Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques
Universitat de Barcelona

MOTION CAPTURE WITH KINECT

Marc Girones Dezsènyi

Director: Oriol Pujol

Realitzat a: Departament de Matemàtica Aplicada i Anàlisi. UB

Barcelona, 20 de junio de 2013

Agradecimientos

Agradecer a Oriol Pujol por darme la oportunidad de poder trabajar en este proyecto y por ayudarme en todas las dudas que se han ido planteando a lo largo del desarrollo del proyecto.

Index

Figures and tables

Abstract

Motion capture, motion tracking or mocap are the terms used to describe the process of recording movement and translating that movement onto a digital model, it is used in military, entertainment, sports, and medical application. In film making it refers to recording actions of human actors, and using that information to animate digital character models in 3D animation.

Unfortunately, most motion capture systems on today's market are prohibitively expensive for educational institutions and small businesses. My goal is to develop a relatively low-cost competitive motion capture system. So in this research I'm looking for the answer how can create an optical mocap player with a single camera.

Current motion capture methods use passive markers that are attached to different body parts of the subject and are therefore intrusive in nature. In applications such as pathological human movement analysis, these markers may introduce an unknown artifact in the motion, and are, in general, cumbersome. So the other key challenge is to produce a system that allows marker-less real-time tracking or near the real time.

My ultimate objective is to build a visual system that can integrate the above mentioned components, wherewith can display the user's movement into an avatar in the virtual space.

Finally, I will propose a validation system to validate the user movements. I will evaluate the different postures or complete movements. Moreover, I want to go future and show the result in different ways, such as different viewing angles, different meshes or mesh with illumination.

1. Introduction

Recognition of human motion stream from 3D motion capture systems can find wide application in many situations, such as surveillance video systems, 3D animation and simulation-based training, gait analysis and rehabilitation and gesture recognition. Marker-less motion capture is a challenging problem, in particular when only monocular video is available.

The aim of this project is to study the viability of creating a human capture system with a single depth and colour camera. I will research some methods wherewith I will be able to create the human motion capture system with the single depth camera. This involves the analysis, design and implementation of an application to verify whether my approach is correct or not.

Finally, I will propose a validation system to validate the user movements. I will evaluate the different postures or complete movements. Moreover, I want to go future and show the result in different ways, such as different meshes or mesh with illumination.

1.1. Project Environments

The aim of this project is find the answer if it is possible to build a human capturing system with a single depth and color camera. I will present many methods, algorithms wherewith I will able to create the human motion capture system. Finally, I will propose a validation methodology to validate the user movements. I will evaluate the different postures or complete movements. Moreover, I show the result in different ways, such as different meshes or mesh with illumination.

Motion tracking or motion capture started as a photogrammetric analysis tool in biomechanics research in the 1970s and 1980s. The idea was to record human body movement (or other movement) for immediate or posterior mapping of the human motion onto the motion of a computer character. The mapping can be direct, such as a human arm motion controlling a character's arm motion, or indirect, such as human hand and finger patters controlling a character's skin color or emotional state.

Disney studio was a first who was traced animation over film footage of live actors playing out the scene. So, any virtual character got convincing motion (e.g. Snow White). This method called "*rotoscoping*" has been used successfully since then.

In 1970, when he began to be possible to animate characters by computer, Animations have adapted traditional techniques, including the "rotoscoping". Today, technology is catching is good and diverse, we can classify them into three broad categories: mechanical, optical and magnetic motion capture.

Today, motion capture is really effective, and it is used in a wide variety of applications, such as surveillance video systems, 3D animation and simulation-based training, gait analysis and rehabilitation and gesture recognition. The application is aimed to non-expert users e.g. this technology can be adopted in health applications. We can imagine that physicians learn surgical techniques from each other form a huge distance. They are able to check key moves from different perspectives. But this technology can also be used in sports. For instance, coaches and athletes could be able to analyse their opponents' techniques, see every posture from difference perspective. So we may take a conclusion that today this technology is one of the major challenge that can radically change the human lives.

Finally, I am referring to the related courses from the Computer Science curriculum. This project is strictly related to the subject of Image Processing and Graphics and Data Visualization, which try and give the first steps in graphics applications, basic knowledge about how to position an objects in a scene and able to view them in a viewport, as well as pipelines and basic frameworks that can help us achieve the goals of the course.

The other signatures which has a relation with the Image Processing and Graphics and Data Visualization, like:

- Programming I: Where introduced to the basis of programing.
- Programming II. Where introduced advanced knowledge of programing.
- Introduction to Scientific Computing: Where introduce the basics of programming computer science.
- Algorithms: Where introduced where the algorithms and the importance of computational costs.
- Design Software: Where introduced to the methodologies followed for the proper design and further development of the software.
- Data Structure: Where introduced when and how to use different data structures.
- Operating Systems I and II: Where introduce the necessary skills to manage the various resources available to a computer and how to manage concurrency.

1.2. Project Motivation

3D human body models are used in a wide spectrum of application that require images of human replicas. It is used in military, entertainment, sports and medical application, and for validation of computer vision and robotics. In filmmaking and video game development, it refers to recording actions of human actors, and using that information to animate digital character models in 2D or 3D computer animation.

In the Graphics and Data Visualization course we work on how we can render a virtual scene which is composed of one or several object viewed through the perspective or parallel projection. In the Image Processing course different techniques wherewith we can segment the one image were shown and, finally, in the Computer Vision course tracking techniques were introduced. I got a wide knowledge during the last four years and raised the question of how I can combine these things to get a revolutionary new thing. The answer for this question was found in the motion capturing field.

So I found the topic, but it remained to choose the **most appropriate devices**. The motion capture system can be divided into magnetic, mechanic and optical.

Magnetic systems use electromagnetic sensors placed on joints of moving limbs, and each sensor record 3D position and orientation (right image of figure 1). The sensors connected with the computer which can process 3D data in real time, which means the system restricts movement due to cabling. This system's advantage is that the processing time has a low cost.

Mechanical systems use special suits with integrated mechanical sensors that register the motion of articulation in real-time (center and left image of figure 1). Each sensor placed on joints of a moving limb, but the system only capture the data, without processing them.

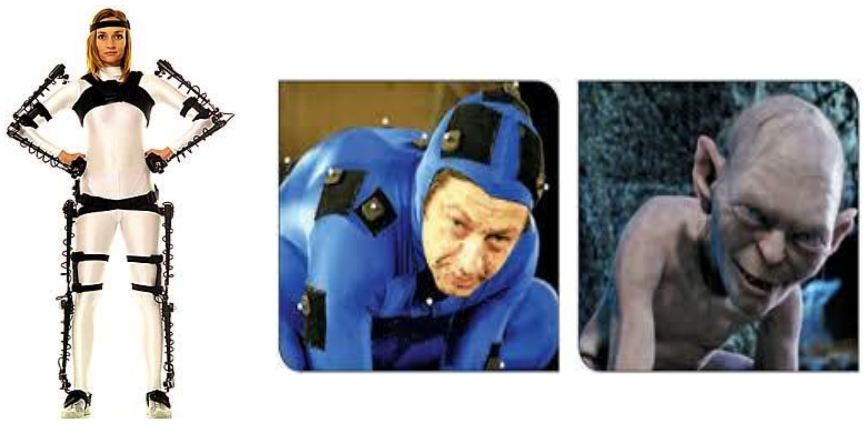


Figure 1: Magnetic and mechanical motion capture system

Finally, the optical systems are based on photogrammetric methods. Optical systems provide high accuracy with complete freedom of movement and support the possibility of interaction between different actors. But the system has an elevate processing time and it has not a mature technology. The quality is not as good as other capturing technologies. Most motion capture systems based on optical techniques need a relatively large number (4-8) of camera views to give effective results. And it requires multiple synchronization because video cameras observing the subject from several different directions.

My computer science interest focus on the visual appearance and in the future I would like to broaden my knowledge in this sector. So there was no question that I will choose the optical system. But in my opinion the system with multiple cameras is too expensive and too hard to implement for a small developer team. So the aim is find the answer if it is possible to create the human captures system with a single depth and colour camera.

The other challenge of the project is a virtual character rig. Because the human motion is too complex and able a many combination of the movement, it was not a possible solution to create the pre-recorded animation of the virtual character. So the other aim is find the solution to create the automatic character rigging from the one static object.

1.1. General Aims and Objects

The core objectives which have been designated as fundamental to the project are:

- *Previous study of the motion capture system*
  In the previous study do a comparative analysis and study of the existing algorithms. Research what are the main components of this techniques and research methods of each component.

- *Design a visualization system that allows viewing objects in 3D*
  Design and implement a system, which allows three dimension views. In addition the interface to be able to shows the scene from different perspective view.

- *Analysis, design and implementation of full human body control.*
  Research any API which provides the hardware and software 3D sensing technologies. Then design and implement the full human body control.

- *Analysis, design and implementation of 3D virtual model visualization and 3D virtual model animation*
  Research any library or graphics engine which provides the 3D virtual model visualization and 3D virtual model animation.

- *Design how to fusion the two previous result*
  Here try to fusion the 2 previous implementations. In other words, recording of human body movement for immediate mapping of the human motion onto the motion of a computer character.

- *Validation of the application.*
  Try to validate the system from the user movements. Evaluate the different postures or complete movements.

- *Conclusions and future lines to continue*
  Get the conclusion of the project, the achievement of objectives and enumeration of future continuation lines. Define what are improvement elements and research better algorithms to subtract them.

1.2. Organization of the document

This document is organized into the following chapters:

Chapter 1: Introduction.
The first chapter is the Introduction. In this chapter contextualize the final year project and defines the problem to subsequently remove the aims of this work. Moreover, there is other paragraph that refers the personal motivation.

Chapter 2: Background
This chapter will introduce the background of this project, analyse the different elemental of the system and approaches to carry out the solution.

Chapter 3: Methodology
This chapter cover the analysis of the application. It will start with the general approach, then step by step go the lower level and explain briefly each level's component.

Chapter 4: System Analysis and Design
This chapter cover the system analysis and design of the application through use cases, class diagrams, sequence diagrams and domain models

Chapter 5: Validation and simulations
Chapter 5 will discuss about the results and validate the user movements. It will discuss the result of the different postures and result of the complete movements.

Chapter 6: Conclusions and future lines to continue
This chapter details the conclusion of the project, the achievement of objectives and enumeration of future continuation lines of this project. There is one paragraph, which explains the future work related to improvement elements of the software.

Chapter 7: References
Party shown external material used

Chapter 8: Annexes
The final chapter list the recommended minimum hardware requirement for the application, explain how to install the application on the PC and explain usage and develop usage.

2. Background knowledge

The purpose of this chapter is to analyse the different elemental component of the system. It introduces and explains briefly those basic theories, whose knowledge is indispensable. Moreover we can place it within a more general context.

This chapter is divided into two sections. Section one will discuss the visual appearance. It talks over how to create the virtual scene and the methods used to display this scene. In section two will define the methods to calculate the joint angles. How we can convert the data from the coordinate features to the BVH feature.

2.1. Projection

As I told at the aims the project, we would like to map directly a human motion into the virtual character. To this mapping, it is an essential components to create the virtual space and visualize one or more 3D component in this space. As in the real word, the scene has a virtual camera, the virtual scene and projection plane.

Let us begin by considering how to transform the scene into camera coordinates. Ergo we want to view the scene from the camera's point of view. The fundamental approach of planar projection there are three steps. Firstly we have to define a plane in 3D space; this is the projection plane or film plane. Then project scene onto this plane and finally map to the window a viewport. Although it is important to know how to create the 3D scene and map to the window a viewport, but in this document I will not discuss. The only topic what I will dissect is the projection onto the plane.

There are many methods to representing a three-dimensional object in two dimensions and ones of them is **orthographic** and **perspective** projection (Figure2).

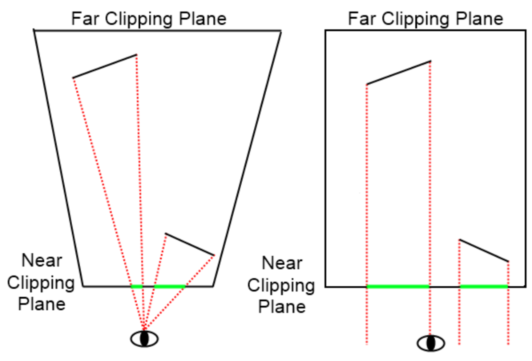


Figure 2: Orthographic projection vs. perspective projection

The **orthographic projection** is a form a parallel projection, where all the projection lines are orthogonal the projection plane. Mathematically, the orthographic projection is what we would get if the camera in our synthetic-camera model had an infinitely long telephoto lens and we could place the camera infinitely far from our object.

Rather than worrying about cameras an infinite distance away, suppose that we start with projections that are parallel to the positive *"z"* axis and projection plane at z = 0.

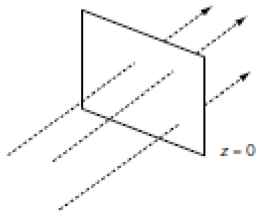


Figure 3: Orthographic projections with projection plane z = 0

We can observe, that not only are the projections perpendicular or orthogonal to the projection plane, but we are able to slide the projection plane along the *"z"* axis without changing where the projections intersect this plane. The orthographic viewing we can imagine of there being an orthographic camera that resides in the projection plane, something that is not possible for other views (Figure 3). In the other words, this projection takes the point *"x", "y"* and *"z"* and project into the point (*x, y, 0).*

In OpenGL, an orthographic projection can be created with the following instruction:

```
void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top,
GLdouble near, GLdouble far)
```

We can observe that all parameters of the function are distances measured from the camera. The orthographic projection "sees" only those objects in the volume specified by the viewing volume. Unlike a real camera, the orhographic projection can include objects behind the camera. Thus, as long as plane z=0 is located between *"near*" and *"far"*, the two dimensional plane will intersect the viewing volume.

The other method what I would like to discuss is the **perspective projection**. This is a form of pictorial drawing that gives the illusion of depth onto a flat surface, very similar to that of viewing of the object through the human eye.

All perspective views are characterized by reduction of size. When objects are moved farther from the viewer, their images became smaller. This size change gives perspective views their natural appearance. However, because the amount by which line is from the viewer, we cannot make measurements from a perspective view.

Normally, the viewer is located symmetrically with respect to the projection plane.

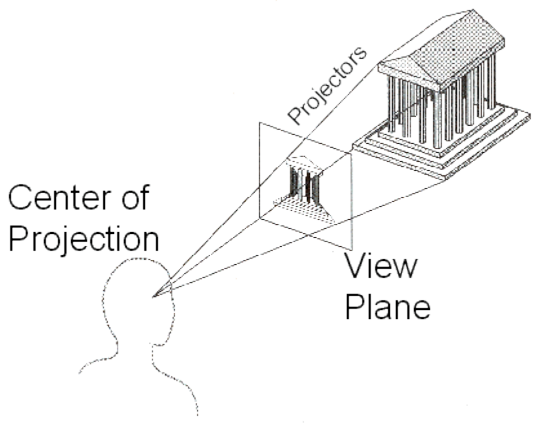


Figure 4: Perspective viewing

Thus, the pyramid determined by the window in the projection plane and the center of the projection is a symmetric or right pyramid (Figure 4). This symmetry is caused by the fixed relationship between the retina and lens of the eye for the human viewing, or between the retina and lens of a camera for standard cameras, and by similar fixed relationships on most physical situations.

