PRIFYSGOL
BANGOR
UNIVERSITY

School of Computer Science
Bangor University

Technical Report Number:	**CS-TR1-2011**
Category:	**MRes Dissertation**
Title:	**Accurate 3d Rigged Avatar Generation With A Kinect**
Author:	**Gaël Charpentier**

Accurate 3D Rigged Avatar Generation with a Kinect

Gaël Charpentier

Supervisor:

Nigel W. John

October 2011

School of Computer Science

Bangor University

Statement of Originality

The work presented in this dissertation is entirely from the studies of the individual student, except where otherwise stated. Where derivations are presented and the origin of the work is either wholly or in part from other sources, then full reference is given to the original author. This work has not been presented previously for any degree, nor is it at present under consideration by any other degree awarding body.

Signed .....(Student)

Date ….....

Statement of Availability

I hereby acknowledge the availability of any part of this dissertation for viewing, photocopying or incorporation into future studies, providing that full reference is given to the origins of any information contained herein.

Signed ..... (Student)

Date ….....

Acknowledgements

Producing a dissertation is without doubt an arduous journey. Fortunately, I was very lucky to receive unselfish help and love from both my friends and family. Without their supports, i would have more struggles in the process of producing the work.

First, I would like to thank Nigel John, my dissertation supervisor who guided me and helped me throughout this research. He inspired me with his professional knowledge and his breeziness. Without him I could not have achieved this.

I would also like to thanks my wife who accepted to climb on a chair to provide me a model for my experiments. She also supports me throughout all my dissertation writing by providing me love and meal.

Thanks to my parents who always support me and encourage me to study in order to surpass myself.

Finally, I would like to thank my future daughter who has given me a reason to study hard and hope for the future.

Abstract :

This paper introduces a new way to generate 3D animated avatars. Current avatar generators typically choose from a library of body parts, some advanced ones also allowing for deformation of these body parts. However, we try to introduce a system that is able to generate an avatar that is a perfect copy of a real person by using the Microsoft Kinect as a 3D scanner. This project determines if such a system is feasible and implements some of the functionality required, but does not provide a fully deployable system. First, the different character, or avatar, generators currently available are reviewed. Then we describe technologies used in character animation and discuss which ones are crucial for character generator, notably skeletal animation techniques. This knowledge is then used to improve the character generator of the <e-adventure> platform, a software application developed to facilitate creation of educational games for people without programing skilsl. After this, a new way to generate an avatar that is a perfect copy of a person is imagined using the Kinect 3D scanning capability. Experiments are carried out to check if the Kinect can deliver data that can be used to generate a 3D model of a person. After having generated such a model, it is animated using auto-rigging software. Thus the imagined system has been proved to be feasible but we saw that the main remaining barrier was the 3D data registration and reconstruction. However, we predict that this system will be soon realizable as Microsoft unveiled a 3D reconstruction system using the Kinect that can be used easily by people inexperienced in 3D scanner, the KinectFusion project.

Contents

Chapter 1: Character Generator and Hypothesis

1.1. Character generator

1.1.1 Presentation of character generator

Character creation tools are now very common in the video game industry. Their principal use was originally in role playing games and MMORPG (Massively Multiplayer Online Role Playing Game) , a type of role-playing video where a large number of players interact in a persistent virtual world. However, they are now used in a great variety of games as developers now give more importance to user-generated content. User-generated content is becoming more and more common in video games due to the fact that every gaming platform is now connected to internet. Previously, The personal computer was the only platform to have this advantage but now games consoles and even hand-held consoles have an internet connection and have network systems such as the X-BOX live, which allows exchange between players. These systems are now turning into social networks . As a result, it is important to allow the player to represent himself/herself in the game.

A simple character generator uses predefined models of body, head, hair, etc. and assembles them together **(fig.1.1).** However, technologies used in video games have greatly evolved allowing more customization of avatars. More advanced character editors allows modification of features like chin, nose shape, etc... **(fig.1.2)** most of the time with sliders but some interfaces permit dragging points on some part of the face to deform them. The player can often use a slider to choose the body shape: muscular / weak, fat / thin. Most character generators also allow the players to attach accessories to their character.

Figure 1.1 *Example of a simple character generator. Here the user can choose the skin color, face, hair style, hair color.*
(World of Warcraft, Blizzard Entertainment, 2004)

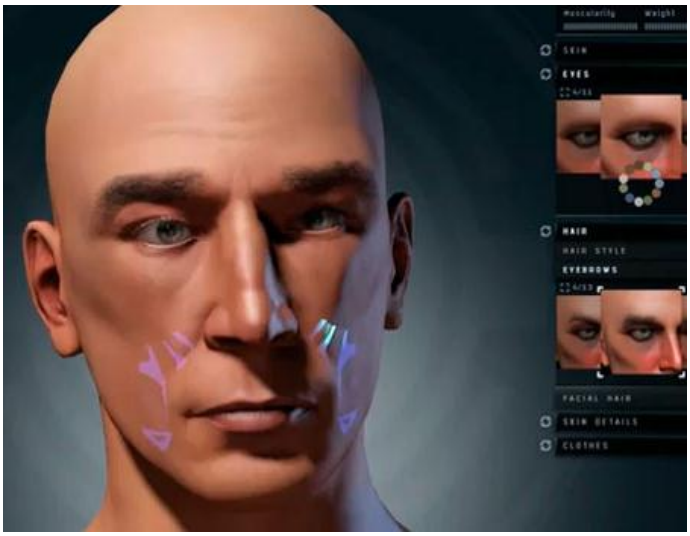


Figure 1.2 A more advanced character generator where the user can choose eye color, eye brown shape. By dragging some area with the mouse, the user can change nose, forehead, chin and cheek shape.
(EVE Online, CCP Games, 2010)

Generally, character creation editors are organized to take the player through a fixed set of different steps in order to avoid providing them with a complex 3D content creation tool. First the player can choose the gender of their character. This is always the first step of character creation as the following customization (body shape, hair style, clothes...) will be associated with a particular gender. In a fantasy or science fiction game, the player can also choose a race or an ethnicity. Selecting those two criteria at the beginning of the character creation reduces the available choices for the rest of the creation process. For example, a male will wear male clothes and the clothes or hairstyle available will depend on the chosen race or ethnicity. These restrictions are set to keep a visual integrity and coherence in a game universe where most of characters are user-generated.

Character creator tools are also used by the movie industry. A 3D CGI (Computer Generated Imagery) representation of a real actor is often used for scene that cannot be performed by the actor or because the scene does not require close up of the actor, which avoids the need of shooting the actor and removes the need for a stunt man **[1]**. Another important advantage of a virtual double is the possibility to use virtual animals instead of trained animals, which require a lot of efforts. In the

case of virtual actors, an interface permitting choice from a library of body parts and clothes is not the solution. Here we need to capture the actor's appearance. Equipments capable of digitalising an actor such as laser scanner are available but they are complex and very expensive. Some research have been done by Ahmed et al **[2]** and Villa-Uriol et al **[3]** to automatically generate a 3D avatar, the former using a multi-camera footage of a real performer while the latter using a set of photography showing each side of the model.

Virtual actors are also used to shoot crowd scene in movies. This removes the need of extra actors and allows an easy shoot of a scene with thousand of characters in harsh environment, which would have been very problematic with real actors. An example of software for crowd simulation is MASSIVE (Multiple Agent Simulation System in Virtual Environment) **[4]** which has been used to create the epic battle scenes in The Lords of the Ring trilogy. Using Artificial Intelligence, this software can simulate thousands of agents acting independently and reacting to the environment conveying the impression of a real crowd. But to make a crowd realistic, all of the virtual individuals in the crowd must be different, in both visual appearance and animation. Thus in this case some kind of character generator is needed. The software will be given a library of models and animations and will build random characters by mixing different body parts together and give them a different animation. To make those characters appear even more unique, their size and animation speed can be tweaked.

Figure 1.3 *A battle scene From the Lord of the Rings trilogy were the character have been randomly generated then animated using the MASSIVE system [17].*

1.1.2 Examples character generator

A good example of character generator which offers a lot of freedom in customization is the one used in Second Life **[5].** First, this editor allows modification of the shape of almost all body parts. Only for the nose, sliders allow modification of nose size, nose width, nostril width, nose tip angle, nose tip shape and more **(fig.1.4)**. Furthermore, character creation is not limited to choice offered by those sliders. One can make one body part invisible and use any 3D model instead. The great strength of the second life editor is to allow users to use their own 3D model as clothes, accessories or body parts. 3D model can be created directly inside second life using the in-game 3D editor **(fig.1.5)**. 3D models can also be created into 3D modeling software such as Maya or Blender and imported into second life. 3D model can be attached everywhere on the character.

The editor includes a clothing section. However, those clothes are just texture applied on the character, making them very easy to customize by importing texture. But if the user wants more than texture, he can use 3D model as mentioned previously. Second Life also allows the players to import their own animations using external tool such as Blender or Poser. One of the most important features of second life is that users can share their creation with everyone. Therefore, a vast choice of clothes, accessories and animation is available in the game allowing the creation of unique avatars.

Figure 1.4 The Second Life character generator. Currently open on the nose customization section

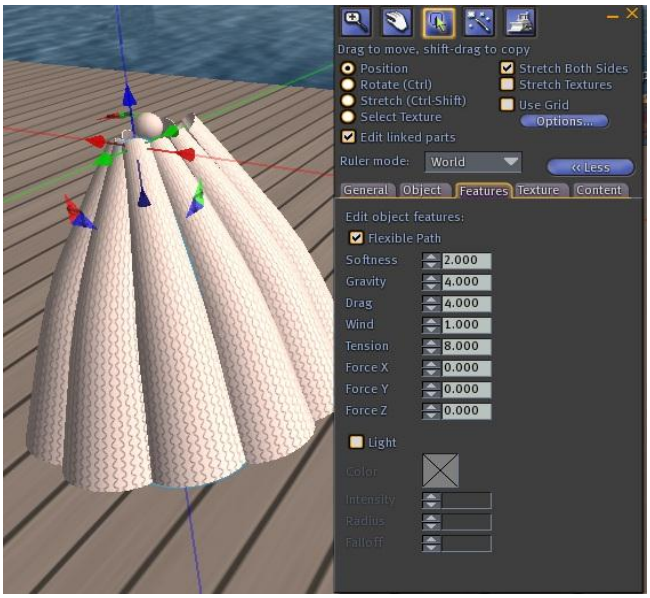


Figure 1.5 The Second Life in-game 3D editor

Another impressive character generator is the one used in Spore **[6]**, a game produce by EA Games. In this game, the player creates his own creatures. The stunning feature of this character generator is that the creature morphology is decided by the user by allowing him to place limbs and organs wherever he wishes **(fig.1.6)**. The generation of the animation is done in real time. Thus, the creature is animated immediately after any modification. We will talk about the technology used in this game in the next chapter.

