NAROVATAR

(AUTOMATED CUSTOMIZABLE AVATEERING)

Project report submitted in partial fulfillment for the award of the degree of

Integrated Master of Computer Applications

by

BASANTA SHARMA

114412

DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

SRI SATHYA SAI INSTITUTE OF HIGHER LEARNING

NAROVATAR

(AUTOMATED CUSTOMIZABLE AVATEERING)

A PROJECT REPORT

Submitted to the

DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

By

BASANTA SHARMA

In partial fulfillment of the requirements
For the award of the degree

Of

INTEGRATED MASTER OF COMPUTER APPLICATION

SRI SATHYA SAI INSTITUTE OF HIGHER LEARNING

Prasanthi Nilayam, Andhra Pradesh, India, 562101

March 2016

SRI SATHYA SAI INSTITUTE OF HIGHER LEARNING

(Deemed University)

DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

BONAFIDE CERTIFICATE

This is to certify that this project report titled "**NAROVATAR (Automated Customizable Avateering)**" was carried out by **Mr. Basanta Sharma** under our supervision. Certified further, that to the best of our knowledge the work reported herein does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Sri Srikanth Khanna,
Asst. Professor, Department of Mathematics and Computer Science,
Sri Sathya Sai Institute of Higher Learning, Prasanthi Nilayam Campus.

Sri BVK Bharadwaj,
Asst. Professor, Department of Mathematics and Computer Science,
Sri Sathya Sai Institute of Higher Learning, Muddenahalli Campus.

Place: Muddenahalli
Date: 19th of March, 2016

TO LOVE

ABSTRACT

Avatar animation is one of the most popular areas in Computer Graphics. Finding its application in varied fields including gaming, interactive user interfaces, education packages, etc., this topic has motivated a great amount of work in sophistication and automation of different stages of Avatar animation pipeline. But unless these significant parts are wedded together in a single complete framework we would not be able to appreciate the true potential and beauty of the whole. So, a need was felt for a framework that integrates these available solutions into a fully automatic and easy-to-use system that can produce realistic Avatar animation without a need for technical expertise.

In this direction, we define our ***Narovatar Framework*** *for* ***creation, rigging*** *and* ***motion retargeting*** *for Avatar animation. We assert automation throughout the pipeline so that a system based on Narovatar will not necessitate any artistic skills or technical expertise on the part of the users to create Avatar animations. We also provide a concrete implementation of Narovatar in C++. It consists of a module for automatic rigging of a human mesh given in any known file format and another one that uses the rigged mesh to retarget motion captured using Microsoft Kinect sensor so that the static mesh given as input now starts mimicking a person in front of the Kinect. The first module of the framework, Avatar model creation however is currently not incorporated in the concrete implementation. Considering available third-party applications that create 3D Avatar of a person using Kinect Fusion (a feature provided by Kinect for Windows API) to fill the gap of the first module, this implementation of Narovatar becomes a complete automated pipeline of Avatar animation.*

Narovatar framework also provides sufficient scope for the system to improvise existing modules of current implementation with better algorithms and to incorporate new modules to implement additional aspects of Avatar animation. With this, Narovatar can potentially produce more sophisticated and realistic animations.

ACKNOWLEDGEMENT

I take this opportunity to recognize and express my heart-felt gratitude to everyone who helped me in different ways during the course of my MCA project. I am thankful for their inspiration, guidance, constructive criticism and friendly advice that brought this project to fruition.

At the outset, I would like to accredit this work to my parents, without whom, I would not be anywhere near to taking this project. I am ever indebted to their warm love and care that keeps me going.

I would like to express my gratitude to my dear Lord, **Bhagawan Sri Sathya Sai Baba**, for His constant inspiration to take up this project and work on it incessantly as an offering to Him.

I extend my ever grateful heart to my guides **Sri Srikanth Khanna** and **Sri BVK Bharadwaj** for their expert guidance and constant moral encouragement.

I fail in my duty if I do not extend my thanks to the **Institute** and the **Department** for providing me with state-of-the art facilities without which I would not have done this project. I take this opportunity to thank the consecutive Heads of the Department Prof. **Chandrasekharan** and **Dr. Pallav Kumar Baruah** and the Deputy Director in-charge of our college **Sri K Sayee Manohar** for kindly providing all necessary facilities and permission to use them.

I also extend my thanks to **Sri V Bhaskaran** and **Sri Sai Manohar** (Deputy Wardens of SSSIHL hostel) and **all other teachers of the department** for their active support and encouragement. I am equally thankful to all my classmates who were always by my side to share my emotions during my ups and downs and to help me in every way they could.

Finally, I extend my thanks to all those whose names I might have inadvertently omitted and seek their pardon.

Basanta Sharma

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TERMS

TERM 1 **Children-with-autism**:

Children diagnosed with a disease in Autism Spectrum Disorder.

TERM 2 **Avatar**:

A 3D model of a human being.

TERM 3 **Avateering**:

[Used interchangeably with Avatar animation] The complete process of animating an avatar.

TERM 4 **Rig**:

Controls that are defined for a mesh, generally used in animation to deform the mesh in order to produce animation.

TERM 5 **Rigging**:

*Rigging is making our characters able to move. The process of rigging is we take that digital sculpture, and we start building the skeleton, the muscles, and we attach the skin to the character, and we also create a set of animation controls, which our animators use to push and pull the body around. [*Frank Hanner, character CG supervisor of the Walt Disney Animation Studios*]*

TERM 6 **Vertex Normal**:

The normal vector to a vertex of the mesh. It is essential for determining the orientation of the mesh.

TERM 7 **Texture**:

A 2D Image that is used for covering or painting the surface dull mesh of an object to give it a real look.

TERM 8 **Texture Map**:

Each vertex of a 3D Model can be mapped to a point in a 2D texture image. This information helps in finding which portion of the texture needs to be applied in a face of the mesh.

TERM 9 **Material and Lighting**:

The material and lighting properties of a 3D Model defines how the model should react to the lighting conditions. It includes specification of properties like diffusive, specular, etc.

CHAPTER 1: INTRODUCTION

1.1 GENESIS

1.1.1 **Children-with-autism** [Term 1] – a motivation

Autism Spectrum Disorder (ASD) is a neurodevelopmental disorder characterized by impaired social interaction, verbal and non-verbal communication, and restricted and repetitive behavior. The signs of this disorder are usually noticed in the first two years of a child's life. It is distinguished not by a single symptom, but by a characteristic triad of symptoms: impairments in social interaction; impairments in communication; and restricted interests and repetitive behavior.

Figure 1.1: (Left to Right) Autism Logo, Repetitive stacking (a symptom of Autism), lining up objects (another symptom of autism)

Although there is no known cure, there have been reported cases of children who recovered. [1] People with autism may be severely impaired in some respects but normal, or even superior, in others. [3] Early speech or behavioral interventions can help children with autism gain self-care, social, and communication skills. [2] From this, we can infer that we can put our hands together to help these children-with-autism grow up to being normal human beings.

Now, to help children-with-autism recover means making them act and communicate in manners they find difficult when left to themselves. But how can they do something which they seem to be unable to do? Probably, a way to achieve this would be to involve them in some activity so well that they start responding to it without putting in their conscious effort. The activity can be anything that interests the child most, like, naturally interacting with their favorite cartoon character or someone very close to them (e.g. parents), playing their favorite video game, etc. So, when the child is fully involved in the activity and if the activity prompts some response from the child that requires it to act in a way which it is otherwise incapable of, there can still be enough chances that it responds successfully because it doesn't put conscious effort to do it.

Now the question remains, how can we achieve this? This gives birth to the idea

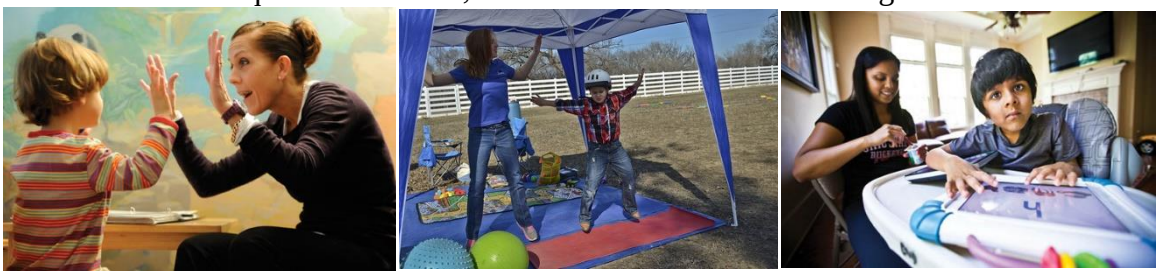


Figure 1.2: Behavioral interventions for curing Autism

of a software system that provides fully involving, fun-filled and involvement demanding activities to the children-with-autism.

1.1.2 The Virtual Interaction System

The idea is to create an **avatar** [Term 2] that involves children-with-autism in a natural interaction. A sensor will be used to feed in the information to the software about movements, expressions and reaction of the child present in front of the system. The information will then be processed to infer the knowledge about the child's state-of-mind or likes/dislikes or reaction that helps the system tailor the interaction according to the situation. The processing will be done by an AI engine in the back-end and will arrive at an appropriate response. The response will then be translated into animation of the avatar in the front end in a way the child can understand and relate to. The look-and-feel of the avatar, therefore, should be customizable to look like any character the child can relate to. There should also

be means to translate any action command (that is generated in the back end) into natural animation.

Clearly, the whole project of building the Virtual Interaction System is a humongous task. So, the system can be broken down into three broad components.

i. **The Capture System**: This system consists of the Sensor and its associated modules, like, the motion capture API and some preliminary data processing modules like data filters, etc.

Figure 1.3: Interaction between human and virtual Avatar

ii. **The Processing System**: This forms the core of the system. It is a context-based module that processes and interprets the input given by the capture system with respect to the context and arrives at an appropriate response for the stimuli. Hence it involves Artificial Intelligence.

iii. **The Front End**: This system translates the response given by the processing system into animation of the Avatar. For this, creation of a realistic and animation-ready avatar also forms a part of this module.

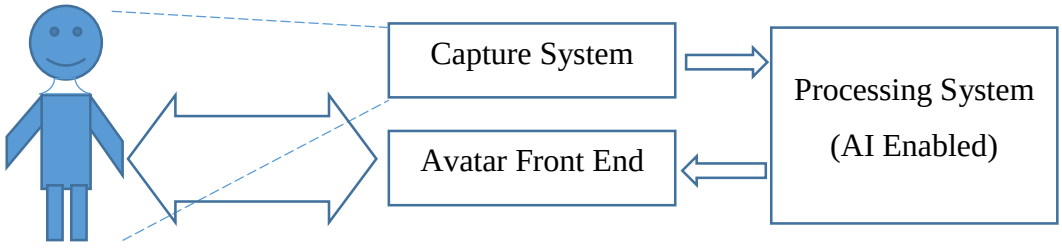


Figure 1.4: The Virtual Interaction System Logical View

1.1.3 Where do I fit in?

An important component of the above mentioned Virtual Interaction System is the Avatar which the child can interact to. The avatar should be *dynamically* created by the users according to the requirement and the system should be able to incorporate the dynamically created Avatar seamlessly in its working. Hence, the system should facilitate *automation* of the whole pipeline of Avatar Animation right from creating it, adding necessary control to it for animation and finally making it move (without technical expertise required from the users). Hence, this project concentrates on building an automated complete avatar animation pipeline, or **Avateering** [Term 3].

The implementation should fulfill the following requirements:

i. It should provide means for automatic and easy creation of a realistic looking avatar. The avatar can be created using images/videos of the person/character or with a 3D scan in case the person is available in flesh.
ii. The avatar should mimic the behavior of the person in terms of gestures, facial expressions, voice modulations, etc. Hence, the system should support behavior learning.
iii. The avatar should have necessary controls for easy animation of any action.