There is a few different ways to setup the view frustum, and thus the perspective projection. In my system, I used the symmetrical frustum, so it look at is as follow:

```
 void glFustrum (GLdouble left, GLdouble right, GLdouble bottom, GLdouble
top, GLdouble near, GLdouble far)
```

Although, the orthographic and perspective projection look similar, but they not the same (Figure 5). The most striking different are the parallel lines. While at the orthographic projection the parallel lines never touch, the at the perspective projection the parallel lines touch at infinity.

Then the other difference is the size of the object. While at the first projection everything seems equal, at the perspective projection the closes things seems bigger.

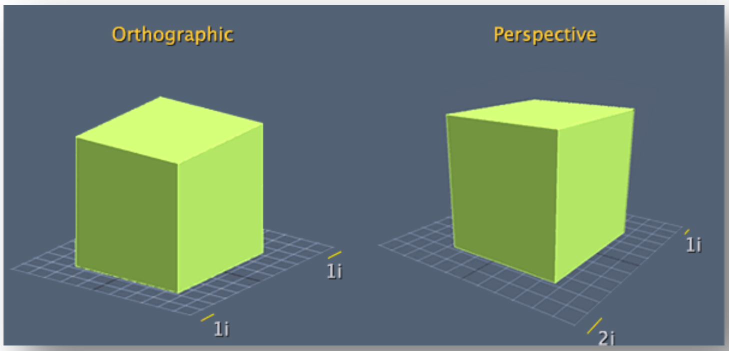


Figure 5: Orthographic projection vs. perspective projection

2.2. Method to calculate the joint angles (Angular kinematics)

Kinematics is the branch of classical mechanics that describes the motion of points, bodies (objects) and systems of bodies (groups of objects) without considering of the causes of motion. Angular motion occurs when all parts of an object move through the same angle but do not undergo the same linear displacement. The object rotates around an axis of rotation that is a line perpendicular to the plane in which the rotation occurs. One example of angular motion are the motion of a bicycle as you pedal across campus , and the motion of your thigh around your hip as you walk to class.

An understanding of angular motion is critical to comprehend how we move. Nearly all human movement involves the rotation of body segments. The segments rotate about the joint centers that form the axis of rotation for these segments. When an individual moves, the segments generally undergo both rotation and translation.

So how can we measure the angles? An angle is composed defined between two lines that intersect at a point called the vertex. In a biomechanical analysis, the intersecting lines are generally body segments and the vertex is their common joint. If you consider the longitudinal axis of the shank segment as one side of an angle and the longitudinal axis of the thigh segment as the other side, the vertex would be the joint center of the knee.

Angles can be determined from the coordinate points you generated in the previous lab. Coordinates of the joint centers determine the sides and the vertex of the angle. For example, an angle at the knee can be constructed using the thigh and shank segments. The coordinates of the ankle and knee joint centers define the shank segment, while the coordinate of the hip and knee joint centers define the thigh segment. The vertex of the angle is the knee joint center.

There are 2 main methods to calculate these angles: **relative angles** and **absolute angles** (Figure 6). The absolute angles or segment angles, where the angle between a segment and the right horizontal of the distal end. In this case it should be consistently measured in same direction from a single reference –either horizontal or vertical. The relative angles or joint angles, where the angle between the longitudinal axis of two adjacent segments. Here should be measured consistently on same joint side.

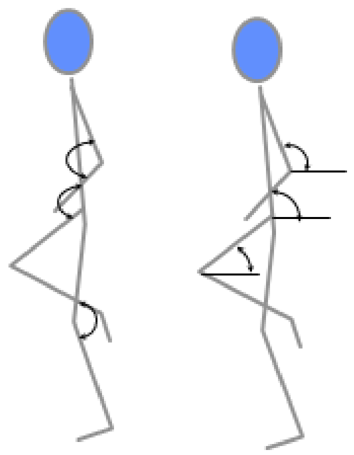


Figure 6: Relative angles vs. absolute angles

The typical data that we have to work at motion capture system are the “x”, “y” and “z” initial and final locations points of each limb. In the next section I will analyse how can get these angles.

Let’s define “$\vec{v}_{\text{moved}}$” and “$\vec{v}_{\text{anatomical}}$” vector, where “$\vec{v}_{\text{moved}}$” is actually situated limb vector and “$\vec{v}_{\text{anatomical}}$” is directional anatomical vector, as the limb naturally situated. With these 2 vectors we can calculate a rotation value to transform a 3d vector “$\vec{v}_{\text{anatomical}}$” to a 3d vector $\vec{v}_{\text{moved}}$”. (Figure 7)

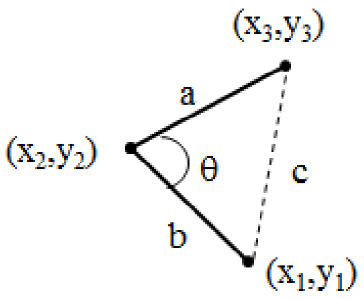


Figure 7: Calculate the absolute angles

Mathematically, it can be calculated using the following formula:

$$cross_{ab} = |\ a_n\ x\ b_n\ |$$

$$dot_{ab} = a_n \cdot\ b_n$$

$ac = \operatorname{acos}(dot_{ab})$ where a_n and b_n are vectors normalized.

Algorismically:

```
        epsilon = 1e-12;

        //Normalize a vector
        an = sl3dnormalize(vec_anatomical, epsilon);
        bn = sl3dnormalize(vec_moved, epsilon);

        axb = sl3dnormalize(cross_product(an, bn), epsilon);
        ac = acos(dot_product(an, bn));
        result = [axb, ac]
```

Figure 8: Calculate rotation between two vectors

This compute (Figure 8) give us the “result” axis-angle row vector, where the first three elements specify the rotation axis, the last element defines the angle of rotation.

2.3. Technologies

In this section I will discuss that, what kind of technology tools available today and which tools I chose for my application. This paragraph is divided into two parts. In section first will discuss what kind of hardware exists to achieve the motion detection. In section two discuss the what kind of graphics engines exist and what are able to visualize the human motion

Kinect device

Motion detection is the process of detecting a change in position of an object relative to its surroundings or the change in the surroundings relative to an object. Motion detection can be achieved by both mechanical and electronic methods. (E.g. Infrared camera, optics like video and camera systems, Radio Frequency Energy likes radar, microwave and tomographic motion detection … etc.)

Like as I said in the project motivation, this system will base on the optical motion capture. So till now I will examine only tools which are based on optical system.

In motion capture sessions, movements of one or more actors are sampled many times per second, early techniques used images from multiple cameras and calculate 3D positions, motion capture often records only the movements of the actor, not his or her visual appearance. This animation data is often mapped to a 3D model so that the model performs the same actions as the actor. This process may be contrasted to the older technique of rotoscope, such as the Ralph Bakshi 1978 The Lord of the Rings and 1981 American Pop animated films where the motion of an actor was filmed, then the film used as a guide for the frame-by-frame motion of a hand-drawn animated character.

Camera movements can also be motion captured so that a virtual camera in the scene will pan, tilt, or dolly around the stage driven by a camera operator while the actor is performing, and the motion capture system can capture the camera and props as well as the actor's performance. This allows the computer-generated characters, images and sets to have the same perspective as the video images from the camera. A computer processes the data and displays the movements of the actor, providing the desired camera positions in terms of objects in the set.

Optical systems utilize data captured from image sensors to triangulate the 3D position of a subject between one or more cameras calibrated to provide overlapping projections. Data acquisition is traditionally implemented using special markers attached to an actor; however, more recent systems are able to generate accurate data by tracking surface features identified dynamically for each particular subject.

Today, only three popular devices can found on the market: the Microsoft Kinect, ASUS Xtion and PrimeSense Carmine. All these are based on the same PrimeSense infra-red technology.

So the basic characteristics of the full-body motion capture are generally the same. Although each of three stable, but the most common used device is the Kinect. It widely used by the users and the developers also and two stable API was designed for this device (Kinect SDK by Windows and OPENNI API). Because many people use the Kinect device, so many forums, documentation and tutorials was help me, there was no question; that the most appropriate tool is the Kinect device.

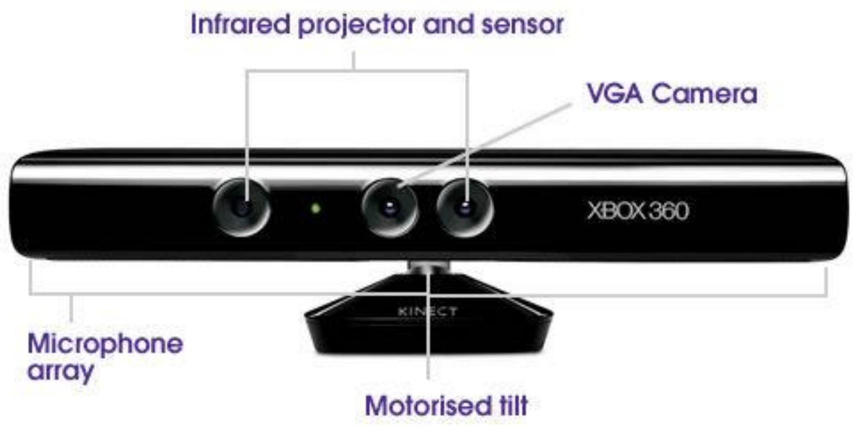


Figure 9: Microsoft Kinect for Windows

Firstly, Kinect has three cameras inside, which allow it to track movement in 3D. Rather than just capturing a 2D image, and trying to work out who and where you are in that image, Kinect is a lot more advanced. The first two cameras work together, to allow Kinect to track a 3D representation of you, and your living room. An infra-red projector, which bathes your living room in points of infrared light, coupled with a camera that detects infrared light allows Kinect to “see” in 3D, which, when coupled with advanced software, allows it to differentiate between items of furniture, and human beings. The third camera is basically a webcam, which allows it to capture a video of you.

Graphics engine

A game engine is a system designed for the creation and development of video games. The leading game engines provide a software framework that developers use to create games for video game consoles and personal computers. The core functionality typically provided by a game engine includes a rendering engine for 2D or 3D graphics, a physics engine or collision detection, animation, artificial intelligence and a scene graph.

So this engine is a great help to render the virtual space and virtual character. The only question is which is the best motor for this project. In the previous study I was used the Motion Capture Interpolation project by Jernej Barbic and Yili Zhao [6]. It was very useful to understand how works the motion capture, but it was very rudimentary graphics engine. Mostly it could be draw a rudimentary virtual skeleton in a simple virtual environment. As you see on the figure 10. Soon I was realizing that I can't reach my goal with this project. In the second iteration I was looking for a graphics engine which provides to load any virtual character and make a rig without the predefined animation.

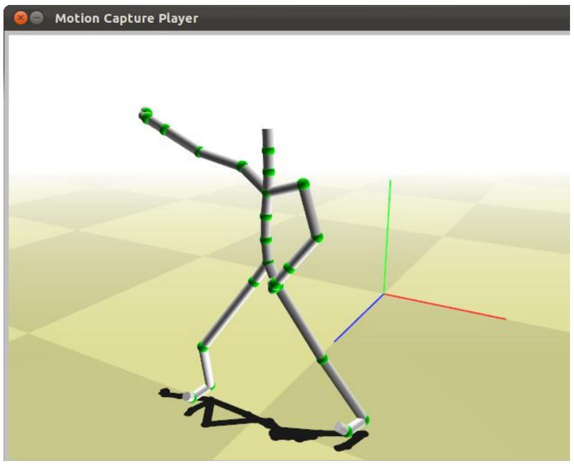


Figure 10: Motion capture by Jernej Barbic

There are many graphics engine can found on the market, like Unreal, Unity, CryEngine or Havok. The first and the second motors I had to reject because the Unreal support the unrealscipt and the Unity support Visual C# and JavaScript. Although it could made a hybrid system, where a full body tracking written in c++, the visual appearance written in other programming language and the communication between the 2 layout via socket. But finally I seemed too complicate solution.

One of the most ideal solutions seemed a Havok engine, but finally I had to reject. Although it support the rig technology and c++ programing, but it does not let me "enter" the visual rendering. So I was unable to visualize Kinect color and depth frame.

Finally the Pinocchio open source SIGGRAPH project was the most appropriate choice. It was written in C++ and supports the automatic rigging and animation of 3D characters. In the 3.2.3. Apply the human motion into the virtual character I will explain in detail how is it works.

3. Methodology

3.1. General analysis of the approach of motion capture

In this section I will make the general analysis of my implementation. The analysis I will start from the high level and step by step go to the lower level. I'll explain briefly each level's components, its functionality and communication with the other components.

Let's start with a quick overview over the whole application. Here we can identify and distinguish three main layouts (See the figure 11): Layout which captures the space, one which applies the angular kinematics methods and finally one which apply the human motion into the virtual character.

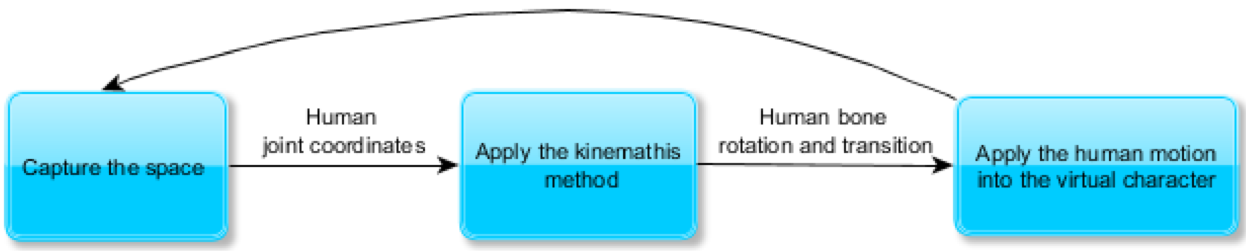

Capture the space
Human joint coordinates
Apply the kinemathis method
Human bone rotation and transition
Apply the human motion into the virtual character

Figure 11: High level layout diagram

In more detail, the first layout communicates with the Kinect device, capture the space with the color and the depth camera. Then it forwards these frames an API, which able to get the 3D space coordinates of each human joint from the frames.

In the next step, the previous layer transfer mentioned coordinates to the second layout. It's functionality to converts the data from the coordinate features to the BVH feature. Say with the other words, the layout get the 3D coordinates of the human joint points and with these values it calculate the individual bone's transition and rotation.

Finally, the last layout map the user motion into the virtual character. The input of the layout is a BVH feature, where each individual is a bone transition and rotation. Now the layer can warping the static mesh to arrive to the same posture what the human is.

When the last layer finishes the rendering, then the sequence starts again. This continuous loop ensures a sequential motion capture appearance.

3.2. Specify analysis of the approach of motion capture

3.2.1. Capture the space

Now I will decompose the overall analysis into minor and detailed components. The first layout, as mentioned above, its function to gets 3D human joins coordinates from the depth frame.

