem within the overall framework is the inferior quality of the motion capture input. This problem manifests itself especially in lateral views of the user in larger distances. The solution of this problem requires fixes on both hardware and software sides. The resolution and quality of depth sensors are expected to increase in the future, resulting in better depth maps and skeletal joint information. Another important aspect on the hardware side is the introduction of multiple depth sensors configured to cooperate and construct a single depth map, in order to overcome the self-occlusion problem. More motion filters and optimizations will help the quality of the animation from the software side. Additional improvements include multi-threaded motion filtering, improving the quality of physics simulations and rendering, however such subjects are secondary in priority.

Bibliography

[1] Fitnect Interactive Kft., "3D Virtual Fitting Dressing Room." http://www.fitnect.hu/, 2012.

[2] Styku, Inc., "Virtual Fitting Room and Body Scanning." http://www.styku.com/, 2013.

[3] FaceCake Marketing Technologies, Inc., "Visual Demonstration System." http://www.facecake.com/, 2013.

[4] Humanoid Animation Working Group, "H-Anim 200x, ISO/IEC FCD 19774 Humanoid Animation." http://h-anim.org/Specifications/H-Anim200x/ISO_IEC_FCD_19774/, 2013.

[5] Blender Foundation, "Blender." http://www.blender.org/, 2013.

[6] Z. Central, "Free Female Base Mesh." http://www.zbrushcentral.com/showthread.php?49053-Free-female-base-mesh-(nudity)., 2012.

[7] ZBrush Central, "Male Mesh." http://archive3d.net/?a=download&id=d2161d4e, 2013.

[8] ShareCG, "Antonia Sundress." http://www.sharecg.com/v/54636/gallery/5/3D-Model/Antonia-Sundress, 2012.

[9] TF3DM, "Neo 3d Model." http://tf3dm.com/3d-model/neo-19284.html, 2013.

[10] TF3DM, "Sub-Zero Unmasked 3D model." http://tf3dm.com/3d-model/sub-zero-unmasked-13941.html, 2013.

[11] A. Borodin, "Flight Suit 3D Model." http://archive3d.net/?a=download&id=cd051531, 2013.

[12] PS3D, "Vest 3D Model." http://archive3d.net/?a=download&id=ffbef632, 2013.

[13] M. Alperin, "Pants 3D Model." http://www.archive3d.net/?a=download&id=a1829cda, 2013.

[14] J. E. Chadwick, D. R. Haumann, and R. E. Parent, "Layered Construction for Deformable Animated Characters," *ACM Computer Graphics (Proceedings of SIGGRAPH)*, vol. 23, pp. 243–252, July 1989.

[15] N. I. Badler and S. W. Smoliar, "Digital Representations of Human Movement," *ACM Computing Surveys*, vol. 11, pp. 19–38, Mar. 1979.

[16] K. Komatsu, "Human Skin Model Capable of Natural Shape Variation," *The Visual Computer*, vol. 3, no. 5, pp. 265–271, 1988.

[17] J. Lander, "Skin Them Bones: Game Programming for the Web Generation," *Game Developer Magazine*, vol. 5, no. 1, pp. 10–18, 1998.

[18] L. Kavan, S. Collins, J. Žára, and C. O'Sullivan, "Skinning with Dual Quaternions," in *Proceedings of the Symposium on Interactive 3D Graphics and Games*, I3D '07, (New York, NY, USA), pp. 39–46, ACM, 2007.

[19] L. Kavan, S. Collins, and C. O'Sullivan, "Automatic Linearization of Nonlinear Skinning," in *Proceedings of the Symposium on Interactive 3D Graphics and Games*, I3D '09, pp. 49–56, ACM, 2009.

[20] N. Miller, O. Jenkins, M. Kallmann, and M. Mataric, "Motion Capture from Inertial Sensing for Untethered Humanoid Teleoperation," in *Proceedings of 4th IEEE/RAS International Conference on Humanoid Robots*, vol. 2, pp. 547–565, 2004.

[21] S. Yabukami, H. Kikuchi, M. Yamaguchi, K.-I. Arai, K. Takahashi, A. Itagaki, and N. Wako, "Motion Capture System of Magnetic Markers Using Three-axial Magnetic Field Sensor," *IEEE Transactions on Magnetics*, vol. 36, no. 5, pp. 3646–3648, 2000.