1.2 AVATAR ANIMATION

Avatar Animation (or Avateering) today forms a part of a lot of applications. Some of them are sophisticated, like characters in modern movies and games, while others are quite simpler. Today, most of the applications tend to include an animated avatar in the front end for a more intuitive human-machine interaction and better involvement of users. The proposed interactive system mentioned earlier, of which this project forms a part, is one such example as it relies in the power of Avateering for the total involvement of children-with-autism.

Animation, as a process, is logically divided into many logically independent phases, like, avatar mesh creation, **rigging** [Term 5] and finally making it move by providing it with motion data (making the avatar "do" things). Each of these phases is a problem of different nature and hence requires different skillset and/or tools. Above this, Animation requires artistic expertise. Hence, it is difficult to automate the process of animation without compromising on excellence.

Regardless of such difficulties, inspired by its application like Gaming and e-Learning industries, there has been a lot of work going around in the field of Computer Graphics to automate the process of animation. Each of the above mentioned phases have been studied separately in depth and various approaches have been proposed for each of them. This project aims at integrating best suitable approaches among them and implement them. The result would be a complete system that automatically creates, rigs and animates an avatar seamlessly without any human expertise required.

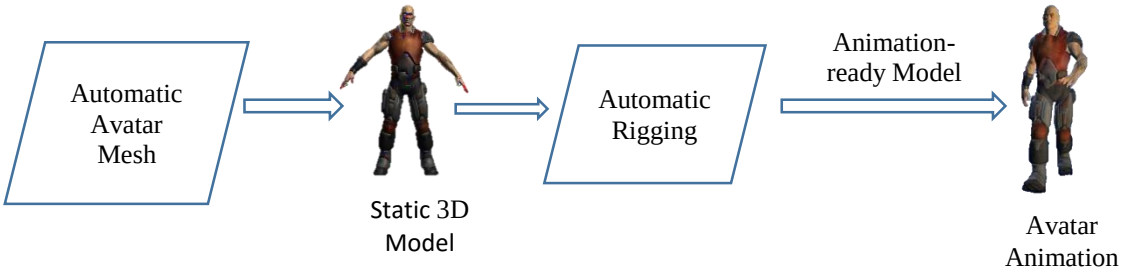


Figure 1.5: A logical view of an automatic and complete avatar animation system

CHAPTER 2: PROBLEM

2.1 PROBLEM STATEMENT

2.1.1 Goal

The goal of this project is to build a prototype of the Virtual Interaction System (explained in Section 1.1.2). It needs to be implemented based on a well-defined framework that encompasses all the necessary aspects of the system, but can have implementation of only a part of it, so that with further enhancements, the whole of the project can be realized in due time. The system created based on the framework should not require any kind of professional knowledge or expertise on the subject like Animation, etc. from its users. The user should be able to use the system with just the knowledge about working of the system.

2.1.2 Specifications

Based on the above goal, the requirement of the project is specified as follows:

i. A conceptual framework should be defined for a fully automatic and complete pipeline of Avatar Animation.
ii. Based on the framework, a prototype should be built with an initial implementation of all the key phases.
iii. Microsoft Kinect™ shall be used as the sensor for capturing data from the environment.

2.1.3 Scope

This project is limited to the Avateering aspect of the Virtual Interaction System. Hence, the advanced aspects of the System, like, implementation of the AI Engine for the back end, are beyond the scope of this project. Also, the deeper aspects of Avateering, like

learning the behavior of the character do not form a part of this project. This project is limited to automation of the complete Avateering pipeline.

2.2 TECHNICAL FEASIBILITY STUDY

This project is found technically feasible according to the following study.

2.2.1 Hardware Requirements:

This project requires the following hardware:

i. A computer system with a good Graphics Card and processing power.

The computer systems in our Computer Lab are 64 bit machines with *Intel® HD Graphics 4600* graphics adapter card that supports OpenGL 4.0. Hence the system is sufficient for building a medium-scale graphics application. The project would not require a sophisticated Graphics Processing Unit within the limited scope defined.

ii. A commodity sensor for capturing motion data, taking 3D scan of a person, etc.

Kinect for Xbox 360 is available for this purpose.

Thus, it is concluded that the project is feasible with respect to hardware requirements.

2.2.2 Software Requirements:

The software requirements are as follows:

i. Drivers and API to work with Kinect:

All the drivers for the Kinect Device and the API to access the data from it are readily available for download in the official website of Windows.

ii. Operating System: Windows

The Kinect API is compatible only in Windows platform. Windows 10 installed in the systems in our computer lab caters to the need.

iii. IDE: Visual Studio 2010 and above

The default IDE supported by Kinect API is the Visual Studio 2010 or above. We have Visual Studio 2013 to build the applications.

iv. Graphics programming libraries

XNA is the default graphics library supported by Windows. It is freely downloadable from official website of Microsoft. But open libraries like OpenGL are also easily available. We can use any of these libraries for graphics rendering interchangeably.

This study show that the project is feasible in terms of software requirements.

2.2.3 Schedule

Time is another important aspect for a project development. This being an academic project, the schedule must consider the time required for acquiring domain knowledge and field study. Only then, one can start studying the problem and provide a solution.

The time available for this project is one academic year, i.e., **two semesters** with dedicated **14 hours a week**. Since, this project demands a deep understanding of concepts in Computer Graphics and thorough study of works done in this field, a semester out of two available needs to be allocated for this purpose. The second semester of project can be dedicated for working on the actual solution to the given problem. This project demands a complete framework with implementation of only a prototype. A **basic prototype** could be implemented in the available time.

CHAPTER 3: LITERATURE REVIEW

3.1 BUILDING CONCEPTS

3.1.1 Computer Graphics

Computer graphics is sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content.

Two of the important computer graphics development platforms that have been explored while preparing for this project are OpenGL and XNA. These are mentioned below separately.

OpenGL

OpenGL is a portable and highly efficient open 3D graphics and modeling library. It provides a C-like runtime library that can be used for creating virtual scenes. It is open and freely distributed. It is widely used in many industries for high quality and performance critical modellings like gaming, flight simulation, etc. It is also available for all important hardware and is supported my most of the OS. OpenGL, therefore is typically the first option for any computer graphics programmer or a professional in this area.

XNA

Microsoft **XNA** (a recursive acronym for **XNA's** Not Acronymed) is a freeware set of tools with a managed runtime environment provided by Microsoft that facilitates video game development and management. **XNA** is based on the .NET Framework, with versions that run on Windows NT, Windows Phone and the Xbox 360. XNA content is built with the XNA Game Studio, and played using the XNA Framework (for Windows games), or published as native executable (for Xbox 360, Windows Phone and Zune).

3D Models

A model is essentially a collection of geometry that is rendered together to form an object in the world. A model can be made up of multiple meshes, each mesh being a **linear graph** of **vertices** (*points in a 3D space*), **edges** (*straight lines that connect the vertices to form a graph*) and **faces** (*set of edges that connect together to form a polygon representing a surface of the mesh*). In addition to the basic graph of vertices, a mesh definition can contain information like the **vertex normal** [Term 6], **texture maps** [Term 8], **material and lighting data** [Term 9], etc.

Avatar

Avatar is a specific instance of a 3D Model representing a character, basically a human model. But an Avatar can be much more than just a 3D Model. The meshes that make up an avatar can be arranged in a hierarchical fashion to form a logical structure of the avatar. The avatar can also contain **rigs** [Term 4] and animation data.

3.1.2 Kinect™ Sensor

Figure 3.1: Microsoft Kinect™ Sensor

Kinect™ is Microsoft's motion sensor add-on for the Xbox gaming console. The device provides a natural user interface (NUI) that allows users to interact intuitively and without any intermediary device, such as a controller. The Kinect comes with SDK that enables its use in various other interactive applications. The technology has been applied to real-world applications as diverse as digital signage, virtual shopping, education, telehealth service delivery and other areas of health IT.

There are two types of Kinect, **Kinect for Xbox 360** and **Kinect for Xbox One** built for **Xbox 360** and **Xbox One** respectively.

Kinect SDK

The Kinect for Windows SDK provides the tools and APIs, both native and managed, that you need to develop Kinect-enabled applications for Microsoft Windows. There are many versions of Kinect SDK, the latest one being **Kinect for Windows v2** which is compatible with Kinect for Xbox One only. The earlier versions upto **Kinect for Windows v1.8** is supported by Kinect for Xbox 360.

Kinect Features [4]

The Kinect SDK provides both managed and unmanaged libraries. If you are developing an application using either C# or VB.NET, you can directly invoke the .NET Kinect Runtime APIs; and for C++ applications, you have to interact with the Native Kinect Runtime APIs. Both the types of APIs can talk to the Kinect drivers that are installed as a part of SDK installation.

The Kinect driver can control the camera, depth sensor, audio microphone array, and the motor. Data passes between the sensor and the application in the form of data streams of the following types:

- Color data stream
- Depth data stream
- Audio data stream

The Kinect SDK provides a library to directly interact with the camera sensors, the microphone array, and the motor. We can even extend an application for gesture

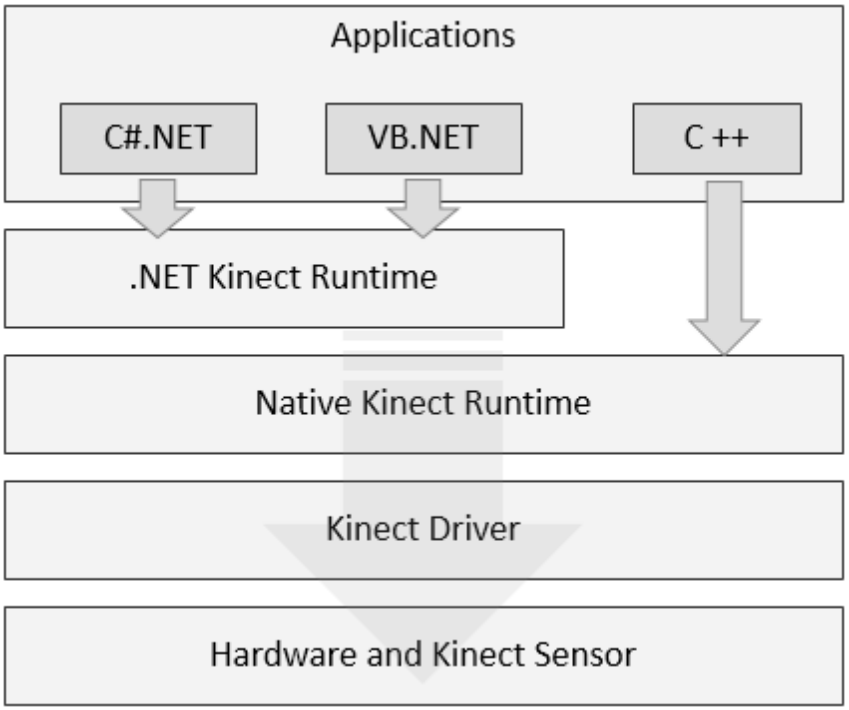


Figure 3.2: The Kinect SDK Architecture

recognition using our body motion, and also enable an application with the capability of speech recognition. The following is the list of operations that one can perform with Kinect SDK.

- Capturing and processing the color image data stream
- Processing the depth image data stream
- Capturing the infrared stream
- Tracking human skeleton and joint movements
- Human gesture recognition
- Capturing the audio stream
- Enabling speech recognition
- Adjusting the Kinect sensor angle
- Getting data from the accelerometer
- Controlling the infrared emitter

3.2 Stepping Stones

This section includes some of the exercises that were undertaken to learn various concepts so that we get sufficiently equipped to dive into solving the problem at hand.

Sprite Display

Aim: To learn the basic game loop of XNA and how to draw 2D sprites in XNA Game Studio.