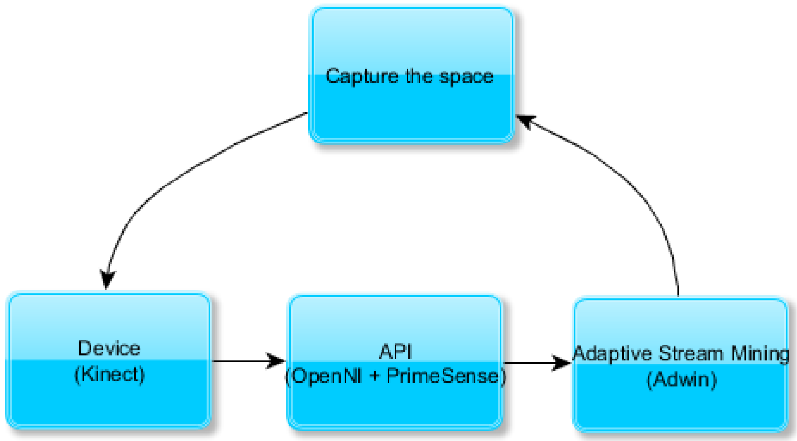


Figure 12: Detailed layout diagram of the first component

Firstly the driver communicates with the Kinect device through the computer bus and gets a synchronized depth and colour frame. (See the figure 12) Then the driver forwards these frames to the OpenNI and Primesense API, which segment the human body from the background. Now with this segmented frame, the API able to determine how is situated the user. This technique calls the full-body tracking. So now we are able to know the 3D space coordinates of each human joint.

With this technique we can distinguish 15 different body parts, like a head, neck, shoulders, elbows, hands, torso, hips, knees and foots. (See the figure 13.)

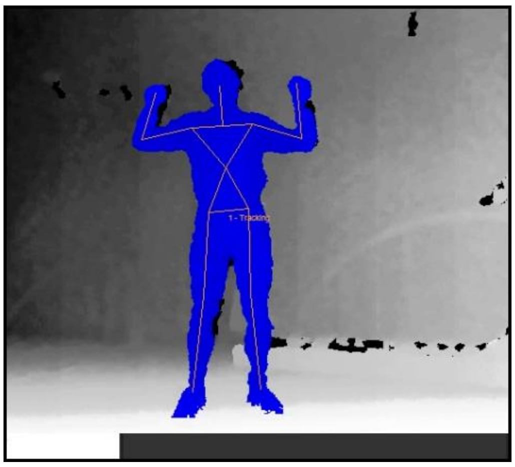


Figure 13: Segmented user from background and his skeleton

However we get the coordinates, but the device precision is not accurate. Map the space with the infrared camera is a good technique, but it is not 100%. Each coordinate has an unknown error component, which is difficult to determine exactly. The measurement results are approximately correct, but sometime are irrelevant. To remove those irrelevant data and minimize the global registration error, I was implementing one adaptive stream mining algorithm.

ADWIN

Thus we have a data stream, where almost every data contains an unknown error value. This stream can contain the weak change (in our case e.g. a body movement) or may can contain a sudden change (in our case e.g. API give an irrelevant coordinate). So my goal is find an algorithm, which blurs the data and generate the relative continuous data stream. It should be taken into account that the most recent data are more valuable than older value.

The adaptive stream mining algorithms used to solve similar problems. In my approach I used the adaptive sliding window (ADWIN)[2]. Let details how it is works:

Our algorithm keeps a sliding window *“W”* with the most recently read “x_i“. Let *“n”* denote the length of *“W”*, “$\hat{\mu}_W$” the (observed) average of the elements in “W”, and “μ_W” the (unknown) average of μ_t for $t \in W$. So we got two different windows. The main idea is, when two windows are sufficiently same than show the mean of the window. When the two windows are sufficiently different, we can be assumed that the trend has been changed and therefore we can remove the oldest part of the window. (Figure 14)

```
ADWIN0: ADAPTIVE WINDOWING ALGORITHM
  Initialize Window W
      for each t > 0
            do W <- W u {$x_t$} (i.e., add $x_t$ to the head of W)
                  repeat Drop elements from the tail of W
                  until $\mu W_0$ - $\mu W1$ < $\epsilon_{cut}$ holds
                        for every split of W into W = W0 · W1
                  output μW
```

Figure 14: Adwin adaptive window algorithm pseudo code

Now we need to calculate the “E” threshold. Our goal the absolute difference between obtained averages will be greater than the threshold “E”. This value say us, that these averages are statistically different and therefore we are facing a significant change that has not been caused by chance. In this context we can use concepts of statistics to find the threshold. The threshold is based on a statistical test which you must provide a parameter of trust between 0 and 1. From this and the sizes of the windows W0 (n0) and W1 (n1) E can be found as follows,

$$m = \frac{1}{\frac{1}{n_0} + \frac{1}{n_1}} \qquad \delta = \frac{\mathrm{d}}{n_0 + n_1} \qquad \mathrm{E} = \sqrt{\frac{1}{2m} ln \frac{4}{\delta}}$$

Finally, in the following two diagrams (figure 15 and 16) can see the result of my approach. I marked the input coordinates with red and the adwin's output coordinates with blue. We observe that the resulting stream is smoother and less noisy. This stream in the final result cause the user's continuous movement and his movement will not be too sharp.

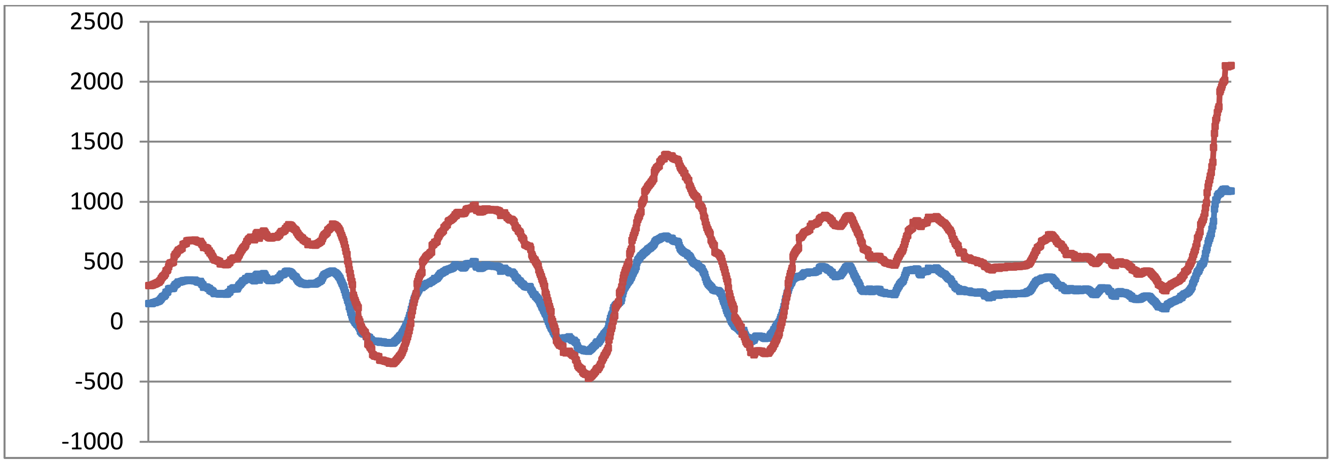


Figure 15: Displacement along the "X" axis

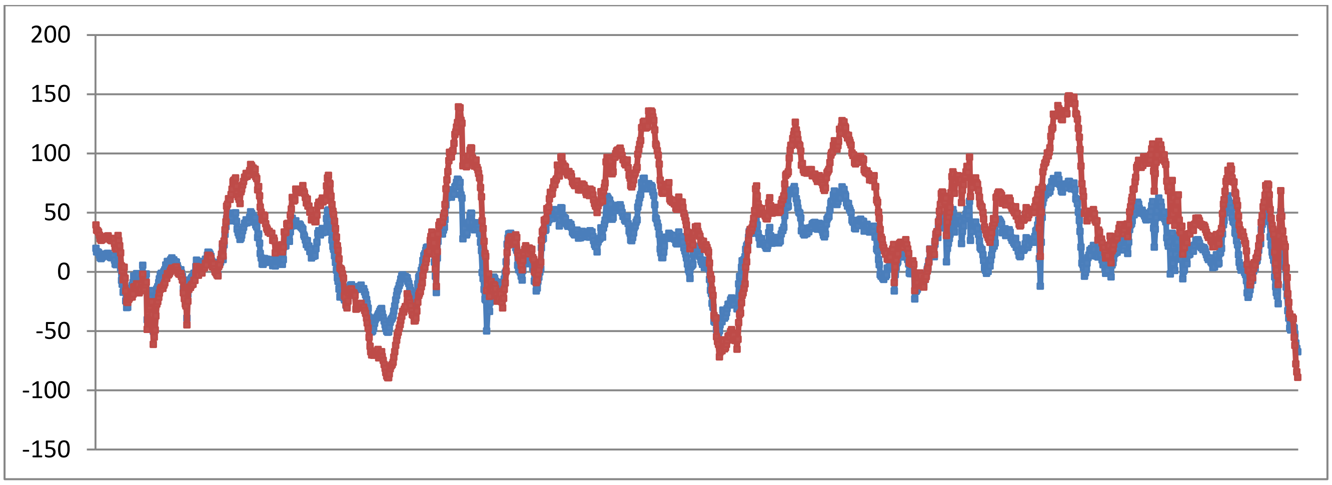


Figure 16: Displacement along the "Y" axis

3.2.2. Apply the kinematics method

The second layout's functionality is converts the data from the coordinate features to the BVH feature. In the other words, the layout get the 3D coordinates of the human joint points and with these values it calculates the individual bone's transition and rotation.

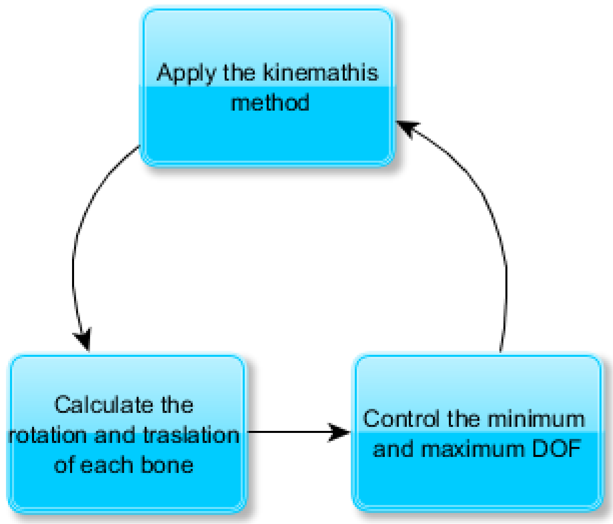

Apply the kinemathis method
Calculate the rotation and traslation of each bone
Control the minimum and maximum DOF

Figure 17: Detailed layout diagram of the second component

Once the system calculated the all limb's rotation and the translation values, I forward these data to examine its accuracy. It is a function which check and limited the minimum and maximum degrees of freedom each bone. (Figure 17)

Calculate the rotation and translation

Let's start with how to calculate the rotation angle and the translation. Here the incoming data are the initial and final points of each limb. Let's define "$\vec{v}_{\text{moved}}$" and "$\vec{v}_{\text{anatomical}}$" vector, where "$\vec{v}_{\text{moved}}$" is actually situated limb vector and "$\vec{v}_{\text{anatomical}}$" is directional anatomical vector, as the limb naturally situated. E.g. the femur and the tibia have $\vec{v}(0, -1,\ 0)$ anatomical vector. With these 2 vectors we can calculate a rotation value to transform a 3d vector "$\vec{v}_{\text{anatomical}}$" to a 3d vector $\vec{v}_{\text{moved}}$".

This compute give us the "R" axis-angle row vector, where the first three elements specify the rotation axis, the last element defines the angle of rotation. The calculation of the angle in this section I will not detail here, because I have already explained in the previous section. (Go to the section 2.2. Method to calculate the joint angles).

In the next step I have to calculate the translation of each bone and the translation of the avatar in the virtual space. The translation values of each bone are static value, which are given by the “Pinocchio” [7]. In the second translation I’m looking for how to “shift” the virtual skeleton so as it completely match with the real user spatial position.

At this point raise the question that what kind of projection will chose for the virtual space. As I was analyse in the section 2.1. Projection, in my approach I have to choose between the orthographic and perspective projection. I was choose the perspective projection because we don’t just want to create a system where the avatar has a static position, but rather a system that return the user’s full movement in space. In this case the reality augmented need to reflect the user distance, so the *“z”* cannot be infinite.

Once I have selected the projection type, and then have to find the transformation that can transform the real space’s coordinates to the virtual space’s coordinates. One of the most trivial methods is *“rule of three”:*

$$\frac{X_{real} - X_{Rmin}}{X_{Rmax} - X_{Rmin}} = \frac{X_{virtual} - X_{Vmin}}{X_{Vmax} - X_{Vmin}}$$

Where the X_{real} is the real word “X” coordinate, $X_{Rmin}\ and\ X_{Rmax}$ the minimum and maximum possible displacement. The computation is very basic and rudimentary, what caused in my approach, that the virtual skeleton is not completely match with the real user spatial position. So the method accuracy is rather vague. Because of the protracted period of development, I had not time to look for a second, more accurate method.

Finally, the received bone translation and rotation values have to organize to BVH hierarch format.

BVH feature reduction

However we have calculated the rotation and the translation, but I haven't any system to converts the data from the coordinate features to the motion feature. It is important, because the visual system will be able to interpret and play the motion and gesture.

On the other hand I haven't any system up to this point, which examined the accuracy of the data. I noticed that is not enough remove the error component and blurred the coordinates with the adaptive sliding window. The device and API work well, but it is not 100%. Its precision is not accurate. There was no question; that I need to create the system which examines the previous data and controls the bone's minimum and maximum values.

At a previous study, I was analysed and discover how to work the most the motion capture systems [6, 8 and 9]. These systems in order to play human motion; used the motion capture file. Motion capture file formats can be roughly divided into two kinds, the **Tracker Format** and the **Skeleton Format**. The former only has three-dimensional location values and accepts the coordinate 3D formats. The latter has skeleton information as well as three-dimensional location values and accepts the BVH and ASF/AMC formats. These formats provide skeleton hierarchy information as well as motion data. In the other words, in the hierarchy section I will define the skeleton structure of the Avatar (define the total 15 joints). It is composed in such a way that the hip assumes the role of the root and each segment is jointed toward the left lower part, right lower part and upper part, in that order. (Figure 18 and 19)

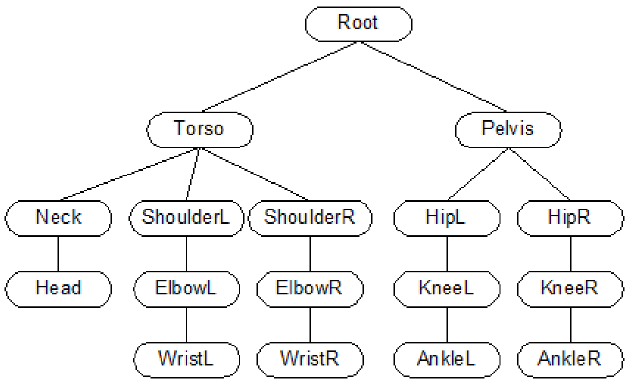


Figure 18: Hierarchical graph of skeleton join

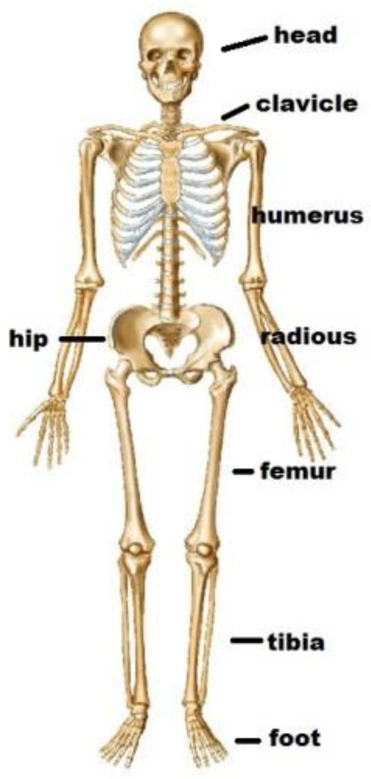
In the motion section is structured with the Euler's angles applied to each joint.

```
OFFSET 0.0000 0.0000 0.0000
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT upperback
    {
      OFFSET -0.2975 11.0962 -0.4289
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT thorax
      {
```

Figure 19: BVH file, motion section

With this technique, we placed the coordinate features to the bvh features. Now the third layout (the visual layout) is able to interpret and play the user's motion and gesture.