Figure 1.6 Example of user-generated creature from Spore
(Picture from **[6]**)

1.2. Hypothesis

The current generation of video games often provides tools that allow the player to create customizable characters (or virtual avatars) that can be imported into the game. However, no tool has yet been developed that will allow the player to insert a perfect copy of themselves inside a game or any other 3D application. The hypothesis of this research is that the new Kinect motion sensing input device by Microsoft, an XBOX 360 accessory developed for gesture recognition but which conceals a powerful 3D scanner, could be used to develop such a tool.

1.3. Structure of Dissertation

In the next chapter we present an overview of current challenges related to character animation in the context of real time rendering 3D media in which most popular and widespread use is video game, and review common technology and techniques used. In Chapter 2 we present work carried out to integrate a character generator using a 3D rigged model into the <e-Adventure> platform that currently uses a 2D sprite to generate characters. In Chapter 3 then we will investigate a new way to generate avatar. We will use the Kinect to capture user appearance and generate a accurate 3D avatar. In the Chapter 4 we will improve the system developed in Chapter 3 by investigating a way to align automatically point cloud. In the last chapter, we will explore different ways to extend the system functionality.

Chapter 2: Background

2.1. Challenges in character animation: The Uncanny Valley

When working on virtual character, we are often faced with a phenomenon called the Uncanny Valley. This term was coined by Masahiro Mori **[7]** a robotic engineer. It originally concerned robots but it is now also applicable to any computer generated character. The more a virtual character gets closer to a real person, the more the viewer will be sensitive to non-human features or behaviors, which will convey the impression of a dead corpse rather that a real person **(fig.2.1)**. This phenomenon is even stronger if the character is animated. If the character has a realistic appearance, this realism will be spoilt if the animation is not at the same level as the graphics. When creating a virtual character, the artist has to make sure that all the character components are matching each other and that the animation of the character is adapted to its body.

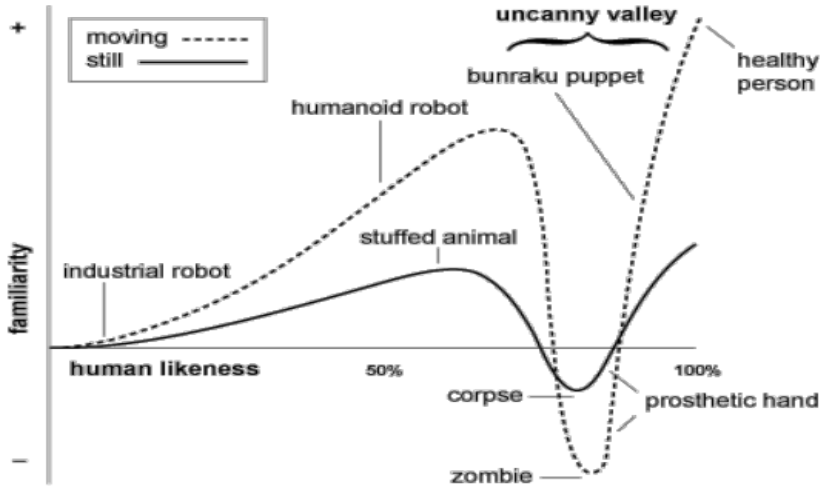


Figure 2.1 Hypothesized emotional response of human subjects plotted against anthropomorphism of a robot

In the case of character generator, there is more risk to fall in the uncanny valley as the character will be composed of different parts randomly assembled together by the player. Furthermore, when allowing feature deformation (size of nose etc...), if too extreme deformation is allowed the result won't be realistic. Therefore the challenge is to make sure that all components at the disposition of the user give a convincing result when put together and that they are animated in a realistic way. One of the possibilities is to restrict the available choices when a part of the character is selected. Furthermore, selection of components should affect the animation of the character

which is often omitted in video games: a character will have the same animation regardless he is wearing pants or a long dress that looks completely unrealistic. Also, providing the player with the possibility to choose the animation of his own character is a great improvement in character creation. Indeed, two characters with the same physical appearance but using different movements will appear to have a completely different personality.

Working on a non-human character gives more freedom with customizations. Indeed, giving too much control on feature deformation for a human character will lead to results that will fall into the uncanny valley. But, in the case of fictitious creatures, there is less standard expected.

2.2. Technologies used in character animation

2.2.1 Surface representation

Two main techniques are used to represent characters geometry: polygons or NURBS patches **[8]**. The former is the most used for real time rendering due to its simplicity, flexibility and due to the fact that polygons rendering is optimized on nearly all graphic hardware. The latter is often used for animation as it permits rendering of smooth curves, but requires a heavy computation overhead. It also offers less flexibility in modeling and in run-time modification than polygonal representation. Indeed, when animating an object made of NURB patches, holes can appears between the patches. This will not happen with polygons which consist of a set of vertices defining faces. Displacing a vertex deforms faces defined by it, which allow easy deformation of a mesh. But with polygonal representation, the object represented will look chunky if it is made of a low number of polygon. Nevertheless subdivisions surfaces, a technique which subdivide one polygon into smaller one, permit to give a smoother appearance to the object though may slow the rendering as more polygon are created. However, computers have now enough processing power to display high number of polygon giving the illusion of curved surface while maintaining high frame-rate. Also, shaders permit to give the illusions of a complex surface even with a flat surface by dynamically modifying the texture rendering at run-time by taking into account the camera position. Character generators are mostly used in video games and are not used in non-real time application so we will only concentrate on the polygonal representation.

2.2.2 Animation

Whether or not a character will be accepted by the audience will depend mostly on its animation so we will review the techniques used for characters animation. In earlier video games, “Morph target animation” was used to animate virtual objects. In this technique, a deformed model is recorded for each key frame then during animation the position of the model is interpolated between key-frames. This method is very fast but it requires saving a complete mesh for each key-frame. Another major drawback is that once those meshes are created, the movement can't be modified at run time, which means that the character will only be able to play a set of recorded animations. This prevents the adaption of the animation to the situation. For example, a dying character animation will show the character falling on the ground. But if this animation is used on a tilted ground, a part of the character will pass through the ground or pass through a nearby wall if is there one.

A more flexible method is now used. To animate characters (or also any kind of object), the technique used by nearly all animation systems today is skeletal animation, which is composed of two layers. The first one is the motion system, which consists of hierarchical collections of limbs (or bones), acting as the armature of the character, placed inside the character mesh. The second part is the deformation system, often called “skinning”, which deforms the mesh according to the skeleton state. If the second layer is configured correctly, animating the character is achieved by only moving the skeleton, which will deform the skin automatically, requiring no additional work at skin level **[9]**. In earlier 3D computer graphics, only the movement system was present. Indeed, character models were made of individual rigid meshes attached together **[10].** So it was enough to link each mesh to a bone to animate a character. At that time, to design a character generator it would be enough to link different body part together to build a character. But the visual result was very poor compared to today’s standards as virtual characters looked like articulated toys due to the brutal transition between limbs **(fig.2.2)**. This technique is still acceptable for a character like a robot or a skeleton as it does not have skin or stretchable tissue so the articulated toy look would not affect them. But for a human being, an animal or even a fictional creature, the viewers now expect the skin or clothes to appear as one entity covering the entire body.

To reach the desired result, the character body is no longer composed of several pieces, but is composed of only one mesh. The deformation system then permits deformation of the mesh according to the skeleton state. This gives a better visual result at the cost of more computation. The most used technique for the deformation layer is vertex weighting. Each vertex composing the character mesh is associated to one or several bones of the skeleton. When animated, the vertices are translated and rotated according to the transformation of the bone they are linked with. If one vertex is linked to several bones, each bone link is weighted to influence more or less the vertex final position. This allows the mesh to form a smooth curved shape at the articulation area instead of a sharp curve when limbs are bent **(fig.2.3)**.

The deformation system can also be used to simulate muscle interaction with skin. Chadwick et al.**[9]** have used Free Form Deformation (FFD) to simulate muscle bulging by wrapping FFD boxes around the model limbs whose shape are controlled by the skeleton state. They also used FFD to simulate the movement of fat tissue by using a force lattice composed of mass and spring elements. This shows that the use of skeleton is not only confined to animation.

Using FFD gives a better visual result than that of vertex weighting for extreme joint bend. But due to its computation cost, it is used for animation movies whereas vertex weighting is used for nearly all real time applications.

Figure 2.2 *An example of a character composed of several rigid mesh assembled together*
(Final Fantasy 7, SquareSoft, 1997)

Figure 2.3 *An example of a character using the skeleton and vertex weighting technique*
(Half Life 2, Valve, 2004)

The use of a skeleton permits to store animation separately from the mesh as it only requires the bone position for each key-frame. The positions of the vertices are calculated at run time.

This allows the use of the same animation with different character only requiring the two skeletons to have the same bones hierarchy. If the characters have the same morphology (same limbs size), no further processing is needed. But most of the time, character will have different limb length. Thus, when animating, the character morphology will be distorted. In this case, Online motion retargetting **[11]** is used to adapt the skeleton to the new character. The animation is modified to fit the target character while preserving the original motion characteristic.

But skeletal animations offer a greater advantage than just allowing separate storage of meshes and animation. The main advantage of skeletal animation is to allow control of the animation at run time. Stored animations can be slightly modified to adapt to the environment, for example, to make sure a character's feet are always at the ground level on an uneven terrain or that a part of it's body do not pass through a wall. To do so, inverse kinematic techniques are used to calculate the position of all bone depending on the expected position of the end bone. In the example of uneven terrain, if the ground is higher that the feet position in the standard animation, inverse kinematic algorithms will calculate which knee angle is needed to put the feet at the right height.

Skeletal animation not only permits slight modification of the original animation but also permits to completely control the body animation, for example, rag-doll animation where the whole skeleton control is given to the physics engine (after a character death, for example).

2.3. Advantage of skeleton for character generator

Skeletal animation is essential for character generator for different reasons. If Morph target animation was still used, when the user finished customizing his or her avatar, he would need to send the generated mesh to a graphic artist to manually generate the animation. With skeletal animation no further work is needed as each part of the character would contain bones. Thus we only have to move the bones using an already existing animation.