Figure 3.3: Sprite Display screenshots

Description and Implementation: This is a very simple exercise. A new XNA Game Studio project was created in Visual Studio 2013. A complete and ready-to-run project gets crated with the implementation of the complete **Game Loop.** An image of Swami (Bhagawan Sri Sathya Sai Baba) was included in the content project. The LoadContent() method was updated to load the image as a sprite in the game. The Draw() method was updated to draw the uploaded sprite.

XNA Game Loop and 'Game' Class: XNA has a certain flow of game execution defined by the framework. The flow is implemented using the Template Method Pattern. The basic flow of any game is to load all the contents, initialize all the elements, and then Update and Draw the contents continually until you exit the game. XNA provides the 'Game' class that forms the root of the whole game and which has got virtual methods for Loading, Unloading, Updating, Drawing and so on. We just need to override these methods in a new class derived from 'Game' class and the game will run smoothly. The sequence or the frequency of invocation of these methods is taken care by the framework.

Sprites and SpriteBatch: A sprite in XNA is a 2D renderable object. SpriteBatch is the object that contain multiple sprites and has methods to render them in different ways. A SpriteBatch also renders a font in the screen. The fonts are defined by SpriteFont content item.

SimplePlayer Game

Aim: To learn how to update scenes in XNA to create games.

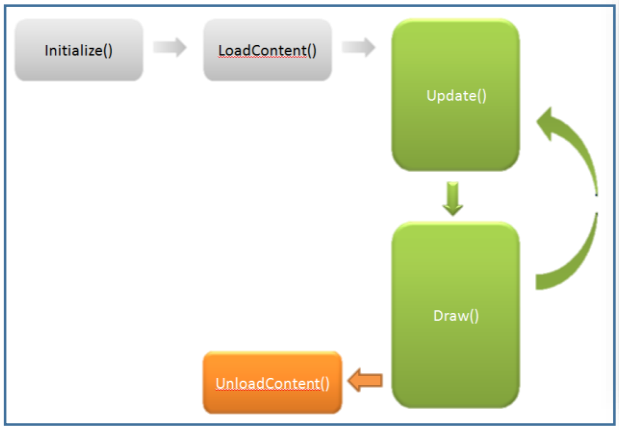


Figure 3.4: XNA Game Loop

Figure 3.5: The SimplePlayer game

Description: This is the simplest game where a space ship is seen travelling in a space. The player can control the movements of the ship.

Implementation: The implementation is rather simple. A sprite is drawn on the background to give the effect of the space and the sprite (image) of the ship is drawn on the foreground. The position at which the spaceship is drawn is changed on key input.

For a next level on the basic game, acceleration and retardation is included. On a long key press, the rate at which the position of the sprite to be drawn on the screen changes (in other words, the velocity of the spaceship) is increased and on leaving the key, the velocity gradually decreases (in contrast to the abrupt stop in the previous implementation)

Model Renderer

Aim: To learn how to import a 3D model from a file and render it in XNA.

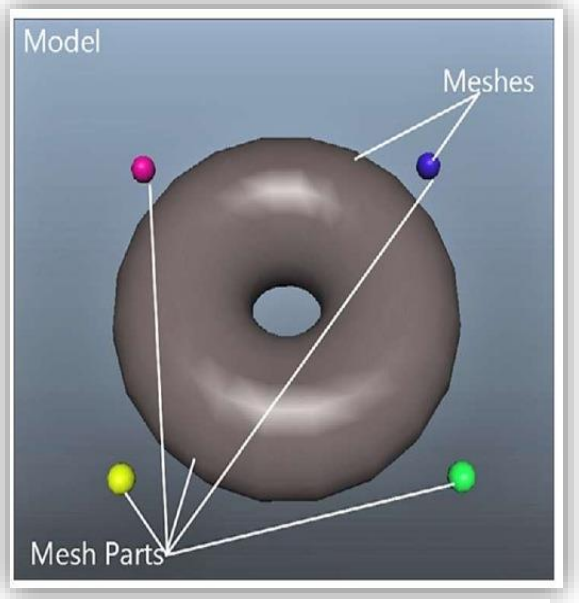
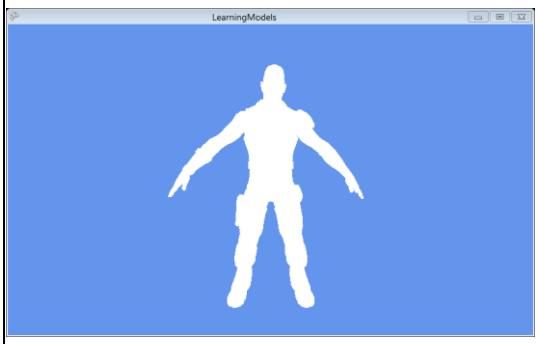


Figure 3.6: Composition of a model in XNA

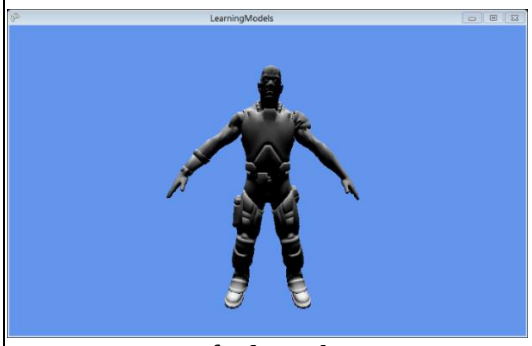
Description: Moving on to 3D graphics from 2D, in this project, I try out rendering the "dude" model used in the Avateering sample of Kinect SDK with different lighting effects.

Implementation: The model can be included in a XNA Game Studio project in the same way as a sprite described above. For rendering, we need to determine the projection matrix for rendering the scene because the model needs to be placed at right place in the viewing volume of the space to get a proper view. I found out the right frustum dimensions, camera position, viewing direction and up direction using trial-and-error method since I am unaware of (I am sure there are ways to find out) the dimensions of the model programmatically to do it automatically.

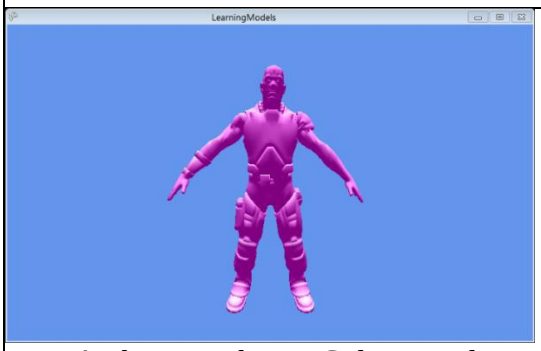
If we render the meshes of the model with the default effect that is loaded from the model data itself, then we get the native look of the model. But, I have tried rendering the meshes with a customized BasicEffect. I try setting the lights like, ambient light, emissive light(the color that the material of the mesh emits back), three directional lights, and specular light. I also try to put a different texture (maple-leaves) on the model and render it. The outputs of different runs with different lighting settings are given below. Their labels indicate the lighting settings used for rendering.



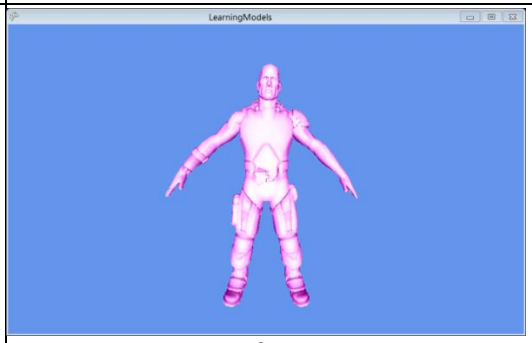
No Lighting



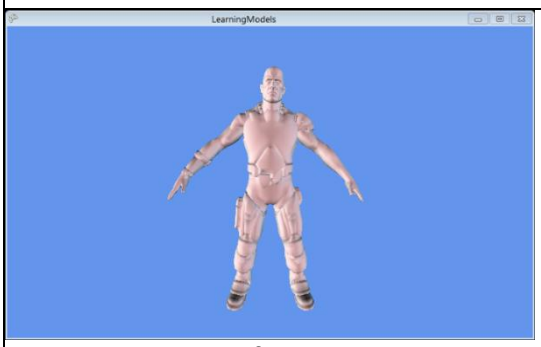
Default Lighting



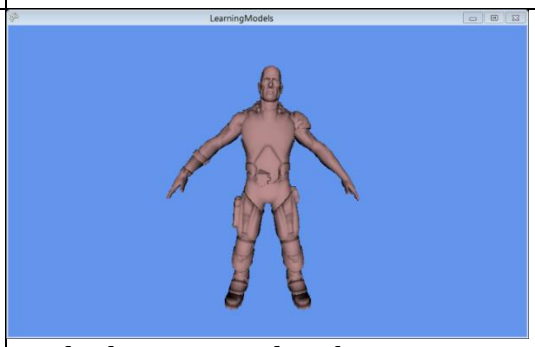
Ambient Lighting, Color: purple



Directional Light(front), Color: white



Directional Light(front), Color: purple



Multiple Directional Lights

Rendering with Texture, (inset) Texture

Rendering with native colors

Figure 3.7: The model rendered in different formats

Rubik's Cube Animation:

Aim: To learn basic animation of a model in XNA.

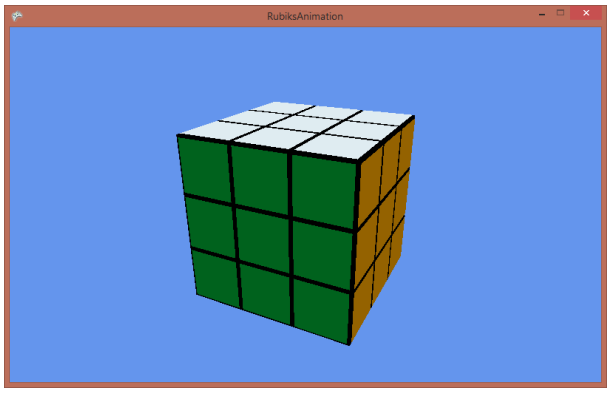


Figure 3.8: The animated Rubik's Cube model

Description and Implementation: I try my first venture into animation by trying to animate rotations of different faces of a 3x3x3 rubik's cube. The project 'RubiksCubeAnimation' is loaded with a model of rubik's cube. I use the techniques learnt in the 'LearningModels' project where I played around with lightings to render the rubiks cube model too. But, this time, I can't afford to give a single static view of the rubik's cube like in the previous project. So, I create a class that handles the camera settings and provides way to move the camera in a "arc-ball" fashion. With this, I am set to start animating the movements of the faces of the Rubik's Cube.

First set of information required before starting the animation is to know which meshpart of the model object refers to which unit of the Rubik's Cube. This information too, I found out by rendering each meshpart one by one and noting down the corresponding unit that was drawn. At last, I converted the mapping to a function that converts the index of a unit of the cube in my system of indexing into the index of the meshpart in the model. Hence, the mapping is done.

Now, my approach of looking at a rubiks cube is as follows. A rubiks cube has got 26 positions, each having multiple candidate cube-elements(I mean a smaller unit of cube as cube element). So, first thing is to create a **CubePosition** class that represents a position of the cube and contains a reference to the the

cube-element that is currently in that position. Now, I create **CubeFace** class that represents the nine rotateable faces (including the mid portions) of the cube. These faces will each have references to nine cube positions that make up the face. (Even the mid portions will have nine positions, the center one being a dummy one which, in turn, has no actual cube unit contained in it.) Now, rotating a face means is as simple as shifting the cube units in the positions defined by the face appropriately. The CubeFace class has methods for doing these. All these cube faces are encapsulated in a class **CubeState** that abstracts all the implementations, providing only public functions to rotate the cube. The rotation, though is just a shift in position of all the cube units in particular face, needs to be animated for showing the transition graphically. This gradual transition is implemented in the class representing the cube-unit, i.e., **CubeUnit.**

Now, it is time for implementing the animation. After loading the cube object in the game class, I need to create an instance of my CubeState class. Once it is created, I map a key input to the rotation of the face of the cube respectively, also in the 'game' class. Now, my rubiks cube is ready to run and play around with.