The last remaining thing is examined the accuracy of the data. The ASF/AMC not only provides the skeleton hierarchy and the motion data, but it provides for putting limits on each of the channels. Its calls control the minim and maxim of degrees of freedom (**DOF**). For each channel that appears there will be a pair of numbers inside parenthesis indicating the minimum and maximum allowed value for that channel. This information is not used for interpreting the motion data; this is useful only for those applications which might apply motion editing functions that put limits on rotation. In my approach I used the next values:



	(rx, ry, rz) in degrees	(rx, ry, rz) in radian
Clavicle Min and max DOF	(null, -20, 0) (null, -10, 20)	(null, -0.3490, 0) (null, -0.1745, 0.3490)
Humerus Min and max DOF	(-90, -90, -90) (60, 90, 90)	(-1.5707, -1.5707, -1.5707) (1.0471, 1.5707, 1.5707)
Radious Min and max DOF	(-10, null, null) (170, null, null)	(-0.1745, null, null) (2.9670, null, null)
Hip Min and max DOF	(null, -90, null) (null, 90, null)	(null, -1.5707, null) (null, 1.5707, null)
Femur Min and max DOF	(-160, -70, -70) (20, 70, 60)	(-160, -1.2217, -1.2217) (0.3490, 1.2217, 60)
Tibia Min and max DOF	(-10, null, null) (110, null, null)	(-0.1745, null, null) (1.9198, null, null)
Foot Min and max DOF	(-45, null, -20) (90, null, 70)	(-0.7853, null, -0.3490) (1.5707, null, 1.2217)

Table 1: Table of the min and max DOF

Although the y-side minimum and maximum DOF are implemented, but it not used because the current technic (Kinect and OpenNI) only allow to calculate the "x" and "z" rotation.

3.2.3. Apply the human motion into the virtual character

This layout's functionality is applying the human motion into the virtual character. We have to note, that the human motion is too complex and each body position consider a certain limbs displacement and rotation. So it was not possible solution to create the pre-recorded animation of the virtual character. Achieve this; I had to use the automatic character rigging methodology from the one static object.

The difficulty of this part is very high, so for this part I used the Pinocchio [7] open source SIGGRAPH project, which provides the 3D virtual model visualization and 3D virtual model animation. To kit the two projects, it was necessary a detailed knowledge of the "Pinocchio" project. Now I'll briefly detail how it is works the "Pinocchio" project.

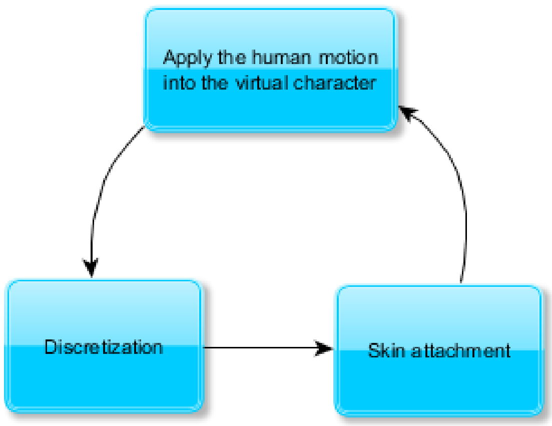


Figure 20: Detailed layout diagram of the third component

This technique used in computer animation in which a character is represented in two parts: a surface representation used to draw the character (called skin on mesh) and a hierarchical set of interconnected bones (called the skeleton or rig) used to animate the mesh. The mesh surface consisting of thousand vertices and nodes, so animating such an object can get very complicated.

The skeleton embedding computes the joints positions of the skeleton inside the character by minimizing a penalty function. To make the optimization problem computationally feasible, they first embed the skeleton into a discretization of the character's interior and then refine this embedding using continuous optimization. The skin attachment is compute by assigning bone weights based on the proximity of the embedded bones smoothed by a diffusion equilibrium equation over the character's surface. (Figure 20 and 2)

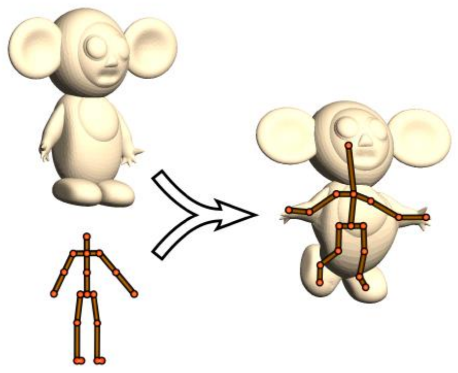


Figure 21: Automatic rigging method

DISCRETIZATION

Next, I will analyse the Pinocchio's discretization approach. In the first step, Pinocchio rescales the character to fit inside an axis-aligned unit cube. As a result, all of the tolerances are relative to the size of the character.

Then, the next step is the **approximate medial surface**, where it makes a very rudimentary proximity to facilitate other posterior computations. (Figure 22) The calculation based on the adaptive distance field to compute a sample of points on the medial surface. The medial surface is the set of C1- discontinuities of the distance field. Within a single cell of our octree, the interpolated distance field is guaranteed to be C1, so it is necessary to look at only the cell boundaries. Pinocchio therefore traverses the octree and for each cell, looks at a grid (of spacing_) of points on each face of the cell. It then computes the gradient vectors for the cells adjacent to each grid point—if the angle between two of them is 120º or greater, it adds the point to the medial surface sample. We impose the 120º condition because we do not want the "noisy" parts of the medial surface—we want the points where skeleton joints are likely to lie. For the same reason, Pinocchio filters out the sampled points that are too close to the character surface (within 2_)

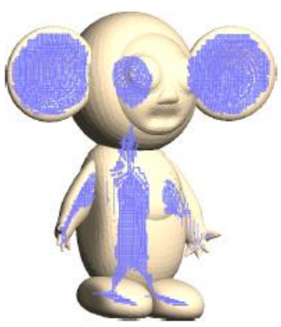


Figure 24: Approximate Medical Surface

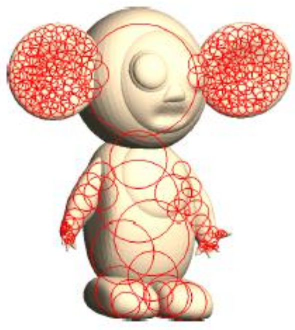


Figure 25: Packed Spheres

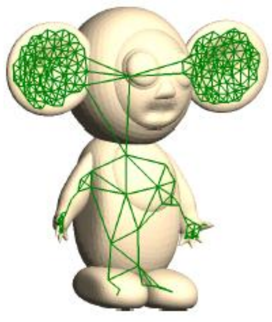


Figure 22: Constructed Graph

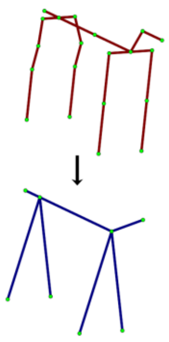


Figure 23: Original and reduced skeleton

Then, the next step is **sphere packing.** Here it sorts the medial surface points by their distance to the surface (those that are farthest from the surface are first). Then it processes these points in order and if a point is outside all previously added spheres, adds the sphere centred at that point whose radius is the distance to the surface. In other words, the largest spheres are added first, and no sphere contains the center of another sphere. (See the figure 23)

The final discretization step is a **graph construction**, where constructs the edges of the graph by connecting some pairs of sphere centers. Say with the other words, Pinocchio computes the shortest paths between all pairs of vertices in this graph and seed up with the penalty function. (See the figure 24).

However we get the skeleton graph, but we have to reduce skeleton. So, all bones chains have been merged, as shown on the above figure. All degree two joints, such as knees eliminated. (See the figure 25)

SKIN ATTACHMENT

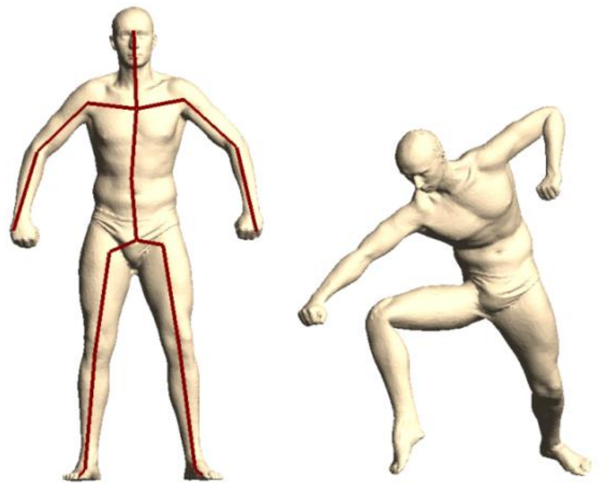


Figure 26: The human discrete embedding and skin attachment

The character and the embedded skeleton are disconnected until skin attachment specifies how apply the deformation of the skeleton to the character mesh. Although it could make use of one of the various mesh editing techniques for the actual mesh deformation, but Pinocchio choose the focus on the standard linear blend skinning method (Figure 26).

The linear blend skinning algorithm works by first placing a hierarchical skeleton inside a static model of a character, typically on some neutral pose. This initial character pose is referred to as "dress pose".

Then, each vertex is assigned a set of **influencing joints** and **blending weight** for each influence. Computing the deformation in some pose involves rigidly transforming each dress pose vertex by all of its influencing joints. Then the blending weights are used to combine these rigidly transformed positions. The deformed vertex position at some skeletal configuration c, $\vec{V_c}$ is computed as

$$\vec{v}_c = \sum_{i=1}^{n} w_i * M_{i,c} * M_{i,d}^{-1} * v_d$$

Where w_i are the weights (usually affine or convex), v_d is the dress-pose location of some vertex v, $M_{i,c}$ is the transformation matrix associated with the *i*th joint in configuration c and $M_{i,d}^{-1}$ is the inverse of the dress-pose matrix associated with ith influence.
There are several properties Pinocchio's desire of the weight. First of all they should not depend on the mesh resolution. Second, for the results to look good, the weights need to vary smoothly along the surface. Finally, to avoid folding artefacts, the width of a transition between two bones meeting at a joint should be roughly proportional to the distance from the joint to the surface.

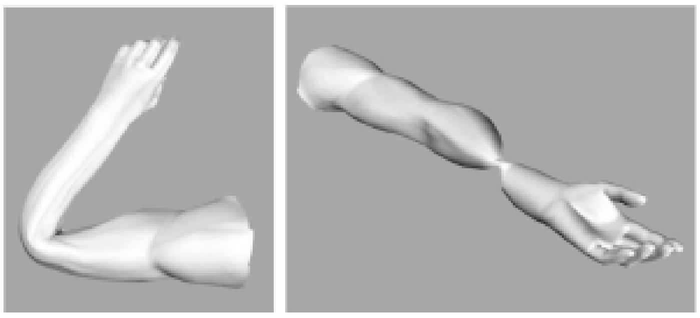


Figure 27: Common problem with linear blend skinning

The common problem of the skinning algorithm is notorious for its failings. It cannot represent complex deformations and suffers from characteristic artefacts such as the candy-wrapper" collapse effect on wrists and collapsing around bending joints as shown in Figure 27.

4. System Analysis and Design

4.1. Use cases

This project implemented a visualization system, under the FLTK environment which allows us to build applications with C + + and OpenGL. This system must be able to validate the approaches of motion capture. Remember that, this project has been introduced some previous project which allowed the visualization of virtual character, so I will not specify those use cases which are related with the previous project, as for example, rig the mesh, discretize the mesh, etc.

Therefore, the use cases listed below:

1. Allow user to view the motion capture
2. Allow user to view the scene with different background
3. Allow user to view the scene from different viewing angles
4. Allow user to set the view (e.g. view the scene with the skeleton mesh, view the scene with the global axis)

4.1.1. Use case 1(UC1)

Allow user to view the motion capture

Actor*:* **User**
Description**: the users want to see the approach of motion capture.**
Pre-conditions**:**
Main stage**:** **1. The user select the virtual characters what want to visualize.** **2. The user starts the application.** **3. The system initializes the Kinect device and the graphics library.** **4. The system discretizes the virtual character.** **5. The system visualizes move the human motion onto the virtual character.**
Post condition**: The simulation is started.**

Table 2: Use case 1

4.1.2. Use case 2(UC2)

Allow user to view the scene with different background

Actor**: User**
Description**: the users want to visualize the virtual scene with different background. (e. g. with the depth image, with the color image or with the black background)**
Pre-conditions**: The user has launched a right away the application**
Main stage**:** stage**:** **1. The user selects the background.** **2. The system changes the camera.** **3. The system gets the frame from the selected camera.** **4. The system visualizes the virtual scene with the new background.**
Post condition**: The system stops active the last selected view.**

Table 3: Use case 2

4.1.3. Use case 3(UC3)

Allow user to view the scene from different viewing angles

Actor**: User**
Description**: the users want to visualize the virtual scene from different viewing angles. They want to zoom in/zoom out, pan and rotate the scene.**
Pre-conditions**: The user has launched a right away the application**
Main stage**:** **1. The user sees the reality augmented.** **2. The user changes the viewing angles with the mouse. (zoom, pan or rotate)** **3. The system changes the attributes of the camera.** **4. The system transforms the virtual scene.** **5. The system visualizes the virtual scene from new viewing angles.**
Post condition**: The system stops active the last selected view.**

Table 4: Use case 3

4.1.4. Use case 4(UC4)

Allow user to set the view

Actor***: User***
Description**: the users want to view the scene with different options. (e. g. with the global axes, with the virtual skeleton …)**
Pre-conditions**: The user initially started and displayed the motion capture**
Main stage**:** **1. The user sees the reality augmented.** **2. The user changes the view options with the “t”, “s”, “a” keys.** **3. The system changes the view options.** **4. The system visualizes the virtual scene with the new view option.**
Post condition**: The system stops active the last selected view.**