[22] A. C. Sementille, L. E. Lourenço, J. R. F. Brega, and I. Rodello, "A Motion Capture System Using Passive Markers," in *Proceedings of the ACM SIGGRAPH International Conference on Virtual Reality Continuum and Its Applications in Industry*, VRCAI '04, (New York, NY, USA), pp. 440–447, ACM, 2004.

[23] L. P. Maletsky, J. Sun, and N. A. Morton, "Accuracy of an Optical Active-marker System to Track the Relative Motion of Rigid Bodies," *Journal of Biomechanics*, vol. 40, no. 3, pp. 682–685, 2007.

[24] K. Cheung, S. Baker, and T. Kanade, "Shape-from-silhouette of Articulated Objects and Its Use for Human Body Kinematics Estimation and Motion Capture," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1 of *CVPR '03*, pp. 77–84, 2003.

[25] E. de Aguiar, C. Theobalt, C. Stoll, and H. P. Seidel, "Marker-less Deformable Mesh Tracking for Human Shape and Motion Capture," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '07, pp. 1–8, 2007.

[26] J. Gall, C. Stoll, E. de Aguiar, C. Theobalt, B. Rosenhahn, and H. P. Seidel, "Motion Capture Using Joint Skeleton Tracking and Surface Estimation," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '09, pp. 1746–1753, 2009.

[27] Y. Liu, C. Stoll, J. Gall, H. P. Seidel, and C. Theobalt, "Markerless Motion Capture of Interacting Characters Using Multi-view Image Segmentation," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pp. 1249–1256, 2011.

[28] G. K. Cheung, T. Kanade, J.-Y. Bouguet, and M. Holler, "A Real Time System for Robust 3D Voxel Reconstruction of Human Motions," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2 of *CVPR '00*, (Los Alamitos, CA, USA), pp. 714–720, IEEE Computer Society, 2000.

[29] K.-M. G. Cheung, S. Baker, and T. Kanade, "Shape-From-Silhouette Across Time Part II: Applications to Human Modeling and Markerless Motion Tracking," *International Journal of Computer Vision*, vol. 63, pp. 225–245, July 2005.

[30] T. Dutta, "Evaluation of the Kinect™ Sensor for 3-D Kinematic Measurement in the Workplace," *Applied Ergonomics*, vol. 43, no. 4, pp. 645–649, 2012.

[31] K. Berger, K. Ruhl, Y. Schroeder, C. Bruemmer, A. Scholz, and M. A. Magnor, "Markerless Motion Capture using multiple Color-Depth Sensors," in *Proceedings of Vision, Modeling and Visualization*, VMV '2011, pp. 317–324, 2011.

[32] K. Khoshelham and S. O. Elberink, "Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications," *Sensors*, vol. 12, no. 2, pp. 1437–1454, 2012.

[33] S. Matyunin, D. Vatolin, Y. Berdnikov, and M. Smirnov, "Temporal Filtering for Depth Maps Generated by Kinect Depth Camera," in *Proceedings of the 3DTV Conference: The True Vision - Capture, Transmission and Display of 3D Video*, 3DTV-CON '2011, pp. 1–4, 2011.

[34] Y. Yang, W. Zhang, and C. Shan, "Investigating the Development of Digital Patterns for Customized Apparel," *International Journal of Clothing Science and Technology*, vol. 19, no. 3/4, pp. 167–177, 2007.

[35] Gerbert Technology, "AccuMark Pattern Design Software." `http://www.gerbertechnology.com/en-us/solutions/apparelretail/productdesign/accumark.aspx`, 2013.

[36] Assyst-Bullmer, "Pattern Design Software." `http://assystbullmer.co.uk/products/software/`, 2013.

[37] Y. Yang, N. Magnenat Thalmann, and D. Thalmann, "Three-dimensional Garment Design and Animation: a New Design Tool for the Garment Industry," *Computers in Industry*, vol. 19, no. 2, pp. 185–200, 1992.

[38] L. Chittaro and D. Corvaglia, "3D Virtual Clothing: from Garment Design to Web3D Visualization and Simulation," in *Proceedings of the Eighth International Conference on 3D Web Technology*, Web3D '03, (New York, NY, USA), pp. 73–85, ACM, 2003.