Yippee! I have achieved my first animation. But, there seems to be a problem. The cube model that is rendered can be viewed from any direction by moving camera in an arc-ball curve. I have mapped keyboard inputs to the rotation of each of the nine faces of the cube. But the rotating faces are defined with respect to the initial position of the camera. That means, when I press 'r' key to rotate the right face of the cube w.r.t the initial position of the camera, when the camera is moved to the back, same key 'r' moves the right face. This is because, when the camera is moved, the definition of faces of the cube w.r.t the camera position changes in reality but not in the program. Hence, we need a model that maps the definition of the face from the present camera perspective into the one from initial camera perspective.

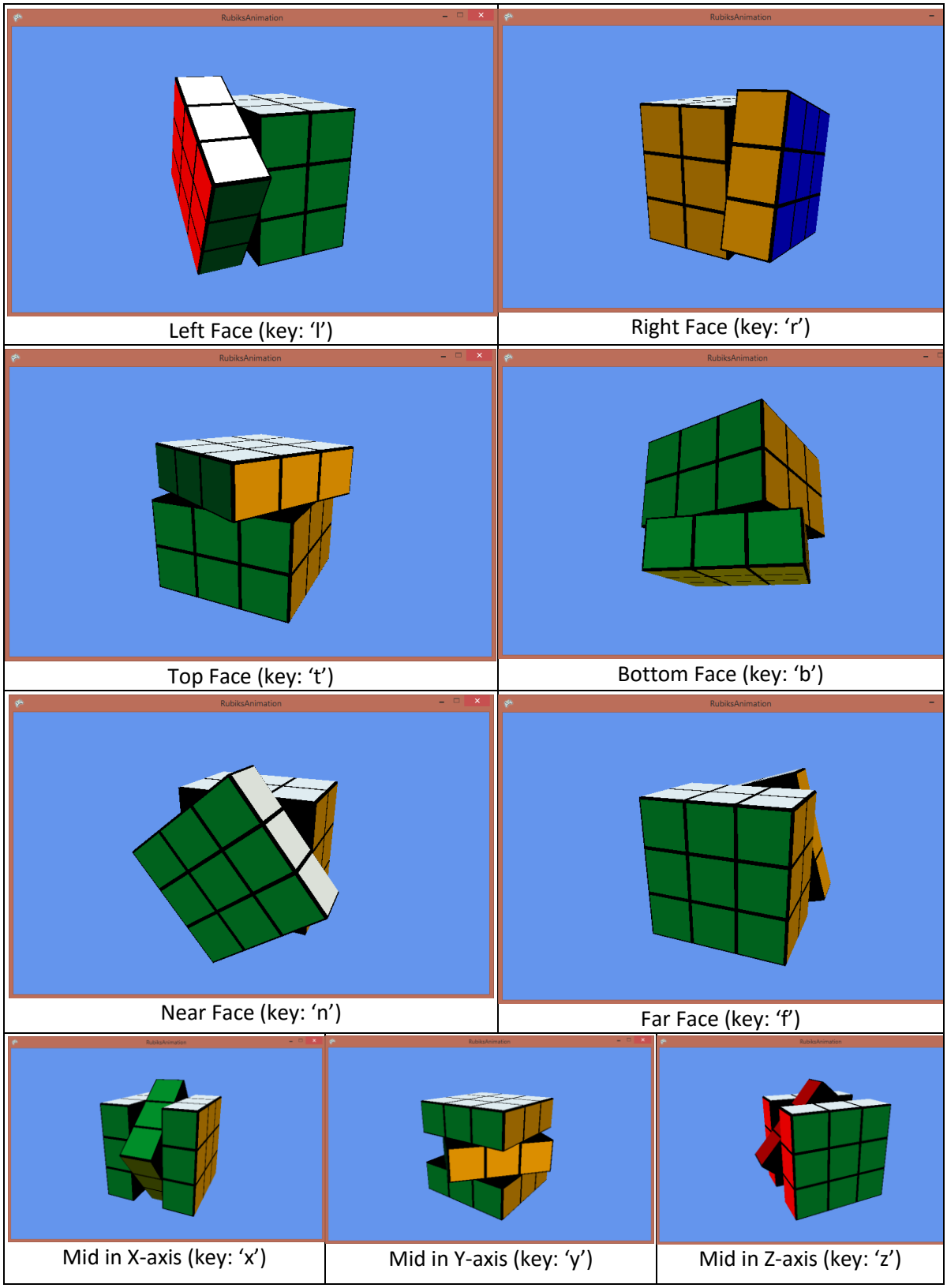


Figure 3.9: Screenshots of Rubik's Cube Animation

The solution approach is as follows: Since, the cube is stationary and the camera is moving, the co-ordinate axes of the cube is same as the axes of global coordinate system. Hence, I find the global axis nearest to i) camera position vector and ii) camera up vector, by calculating the angle between the camera/up vector and each of the axis vectors and comparing them to find the minimum. So, we now have two global axes that are 'front' and 'up' according to the current camera position. We can easily map these global axes to the faces of the cube when the camera is initially at the positive z-axis. Having mapped two faces, (the 'top' and 'near' faces w.r.t to current camera position to two faces w.r.t the initial camera position) we can derive the mapping for any other faces. Once this mapping is established, we can rotate camera to view the cube from any angle and press a key to rotate a face of the cube, the face that is intended to rotate only is rotated.

3.3 Field Study

A lot of work has been carried out in the field of Avatar Animation. This field is presently one of the most researched area. The following sections deals with some of the study undertaken on work done in each of the field of Avatar Animation, i.e., Avatar Mesh Creation, Rigging and Animating the Avatar.

3.3.1 Avatar Mesh Creation

There are numerous means of creating an avatar. Some of them are:

1. **Manual creation using 3D Modelling Software**: There are numerous 3D Modelling applications available in the market with variety ranging from open source to proprietary, very expensive to free ones, sophisticated to simple, and so on. Some of the popular ones are Autodesk Maya, Blender, 3ds Max, etc. One can have full control on the features of the avatar created

by this technique. But it also requires artistic talents and professional expertise supported by knowledge about the software.

2. **API based creation**: There are APIs available for different platforms that define set of methods for programmatically creating an avatar based on templates. One such example is *Avatar API* provided by *XNA Game Studio*.
3. **3D Shape Acquisition by Scanning**: There are various commodity sensors available that can be used to take a 3D scan of a human being. The data from the sensor can be processed and combined to get a wholesome 3D mesh of the person. This has become one of the most widely used technique for avatar creation of late because of its inherent simplicity.
4. **3D Reconstruction**: This technique involves processing 2D image/video data to create the corresponding 3D model. Usually, it is a pipeline involving several phases, where each phase extracts a different feature from the data adding value to the target 3D model.

Among the four broad categories of methods mentioned above to create an Avatar, the first one is not automatic and requires artistic skills and professional expertise. So, the remaining three are studied further.

3D Shape Acquisition by Scanning:

There has been many works done and applications developed for creating a 3D model of a person dynamically by scanning the person using commodity sensors like Kinect. *Rapid Avatar Capture and Simulation using Commodity Depth Sensors* by **Ari Shapiro** [14] et al. is a good example.

One of the most important tool for 3D shape acquisition is this feature provided by Kinect SDK called *Kinect Fusion*. The Kinect Fusion system reconstructs a single dense surface model with smooth surfaces by integrating the depth data from Kinect over time from multiple viewpoints. The camera pose is tracked as the sensor is moved (its location and orientation) and because we now know each frame's pose and how it relates to the others, these multiple viewpoints of the objects or environment can be fused (averaged) together into a single reconstruction voxel volume. Think of a large virtual cube in space (the reconstruction volume), located around your scene in the real world, and depth data (i.e.

measurements of where the surfaces are) are put (integrated) into this as the sensor is moved around.

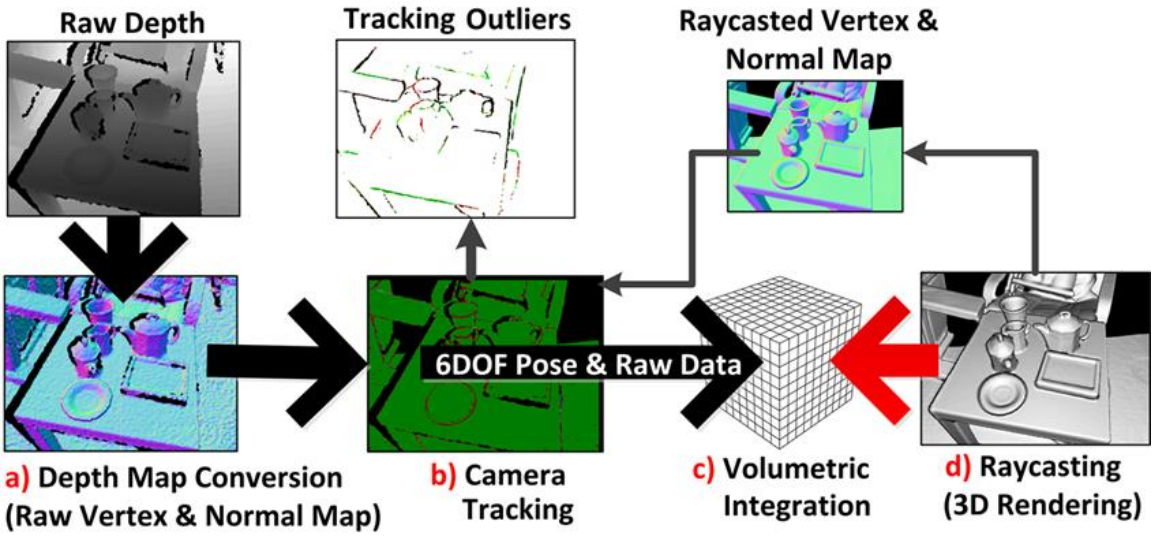


Figure 3.10: The Kinect Fusion pipeline

3D Reconstruction

A lot of work has been done in creating a 3D Avatar form Image samples. Specifically, creation of 3D model of face has been done even using single images [8] or video [9].

The work of Naveed Ahmed et al. [] explains method for full avatar mesh creation. A pipeline is presented that combines a set of hardware-accelerated stages into one seamless system. Primary stages in this pipeline include pose estimation, skeleton fitting, body part segmentation, geometry construction and coloring, leading to avatars that can be animated and included into interactive environments. This is a typical flow for creating 3D avatars from 2D image/video data.

API based Mesh Creation

There are APIs that provide predefined avatars that one can customize. The APIs also provide parameters that can be tuned to create variations of the avatar. Methods are also provided for animations, like laughing, yawning, jumping, sitting down, getting up, etc. An example of such API is the Avatar API of XNA. [11]

3.3.2 Avatar Mesh Rigging

Before we choose the method of rigging, we need to decide upon the method of animation that we have to follow, because that defines what rig do we need to define for the mesh. There are many different types of animation processes. [12] Out of all, **Skeletal Animation** is nearest match to the actual anatomy of human body.

Skeletal Animation

Skeletal animation is a technique in computer animation in which a character is represented in two parts: a surface representation used to draw the character (called *skin* or *mesh*) and a hierarchical set of interconnected bones (called the *skeleton* or *rig*) used to animate (*pose* and *keyframe*) the mesh.

This technique is used by constructing a series of 'bones,' sometimes referred to as *rigging*. Each bone has a three-dimensional transformation (which includes its position, scale and orientation), and an optional parent bone. The bones therefore form a hierarchy. The full transform of a child node is the product of its parent transform and its own transform. So moving a thigh-bone will move the lower leg too. As the character is animated, the bones change their transformation over time, under the influence of some animation controller.