Table 5: Use case 4

4.2. Domain model of the application

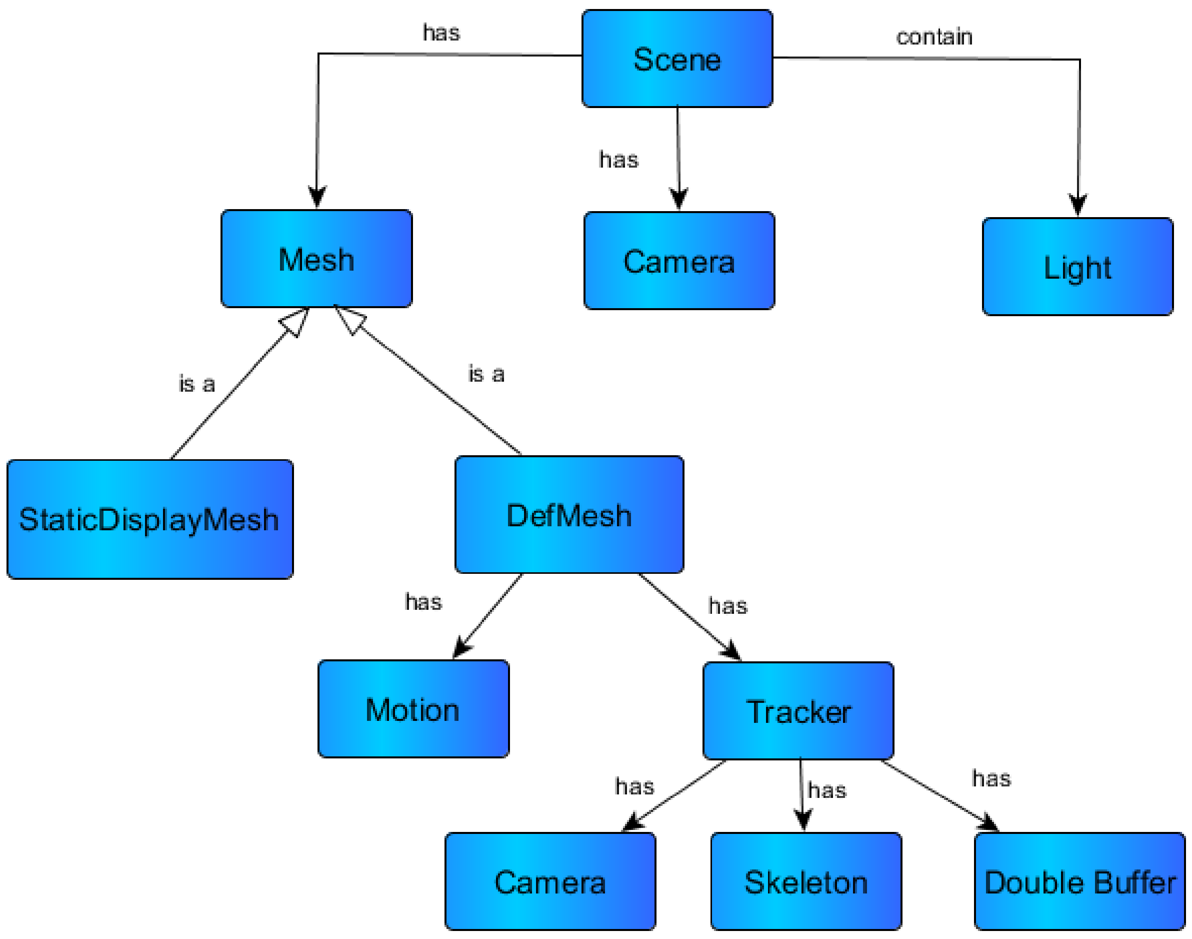


Figure 28: Domain model of the application

Entity details:

Scene: Is the entity that renders the virtual avatar and the camera's captured image with the possible options. Moreover, the entity controls the rotation, panning or zoom of the camera and the illumination.

Mesh: Abstract entity which represent the virtual avatar on the virtual scene.

Camera: Entity which represent the virtual camera on the virtual scene. Contain attributes and methods necessary to compute the camera's projection.

Light: Entity which represent the illumination of the scene.

StaticDisplayMesh: Entity used for represent a static virtual avatar. This entity used, when the user not launched the application correctly.

Defmesh: Entity used for represent a dynamic virtual avatar. The user’s movement will display on this virtual character.

Motion: Entity which contain one user’s motion posture. Contain the each bone rotation and translation value and it updated each frame.

Tracker: Entity which control the full-body tracking. Segment the human body from the background and determine how is situated the user.

Camera: Entity which represent the real world camera with the color and depth frame.

Skeleton: Entity which contain the user’s instance posture. Contain the attributes and methods necessary convert the data from the coordinates to bone’s transition and rotation.

Double buffer: Is an entity that controls the captured color and depth frames of the camera.

4.3. Class diagrams

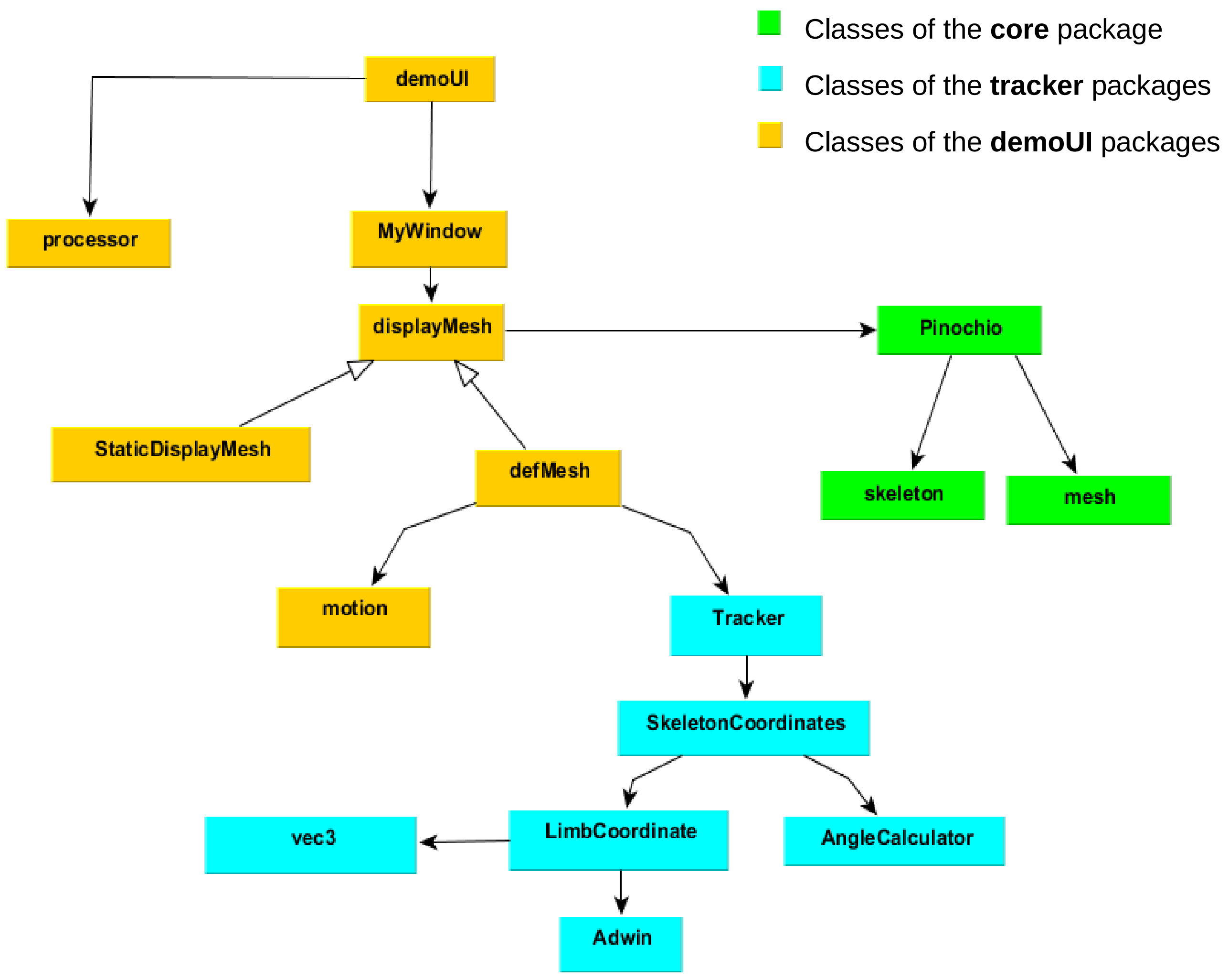


Figure 29: Simplified class diagram

The application follows the MVC design pattern (Model View Controller). This pattern separates the data, the logic and the GUI of the application. In my approach consists three different packages: Core, Tracker and DemoUI, which identified with different colors on the Figure 29. The “Tracker” package, which contains the logic of the full-body tracking and store the human limbs coordinates. The “core” package contains the logic of the automatic rig and the animation of the avatar. Finally the “demoUI” has a controller and GUI function.

Green color classes used as external resources files without the modification. The yellow color classes created for this project and the blue color classes are the “Pinocchio” classes, but have been modified for this project.

4.4. Sequence diagram

The following diagrams show the interaction between the objects of the application. These descript the objects and classes how involved in the scenario and which order exchanged the messages. These diagrams are related with the user cases, which I was discuss at the previous paragraph. Same way as a previous paragraph, the previous project's objects will not appear in the sequence diagram. For reason of readability, the sequence diagrams separated into modules and some parts of the implementation are simplified.

4.4.1. Sequence diagram 1

The users want to want to see the approach of motion capture. Initially the user selects the virtual character and launches the application with this character. The main function forwards the execution line to the "*process*" static function which analyzes the command line arguments and creates the virtual character. Then it launches the motion capture system and maps the human motion into the virtual character.

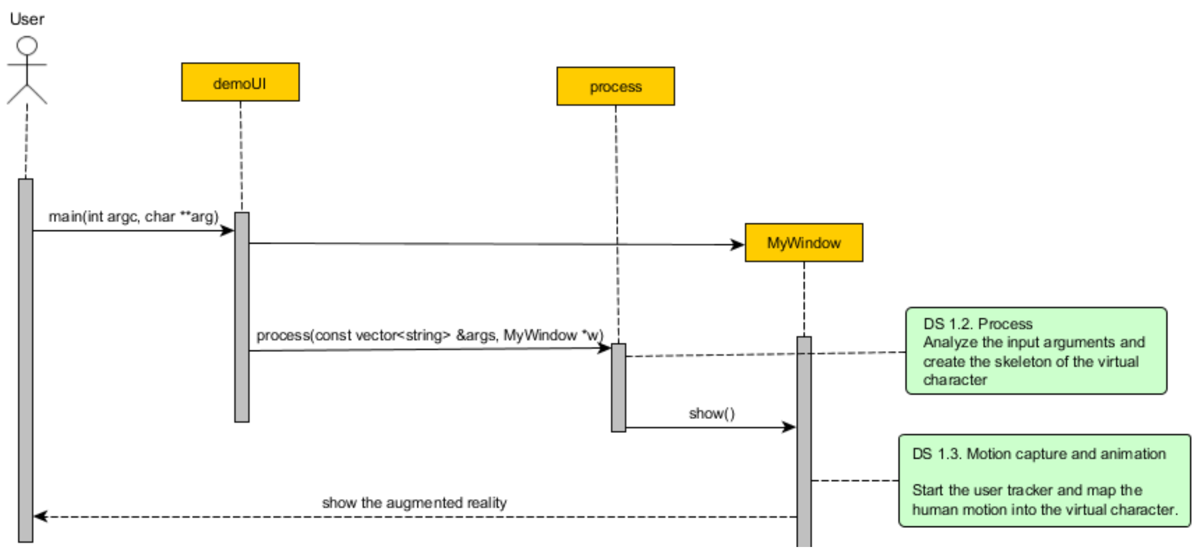

User
demoUI
process
MyWindow
main(int argc, char **arg)
process(const vector<string> &args, MyWindow *w)
show()
show the augmented reality
DS 1.2. Process
Analyze the input arguments and create the skeleton of the virtual character
DS 1.3. Motion capture and animation
Start the user tracker and map the human motion into the virtual character.

Figure 30: Sequence diagram 1

In the 1.2 sequence diagram (see the figure 31) I will extract the argument processing. Here the process function gets the input command line arguments. One of these parameters should be the virtual avatar filename and extension. If the user has been entered correctly, then the system creates the mesh object. Later the execution line calls the discretization and skin attachment function. Once it finished, the system create the motion object. This object contains the human posture, which updated every frame.

Up to this point I create the mesh object and the motion object. Now I'll call "*addMotion*" function to assign the motion object reference to the mesh object. So now the system is able to do the rig.

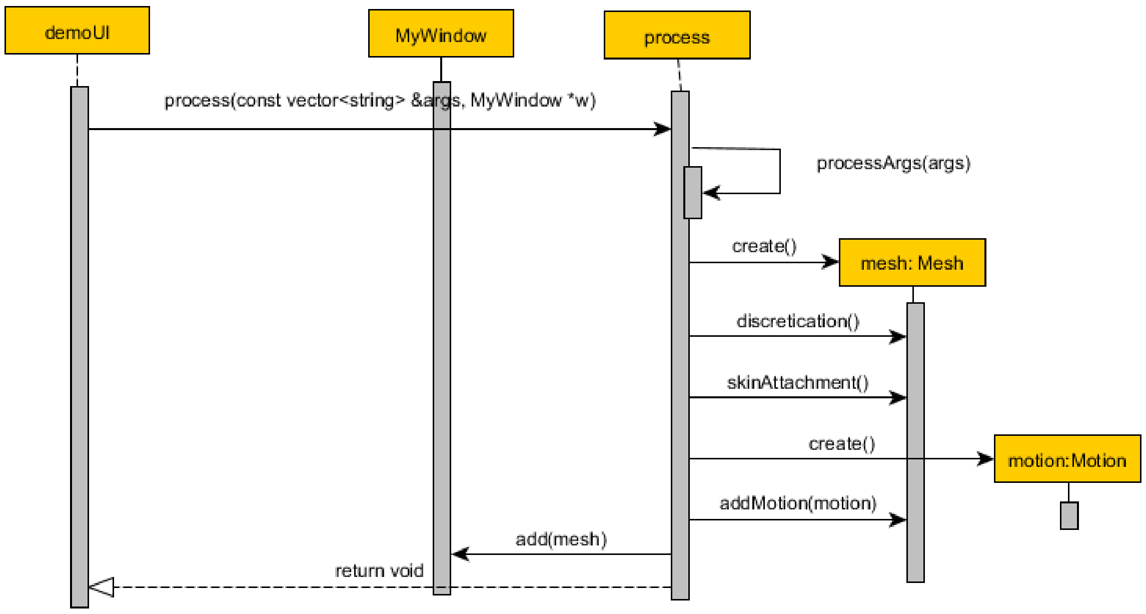


Figure 31: Sequence diagram 1.2

In the 1.3 sequence diagram (See the figure 32) I will extract the motion capture initialization and the full-body tracking process.

Firstly create the tracker object. In the constructor of the tracker create the "*Skeletoncoordinate*" for each body part object. Create the object for the head, neck, shoulders, elbows, hands, torso, hips, knees and foots. Secondly call the "*deviceInitialize*" function, which initialize the depth and the color camera. Now we get independent thread, which automatically update the color frame, the depth frame and the skeleton coordinates.