[39] E. Turquin, J. Wither, L. Boissieux, M.-P. Cani, and J. Hughes, "A Sketch-Based Interface for Clothing Virtual Characters," *IEEE Computer Graphics and Applications*, vol. 27, no. 1, pp. 72–81, 2007.

[40] T. Bonte, A. Galimberti, and C. Rizzi, "A 3D Graphic Environment for Garments Design," in *From Geometric Modeling to Shape Modeling* (U. Cugini and M. Wozny, eds.), vol. 80 of *IFIP The International Federation for Information Processing*, pp. 137–150, Springer US, 2002.

[41] U. Cugini and C. Rizzi, "3D Design and Simulation of Men Garments," in *The 10th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (Short Papers)*, WSCG '2002, pp. 9–16, 2002.

[42] I. Autodesk, "Autodesk Maya." `http://www.autodesk.com/products/autodesk-maya/overview`, 2013.

[43] F. Durupınar and U. Güdükbay, "A Virtual Garment Design and Simulation System," in *Proeedings of 11th International Conference Information Visualization*, IV '07, pp. 862–870, 2007.

[44] Wikipedia, "Physx - Wikipedia, The Free Encyclopedia." `http://en.wikipedia.org/wiki/PhysX`, 2012.

[45] NVIDIA Developer Zone, "APEX Multi-platform, Scalable Dynamics Framework." `https://developer.nvidia.com/apex`, 2013.

[46] I. Autodesk, "Autodesk 3dsMax." `http://www.autodesk.com/products/autodesk-3ds-max/overview`, 2013.

[47] J. Weil, "The Synthesis of Cloth Objects," *ACM Computer Graphics (Proceedings of SIGGRAPH)*, vol. 20, pp. 49–54, Aug. 1986.

[48] D. R. Haumann and R. E. Parent, "The Behavioral Test-bed: Obtaining Complex Behavior from Simple Rules," *The Visual Computer*, vol. 4, no. 6, pp. 332–347, 1988.

[49] X. Provot, "Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior," in *Proceedings of Graphics Interface*, GI '95, pp. 147–154, 1995.

[50] T.-Y. Kim, "Character Clothing in PhysX-3." Tech Talk SIGGRAPH ASIA 2011, 2011.

[51] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer, "Elastically Deformable Models," *ACM Computer Graphics (Proceedings of SIGGRAPH)*, vol. 21, pp. 205–214, Aug. 1987.

[52] D. Baraff and A. Witkin, "Large Steps in Cloth Simulation," in *Proceedings of ACM SIGGRAPH '98*, (New York, NY, USA), pp. 43–54, ACM, 1998.

[53] D. Protopsaltou, C. Luible, M. Arevalo, and N. Magnenat-Thalmann, "A Body and Garment Creation Method for an Internet Based Virtual Fitting Room," in *Advances in Modelling, Animation and Rendering* (J. Vince and R. Earnshaw, eds.), pp. 105–122, Springer London, 2002.

[54] W. Zhang, T. Matsumoto, J. Liu, M. Chu, and B. Begole, "An Intelligent Fitting Room Using Multi-camera Perception," in *Proceedings of the 13th International Conference on Intelligent User Interfaces*, IUI '08, (New York, NY, USA), pp. 60–69, ACM, 2008.

[55] Y.-J. Chang, S.-F. Chen, and J.-D. Huang, "A Kinect-based System for Physical Rehabilitation: A Pilot Study for Young Adults with Motor Disabilities," *Research in Developmental Disabilities*, vol. 32, no. 6, pp. 2566–2570, 2011.

[56] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "RGB-D Mapping: Using Kinect-style Depth Cameras for Dense 3D Modeling of Indoor Environments," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 647–663, 2012.

[57] L. Gallo, A. Placitelli, and M. Ciampi, "Controller-free Exploration of Medical Image Data: Experiencing the Kinect," in *Proceedings of 24th International Symposium on Computer-Based Medical Systems*, CBMS '2011, pp. 1–6, 2011.

[58] Y. Meng, P. Mok, and X. Jin, "Interactive Virtual Try-on Clothing Design Systems," *Computer-Aided Design*, vol. 42, no. 4, pp. 310–321, 2010.

[59] S. Giovanni, Y. C. Choi, J. Huang, E. T. Khoo, and K. Yin, "Virtual Try-on using Kinect and HD Camera," in *Proceedings of the International Conference on Motion in Games, MIG '2012*, vol. 7660 of *Lecture Notes in Computer Science*, pp. 55–65, Springer, 2012.