Each bone in the skeleton is associated with some portion of the character's visual representation. *Skinning* is the process of creating this association. In the most common case of a polygonal mesh character, the bone is associated with a group of vertices; for example, in a model of a human being, the 'thigh' bone would be associated with the vertices making up the polygons in the model's thigh. Portions of the character's skin can normally be associated with multiple bones, each one having a scaling factors called vertex weights, or blend weights. The movement of skin near the joints of two bones, can therefore be influenced by both bones. In most state-of-the-art graphical engines, the skinning process is done on the GPU thanks to a shader program.

This technique is used in virtually all animation systems where simplified user interfaces allows animators to control often complex algorithms and a huge amount of geometry; most notably through inverse kinematics and other "goal-oriented" techniques.

Automatic Rigging Process

Number of works are available on automatic rigging of Avatar. Some of them are studied here.

Natapon Pantuwong and *Masanori Sugimoto* in their work, "*A fully automatic rigging algorithm for 3D character animation"* [15] propose an automatic algorithm to generate an inverse kinematic (IK) skeleton for 3D characters. The extracted curve skeleton is then analyzed and classified by comparing it with the characteristics of templates for real animals. The outcome of this classification step is an associated class for the given 3D character model, together with the meaning of each skeleton segment. Next, each bone of the template skeleton is located in the appropriate skeleton segment of the input skeleton graph.

"*RigMesh: Automatic Rigging for Part-Based Shape Modeling and Deformation*", the work of Peter Borosan et al. [16], introduce algorithms and a user-interface for sketch-based 3D modeling that unify the modeling and rigging stages of the 3D character animation pipeline. Their algorithms create a rig for each sketched part in real-time, and update the rig as parts are merged or cut. As a result, users can freely pose and animate their shapes and characters while rapidly iterating on the base shape.

"Automatic rigging and animation of 3D characters" is another well-accepted work by Ilya Baran and Jovan Popovic [17]. The present a method for animating characters automatically. Given a static character mesh and a generic skeleton, our method adapts the skeleton to the character and attaches it to the surface, allowing skeletal motion data to animate the character.

3.3.3 Motion Retargeting

Applying a given motion data to a target mesh for creating dynamic animation is referred to as Motion Retargeting. The source of motion data can be anything, like, a predefined motion data from a file, motion commands or stream data from sensors like Kinect. AvateeringXNA sample is a good example for learning Kinect Motion Retargeting.

Chris Hecker et al. in their paper [18] introduces a novel system for animating characters whose morphologies are unknown at the time the animation is created.

CHAPTER 4: APPROACHING SOLUTION

4.1 THE START

In the start of the academic year, when the project was allocated to me, I was a novice in Computer Graphics and Animation. So, the first step for me was to install the drivers and API for the Kinect and explore the features. With this intention, I downloaded and installed the **Kinect for Windows SDK v1.8** and **Kinect for Windows Toolkit v1.8** for the Kinect for Xbox 360 that the department had provided for me. The Toolkit came with **Developer Toolkit Browser**, an application that has many sample programs with options to **install** the source code for developing with Visual Studio and to **run** the application.

Once I started exploring all the features of Kinect through the sample programs, I came across this beautiful sample called **AvateeringXNA**. The sample has a beautifully created and rigged model called "*dude*" that shows up on the screen and if a Kinect is connected to the system, the avatar mimics the movement of a person standing in front of the Kinect. Yes, *that was my starting point*.

I started by studying and trying to understand the various techniques used in the sample. The terms like *mesh*, *skeleton*, *bone*, *skinning*, etc. used in the sample are the ones which pushed me into the ocean of Avatar Animation. When I found out that the method of animation used in the sample was just one among many, I started studying about other alternatives that would fit the requirement of this project better.

After a considerable amount of study spanning a few months, with a considerable knowledge in Computer Graphics and animation techniques, supported by hands on exercises undertaken to understand various concepts (as described in Section 3.2), I could comprehend the design approach of the AvateeringXNA sample quite well. It was then, I found out the following differences between the implementation of AvateeringXNA sample and the requirement of my project.

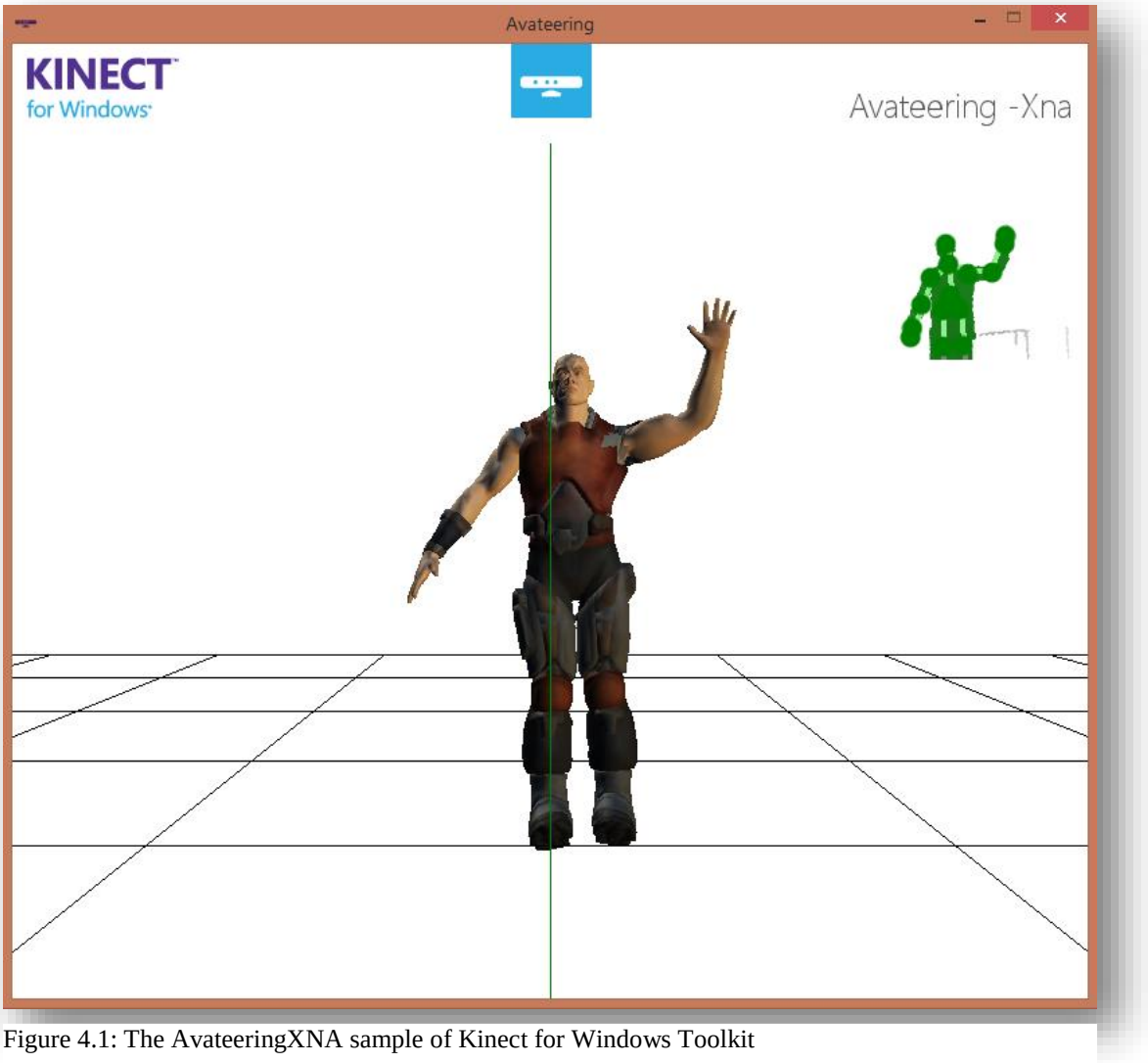


Figure 4.1: The AvateeringXNA sample of Kinect for Windows Toolkit

i. The sample works with the model "dude" that was most certainly created by an expert using sophisticated 3D modelling application. *Our system needs automatic avatar model creation.*

ii. The model is pre-rigged. That is, the model comes with the controls that are required to deform the avatar mesh in order to create animation. *A model that is automatically created in our system has to be automatically rigged so that they can be deformed during animation.*

iii. To make the AvateeringXNA work for any other model, the source code needs to be modified to cope with the *anatomy* and *control rigs* of the new

avatar. *The system to be built should be designed to seamlessly integrate the dynamically created avatars into its pipeline.*

iv. The sample does not include features like fingers movement tracking, facial expression recognition, etc. Our system needs to be 'realistic' that includes every perceivable feature in Avatar Animation. Gestures and expressions hence are all the more important.

v. The sample does not involve sound at all. The system we need to build would be incomplete without the voice of the avatar.

With this understanding, I was set to venture into thinking about the solution model to the problem at hand.

4.2 NAROVATAR FRAMEWORK

In this section, I propose a conceptual framework for an automatic and complete avatar animation pipeline, called the Narovatar Framework.

As shown in Figure 1.5, the framework shall consist of three broad modules – automatic mesh creation, automatic rigging and motion data retargeting.

Mesh Creation

There are two methods of automatic creation of avatar mesh as discussed in the Section 3.3, first one being shape acquisition by 3D scanning of a person and the second being 3D reconstruction from 2D image/video data. The framework shall include both these methods in the pipeline as alternatives. This is because both of them are relevant for their own scenario and input sets. When the person whose avatar is to be created is available in person, the 3D scanning technique suits well instead of taking his image and running a 3D algorithm to create the mesh. In cases where the avatar to be created is not available in person, or when the avatar is modified human being, like that in the famous movie called 'Avatar', the 3D reconstruction from images and artistic sketches would be more relevant.

Rigging

Avatar rigging is another area where wide variety of options are available. This is the phase where the technique for animation has to be decided upon. The most popular technique used in most of the animations is the *skeletal animation*. This techniques involves defining *bones* that combine in a hierarchy to form a *skeleton* which deforms the mesh. Since this technique closely resembles the actual human anatomy and hence enables a realistic animation of a human avatar, this technique is chosen for the framework.

Rigging the body with skeleton for the movement of limbs, head, etc. are not enough. For a more subtle animations like facial animations, special rigging is required. Hence, the face of the mesh needs to be separately rigged providing finer control. So, the framework has separate modules for rigging the whole mesh and the face separately.

Motion Retargeting

The last phase of motion retargeting is the one that fructifies the background work done in the previous two phases. In our scenario, we consider two dynamic motion sources for automatic motion retargeting.

The first one is from a commodity sensor like the Kinect. We can extract the data like skeleton stream (the whole body pose of a person for each frame), points in the face for facial expression recognition, speech command, etc. These data can be directly applied in the avatar animation.

The second source of animation of data would be action command. A complete set of basic actions can be defined, like, *stand up, sit down, turn, look up, look down,* etc., which can make up complex animations when executed in sequence. Each of these basic actions will have a set of animation data that can be predefined. Like sitting down requires bending two legs, which in animation terms will be rotating the knee joint by certain degrees.

The following diagram (Figure 4.2) illustrates the logical view of the framework.