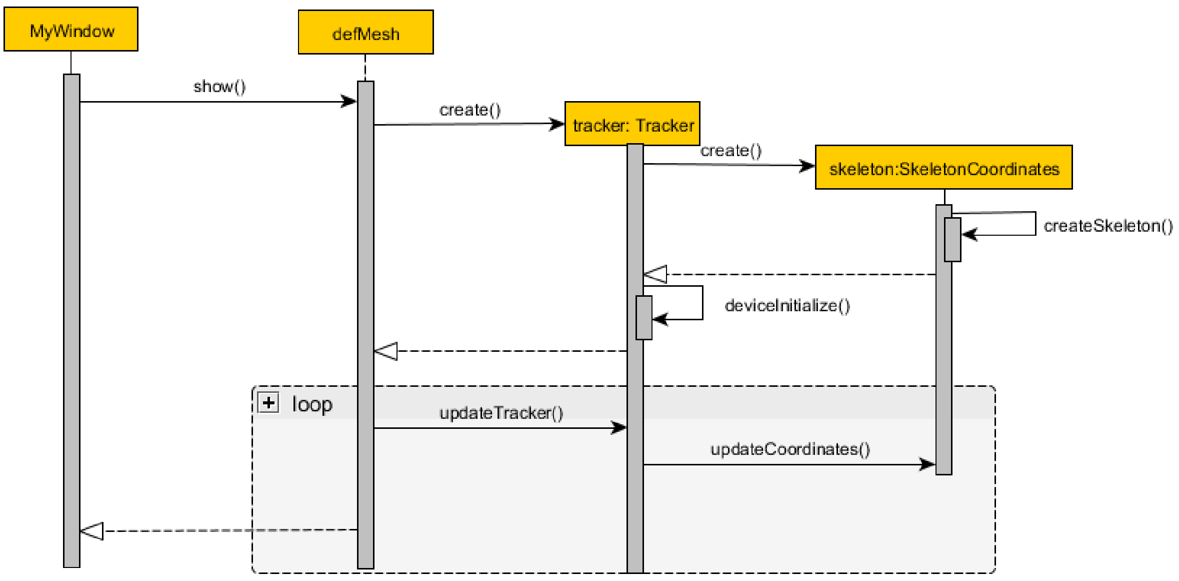


Figure 32: Sequence diagram 1.3

4.4.2. Sequence diagram 2

Initially the users want to visualize the virtual scene with different background (e. g. with the depth image, with the color image or with the black background). See the Figure 33. So the user calls the event function through user interface. The event function analyzes the user which camera’s photos want to visualize and save the selected camera’s code into background flags.
I should be note that the depth and the color should not activate or deactivate, because the app uses those to get the human joint coordinates. The two cameras launched with the application and it stays active till running the program.
Finally it calls immediately the getTrackerColorFrame or getTrackerDepthFrame function, which will return with the frames of the selected camera.

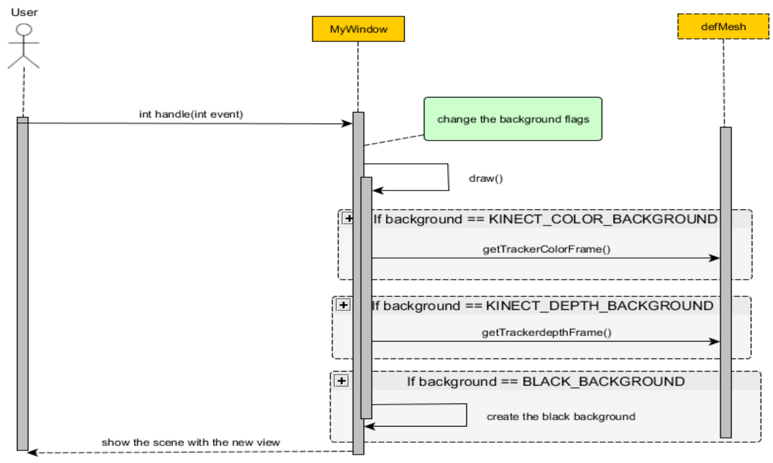

User
MyWindow
defMesh
int handle(int event)
change the background flags
draw()
If background == KINECT_COLOR_BACKGROUND
getTrackerColorFrame()
If background == KINECT_DEPTH_BACKGROUND
getTrackerdepthFrame()
If background == BLACK_BACKGROUND
create the black background
show the scene with the new view

Figure 33: Sequence diagram 2

4.4.3. Sequence diagram 3

In this case the users want to visualize the virtual scene from different viewing angles (Figure 34). They want to zoom in/zoom out, pan and rotate the scene. So the user calls the event functions through user interface. The event function analyze the user how want to change the viewing and calculate the scene transformation. Then it executes the scene transformation and visualizes the transformed scene.

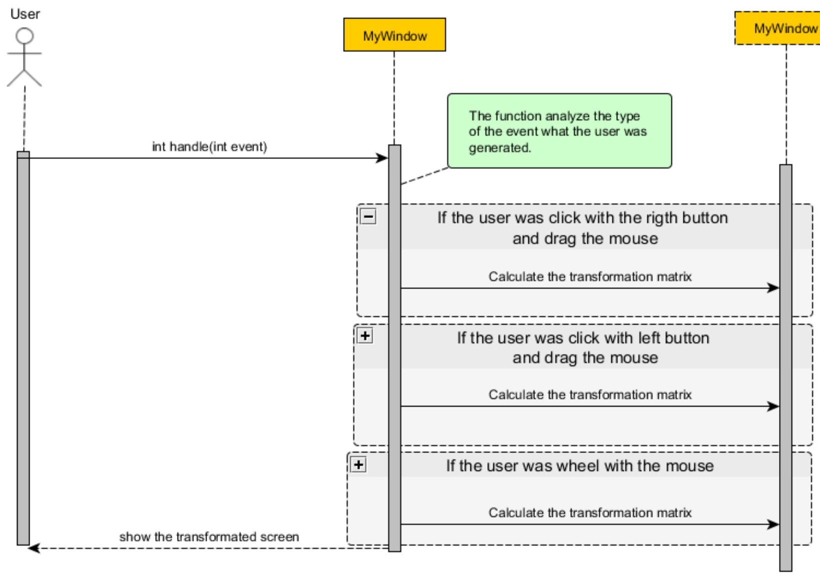

User
MyWindow
MyWindow
int handle(int event)
The function analyze the type of the event what the user was generated.
If the user was click with the rigth button and drag the mouse
Calculate the transformation matrix
If the user was click with left button and drag the mouse
Calculate the transformation matrix
If the user was wheel with the mouse
Calculate the transformation matrix
show the transformated screen

Figure 34: Sequence diagram 3

4.4.4. Sequence diagram 4

In this case the users want to visualize the virtual scene users want to view the scene with different view options. (e.g. with the global axes, with the virtual skeleton). See the Figure 35. So the user calls the event functions through user interface. The event function analyzes and determines which option is turned on. One it determine the options type, the system change the selected options flag.

Then, the system has to render the virtual scene with the active view options. Step by step check which option flag is active, and if it is active render on the scene. After the draw method execution the user view the scene with the active view options.

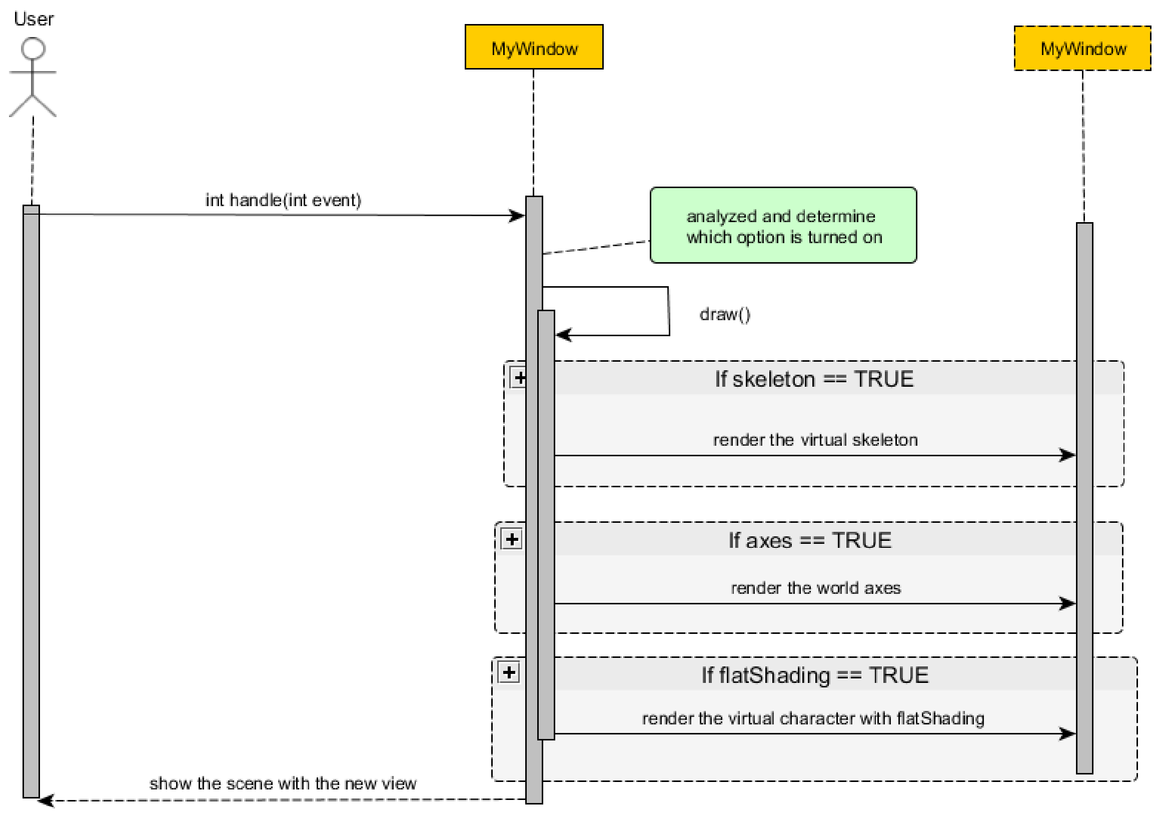

User
MyWindow
MyWindow
int handle(int event)
analyzed and determine which option is turned on
draw()
If skeleton == TRUE
render the virtual skeleton
If axes == TRUE
render the world axes
If flatShading == TRUE
render the virtual character with flatShading
show the scene with the new view

Figure 35: Sequence diagram 4

5. Simulation and validation

In this section I will evaluate the result of this work. This evaluation I will divide into two parts. Firstly I will evaluate individually each bone movement, then the avatar space positioning accuracy. Secondly validate the limit of the system, determine which postures are not able to render the system.

5.1. Movement and positioning validation

The application keeps sixteen different biped avatars (figure 36), which builds by Cosmic Blobs. Many of these character challenging due to their cartoony proportions and features that may be mistaken for limbs. The application correctly rigged 13 of these characters automatically, and the remaining 3 were correctly rigged with a single joint placement hint.

If you would like to animate with the different avatar, you have to create it with Blender's "Bone Heat" and the PM_heatWeight plugin for Maya.

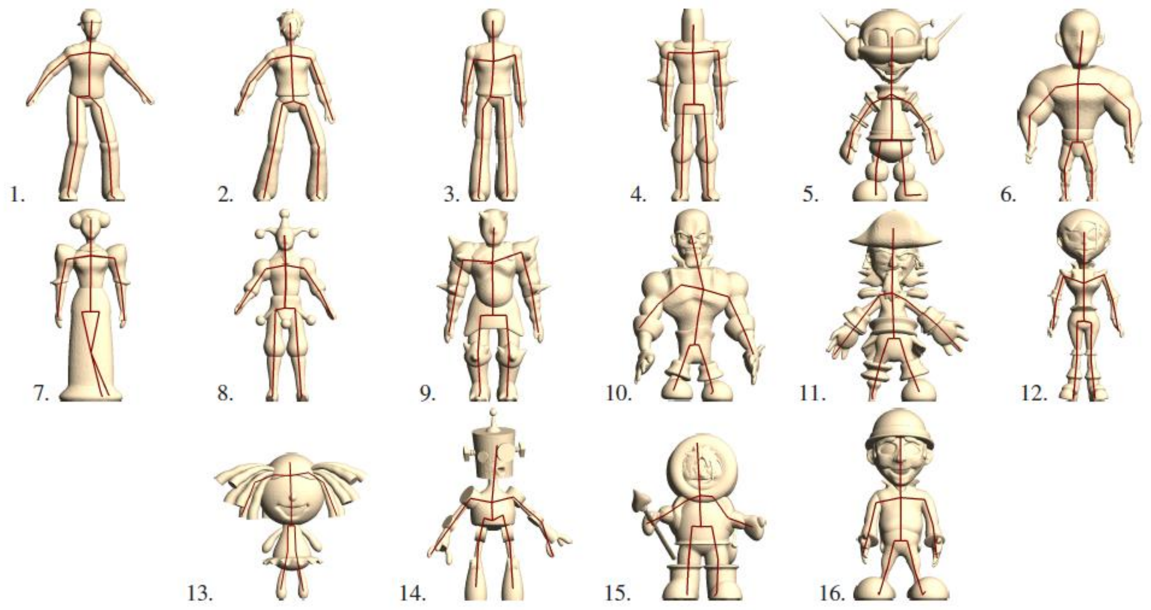


Figure 36: Meshes of the application

Figure 37 shows the individual movements of the left and right tibia. We can see that the application returns the user's resemblance to fairly accurately, although we observe that the left tibia angle a little inaccurate.

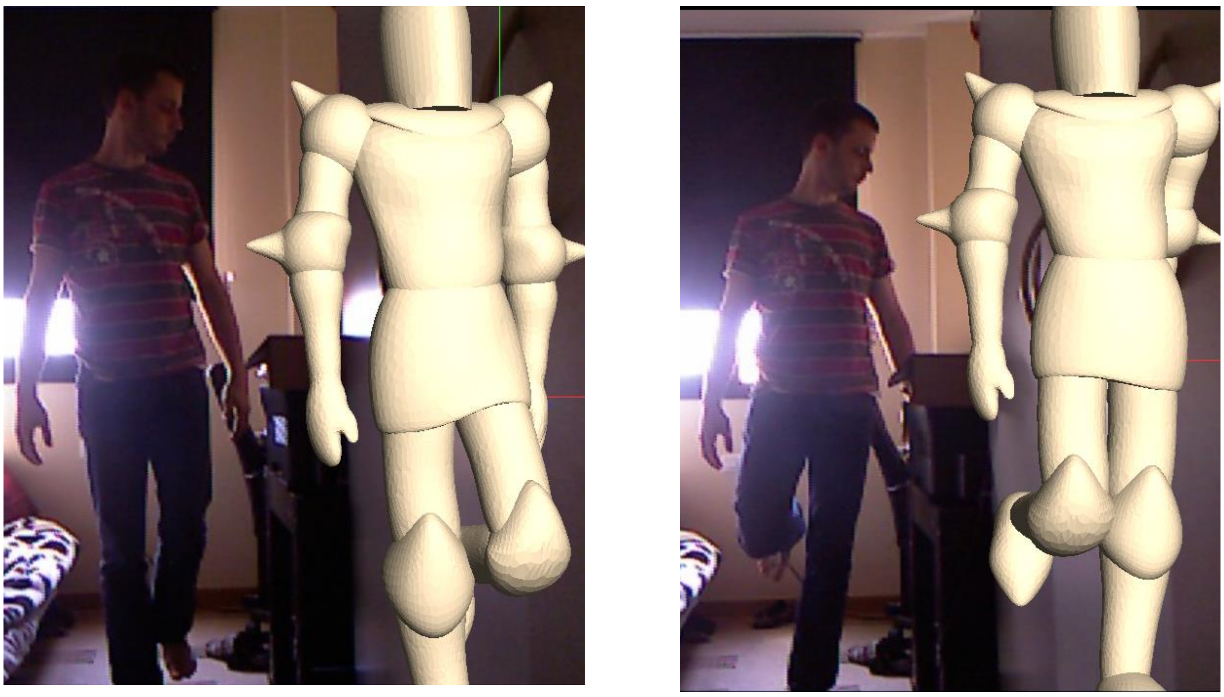


Figure 37: Left and right tibia simulation

The next figure (figure 38) shows the complete leg movements (tibia and femur). My approach returns the user's resemblance to fairly accurately, although the left foot took a more open position than the user.