[60] OpenNI, "Programmer Guide-OpenNI." `http://openni.org/Documentation/ProgrammerGuide.html`, 2012.

[61] I. Microsoft, "Kinect for Windows." `http://www.microsoft.com/en-us/kinectforwindows/`, 2013.

[62] S. Hauswiesner, M. Straka, and G. Reitmayr, "Virtual Try-On through Image-Based Rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 9, pp. 1552–1565, 2013.

[63] Z. Zhou, B. Shu, S. Zhuo, X. Deng, P. Tan, and S. Lin, "Image-based Clothes Animation for Virtual Fitting," in *ACM SIGGRAPH Asia, Technical Briefs*, (New York, NY, USA), ACM, 2012.

[64] Y. Cui, W. Chang, T. Nll, and D. Stricker, "KinectAvatar: Fully Automatic Body Capture Using a Single Kinect," in *11th Asian Conference on Computer Vision (ACCV 2012) Workshops* (J.-I. Park and J. Kim, eds.), vol. 7729 of *Lecture Notes in Computer Science*, pp. 133–147, Springer Berlin Heidelberg, 2013.

[65] Y. Cui, S. Schuon, D. Chan, S. Thrun, and C. Theobalt, "3D Shape Scanning with a Time-of-flight Camera," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '10, pp. 1173–1180, 2010.

[66] Z. Yasseen, A. Nasri, W. Boukaram, P. Volino, and N. Magnenat-Thalmann, "Sketch-based Garment Design with Quad Meshes," *Computer-Aided Design*, vol. 45, no. 2, pp. 562–567, 2013. ¡ce:title¿Solid and Physical Modeling 2012¡/ce:title¿.

[67] F. Cordier, H. Seo, and N. Magnenat-Thalmann, "Made-to-measure Technologies for an Online Clothing Store," *IEEE Computer Graphics and Applications*, vol. 23, no. 1, pp. 38–48, 2003.

[68] Y. Meng, C. C. Wang, and X. Jin, "Flexible Shape Control for Automatic Resizing of Apparel Products," *Computer-Aided Design*, vol. 44, no. 1, pp. 68–76, 2012.

[69] C. C. Wang, Y. Wang, and M. M. Yuen, "Feature Based 3D Garment Design Through 2D Sketches," *Computer-Aided Design*, vol. 35, no. 7, pp. 659–672, 2003.

[70] S. M. Kim and T. J. Kang, "Garment Pattern Generation from Body Scan Data," *Computer-Aided Design*, vol. 35, no. 7, pp. 611–618, 2003.

[71] D.-E. Kim and K. LaBat, "Consumer Experience in Using 3D Virtual Garment Simulation Technology," *Journal of the Textile Institute*, In Press.

[72] L. Kavan and J. Zara, "Real-Time Skin Deformation with Bones Blending," in *WSCG Short Papers Proceedings*, 2003.

[73] M. Azimi, "Skeletal Joint Smoothing." White Paper, `http://msdn.microsoft.com/en-us/library/jj131429.aspx`, 2013.

[74] C. C. Holt, "Forecasting Seasonals and Trends by Exponentially Weighted Moving Averages," *International Journal of Forecasting*, vol. 20, no. 1, pp. 5–10, 2004.

[75] P. S. Kalekar, "Time Series Forecasting Using Holt-Winters Exponential Smoothing." Kanwal Rekhi School of Information Technology, 2004.

[76] L. Kovar, J. Schreiner, and M. Gleicher, "Footskate Cleanup for Motion Capture Editing," in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '02, (New York, NY, USA), pp. 97–104, ACM, 2002.

[77] L. Ikemoto, O. Arikan, and D. Forsyth, "Knowing When to Put Your Foot Down," in *Proceedings of the Symposium on Interactive 3D Graphics and Games*, I3D '06, (New York, NY, USA), pp. 49–53, ACM, 2006.

[78] M. Sung, "Automatic Fixing of Foot Skating of Human Motions from Depth Sensor," in *Multimedia and Ubiquitous Engineering, MUE '2013* (J. J. J. H. Park, J. K.-Y. Ng, H. Y. Jeong, and B. Waluyo, eds.), vol. 240 of *Lecture Notes in Electrical Engineering*, pp. 405–412, Springer Netherlands, 2013.