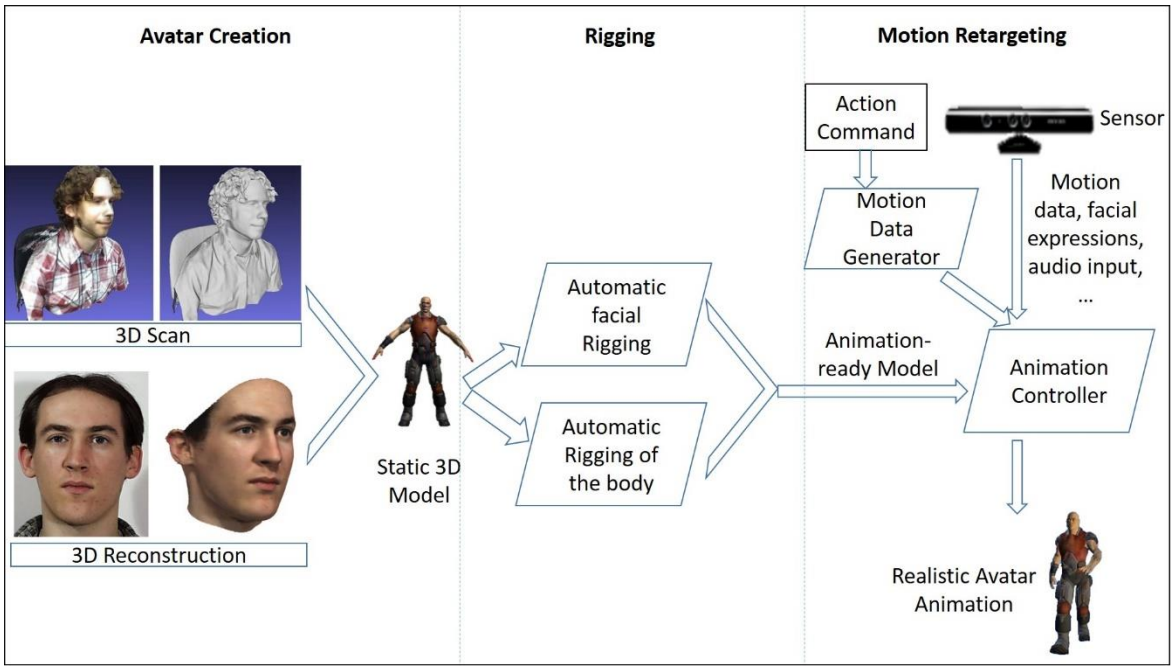


Figure 4.2: Logical view of the Narovatar Framework.

CHAPTER 5: IMPLEMENTATION

5.1 NAROVATAR

5.1.1 System Design

Based on the Narovatar framework, a system (also called **Narovatar**) is built. The choices of methods for each of the modules of the Narovatar Framework in this implementation are discussed below.

Avatar Creation

For this initial implementation, among the two basic methods of avatar model creation, the method of avatar mesh acquisition by 3D scanning was selected for its convenience. For this, we have a Kinect sensor with Kinect Fusion API available in its SDK. Also, there are various programs built in for this purpose. On the other hand, reconstructing a mesh from 2D images/videos would involve a more in-depth knowledge in the field and would therefore be difficult to be achieved in the available time.

Automatic Rigging

There are two rigging steps required to be performed as specified in the Narovatar Framework, rigging of the whole body for pose animation and specific rigging of head for facial expressions. Clearly, the rigging of whole body for body pose animation should be done first.

There are many works available for this. But we have chosen the implementation by Ilya Baran and Jovan Popovic, titled *"Automatic Rigging and Animation of 3D Characters", SIGGRAPH 2007.* They also provide demo implementation in code for their work [5]. Taking their implementation, with necessary changes, we fill in the second module of automatic rigging of the Avatar model.

Kinect Motion Retargeting

With the Kinect for Xbox 360 sensor available, it becomes our obvious choice as the source of motion data for avatar animation. We also have the *AvateeringXNA* sample of the *Kinect for Windows SDK* for reference.

With the above choices made, the system view of Narovatar would be as shown in the following diagram.

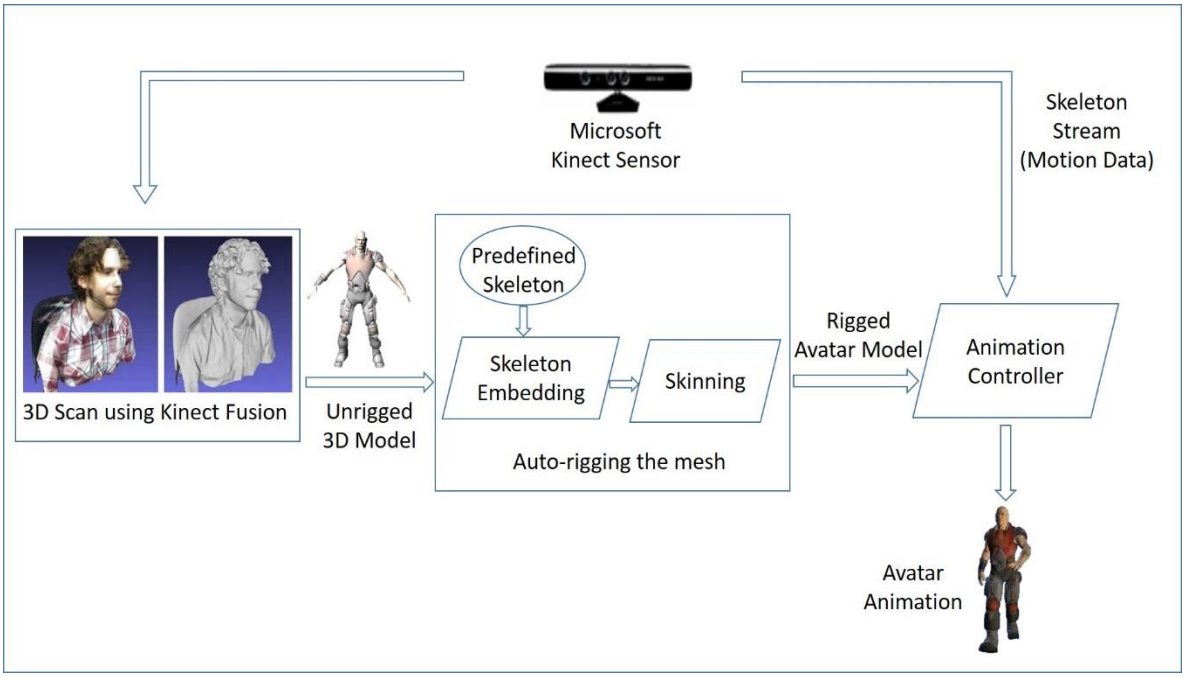


Figure 5.1: Narovatar System View

5.1.2 Implementation Details

Intuitively, the first module should have been implemented first and the pipeline should have continued. But we decided to implement Narovatar starting from the rigging phase. We abstracted out the first phase by taking available 3D mesh of a person as input to build the second phase of rigging.

The Rigging Module

For the rigging module, we first downloaded the demo implementation of the work by Ilya Baran and Johan Popovic (as mentioned in the previous section) called **Pinocchio** from their website.

Firstly, a thorough study on the working of Pinocchio by code walkthrough using readme files and the paper published by the authors was undertaken. The first thing that we found out was that Pinocchio could import and hence rig only the models in **Wavefront object (.obj) file** format. But according to the requirements of our project, the avatar mesh could be created in any file format depending upon the process of creation and the application that creates it. So, a general 3D model importer was required for this sake. **AssImp (Open Asset Import Library)** was found to be suiting the requirements.

AssImp imports a given 3D model from files of supported format into its own common data structure. The Pinocchio library on the other hand, defines its own data structure to rig the mesh. So, the next task is to understand the design of both the data structure and copy the mesh data from the AssImp *Scene* data structure to the Pinocchio *Mesh* data structure.

Once this adaptation work was successfully completed, the Pinocchio rigging library function could be invoked after the data is imported using AssImp library. But, now we need to render it. Since we are using Pinocchio rigging library code and also AssImp importer library both of which are implemented in C++, the whole is getting built up in C++. Hence XNA, which is in C#, is not an option for mesh rendering. So, the other option is OpenGL.

Now, it's the time to get started with advanced OpenGL rendering techniques. The tutorials series: OpenGL Step by Step - OpenGL Development [6] came as help in this aspect. The Tutorial 22 of the series specifically talks about rendering 3D scenes imported using AssImp library. The author also provides demo programs in the tutorial. We could easily adapt the code into our project and render the rigged model. We have now got a rigged model on our screen.