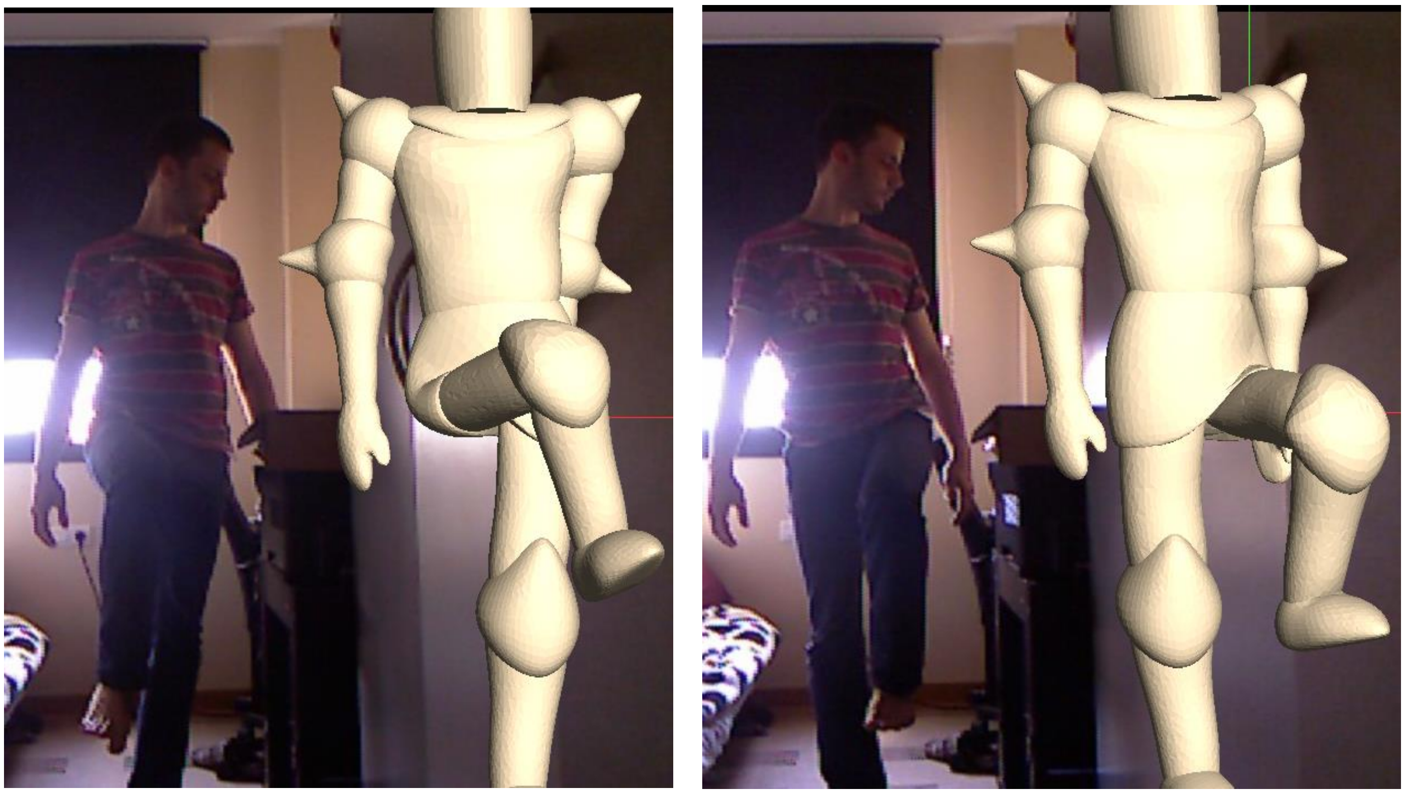


Figure 38: Left and right complete leg movement simulation

The last figure (figure 39) shows the complete arm movement (clavicle and humerus) and the trunk rotation. In this case, my approach returns 100% accurately with the user's posture.

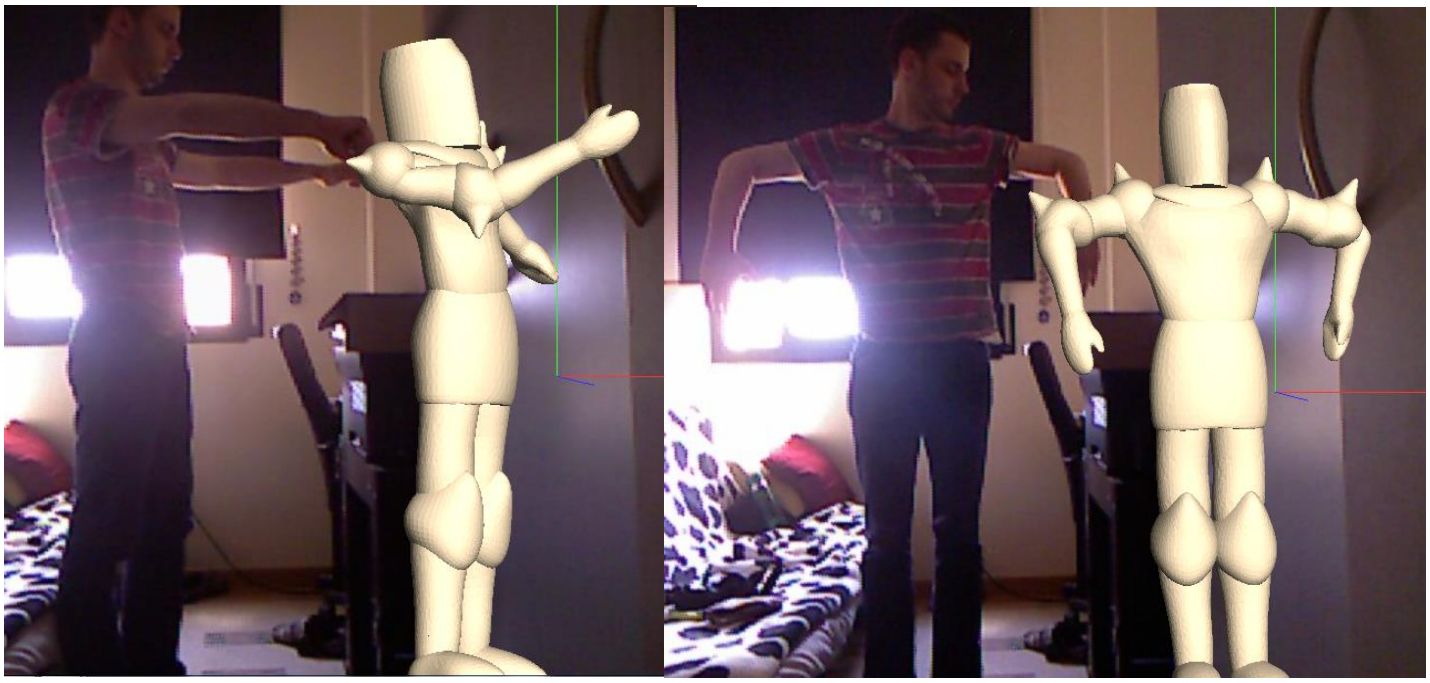


Figure 39: Rotation and complete arm simulation

Moreover, the application not just can render a standing posture, but it can render a sitting position too. (Figure 40)

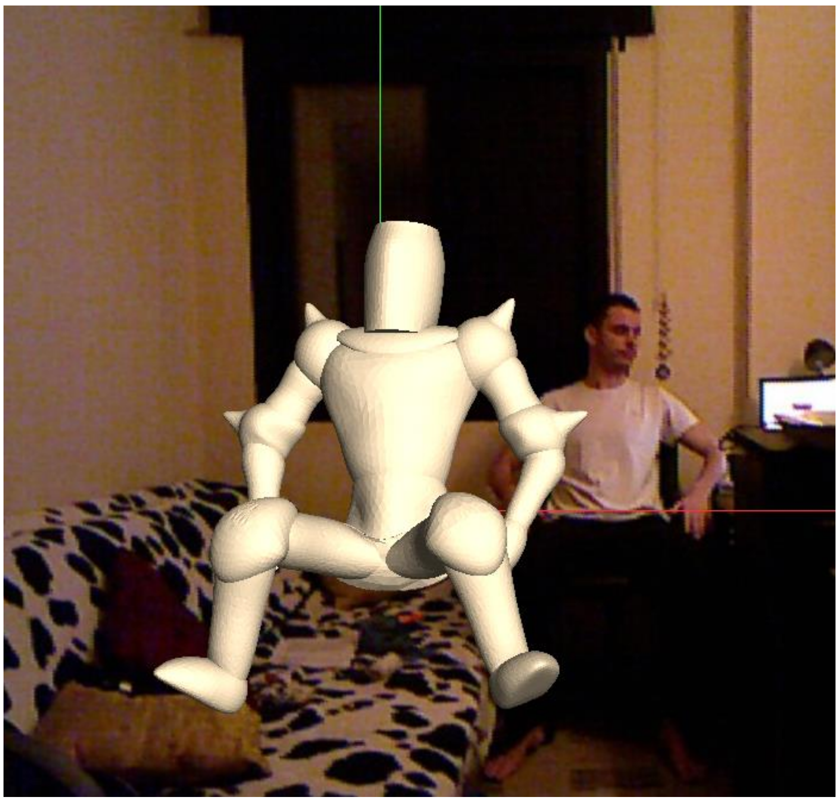


Figure 40: User seated position

5.2. Limitation

The test phase was observed that the approach is a fairly accurate, but there are limitations. The first thing we should note that the system is can replicate from a slow to fast motion. But if the user makes too rapid movement, then the virtual avatar little movement is slightly delayed.

The second key limitation is the user and the camera position. The user should be located front-facing with the camera. A little trunk rotation allowed (approx. max 60 degrees), but if the rotation is too big, then the avatar’s movement will be imprecise.

The final key limitation is the system is able to track only a one person. If located several users in the Kinect camera vision angle, then the program loose the previous detected user.

6. Conclusion and future line to continue

The aim of this project was to analyse, design and implement a motion capture system which allows loading any virtual avatars and applies the user motion into the virtual character. I don't want to create just a system where the avatar has a static position, but rather a system that return the user's full movement in 3D space. The primary aim was "shift" the virtual skeleton so as it completely match with the real user spatial position and play back the user movement.

Leveraging the success of the motion capture, it also seeks to interact with only a single motion camera. Today's motion capture designed with multiple cameras, which are too expensive and too hard to implement for a small developer team. So the aim was find the answer if it is possible to create the human captures system with a single depth and colour camera.

The final challenge was the user can be interacting with the system in real time. Although I said a percussion goal, but it was very important from the beginning of the project. For each analysis, design and implementation phase was a key consideration to reduce maximally the computational costs.

The analysis, design and implementation of the final application reached the goals. As we have seen in the previous paragraph (5. Simulation and Validation), the virtual avatar fairly accurately reproduces the user movement and position in space. Although the movement is not 100% accurate, but it's a good approximation. Involved a learning step, because this technique was required depth knowledge of motion capture and the automatic rigging techniques. Although the Graphics and Data Visualization, the Image Processing and the Computer given me a basic knowledge but it was necessary extended widely.

Below are listed the possible lines of continuation of the project.

- Improve the graphics engine or replace, that able to detect the collision between the parts of the body, textural appearance of the avatar and improve the rig methodology.

- Improve the translation method, that the virtual skeleton so as it completely matches with the real user spatial position.

- Although the project is designed to motion capture with a single camera and the result of my approach is very accurate, but can be improved if we increase the number of the cameras.

- As I sad many times, the motion capture has a wide range of applications. The current approach is a core, which can be implemented in any of these techniques. (E.g. virtual dressing room, animation movie maker, surgical education software… etc.) Need to define the aims of the software and which type of user will be directed the software and continue on this line.

7. Bibliography

[1] RUSINKIEWICZ S., HALL-HOLT O., LEVOY M.: Real-time 3D model acquisition. ACM SIGGRAPH 21, 3 (2002), pp. 438–446.

[2] Learning from Time-Changing Data with Adaptive Windowing by Albert Bifet & Ricard Gavaldà, http://www.lsi.upc.edu/~abifet/Timevarying.pdf

[3] http://www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-get.cgi/2008/MSC/MSC-2008-06.pdf

[4] http://www.mit.edu/~ibaran/autorig/pinocchio.html

[5] Skeletal Representations and Applications, by Andrea Tagliasacchi, http://arxiv.org/pdf/1301.6809.pdf

[6] MOTION CAPTURE INTERPOLATION by Jernej Barbic and Yili Zhao, February 2012, http://run.usc.edu/cs520-s12/assign2/

[7] Automatic Rigging and Animation of 3D Characters by Ilya Baran & Jovan Popović, 2007, http://www.mit.edu/~ibaran/autorig/

[8] MOTION CAPTURE FILE FORMATS by M.Meredith, S.Maddock, http://www.dcs.shef.ac.uk/intranet/research/public/resmes/CS0111.pdf

[9] MCML: motion capture markup language for integration of heterogeneous motion capture data by Hyun-Sook Chung*, Yilbyung Lee, http://www.motioninplace.org/MiPP_Articles/Chung.pdf

[10] Building Efficient, Accurate Character Skins from Examples by Alex Mohr and Michael Gleicher, http://research.cs.wisc.edu/graphics/Gallery/SkinFromExamples/skin-from-examples.pdf

[11] http://research.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/ASF-AMC.html

[12] http://www.mathworks.es/es/help/sl3d/vrrotvec.html

[13] http://en.wikipedia.org/wiki/Iterative_closest_point

[14] https://en.wikipedia.org/wiki/Motion_capture

8. Annex

Annex I: Installation, minim requirement

- Requirement:
  The software requirements to run this application:

  - Windows operating system, is possible Windows 7 (32 bits)
  - C++ library
    - Sources: http://www.microsoft.com/visualstudio/esn/products/visual-studio-express-for-windows-desktop
  - OpenNI SDK (version 2.x)
    - Sources: http://www.openni.org/openni-sdk/#.UbBp9tI3Dqg
  - Primesense Nite 2.x Middleware
    - Sources: http://www.openni.org/files/nite/#.UbBqQdI3Dqg
  - Kinect SDK (version 1.6)
    - Sources: http://www.microsoft.com/en-us/download/details.aspx?id=34808
  - FLTK libraries (version 1.3.0)
    - Sources: http://www.fltk.org/index.php

To successfully run the program, you has to install the above-mentioned software version. It is possible that other version also works, but is untested.