We saw previously that character generator allows users to select from a library of different meshes for different body part. Even if a lot of meshes are provided, it will limit the character morphology to the ones of available meshes. Another strength of rigged model is that their

morphology can be altered by manipulating bones position and scale. This requires motion retargetting to adapt the animation to the new skeleton morphology which can be done in real time. So when a player changes the height of his or her avatar, the change will be displayed immediately without interrupting the current animation.

Traditionally, character skeletons are made of a single unified hierarchy of joints but now modular skeletons are appearing. This kind of structure permits each part of the body to have its own animation that can be modified without affecting others parts. Another advantage is that it permits modification of sub parts of the character body easily if the sub-skeleton is associated with a different mesh. For example, we can first design a character with normal arms and then replace one of the arms by a tentacle that has a totally different animation. This technique is obviously very useful for character generator.

The skeleton is not only used for the character animation. It also permits other meshes to be attached to the character model. So it is used in character generator to add accessories to customize the character. To allow for more flexibility in the accessories placement, additional joints might be placed on the skeleton. A simple skeleton would be a single link across each limb. However, if the object is not placed on a bone root, its position will be interpolated between two bones, which can lead to bad visual results such as “jumping” objects due to interpolation errors. Although, whereas many joints will provide greater customization possibilities, if it is overdone it will slow down the system, particularly when several characters are displayed **[12].** So when creating a character generator, a balance between customization and execution speed must be found.

We saw that motion retargetting permit to adapt an animation to skeleton with different proportion. But this method can't be applied if skeleton have a completely different morphology (a different number of limbs). In the Spore game, a new motion retargetting algorithm has been developed to generate animations at run-time in order to fit an user generated creature which can have any morphology **[6]**.

To do this, the animation tool uses a generalized and a specialized space with a function permitting to move from one to the other. The animator first create the animation for a given creature in its specialized space before this animation is transposed to a general space where it can be transposed to a specialized space again at run-time. When animating, the animator can select body part by query as each limbs have tags like "grabber". Those tags are reused at run-time by the AI system. Indeed, with traditional skeleton animation, joint name are known but in this case, the skeleton is

unknown at run time. So if the creature has to grab an object, the algorithm will query all limbs having the "grabber" tag and then will use the one closest to the target. The result is very impressive as the creature moves immediately after having been given news body parts. Not only the movement is immediate, it is also realistic whatever morphology is given by the user.

Now that we saw all the advantages brought by the use of rigged model, we will try to integrate a character generator using 3D rigged model into the <e-Adventure> platform, which currently use 2D sprite to generate character.

Chapter 3 The <e-Adventure> platform character generator

3.1. The <e-Adventure> platform

It is often hard for a teacher to catch students' attention during lectures. But the same students who seem unable to concentrate on lectures can display very high concentration capability when playing video games. Indeed, a student can often think that the taught material in a lecture won't be of any use to him or her. Conversely, a game player will often pursue the game objectives without reluctance. Also, the player can often lose sense of time and be so focused on the presented game. Additionally, the adaptive difficulty level found in most video game can be useful for education purposes **[13]**. So if teachers are able to make their own video game and use it as teaching material, they will be able to transmit their knowledge to even the most unmotivated students. But creating video games require programming skills that a teacher can't afford to acquire. It's with the objective to solve this problem that the <e-Adventure> platform has been developed. The <e-Adventure> platform is a research project aiming to facilitate the creation of educational games. It is being developed by the <e-UCM> e-learning research group **[14]** at Universidad Complutense de Madrid. The <e-Adventure> platform provides a user friendly interface to easily create Point & Click educational games.

The <e-Adventure> platform is open-source and cross-platform as it is written in JAVA. It can be deployed as a standalone application but the use of JAVA also permits applets to be deployed for online education. Hence the game can be played via a web browser only requiring the user to have JAVA installed on their computer.

The <e-Adventure> platform allows the creation of Point & Click games. This game format is often used for education for two main reasons. First, it only requires a mouse to be played, optionally the keyboard can be used to input text. This results in simple interfaces quickly understandable. Indeed, the player only has to place his or her cursor on an object of interest and click on it to interact. In most of point & click games, there are only two types of action possible: examine and use (or talk if it is a character), each action being affected to one mouse button. This simple interface makes the game accessible to everyone, even those with no gaming experiences.

The second main reason why point & click game style is often used for educational games is that these kind of games do not require a lot of resources to be created. There is no need to model complex 3D environment nor 3D animated characters. In a point & click game, the environment can be simply composed of still 2D picture. The need for animation can be removed by using text describing the ongoing event or story. Therefore these games are very easy to create even for people having no experiences in video game creation. The only remaining barrier is to master a programming language which is removed by the e-Adventure platform game editor.

As a platform designed for education, the e-adventure platform features a built-in assessment mechanism. The assessment system can check if the player has performed some particular action and how much time he spends on it. It then computes the result to give feedback on the player's performance. The player's performance can be then reused by other learning software. Indeed, the e-adventure platform has been developed to generate games that can be integrated easily into learning management systems as they are packaged with meta-data in accordance with the IEEE Learning Object Meta-data standard **[15]** which defines the description of learning object to facilitate their exchange, evaluation and management

The e-adventure platform game editor **(fig. 3.1)** offers all the standard features of traditional point & click game. The user can create a scene by simply choosing a background image then select interactive areas. An interactive area is an area of the screen that will trigger an event when clicked. It can be an examinable area, a pickable item, a way to another scene, etc. The editor provides an inventory system where the objects picked by the player are stored. The editor features a special kind of item, notebooks. This feature is really important for educational game as it can force the player to read by putting information required to progress in the game in one of those notebooks.

The editor also provides an interface to create dialogue. Dialogue is created using a graph structure where the player can choose his next line.

When creating a new project, two styles of game will be proposed to the user: a first person game, where the action is seen through the eye of the controlled character; or a third person game, where the player can see his character. In the first case, there is no need to create a character – or avatar - to represent the player as the player character won't appear on the screen. Also, creating an animated representation of non-playable character is not necessary as in the first person game still image is acceptable. But for the third person game, an animated representation of the player and non-playable is required as people are used to animated character in this kind of game. Additionally, even if still image is acceptable in first person game, a game will be more appealing if it features animated character. This is when the need for a character avatar generator appears.

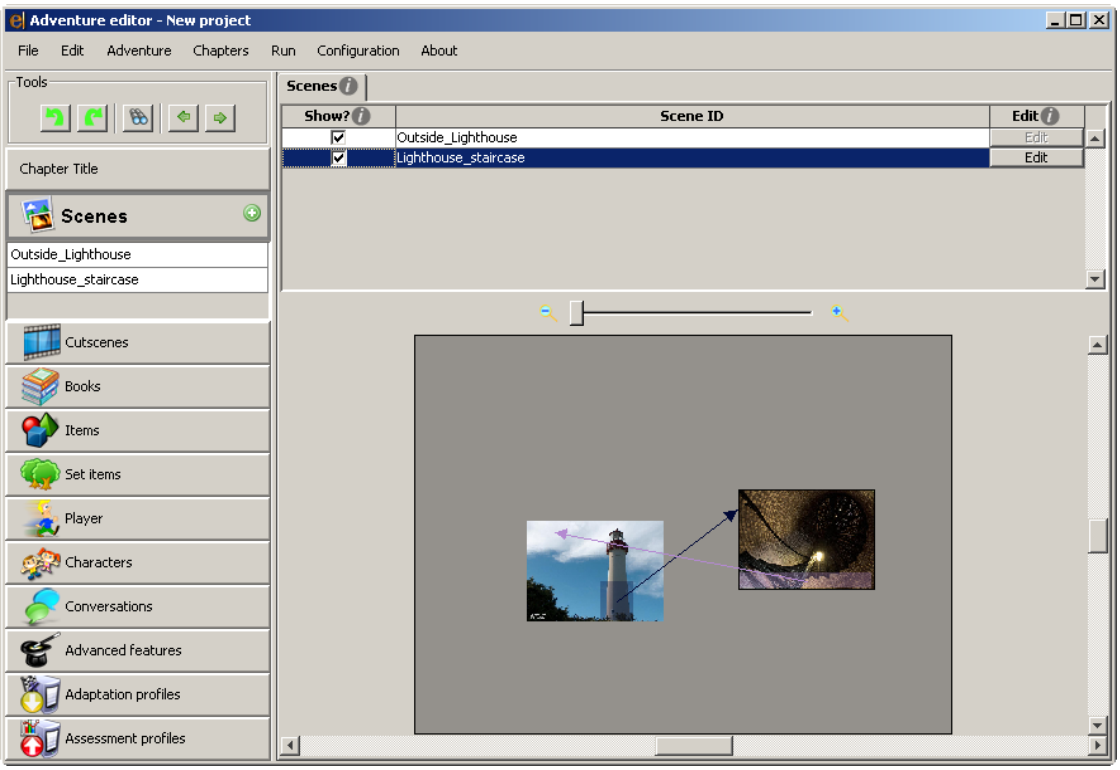


Figure 3.1 Screenshot of the e-adventure game editor. The editor is currently in the scene editor in which the user can see how the different scenes (areas) of the game are linked together

3.2. Generating avatars

The <e-Adventure> engine relies on 2D sprites to display animated characters. To create a new character, the game creator must provide a series of picture files for each animation (standing, running, talking, etc...). On top of that, they must also provide back, front, left and right views. Even though the use of 2D characters instead of 3D greatly reduces the amount of work required from the game creator by removing the need of mesh modeling, rigging and texture mapping, it is still a lot of work. It obviously requires the game creator to have artistic talent that many people do not possess. Even with artistic skills, they would have to produce numerous pictures, which is very time consuming. Another approach would be to take picture of a real person instead of drawing a character. This does not require artistic skill, but this approach is still hard to carry out. Indeed, it requires taking a series of photography for each animation and editing those pictures individually to remove the background. Furthermore, to get a animation of someone walking which looks natural, it require equipment such as treadmill.

For these reasons, a character generator is needed here to allow any game creators to easily create a character without having to be an expert in character design. Although creating a 2D character is faster, a 2D character is less flexible than a 3D character due to its lack of an internal skeleton structure. Conversely, We saw in previous section that a skeleton offers more flexibility to a virtual character as it allows easy animation change, easy modification of a character morphology by manipulating bone position and scale and allows accessories to be attached easily on a character.