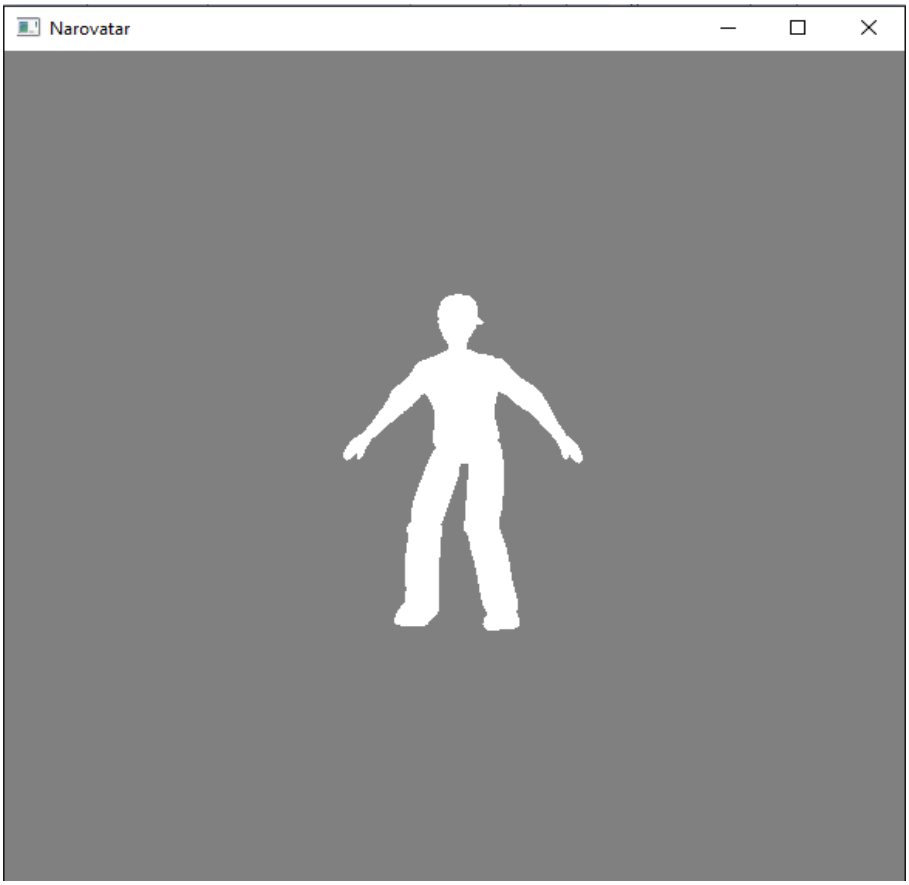


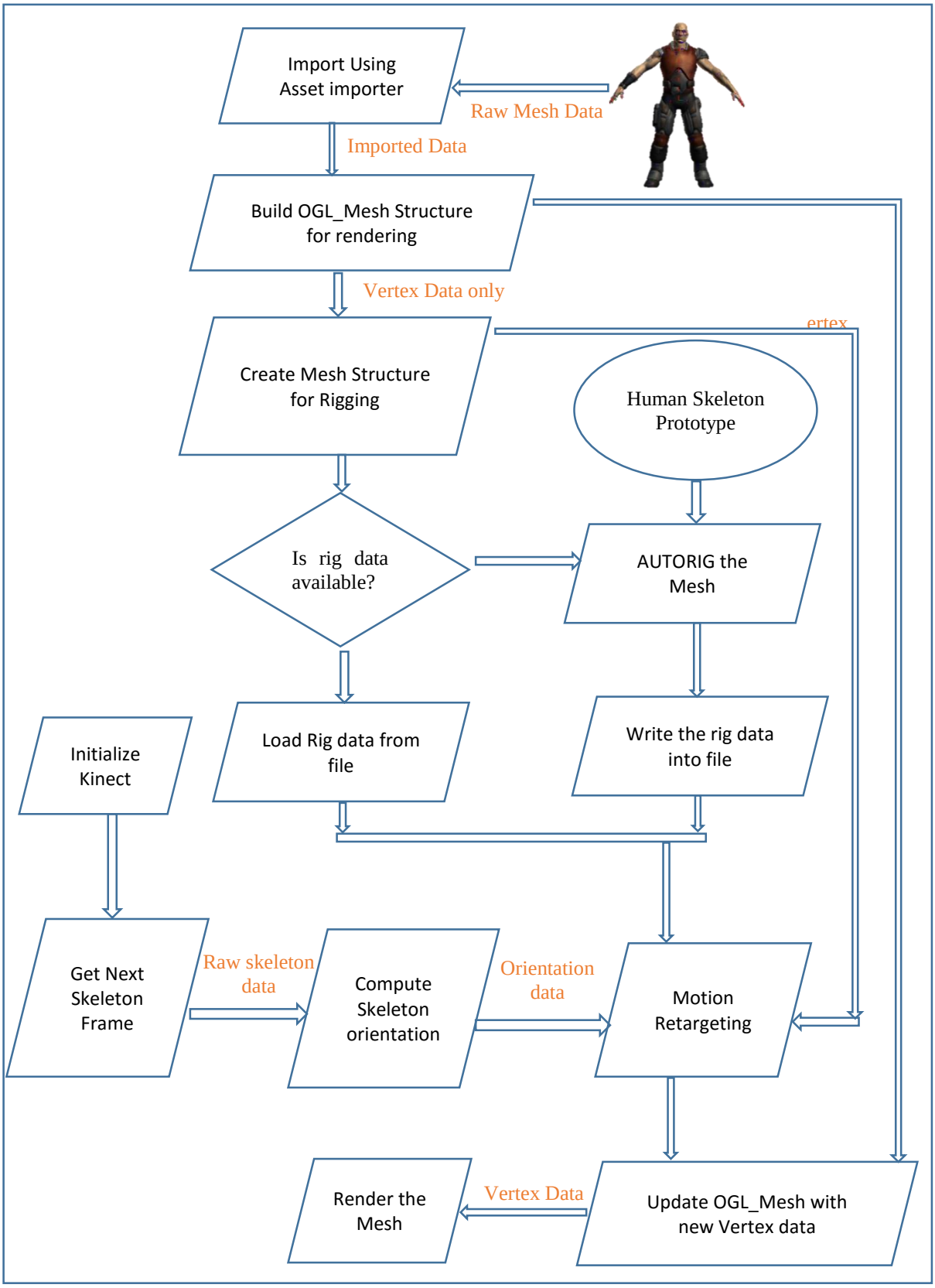
Figure 5.2: The rigged 3D model rendered using OpenGL

Kinect Motion Retargeting

Now, we are ready to move to the third module of the Avateering, i.e., Motion Retargeting using motion captured using Kinect Sensor. Our first goal would be to capture motion data from Kinect.

We know that Kinect SDK is available for two languages, C++ and C#. As we are working in C++, we use the C++ API of Kinect SDK. The skeleton stream of the user standing in front of the Kinect can be accessed as a stream using the API. Each skeleton frame contains the orientation of the all the bones. This orientation is then applied to the skeleton of our rigged model. Then, the mesh is deformed based on the new skeleton orientation.

Now, we have completed the basic implementation of Narovatar Framework with the first module to Avatar Mesh Creation abstracted out. The system by far is as follows:


Import Using Asset importer
Raw Mesh Data
Imported Data
Build OGL_Mesh Structure for rendering
Vertex Data only
ertex
Create Mesh Structure for Rigging
Human Skeleton Prototype
Is rig data available?
AUTORIG the Mesh
Initialize Kinect
Load Rig data from file
Write the rig data into file
Get Next Skeleton Frame
Raw skeleton data
Compute Skeleton orientation
Orientation data
Motion Retargeting
Render the Mesh
Vertex Data
Update OGL_Mesh with new Vertex data

Results Analysis

On running this code, we can see the first animation of auto-rigged mesh as seen in the screenshots given in the next section.

As you can notice in the output, there is a gap between the actual body pose and the orientation of the mesh. There can be two reasons for this gap:

i. Improper mapping of skeleton orientation from the Kinect Skeleton to the skeleton of the avatar.

ii. Noise in the skeleton input stream.

The first issue can be solved by revising the logic in implementation and debugging. The second issue is more serious. So, again, we refer to the AvateeringXNA sample of Kinect Developer Toolkit. On analyzing the implementation of the sample, we find out that there are ***filters*** that they use to filter out the noise and for post-processing steps like mirroring the skeleton orientation, etc.

Now, the next step would be to implement the filters. But that would require some more study on filtering techniques. So, an alternate plan was conceived.

Another way

The basic Idea of taking another way is that, we have the filters implemented in the AvateeringXNA sample. We can, therefore, make use of the existing code to animate our rigged model. So, now Narovatar is no longer a single program that includes all the modules, but different programs for different modules.

The rigging module shall be based on the existing implementation in C++. But the rigged model shall now be exported to a file with skeleton and skinning data. The rigged model then can be imported in a C# application, which will be a modification of the AvateeringXNA sample so that the animation can be done using XNA framework using the existing implementations of motion data filters.

So, the two major tasks involved are, exporting the mesh with rigging data to a file and adapting the AvateeringXNA sample implementation to animate the model rigged by our application.

AssImp, in addition to being an Asset Importer, it also supports exporting the scenes to a few file formats, *and that's exactly what we want!* So, we can use the AssImp

exporter to export our rigged model in **COLLADA (.dae)** format. This format is chosen because it can include skeleton data and is well documented open-source format widely used in many commercial applications.

With exporter in place, the next task ahead of us is to import the rigged mesh in COLLADA format into the C# application for animation. This could be easier if XNA had come with a default COLLADA importer. But XNA by default supports import of *FBX* and *Object (.obj)* formats only. So, we need a COLLADA importer for XNA. Luckily we found an implementation in GitHub. [7] We could build and link the library as importer for our model.

After importing the model, in order to animate it, the AvateeringXNA implementation could be adapted with modifications of a few methods to adapt to the target model.

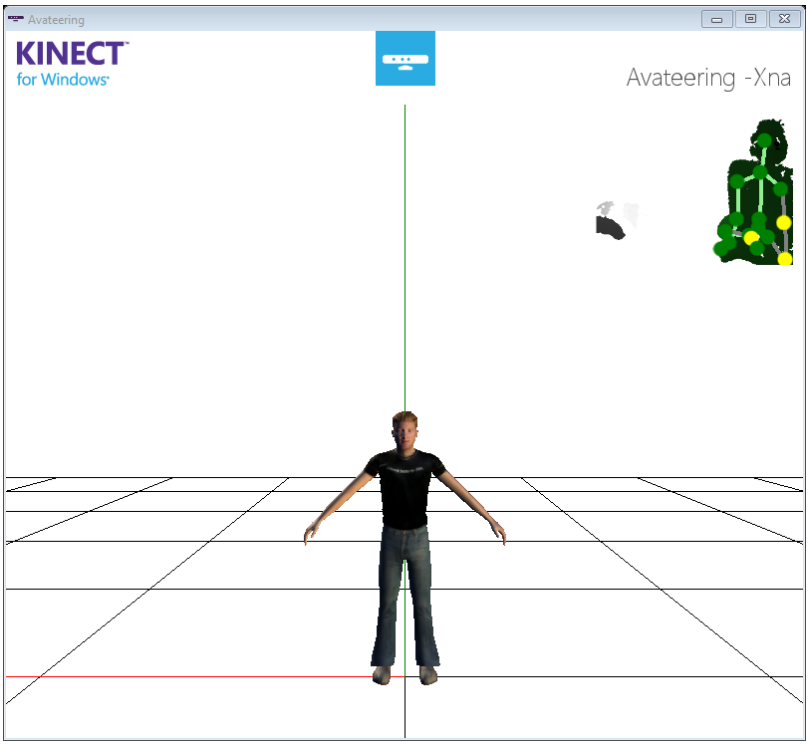


Figure 5.3: COLLADA model imported in modified AvateeringXNA sample in bind pose

The technique has worked well till now. So, the next task is to combine the two modules in a sequence. So, we updated the Scene imported by AssImp with rigging data

obtained from the Pinocchio library. Then, we used the AssImp exporter for exporting the updated scene.

Result Analysis of the second implementation

The output file from the rigging module was imported in Blender (a 3D modelling tool) to study its integrity. But it was found out that the avatar model contains only the mesh but not the skeleton information. There are two possible reasons for this:

i. The rigging information was not set in the data structure of AssImp during update.
ii. The existing export method of AssImp does not suit our requirement.

To the best of our knowledge, the first possibility is well taken care. On a quick study of the implementation of export method of AssImp library, it was found out that it didn't include logic for creating <*bone*> tags used for storing skeleton information in a COLLADA file. Hence, it is concluded that for this implementation to function, we need a **custom exporter** of an avatar model in COLLADA format. The implementation of the custom exporter could not be completed as part of this project.

5.2 OUTPUT SCREENS

The following screenshots show rigged mesh deform in Narovatar.