[79] Human Modeling and Simulation Laboratory, University of Pennsylvania, "Inverse Kinematics using ANalytical Methods-IKAN." `http://cg.cis.upenn.edu/hms/software/ikan/ikan.html`, 2013.

[80] D. Tolani, A. Goswami, and N. I. Badler, "Real-Time Inverse Kinematics Techniques for Anthropomorphic Limbs," *Graphical Models*, vol. 62, no. 5, pp. 353–388, 2000.

[81] M. Muller, B. Heidelberger, M. Hennix, and J. Ratcliff, "Position Based Dynamics," *Journal of Visual Communication and Image Representation*, vol. 18, pp. 109–118, Apr. 2007.

[82] R. Tonge, "Collision Detection in PhysX," in *Recent Advances in Real-Time Collision and Proximity Computations for Games and Simulations, ACM SIGGRAPH 2010 Courses*, 2010.

[83] K. Khoshelham and S. O. Elberink, "Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications," *Sensors*, vol. 12, no. 2, pp. 1437–1454, 2012.

[84] B. Willis, "Body Proportions in Art." `http://www.worsleyschool.net/socialarts/body/proportions.html`, 2012.

[85] I. Samejima, K. Maki, S. Kagami, M. Kouchi, and H. Mizoguchi, "A Body Dimensions Estimation Method of Subject from a Few Measurement Items Using KINECT," in *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*, SMC '2012, pp. 3384–3389, IEEE, 2012.

[86] Torus Knot Software, "OGRE – Open Source 3D Rendering Engine." `http://www.ogre3d.org/`, 2012.

[87] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, vol. 25, no. 11, pp. 120, 122–125, 2000.

Appendix A

OGRE Framework

We embrace the "do not repeat yourself" mindset and extreme programming methodology in the implementation of the simulation software. Furthermore, since this study required utilization of many different techniques in different fields in Computer Science, we required various third party software development kits (SDKs) to cover the lowest level code. The search for the most appropriate rendering engine yielded the following requirements:

1. must be code-oriented rather than designer-oriented;
2. must be able to utilize both DirectX and OpenGL (for compatibility reasons);
3. must take care of mundane and routine programming such as the rendering pipeline and input handling;
4. must be able to integrate easily with 3rd party libraries;
5. must be stable and mature;
6. must have an associated 3-D designing program which can be used to easily produce content and load into the program; and
7. must have accurate and extensive documentation.

Other than these, automatic material rendering, skeletal animation support were considered as extra useful features. After evaluating Unity, UDK and native OpenGL programming, we decided on Object-Oriented Rendering Engine (will be referred as OGRE in the rest of the paper) for it fits the requirements of the software best.

A.1 The Features

OGRE is more than just a rendering engine, despite what name implies. Among all the features, the ones utilized considerably in the software are as follows [86], which led us to choose OGRE as the base of our framework.

- Render state management.
- Spatial culling and transparency handling.
- Material rendering:
  - easy material and shader management, custom shader support,
  - multitexture and multi pass blending,
  - lighting shader and different shadow rendering techniques,
  - material level of detail support,
  - support for a variety of image formats, volumetric textures and DXT textures, and
  - render-to-texture (frame rendering buffer) support.
- Meshes:
  - native mesh format, which can be exported from Blender Designer,
  - level of detail support,
  - skeletal animation feature, can be used with models exported from Blender:

    - multiple-bone weighted skinning,
    - hardware acceleration.

- Easy scene management.
- Easy camera and input management.
- Easy integration with third party libraries due to code-based nature.
- Overlay feature which enables easy information tracking about the feature.

A.2 High Level Overview

A.2.1 The Root Object

The root object is the entry point and core of the framework.

- It is created first and destroyed last in the application life cycle.
- It configures the system, delivers pointers to the managers for various resources.
- Provides automatic rendering cycle, continued until an interrupt from FrameListener objects.

A.2.2 The RenderSystem Object

RenderSystem is an abstract class to define the underlying 3D API (either Direct 3D or OpenGL). This class is not accessed and modified by the application programmer.

A.2.3 The SceneManager Object

SceneManager is the most frequently used object by the application programmer because it is in charge of the contents in the scene to be rendered.

- It is used to create, destroy and update the objects.
- It sends the scene to the RenderSystem object for rendering.
- Multiple SceneManagers can be used to create other visual resources (e.g., RenderToTexture environment).