To install the software, download the sources from the link and follow the official installation instructions.

Annex II: User guide to develop

To develop this application, has to use the mentioned tools from the annex I. In addition, has to use the Visual Studio 2010 IDE, which can be downloaded from the same link as C++ library.

To develop the application, I'm recommended to following the next steps:

a) Install components and development environment
b) Once installed all components and the project located in a directory, open the project as follows:

- Open Visual C++ if it is not already open, and then from the menu select *File>Open>Project/Solution.*

- Navigate to the location where your new project has been created, and

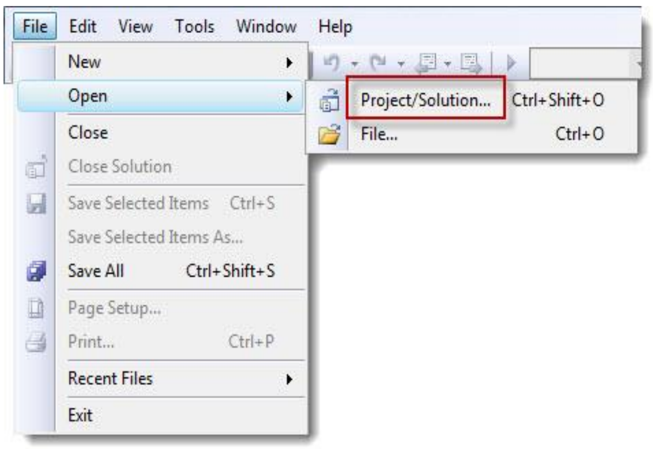
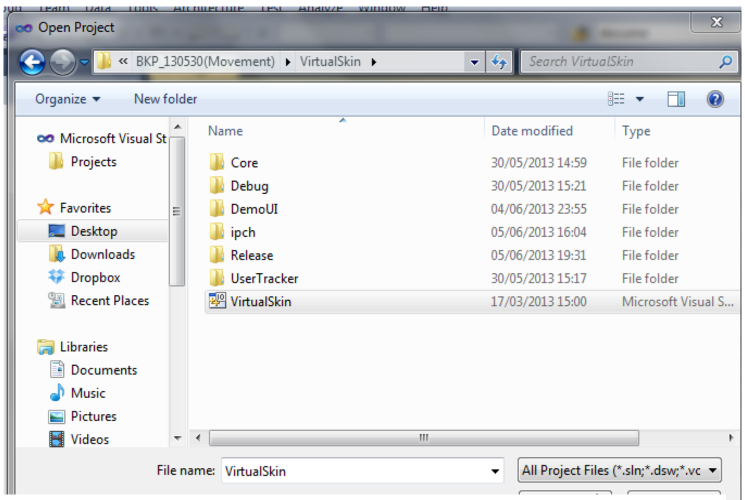


Figure 41: Open project on Visual Studio

- then move down into the folder and open the "VirtualSkin.sln" file:

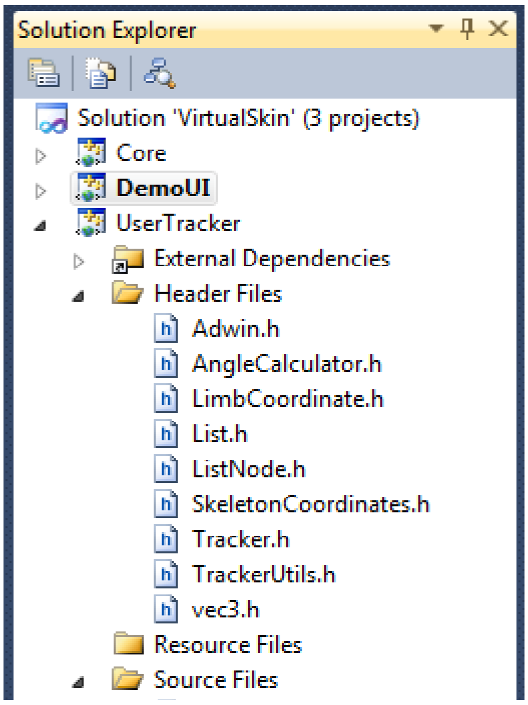


Name	Date modified	Type
Core	30/05/2013 14:59	File folder
Debug	30/05/2013 15:21	File folder
DemoUI	04/06/2013 23:55	File folder
ipch	05/06/2013 16:04	File folder
Release	05/06/2013 19:31	File folder
UserTracker	30/05/2013 15:17	File folder
VirtualSkin	17/03/2013 15:00	Microsoft Visual S...

File name: VirtualSkin
All Project Files (*.sln;*.dsw;*.vc

Figure 42: Open the ".SLN" file

- Once opened, you will see our project in the project file tree.

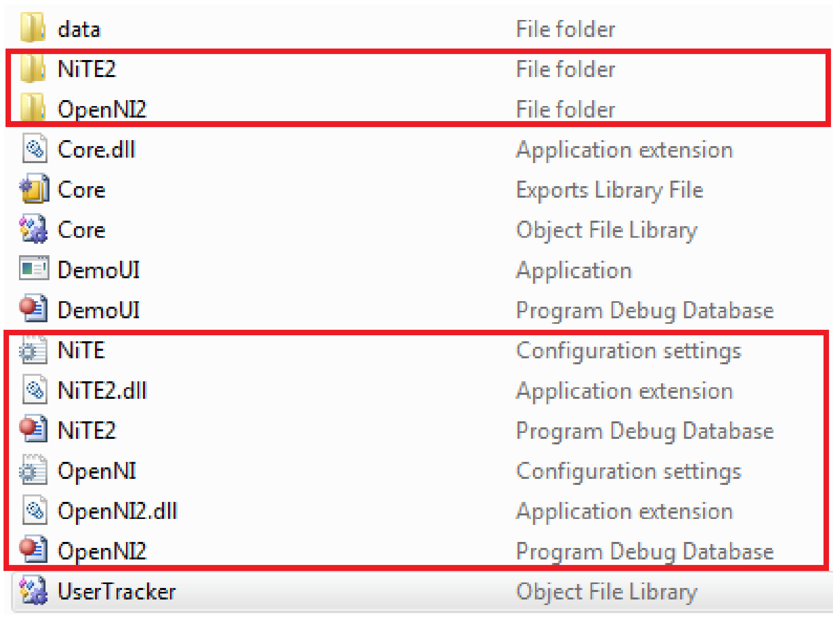


```
Solution Explorer
Solution 'VirtualSkin' (3 projects)
  ▷ Core
  ▷ DemoUI
  ◢ UserTracker
      ▷ External Dependencies
      ◢ Header Files
            Adwin.h
            AngleCalculator.h
            LimbCoordinate.h
            List.h
            ListNode.h
            SkeletonCoordinates.h
            Tracker.h
            TrackerUtils.h
            vec3.h
        Resource Files
      ◢ Source Files
```

Figure 43: Project file tree

The “Header Files” folder contains the headers with “.h” extension and the “Source Files” folder contains the source code with “.cpp” extension. I recommend, that once the project is opened, then set the building folder.

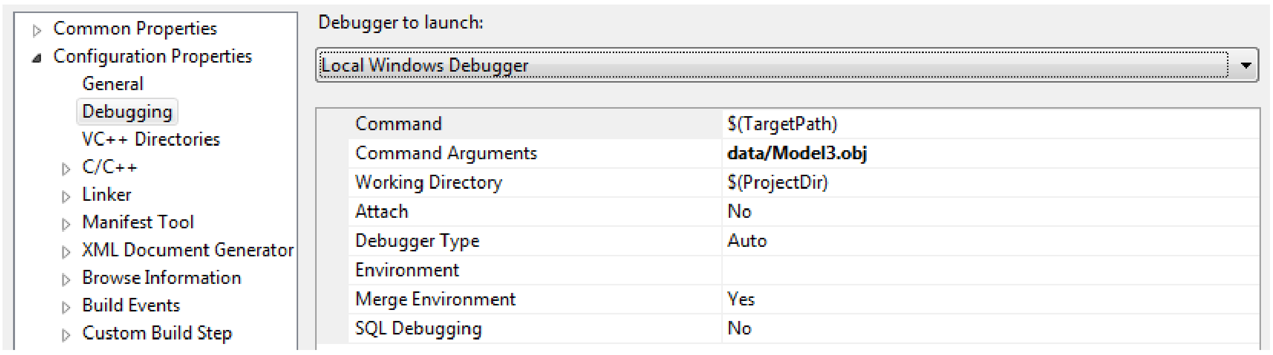
You should copy the OpenNI2 folder from C:\Program Files\OpenNI2\Redist and Nite2 folder from C:\Program Files\Nite2\Redist and insert into builder folder. Then should copy Nite and OpenNI execution files from the same folder and insert into “DemoUI” folder.



data	File folder
NiTE2	File folder
OpenNI2	File folder
Core.dll	Application extension
Core	Exports Library File
Core	Object File Library
DemoUI	Application
DemoUI	Program Debug Database
NiTE	Configuration settings
NiTE2.dll	Application extension
NiTE2	Program Debug Database
OpenNI	Configuration settings
OpenNI2.dll	Application extension
OpenNI2	Program Debug Database
UserTracker	Object File Library

Figure 44: Build folder with the NITE and OpenNI files

- Finally to run or debug the application with the command-line go to the "*Project/Properties/Debugging section/Command line arguments box* introduce the next command: "*object_file_name.obj*" or "*object_file_name.obj -motion onifile.oni"*



- Common Properties
- Configuration Properties
  - General
  - Debugging
  - VC++ Directories
  - C/C++
  - Linker
  - Manifest Tool
  - XML Document Generator
  - Browse Information
  - Build Events
  - Custom Build Step

Debugger to launch:

Local Windows Debugger

Command	$(TargetPath)
Command Arguments	**data/Model3.obj**
Working Directory	$(ProjectDir)
Attach	No
Debugger Type	Auto
Environment	
Merge Environment	Yes
SQL Debugging	No

Annex III: User guide

First, I will explain how to run the application from terminal:

- Open a terminal
- Navigate with the terminal to the source project folder where is the executable files
- Copy the OpenNI2 folder and OpenNI.ini, OpenNI2.dll and OpenNI2.pdb files from C:\Program Files\OpenNI2\Redist and insert into Exe\
- Copy the Nite2 folder and NiTE.ini, NiTE2.dll and NiTE2.pdb from C:\Program Files\Nite2\Redist and insert into Exe\
- Once you find the folder, type the following command without the quotes: *"DemoUI object_file_name.obj"* or *"DemoUI object_file_name.obj -motion onifile.oni"*

Once executed, the first thing what you will see the following GUI:

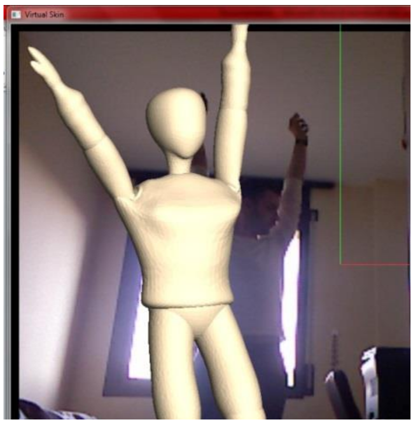
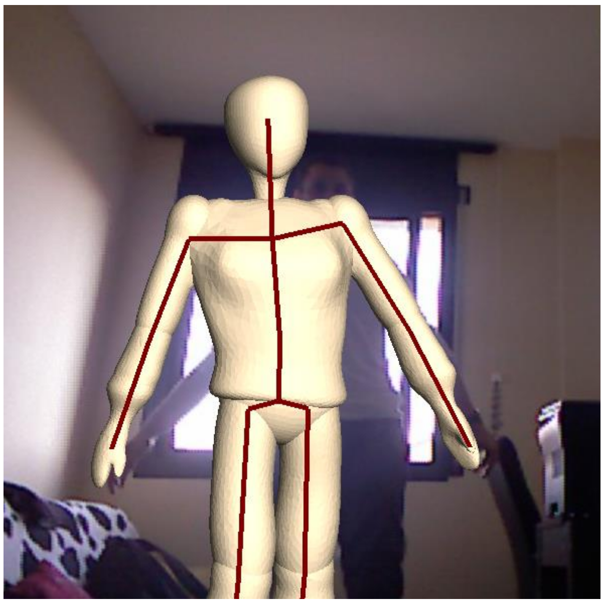


Figure 45: Application GUI

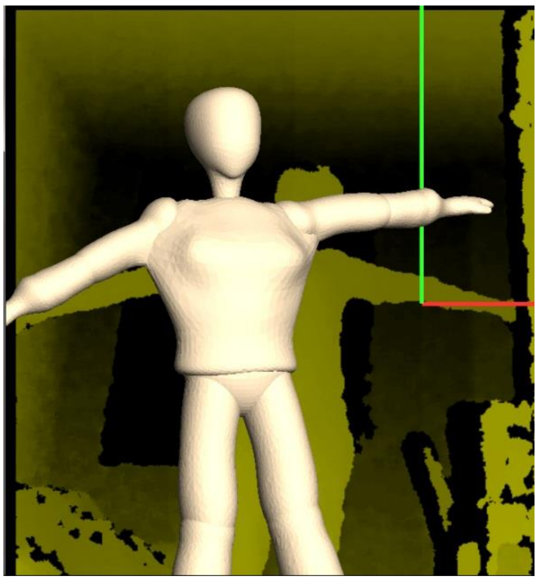
The application consist a terminal window, where display the key information of the application state and the graphic interface where display a motion capture with the character what we was choose at the previous step.

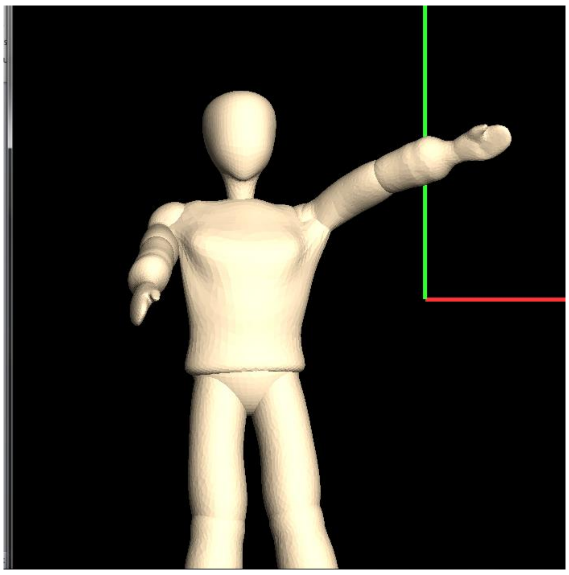
To rotate the volume, click with the left mouse button on the volume and rotate it to get the desired position. To move the volume, click the right mouse button and move the volume to obtain the desired position. To zoom, click with the middle mouse button on the volume and move the mouse forward or backward to zoom in for more detail.

The animation represented in two parts: a surface representation used to draw the character (m*esh*) and a hierarchical set of interconnected bones (called the *skeleton*). If we want draw also the rig skeleton, then we have to press the “s” key. To deactivate that, press the “s” key again.



To change the background of the scene, press the “b” key. Now the application switches the camera and rendering the new camera’s frames.





Finally, to view the scene with the global axes, press the “a” key. Now the system renders the virtual character with the global axes. To deactivate that, press the “a” key again.