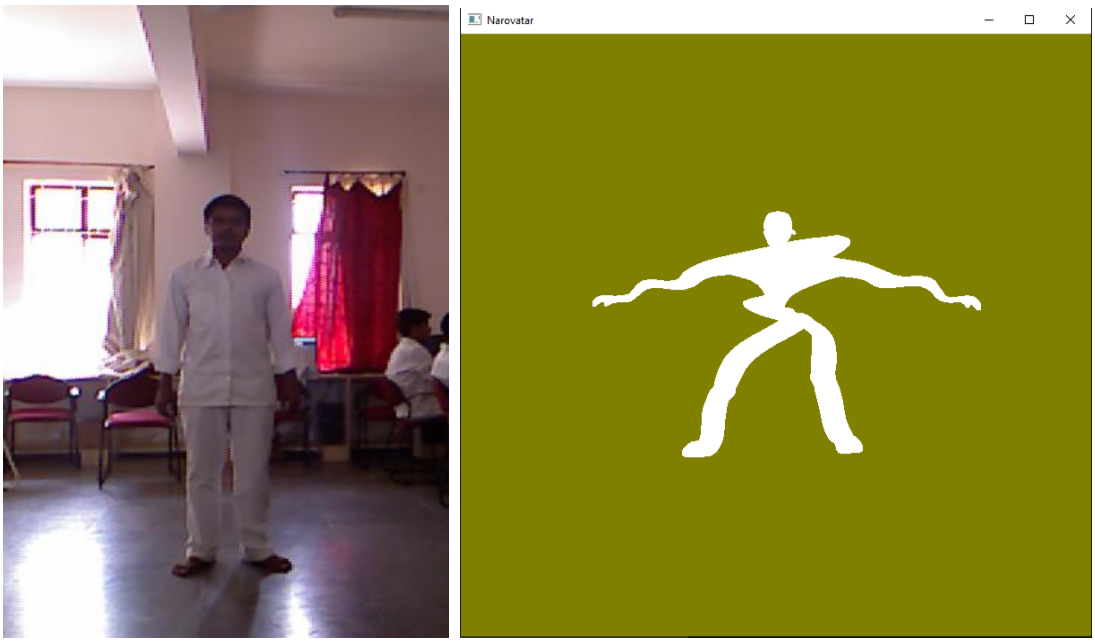


Figure 5.4: Pose 1 Normal Standing Posture

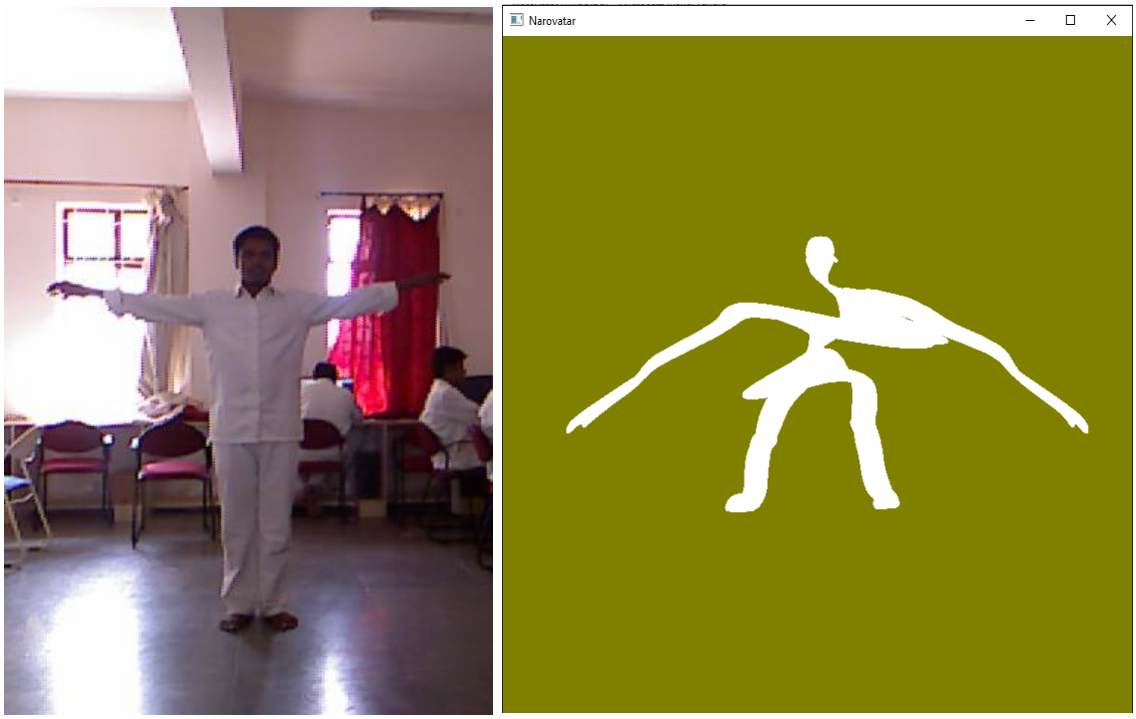


Figure 5.5: Pose 2 T-pose

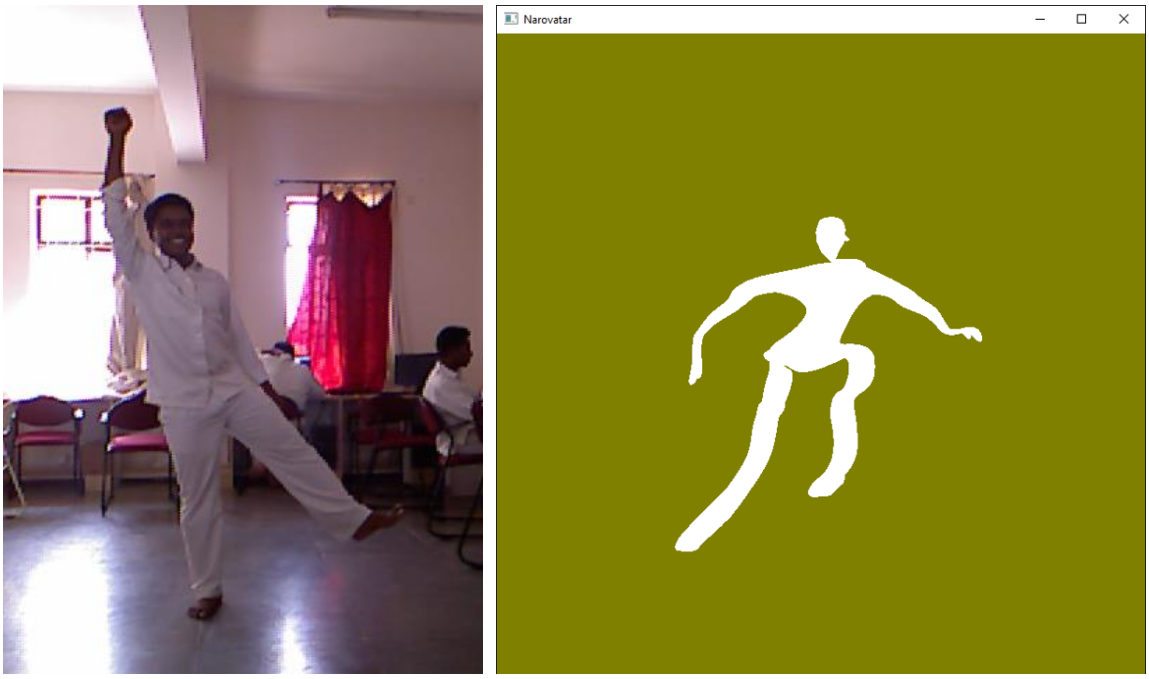


Figure 5.7: Pose 3 Right hand up, left leg lifted sideward

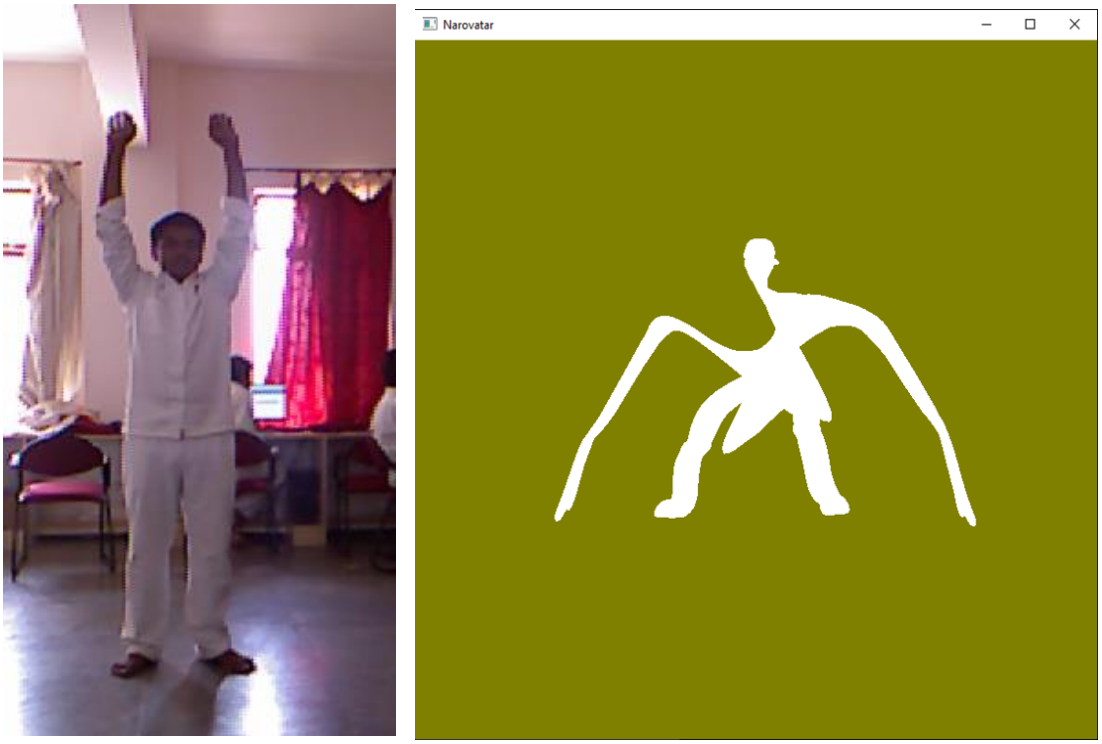


Figure 5.6: Pose 4: Both the hands up

CHAPTER 6: FUTURE SCOPE

As mentioned earlier, this project is just the first step taken towards a much sophisticated software that produces a realistic avatar animation. So, the future scope for this project can be divided in the following three broad categories:

1. Improvements
2. Filling in
3. Enhancements

Improvements:

The modules implemented in this project were implemented with an aim to provide a prototype of an automatic Avateering system but with sufficient scope for improvement. So, any part can be replaced with a better implementation.

Filling in:

There are many modules missing in the current implementations of Narovatar Framework that keeps from being a complete animation pipeline. Some of them are:

- The entire first module- *Avatar Model Creation*
- *Custom exporter of the avatar model in COLLADA format* in the second implementation of Narovatar.
- *Facial Rigging* of the avatar for animating facial expressions.
- Animating the avatar using *motion commands*.
- Including *speech* in the Avatar Animation.

Enhancements:

Enhancement is possible in the Narovatar Framework itself. The ultimate goal of the project is to implement the Virtual Interactive System. Any further enhancement can be done on the Narovatar Framework for achieving this goal, or to create a realistic animation.

REFERENCES

1. Atun-Einy O, Lotan M, Harel Y, Shavit E, Burstein S, Kempner G. Physical Therapy for Young Children Diagnosed with Autism Spectrum Disorders–Clinical Frameworks Model in an Israeli Setting. Frontiers in Paediatrics. 2013; 1:19.
2. Myers SM, Johnson CP (2007). *"Management of children with autism spectrum disorders"*. Pediatrics **120** (5): 1162–82.
3. Pinel JPG. *Biopsychology*. Boston, Massachusetts: Pearson; 2011. ISBN 978-0-205-03099-6. p. 235.
4. Abhijit Jana (2012), *Kinect for Windows SDK Programming Guide*: ISBN 978-1-84969-238-0. Looking Inside the Kinect SDK, p. 34.
5. http://www.mit.edu/~ibaran/autorig/pinocchio.html
6. OpenGL Step by Step - OpenGL Development http://ogldev.atspace.co.uk/ http://ogldev.atspace.co.uk/www/tutorial22/tutorial22.html
7. Bunkerbewohner/ColladaXna Wiki https://github.com/Bunkerbewohner/ColladaXna/wiki
8. Ira Kemelmacher-Shlizerman, Ronen Basri, *Member, IEE3D Face Reconstruction from a Single Image using a Single Reference Face Shape*, IEEE Transactions On Pattern Analysis And Machine Intelligence
9. P. Breuer, K. I. Kim, W. Kienzle, V. Blanz, B. Scḧolkopf *Automatic 3D Face Reconstruction from Single Images or Video* Max Planck Institute for Biological Cybernetics.
10. Naveed Ahmed, Edilson de Aguiar, Christian Theobalt, Marcus Magnor, Hans-Peter Seidel *Automatic Generation of Personalized Human Avatars from Multi-view Video*
11. http://what-when-how.com/xna-game-studio-4-0-programmingdeveloping-for-windows-phone-7-and-xbox-360/introduction-to-avatars-xna-game-studio-4-0-programming/
12. http://www.the-flying-animator.com/animation-techniques.html
13. Soriano, Marc. *"Skeletal Animation". Bourns College of Engineering.*
14. Shapiro, A., Feng, A., Wang, R., Li, H., Bolas, M., Medioni, G. and Suma, E. (2014), *Rapid avatar capture and simulation using commodity depth sensors*. Comp. Anim. Virtual Worlds, 25: 201–211. doi: 10.1002/cav.1579
15. Natapon Pantuwong *and* Masanori Sugimoto, *A fully automatic rigging algorithm for 3D character animation*, SA '11 SIGGRAPH Asia 2011 Posters, Article No. 30
16. Peter Borosan et al., RigMesh: Automatic Rigging for Part-Based Shape Modeling and Deformation, SIGGRAPH Asia 2012.
17. Ilya Baran and Jovan Popovic, *Automatic rigging and animation of 3D characters*, ACM SIGGRAPH 2007.
18. Chris Hecker et al, *Real-time motion retargeting to highly varied user-created morphologies*, ACM SIGGRAPH 2008.