Several 2D character generator applications programmed in flash can be found on the internet **(e.g, fig 2.2)**. These generator applications simply superimpose several layers of images to "dress" the character. Although good quality results can be achieved, this approach is not well suited to the requirements of this project. The output of these avatar generator applications is a picture of the character or a flash animation, but this animation will only focus on a small part of the character like eye blinking or hair flowing. Furthermore, the output generated only shows one side of the character. To have an output showing every side of the character, each resource must be drawn for each side. Also as these resources are only pictures, they can't be animated to follow body movement which would be easily achieved with a rigged 3D model.

We saw that Rigged 3D characters offer more flexibility and so are more suitable for an avatar generator than the 2D approach if we want fully animated character. However, the <e-

adventure> editor requires image files for character creation. So the idea investigated in this project for the character generator is to customize a 3D character using an interface similar to the one used in most video games avatar generator and then render the animation to picture files. These pictures will be packed into an archive format reusable by the <e-adventure> game editors.

Figure 3.2 Example of a flash character generator

3.2.1 Requirements

The main requirement for the character generator is to be simple to use with a clear interface. Although current technologies allow avatar generators to offer great control over the character appearance as we saw in previous section, the <e-adventure> platform is intended to be used by people who probably do not have gaming experiences and have never been confronted to a character generator. They would likely be confused if confronted with an authoring interface containing many “sliders” and other widgets as the one from Eve Online **(fig.1.2)** where even small detail like chin shape and distance between the eyes can be customized.

Therefore, the character generator should only allow minimal control of the character appearance. The requirement for these control are the following:

- First, to allow the game creator to choose between a selection of pre-generated models for different body parts and clothes (face, hair, torso, leg).
- Then to be able to choose a skin tone and change clothes and hair colour.
- Finally, allow the game creator to change the size of the character. Changing the age of the

character can also be achieved by modifying the relative proportions of the different body parts. Indeed, children have a bigger head compared to their body.

3.2.2 Irrlicht

To be integrated into the <e-adventure> platform, the character generator has to be cross-platform and easy to integrate. The Irrlicht game engine **[16]** satisfies these requirements and has been chosen for our purposes. Irrlicht is open-source, cross-platform and relatively light weight. It is one of the most widely used open-source game engines after Ogre **[17]**. Although Irrlicht offers less possibilities than Ogre, it is simpler to use and to integrate in other applications and offers the features necessary for a simple avatar generator. Indeed it can handle skeletal animation which is the minimal requirement for the character generator needed here.

3.2.3. Implementation

As with the majority of game engines, Irrlicht uses a hierarchy of nodes to handle graphical objects (e.g., meshes, lights, camera, etc..). Irrlicht allows the manipulation of a rigged mesh via the use of the `IAnimatedMeshSceneNode` class. This class not only contains a mesh object but also a collection of bones, each bone being a node, which allows objects to be easily attached to a character. The initial idea was to add to the scene an Irrlicht animated mesh node containing a skeleton but with no geometry, thus having no visual display before attaching the different body parts selected by the user to this skeleton. These body parts would be added to the scene still using `IanimatedMeshSceneNode, with both` geometry and texture **(fig.3.3)**. The skeleton contained in the nodes would be sub-parts of the main skeleton, for example only the neck and head bone for the head model. As bones are also nodes, it is possible to make them children of other bones. A child inherits all the transformations of its parent node. Therefore, if each bone is attached to his correspondent bone in the main skeleton, the skeleton of the body part will have exactly the same animation as the main skeleton. All independent body parts will move synchronously, conveying the impression of a one piece character.

Unfortunately, problems arose when trying to apply this idea. When defining manually the bones positions of an animated mesh node that had been created by importing a directX model, the calculation of the mesh vertices position is corrupted resulting in the mesh being torn apart.

This problem only occurs when importing DirectX model into Irrlicht. Some tests have been done using the B3D format, another rigged mesh format readable by Irrlicht. When using this format we obtained the expected result. Thus it proved that the problem came from a bad handling of DirectX format by Irrlicht. Unfortunately, the library of models generated for the <e-Adventure> platform have been generated using Blender, and currently DirectX is the only format which can be exported by Blender and imported into Irrlicht.

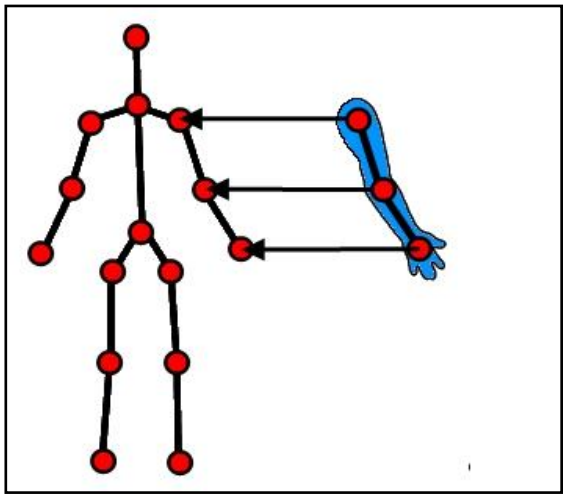


Figure 3.3 Each bone of the rigged mesh is attached to the matching bone in the reference skeleton

The solution found to get around this problem was to use the `UseAnimationFrom` function from the `IanimatedMeshSceneNode` class, which takes as an argument a pointer to another `IanimatedMeshSceneNode`. `It` reads the animation -the position of bones- from the given node, and copies this animation to its own skeleton. By using this function, the result is the same as the first method. Nevertheless, this method does have a drawback. The prerequisite for this function to work correctly is that all bones from the source skeleton have a match for the target one. So to generate a body part model that can be animated using a reference skeleton, this body part must have the same skeleton, i.e. a complete skeleton**(fig.3.4)**. This means that in the case of a torso model, the torso mesh will have a complete skeleton even if most of the bones of this skeleton are not associated with any vertex. This implementation would have been problematic in an application such as a video game as it would result in numerous copies of a similar skeleton being stored in memory for each character displayed. However, in our case, there will be only one character displayed. So here this is acceptable until an update in the Irrlicht engine code solves the vertex position corruption problem.

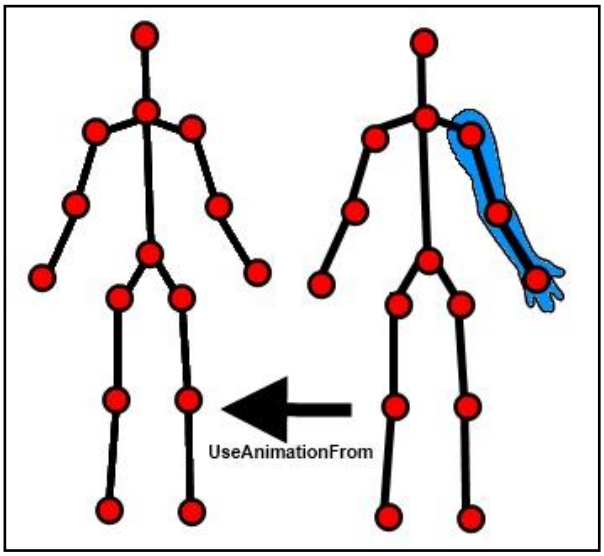


Figure 3.4 Each mesh must have a complete skeleton that is animated by copying the animation from the reference skeleton

Swapping meshes is a method to completely change a part of a character but there is a simpler solution. By only changing the texture, the character can be given a completely different appearance. For example if we have a mesh representing a long coat, this mesh can represent a scientific white coat or a detective's long coat by simply changing the texture being used.

This implementation did not pose any difficulty as it simply needs to use the node material function `setTexture`.

Texture manipulation is also suitable to change the skin tone or the colour of clothes. The simplest solution here would be to just change the material property of the geometry being rendered. However, this would change the colour for the complete mesh. In a character generator, more control over colour change is desirable. Indeed, if we want to change the skin tone, we must be able to do it even if body and clothes are one same mesh. Similarly, we may want to change only the colour of a logo on the avatar's clothes but not the rest of the clothing.

A shader program using a mask has been used to implement this required functionality. The shader program will read mask images **(fig.3.5)**. If the selected pixel in the mask images has an alpha channel different than zero, then the pixel hue is changed to the colour associated with the mask, chosen by the user via a RGB slider or preselected colour button. If the alpha channel is null, the colour is not modified.

The shader program calculates the final colour by converting the initial colour from RGB to HSV space. Then it changes the hue to the hue of the selected colour and converts this value back to RGB. This allows for modification of colour tone without affecting the lightness. One drawback is that if the initial colour is white or black, it won't be affected by change of the hue.

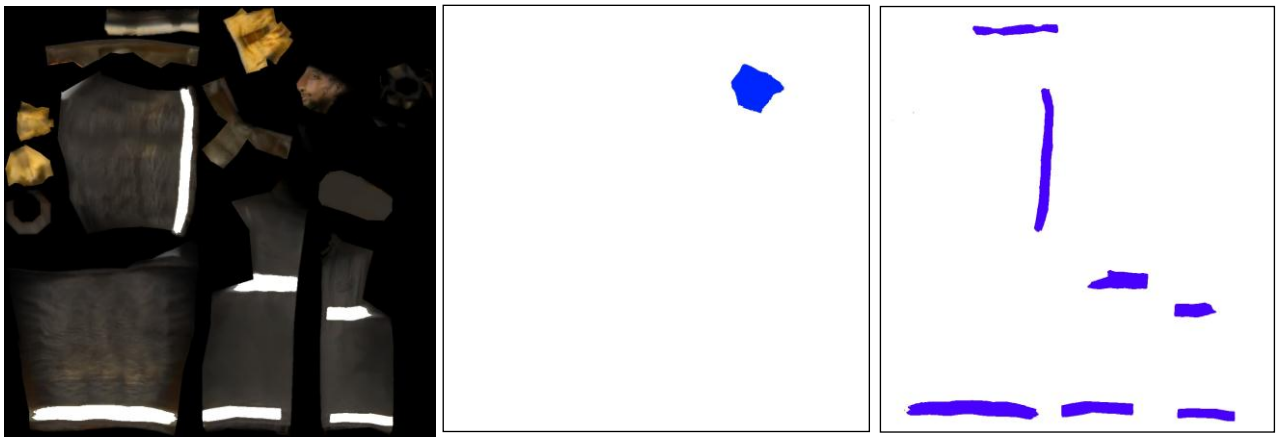


Figure 3.5 An example of a texture and the corresponding mask. The mask on the middle picture is used to modify the head skin tone while the mask on the last picture is used to modify the colour of the stripe on the character clothes

When the user is satisfied with the character being created, he or she can export it into a picture file. This is done by using the render to texture Irrlicht feature. For every frame of the character animation, the character is rendered to a texture from four camera positions (front, back, left, right). Then this texture is saved to an image file. Currently, the program just generates the image files but in a future version it will pack those images into an archive format that then can be directly read by the game editor, which will create the character from those picture automatically.