A.2.4 Resource Manager

ReosurceManager object is an abstract class, which used to create, keep and dispatch a type of resource it is associated with.

- The associated type is defined by the class inheriting the ResourceManager, such as MaterialManager.
- There is always only one instance of every child of ResourceManager in an application.
- Resource managers search the pre-defined locations of the file system and automatically indexes the resources available, ready to be loaded upon demand.

A.2.5 Entities, Meshes, Materials and Overlays

Entities are the instances of movable objects in the scene. They are based on meshes, which define the geometric and material properties of the entity. Materials describe material properties of objects that determine the color and intensity of pixels during rendering.

- Entities are attached to scene nodes for moving and rendering. Scene nodes can be nested, which greatly simplifies the process of rendering complex scenes.
- Meshes consist of sub meshes, which can have different material associations. Therefore, a mesh can be composed of various parts with various materials.
- Materials are defined either in run-time or in .material scripts, with detailed information. They also support custom shaders.
- Mesh files can be created and saved with manual objects, or exported through designer programs such as Blender. The .mesh files are in binary format.

Overlays are used to create panels for control and Head-Up-Display (HUD), which are rendered above the scene. They are 2D elements are placed either by screen proportion or pixel size and rendered orthographically, last in the rendering pipeline by default (this can be overridden).

Appendix B

User Tracking

B.1 Hardware

User tracking has always been both a challenge and a valued feature in image processing. Until the availability of time-of-flight cameras, user tracking was dependent on RGB cameras. Although RGB cameras are sufficient for user tracking, they are proven to be harder to use for body articulation and joint estimation, mostly because of the complexity of human body, self occlusions, and the difficulties of body segmentation based on pixel colors alone. Researchers started with still images, than experimented with image sequences from multiple cameras. Most common technique, called *Shape From Silhouette (SFS)*, extracts the body silhouette from the image and constructs a body shape with silhouettes from multiple calibrated cameras [29]. SFS algorithms evolved from spatial to temporal tracking and their accuracy improved even more [29].

However, RGB based accurate user articulation techniques required many cameras-a financial problem. After the need for extensive imaging hardware, the processes required to calibrate the cameras initially, extract and combine the silhouettes, build a shape and articulate the result. These processes require very complex algorithms, many man-hours to implement and very powerful computing infrastructure. Instead of RGB imaging, we searched for an alternate type of

device which can capture the depth of the field. Although there are a variety of such devices, from sonars to Laser Scanners, the one most appropriate for user tracking is time-of-flight cameras. They have significant advantages over stereo vision and laser tracking, such as simplicity, speed (up to 100 frames per second), and accuracy in distance [83].

Choosing the most accurate time-of-flight camera is easy, as Microsoft Kinect is not only the cheapest and the most available of them all, it is also the most powerful and have an extensive developer community. We utilize the Kinect for XBOX rather than Kinect for Windows in this study because of its distance and performance characteristics.

B.2 Software

The user tracking process with Kinect is much simpler compared to SFS techniques, due to the shape being available mostly with the depth field output. However, proper articulation still requires lots of different algorithms and time. In order to speed up the user tracking and articulation, we utilize the Kinect for Windows SDK framework. KFW provides the framework for capturing and utilizing the various types of streams from natural interaction devices, also provides abstract modules for accessing complex functionalities, such as skeletal tracking [60].

With the integration of these modules to the simulation software, we are able to acquire the joint positions and orientations from the depth sensor with almost no effort except the integration itself. With the current hardware/software configuration, we acquire 20 joint positions and orientations at 30 frames per second, enough to reproduce the movement of the user on the virtual model. Other than the joint information, we also acquire the depth and image streams from the depth camera for user body measurement purposes. We also use the image stream to present the obtained results, comparing the user movement with the simulated environment.

Appendix C

Hand Tracking

C.1 OpenCV

The user interface within a natural interaction framework requires functions such as hand state recognition or hand swipe filters, which are omitted in the Kinect for Windows SDK. In order to simulate mouse-clicking behavior, we track the hands of the user to be used as cursors, notice open/close hand gestures for clicking events. In order to implement the algorithms for the proposed hand-track solution, we selected to work with OpenCV, due to its maturity, community and integration with other libraries. OpenCV not only provides basic functions to perform complex and processor-intensive image processing functions such as, facial recognition system, gesture recognition, stereoscopic 3D and segmentation, it also has a very vast machine learning aspect, including boosting, decision three learning and many more features [87].