3.3 Summary

Using 3D rigged meshes instead of 2D sprite allow for more flexibility and ease in character creation for people with no experiences in character design which is very important for the <e-Adventure> platform as its objective is to allow people with no programming or artistic skill to create video game.

However, it can be seen as a drawback that 3D model are converted to picture files. Indeed, after this conversion, we lose one of the advantages of rigged model which allows for animation modification at run-time. To allow this, one approach could be to use a skeletal animation system adapted to 2D. Mark Ten Bosh have developed such a system along with a character editor **[18]**. However, this system is good when you only see characters from side, like in fighting game or 2D platform games. But it will cause problem if the character has to be seen from front or back. This remark is for information purpose only as in the next chapters we will focus on other problems.

The work that was presented in this chapter has been integrated into the production version of the <e-Adventure> software and is going to be used on the European project that is developing this educational platform. Thought with the developed character generator we can offer users a large library of model, the user will still be limited to what is offered to him. However, we saw that an approach to produce 2D sprite was to take photography of a real person. The use of 2D sprite offers freedom to the user but only if he got the required skills. However, our approach allows users with no artistic or computing skill to easily create character but at the cost of this freedom. It would be a great breakthrough into customization if a similar approach could be done with 3D character instead of 2D spite. Hopefully, with the latest innovations in games hardware, in particular the Microsoft Kinect, comes the possibility to create more sophisticated 3D avatars that are based on the actual game player, in an in-expensive and easily accessible manner. This is explored in the next chapter.

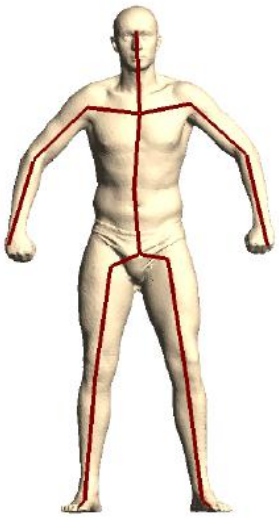
Chapter 4 Avatar Creation using Microsoft's Kinect as a 3D Scanner

In this section, we will investigate a way to allow users to import 3D custom model into a game easily. The first problem we will encounter is model without or with incompatible skeleton. Thus first, we will investigate auto-rigging solution. To customize in-game material, some games allow users to import their own texture into the game. Indeed, it requires users to simply scan a picture. So we can imagine the same process but with 3D data. Thus we will focus on 3D scanner, in particular the Microsoft Kinect. Before attempting any model creation from Kinect, we will first explore the possibilities available for rigging a 3D model.

4.1. Auto-rigging

We saw in previous chapter the importance of skeleton for virtual character. With the <e-adventure> platform, we considered the case of a character generator offering a library of already rigged meshes. But if we consider a character generator where users can import their own 3D models, imported models will probably have a skeleton with bone having different name from the one used in the generator or it can have no skeleton at all. In that case, those custom meshes would be of no use except with an auto-rigging system. Rigging a mesh is a arduous and time-consuming task. Unfortunately, it has to be done to have a complete control on a character animation. We saw in previous chapter that a major advantage of skeletal animation is that an animation can be reused for several characters, sometimes requiring motion retargetting, which is not an issue as this algorithms is now fast and give good result. However, this advantage can be used only if the model possesses a skeleton and that skeleton has the same structure. It poses a barrier when one wants to reuse a character model, even if it already possesses a skeleton but with different structure. Thus, to ease reuse and remixing of character model, a lot of researches have been done into automatic mesh rigging. Baran et al.**[19],** Pan et al. **[20],** M Poirier et al. **[21]** have developed such algorithm. We focused on Baran et al. Algorithms as they provide their software and its source code for download. Their Pinocchio software is able to rig a character mesh in approximately one minute on a mid-range computer. The software gives very good result on well-proportioned human shape (**fig.4.1**). When attempting to rig humanoid with wrong proportion, the software misplaces some bones but the overall result is acceptable except in extreme cases like when hair is bigger than arms (**fig. 4.2**)

so the algorithm will rig hair as arms. The software is not limited to humanoid skeleton and allows for input of custom skeleton file. Thus quadruped animal can also be rigged in addition to fictional creatures such as centaur, a human torso on a horse body. The limitation of the algorithms is that it generates simple skeleton with no complex hands. Also, the algorithm requires meshes to be closed. So meshes must be processed to close all the holes before attempting a rigging.



· ***Figure 4.1*** Example of a mesh rigged with the Pinocchio software (Picture from **[19]**)

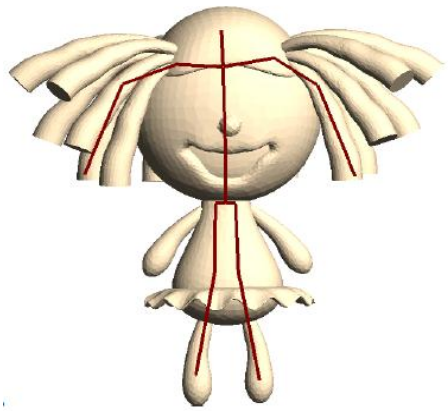


Figure 4.2 missrigging due to disproportioned character in Pinocchio software (Picture from **[19]**)

Auto-rigging algorithms broaden the possibility for character generator as they permit users to import their own 3D model. But here it requires the users to have some 3D modeling skills. In order to allow the player to integrate himself in games, the best way would be first to allow the user to capture 3D data himself and then feed it in the character generator. To do so, We need to find a way to capture geometry and color data using cheap and easy to deploy device.

4.2. 3D scanner

We saw that there are two ways to generate virtual character if one does not want to manually modelize it. The first one is the one used in video game where the player use an editor to modify a character appearance using per-generated meshes library, and the second one is the virtual character that is created by acquiring geometric data from the real world. The second case would permit players to use an avatar which is a perfect virtual reproduction of himself instead of trying to put together elements that best match his appearance from a library of pre-generated meshes and texture. Unfortunately, acquiring this kind of data requires expensive and hard to deploy equipments like laser scanner or multiple-camera capture system **[2], [3]**. Such equipments can be available for companies but not someone such as an individual player. A summary of the evolution of 3D scanner can be found in **[22]**.

3D scanners are used in various fields such as Construction industry and Civil engineering, Medicine, Cultural heritage, Entertainment. 3D scanners are used in movie and video game industries. In the first one it is used to create a double of actor. In the second one, it is used in video games adapted from a movie or in sport video games where the virtual ego of the actor or sport-person is created by scanning the real person. They can also be motion captured to make his virtual ego closer to the original. In this project, we want to offer players the same above possibilities.

4.3. Kinect

4.3.1 Kinect Presentation

So we need a system capable of capturing 3D data which can be easily set up in a small room and is relatively cheap. The perfect candidate for this could be the Microsoft Kinect **[23]**, an motion sensing input device for the XBOX. The Kinect is Microsoft response to Nintento Wiimote and Sony Playstation Move, which are also gesture interface for concurrent video game consoles. The Kinect is designed to be a touch-less interface which allows users to control the XBOX using gesture and voice command. It allows the users to navigate through menu by “touching” them as the Kinect is able to track user hands. The Kinect is not limited to hand tracking as it provides full-body 3D motion capture, facial recognition and voice recognition capabilities. So it allows for the

control of a virtual character by moving it accordingly to user movement via it's motion capture capabilities. The Kinect has been made compatible only with the XBOX 360 but numerous programmer understood that the Kinect could be more than a video game console accessory as the Kinect functionalities are supported by a fast and powerful 3D scanner.

4.3.2 Kinect hacks

At the time of the Kinect release, Microsoft did not provide any SDK to allow developers to use the Kinect capabilities and wasn't planning to release any as they wanted the Kinect to stay an XBOX 360 exclusivity. However, programmers soon realized that the Kinect can be used for more purposes than it was initially designed for. Adafruit Industries , an open-source hardware company, even launched a race to hack the Kinect by offering a money price to whoever would produce an open source driver for the Kinect **[24]**. Thus, the Kinect have been successfully hacked through reverse engineering on November 10 2010, only 6 days after its US release on the 4 November by Héctor Martín who developed a Linux driver allowing access to the Kinect RGB camera and depth sensor called libfreenect. This driver have been given to the OpenKinect project **[25]**, a community created to facilitate exchange of idea or code about the Kinect. Numerous examples of applications using the Kinect can be found on their website.

the libfreenect software have since been extended to be cross-platform and now provides an API allowing to communicate with the Kinect. These API have wrapping to several language including C++, JAVA, MATLAB, Python Ruby etc.. However, this software only provide low level interaction with the Kinect which means that functionalities such as body motion capture have to be developed by the users. A major inconvenient of this software is to require the user to calibrate his Kinect, which requires to print a chessboard pattern. Without this calibration, the color map and depth map will be misaligned.

In December 2010, PrimeSence, the company which designed the depth sensor used by the Kinect, released their own open-source drivers along with a motion tracking middleware called NITE **[26]**.

NITE is used by OpenNI **[27]**, an industry-led, not-for-profit organization whose goal is to improve interoperability of Natural Interaction devices, applications and middleware, one of this organization’s main member being PrimeSence. OpenNI, which stands for *Open Natural*

Interaction, provides open-source APIs for Natural Interaction devices, including the Kinect. Like libfreenect these APIs cover low level communication with the Kinect , not only allowing access to color and depth map, but also providing high level middleware solution for Body Motion Tracking , hand gesture recognition and voice recognition. The main advantage of the OpenNi API over libfreenect is to provide access to the Kinect factory calibration removing the need to calibrate it.

Thus for the above reasons, we will use OpenNI API in this project. We will investigate if the feature provided by the NITE middleware could be of any use in this project.

One feature which could be of great help in his project is the body motion capture capability. Indeed, a skeleton reproducing the user movement can be generated. Three use can be considered for this feature. First, it can be used as an alternative of the auto-rigging software discussed previously. Also, the joints of the generated skeleton can be used as markers for the alignment problem that we will discuss later. Lastly, if we successfully generate a 3D rigged model from Kinect scan, to improve the similarity between the real model and the virtual avatar, the Kinect can be used to do motion capture as it was what it has been designed for.

Another useful feature is the user detection and segmentation. A user map can be generated which indicates for each point of the depth and colour map if it corresponds to an user. In that case it will give it's ID. If it corresponds to the background, the value will be zero. This feature will be very useful when capturing people as it allows for keeping only the scanned person geometry without capturing the background. As this operation is performed only in two dimensions, the depth map, it is much faster than if we process the 3D cloud afterward to remove the background.

Now we will look closer at the Kinect 3D scanner capabilities to see if it can be used for our purpose, which is to obtain a 3D model of a person with fine details.

4.3.3 The Kinect 3D scanner

Microsoft who was planning to release a touch-less interface did not realize that they developed a powerful, cheap and easy-to-use 3D scanner. The Kinect can also be considered as a 3D camera due to its fast processing speed. This shouldn't be mistaken with stereoscopic 3D camera which only takes two pictures with a slight different angle to provide depth impression. The Kinect is a real 3D camera as every point of the "picture" generated by the Kinect contains not only colour information but also depth. So using some projection calculation, it is easy to get a 3D version of this picture. To do this 3D reconstruction, the Kinect rely on a range camera which measures depth using infrared light. An infrared laser projector projects a grid of laser point which is then captured back by a monochrome CMOS sensor. The depth is calculated using the time elapsed between emission and reception. This type of scanner is called a time-of-flight 3D scanner. The advantage of this type of scanner is that they can scan a scene instantly as they can calculate the distance of approximately 10,000~100,000 points per second. The resolution of the map generated by the Kinect being 640*480 point, which represent 307200 point, the Kinect can deliver colour map and depth map at a rate of 30Hz which means that the Kinect is able to deliver "3D movie". Thus, when used as a scanner, a scene can by captured in 3D in a few second by moving the Kinect around. In that aspect, the Kinect is better than some of the professional 3D scanner we saw earlier as some of them need to calculate each point one by one and so demand more than one minute for one frame. Of course, to attain this processing speed, some sacrifices have to be made. The depth map generated by the Kinect is noisier that the one obtained with these professional laser scanners. Thus, a smoothing has to be done on the data delivered by the Kinect to obtain smooth surface.

The quality of the point cloud greatly depends on the distance between the Kinect and the captured point. Within short range, between approximately 50 centimeters and 100 centimeters, the Kinect is able to deliver point cloud with very fine details, no as fine as professional scanner of course but good enough for our purpose. But when points are above 100 centimeters, the data obtained become noisier and thus requires some further processing. Also, as you go farther from the sensor, the distance between each Z plane becomes greater creating aliased surface.

Y Cui et al. **[28]** And D. Striker et al. **[29]** have attempted 3D scanning with a Time-of-flight scanner, the latter using a Kinect. They reduced the noise by averaging several depth maps. An alternative method to reduce noise is to use the Moving least squares **[30]** algorithms to smooth the

surface. This is the approach used on the following example of captured point clouds.

On the following picture (**fig.4.3**), we can see that the Kinect generates a point cloud with a high level of detail. However, an aura phenomenon can be noticed is the border of scanned objects(**fig.4.4**). This aura is due to a miscalculation of the depth at object border. The misplaced points are called veil points **[31]** and are a typical phenomenon in data obtained from 3D scanner. This phenomenon generates colour but also geometry distortion in the data.

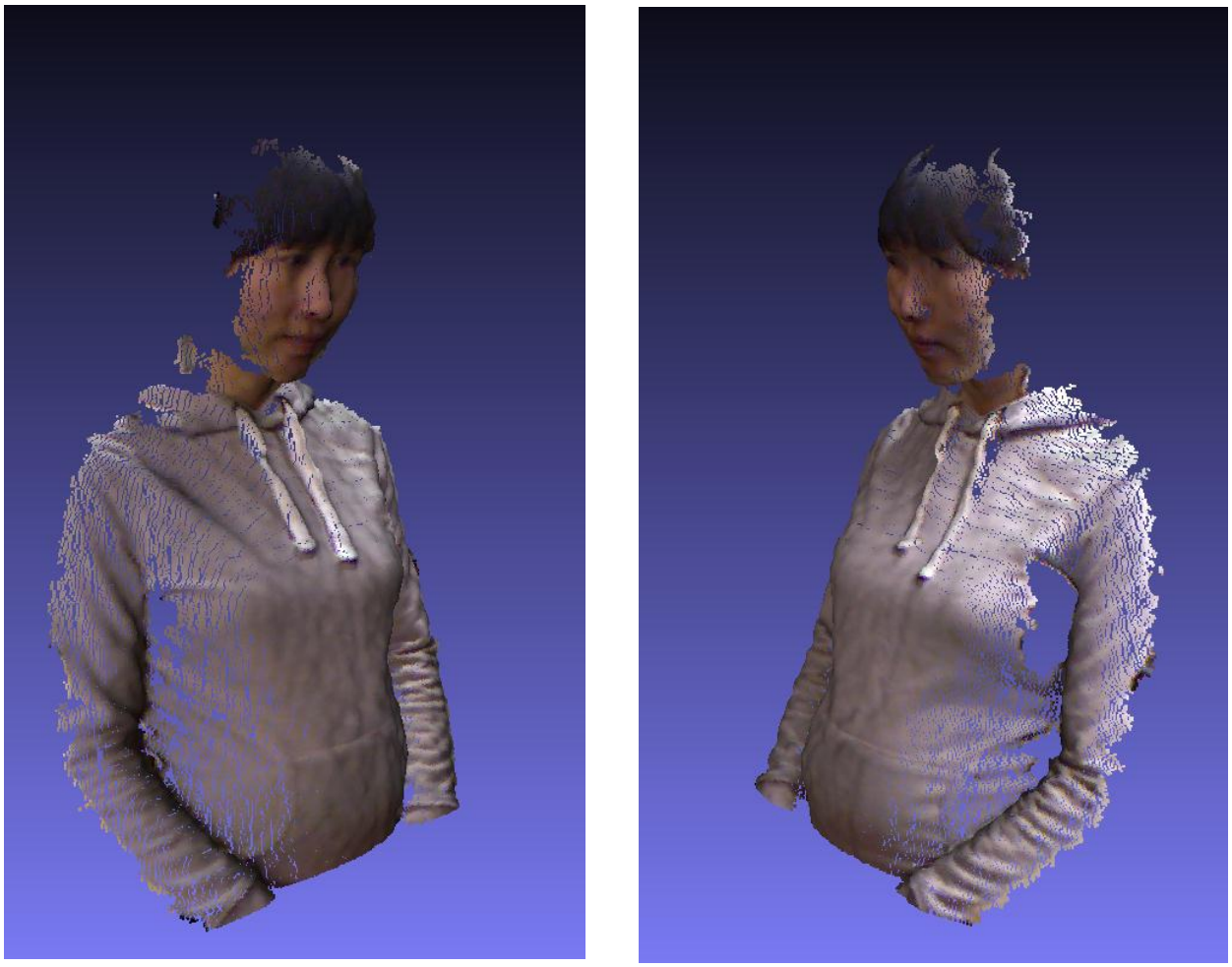


Figure 4.3 : Example of a point cloud obtained with the Kinect. This point cloud has been acquired in a range below 100 cm.

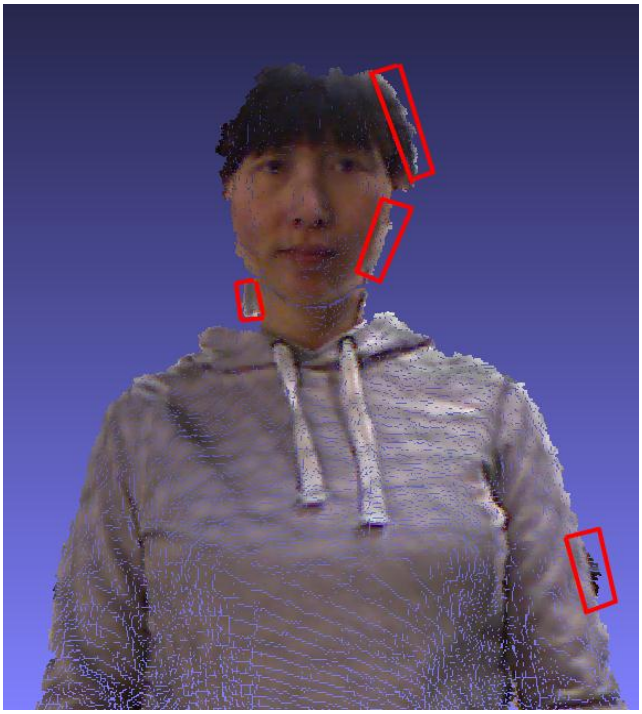


Figure 4.4 The red mark show area where veil points are appearing, causing colour and geometry distortion

Thus, although the data obtained is not perfect, the Kinect solves the problem of the acquisition of geometric and colour data from a player as it can capture fine detail. Furthermore, the Kinect being a Xbox 360 accessory, it is available in commerce for players at a price of approximately 150$, making it available for a large public. It is also easy to use as the Kinect does not need any configuration step and can be used to capture 3D data immediately and it does not require special environment, for example, uniform colored background often used when scanning real object.

The Kinect solves the hardware part of the problem. But the data captured by the Kinect still need to be processed to generate fully rigged character. As we said, the Kinect can be seen as a 3D camera. When taking a picture with a standard camera, you can only see one side of the object on the picture and object in front will occlude object behind them. The same phenomenon appears with 3D scanner. You will get your object in 3D but only one side of it. Furthermore, the front object will create shadow hole in background object. To capture a complete object, several captures must be done at different angles which can be done quickly and easily by hand-holding the Kinect around thanks to its processing speed.

But those captures won't be aligned to each other as each of them would have been captured

from a different position with a different angle. So we need to align each capture which will be the main challenge of this project.

A similar project has been introduced by Weiss et al.**[32]**. Like us, their goal was to generate a 3D model of a person using the Kinect. However, their goal was to generate a 3D model whose body have the same proportions as the captured person but not capturing fine details. Thus to solve the alignment problem, they used a SCAPE model **[33]**, a parametric model of human body learned from a database of several thousand laser scans. After a pose estimation, the model is deformed to fit the Kinect data. This approach permits to solve non-rigid transformation problem as the scanned person can't retain the same exact pose when he is required to turn around for the purpose of a full body capture. But this inevitably sacrifices the level of detail offered by the Kinect. Therefore, this approach can't be used for our purpose as we want to generate model being a virtual copy of the user. We can't afford to lose fine detail. In the next section we will explore into different methods for point cloud alignment.