C.2 The Process

Utilizing such a powerful middleware as a depth sensor, we are able to perform very robust background subtraction with almost no effort. Skeletal body tracking

is another embedded property of the software that we use, giving a speed boost. We implemented two different techniques: *hand state recognition and* hand swipe recognition. Hand swipe recognition is simpler compared to hand state recognition. It is just a matter of keeping track of the 3D position of the hand, and keeping a listener that is activated when the 3D velocity of the hand exceeds a certain threshold. We also perform a number of optimizations to make it invariant with the size and location of the user. Open/close hand recognition is harder than hand swipe recognition. We perform image processing in order to determine the state of the hand successfully at relatively low resolutions and large distances. Figure C.1 shows the overview of the hand recognition algorithm. The steps of the algorithm are as follows:

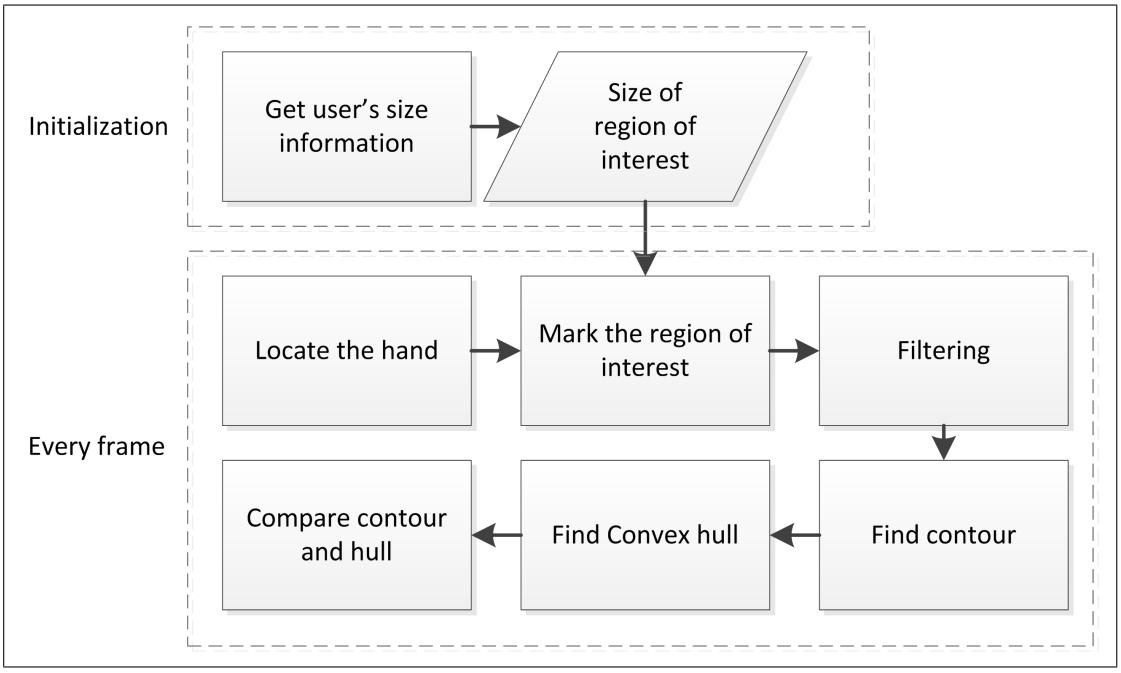


Figure C.1: The overview of the hand recognition algorithm

1. The size information for the region of interest is drawn from the distance between the user's head and neck.

2. The hand is located using skeletal body tracking.

3. The marked region is copied from the depth stream.

4. Two dilation and one erosion operations are performed to smooth the hand image.

5. The contour is found on the filtered image.

6. The convex hull of the found contour is calculated.

7. The depth difference between the hull and the actual contour is taken as the reference for hand-state.

Figure C.2: Images and contours of hand regions from the depth stream. Left: open hand and right: closed hand.

Figure C.2 depicts the results of experiments for hand recognition. The hand tracking feature is not implemented within the simulation framework yet, as the consumer level usage requires an extensive amount of available apparel meshes as well as better user tracking and motion smoothing. Hence, the user interaction based on hand gestures is left as a future work.