4.4. Avatar creation with the Kinect

4.4.1 Capture method

When scanning someone, first issue we are confronted with is that we have to capture a person from several angles. An approach to this problem could be to use multiple Kinect to get a full 3D capture. However, this solution will obviously multiply the cost, and will be more complicated to set up as each Kinect can't be placed arbitrary and it will require much more space. This solution would allow us to have a complete scan quickly but at the cost of the ease of use of the Kinect. This approach will not be conceivable if this project is aimed for player as it is complicated in set-up and we can’t expect them to buy several Kinect.

Another simple approach could be to have the person turning himself/herself in front of the Kinect, which is done in **[32].** However, In this case, the scanned person will probably not position himself/herself in the exact same position as in the previous scans, creating non-rigid transformation that can't be handle by registration algorithms such as ICP. We saw that non-rigid registration can be done using SCAPE model but at the price of the scan details. Thus, if we decide to move the person instead of the scanner, we have to use something like a turning table.

On the contrary, we can choose to move the scanner around the scanned person. The advantages of this approach are that it requires no devices such as turning table, but only requires

some space. But it can be hard to perform as you can't always saw the Kinect output when scanning this way and so, can't know if you are correctly aligned and if you put the Kinect at the correct distance, as the Kinect can't see object closer than 50 cm. In addition, moving the Kinect will generate change in the lightning condition and thus can cause great differences in colour.

Whether we shall move the person or the scanner will depend on the registration method used. Indeed, in next chapter we will saw registration methods that make use of the background, thus making the approach, with which we turn the scanned person, inappropriate. As at first we will attempt a manual alignment, we will use the approach where we turn the person by using a turnable chair for the capture. We will talk more about alignment in the next section.

4.4.2 Alignment

The first challenge we encounter when working with 3D scanner is to align the obtained point cloud. This procedure is also called registration. As registration of point cloud has been a common problem in computer vision, a lot of researches have been done about it. When obtaining data from scanner, each data set will have a different reference frame, in other word, two points corresponding to the same point in the real space will have different coordinates. Thus, the goal of registration is to estimate the transformation between the different data-set. In the field of registration, The most popular algorithms is the Iterative Closest Point algorithm or ICP, which was introduced by Besl **[34]** and Zhang **[35]**. The ICP algorithm is simple and only consists of three steps.

Considering two point clouds which overlap each other, one is referred as the source and the other one as the target. We want to compute the transformation which put the source cloud into the same reference frame as the target cloud.

First, for each point in the source cloud, the nearest neighbor in the target cloud is computed. Secondly, the transformation that transforms all points into its nearest neighbor is estimated by using a mean square function.

Thirdly, the source point cloud is transformed using the estimated transformation.

These three step are reiterated until one of the stopping conditions is satisfied which can be a

maximum number of iteration or a minimal error rate calculated by measuring the distance to nearest neighbor for each point. The algorithm will converge to a local minimum of the calculated error rate. As a result, ICP will only give a good registration if the transformation between the two point cloud is not too important. So when ICP is used, it is often used in combination of a coarse registration, where it will refine this coarse registration. This initial alignment can be done manually or by placing marker on corresponding point onto the two point cloud, then estimating a transformation between those markers. Later we will explore more approach for coarse alignment but for now we will simply check if good result can be obtained with manual alignment applied on the Kinect data.

4.4.3 Test with manual alignment

Before trying to develop a program that can align automatically several point cloud from the Kinect without any human intervention, first it must be verified if a satisfying 3D mesh can be obtain by aligning those point semi-automatically, which means by first roughly aligning the point clouds manually using a 3D editing software and then, refining this alignment by using ICP as discussed in the previous section. To do this, we choose to use MeshLab **[36]**, an open-source cross platform mesh processing software which was designed to process data obtained from 3D scanner. MeshLab has been chosen as it's a free software and has been used in various academic and research contexts. It features various tools to edit, repair meshes. One of the tools we are interested in is the alignment tool which uses the ICP algorithms. MeshLab also features surface reconstruction tools that we will need later.

We attempted to fully capture a person's body with the Kinect. To do so, the person needs to stand on a turnable chair and should be rotated by 90 degree after a series of four captures done at different heights.

We save the captured point cloud to the PLY format (Polygon File Format) to import it into MesLab**.** The PLY format is very simple and has been designed to store data from 3D scanner. It has been designed in the Stanford graphics lab and is based on the Wafront OBJ format. This format wasn't usable to store 3D scanner data as it does not allow storage of point cloud due to the need of face definition. The PLY format has been used by Stanford University for The Digital Michelangelo Project **[37]** ,which goal is to produce digital version of sculptures and architecture of Michelangelo.

Before attempting any alignment, the captured point clouds are smoothed using the least moving square implementation of PCL **[38]** to remove noisy data and to obtain smooth surface. We will discuss about PCL in next chapter. Point which does not belong to the model body but to its surrounding have been removed manually in MeshLab.

To proceed to a rough alignment, MeshLab offers a point-based alignment tool. The users place markers onto the two point clouds. Then, a rough transformation is calculated from those markers. We used this tool to get a coarse alignment. After this rough manual alignment, the ICP implemented into MeshLab managed to give a satisfying fine alignment of the Kinect point cloud (**fig.4.5**). It can be noticed that sharp differences between colour from different point clouds appear due to changing lightning condition as we move the model and not the scanner. Also, it can be obviously observed that the veil points create bright lines at one point cloud border.

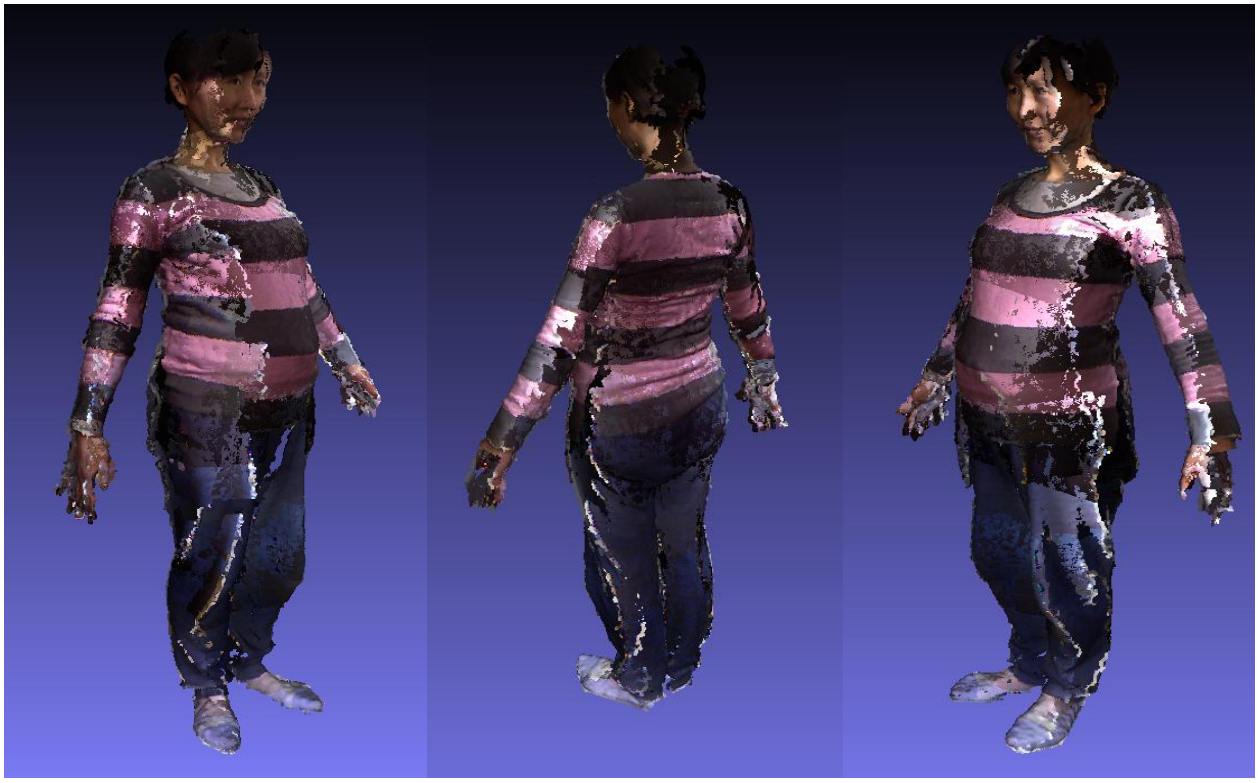


Figure 4.5: All captured point cloud after semi-Manual Alignment in MeshLab

Now that we have a complete point cloud of the model, the next step it to turn it into a triangle mesh. To do this we will use the Poisson surface reconstruction **[39]** implemented into MeshLab. The Poisson surface reconstruction is perfect in our case as it has been designed to handle noisy point cloud obtained through 3D scanner. Other algorithms, like the Ball Pivoting Algorithm, developed by Bernardini et al. **[40]**, which have also been implemented in MeshLab, will output a mesh containing all the points from the input point cloud linked together with triangles. However, the triangle mesh produced by the Poisson surface reconstruction does not contains the same points as the input point cloud due to the fact that the algorithm generates a surface that "wrap" the points but not compulsorily contains them as a vertex. Thus this algorithm can generate mesh with smooth surface even with noisy point cloud and minimize the creation of holes even if some parts of the surface are missing in the point cloud. In addition to not being esthetic, meshes containing holes would prevent us to use the Auto-rigging software developed by Baran et al. as it requires meshes to be closed. The Poisson surface reconstruction solves this problem by always considering all the points at every stage of the algorithm instead of a local neighbourhood. This surface reconstruction method also permits to avoid the generation of doubled surfaces that occur when point clouds are not perfectly aligned.

The result can be seen on the following picture (**fig.4.6**). The surface reconstruction is able to produce a high detail mesh. The mesh does not have any holes in its structure even with the presence of important holes in the points cloud due to occlusion **(fig.4.7)**. It also handles misaligned parts, such as the back of the head. The hands of the model are very badly modeled but it is due to the bad quality of point cloud around the hand area and not the surface reconstruction. Unfortunately, it is very hard for the model to keep hand completely immobile. It should be noted for later scan that hand should be perpendicular to the body to successfully capture each side of hands and also cause less occlusion on the body.

The colour information is lost when using the Poisson surface reconstruction in MeshLab but it can be easily recovered by giving to each vertex of the constructed mesh the color of the closest point in the initial point cloud.

Figure 4.7 The red mark show large holes which do not appear in the triangle mesh.

The next step is the rigging and animation of the mesh. This mesh has been fed into the Pinocchio software which successfully rigged the model (**fig.4.8a and 4.8b**). Except for a slight misplacement for the spine middle joint and for the height of the pelvis, the positions of all joints are correct. When the model is animated, we can see that the model legs are linked together where they should not. Thus it shall be noted that during scan, the model should spread arms and leg and not wear too large clothes as it will be hard to automatically delimit limbs.

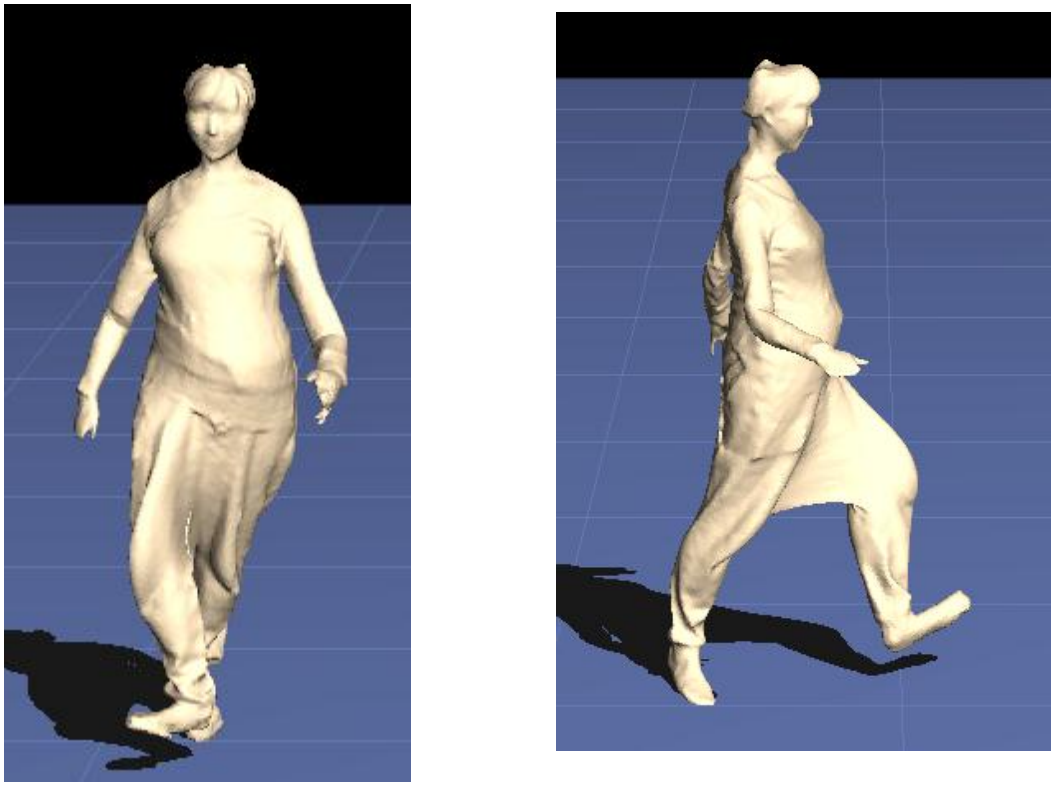


Figure 4.8a Screenshot of the model animated in the Pinocchio software. On the left picture, a stretched surface appear between the leg caused by a wrong delimitation of leg during triangulation.

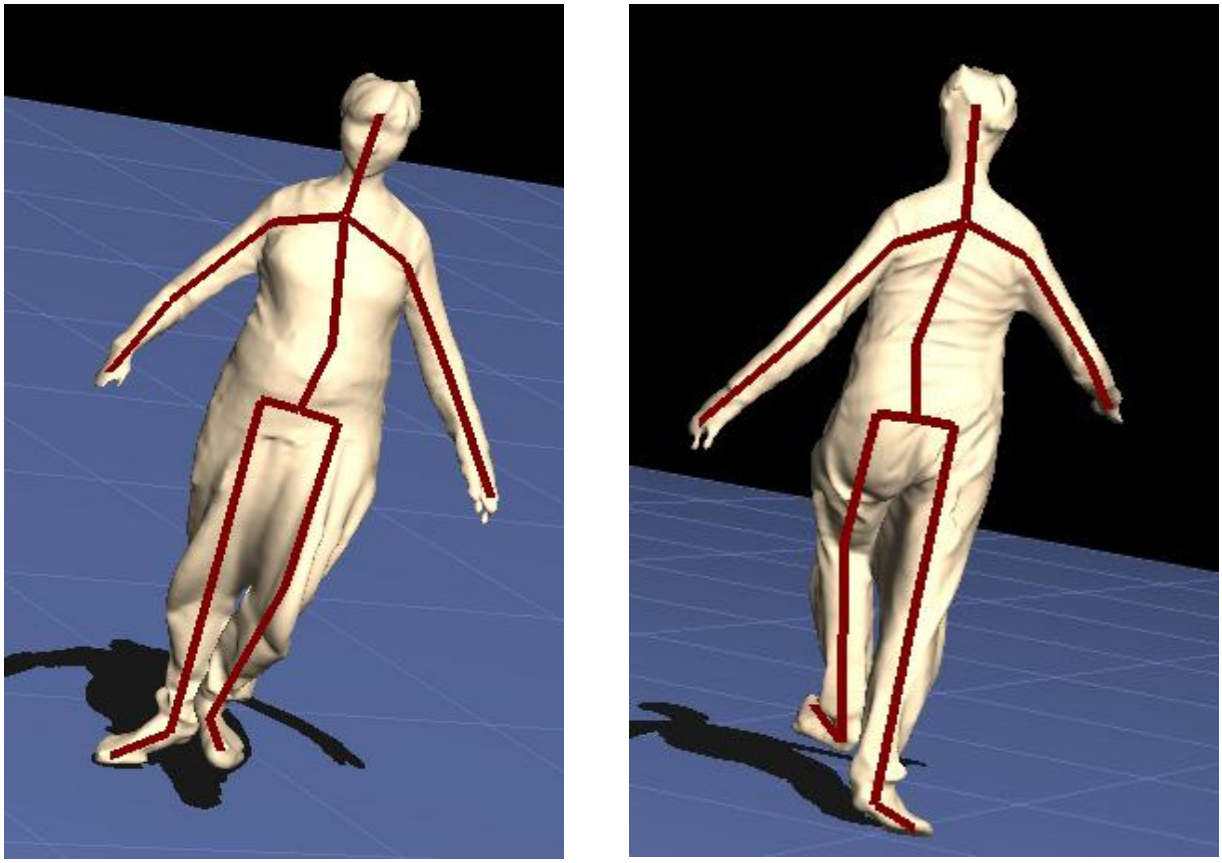


Figure 4.8.b Screenshot of the model animated in the Pinocchio software with skeleton visible.

4.5. Summary

We confirmed the first part of this project hypothesis which was that the Kinect can be used to generate 3D rigged model of real people. But different points need to be improved. The first point is the colour differences between captured points cloud which would require a colour equalization. The second is that the veil points need to be removed as they create colour stripe over the model and also corrupt the geometry. This geometry corruption can be handled by the Poisson surface reconstruction but at the price of a loss of detail. Steder et al. **[31]** have investigated a way to recognize those points in order to treat them properly when processing 3D scanner data.

However, we will not investigate this problem further as we will focus on the alignment problem. Indeed, this result has been obtained by manually aligning the obtained point clouds but this task is greatly time consuming. Several hour have been spent to align all the point cloud. Thus, we will now investigate a way to fully automatize the alignment step.

Chapter 5 Automatic Point Cloud Alignment

Manually aligning several point clouds is a very time consuming task and we can't expect the user to align his scan himself. That is why we need to investigate how to align automatically point clouds without any user intervention. As we saw previously, ICP allows a good alignment of overlapping point clouds provided that a coarse alignment is done beforehand. Indeed, ICP blindly brings closer neighbouring points without knowing if they are right correspondences or not. Thus rough correspondences need to be computed first to provide a rough alignment. One strength of the Kinect is that it gives two possible spaces to search for these correspondences: either in 2D space using the color map or in 3D space using the point clouds. We will explore several different ways to compute a coarse alignment.

There are two ways to approach the registration problem. One is to register at the capture time, thus permitting real time reconstruction. The other is using non real time techniques to register the data after the capture session. First we will try the last approach to see if the data captured previously can be registered automatically. Then we will focus on real time reconstruction.

5.1. Point cloud library

In this project, we will use the Point Cloud Library (PCL) **[38], [41].** PCL is an open-source, cross-platform framework for point clouds processing developed by Willow Garage, a robotics research lab developing hardware and open source software for personal robotics applications. This framework contains numerous state-of-the art algorithms for filtering, feature estimation, surface reconstruction, registration, model fitting and segmentation. PCL is supported by an international community of scientist and engineer.

Here we will focus on the registration and feature estimation algorithms offered by PCL. Indeed, PCL provides several ways to register point clouds with different 3D descriptor and implement several variants of the ICP algorithm.

5.2. 3D local features

To find correspondences into two point clouds, a 3D descriptor can be computed for each point. However, this approach can't be used in real time applications as the computation in 3D space will take more time. PCL offers various 3D feature algorithms. Contrarily to 2D features algorithms , 3D features algorithms do not compute key points. Instead they compute descriptor for each point. An exception is the NARF descriptor which generates a set of key points.

We will review the different feature descriptors available in PCL to see which one is best suited for our purpose.

5.2.1 The NARF descriptor

The NARF descriptor **[42]** (normal aligned radial feature) , though it describes a 3D feature, is fast to compute as its computation happen in a 2D space. Indeed, this algorithm uses the depth map for its computation. Unfortunately, the NARF descriptor is not suitable for our problem. It has been designed for object recognition and thus can recognize an object into a range map but only when it is trained on this object first. Indeed, the key point matching is done on a collection of model. Its recognition is based on the outer form of an object and thus the generated key points depend on the viewpoints. For our problem, we are looking for features that will be consistent for any viewpoints.

5.2.2 FPHF

When attempting to find correspondences into two point clouds, we need a descriptor that is not affected by rigid transformations, nor affected by reasonable noise in the data or by variation of the density of data. Such a descriptor was developed by R.B.Rusu et al., the Persistent Feature Histograms (PHF) **[43]**. They have then modified this descriptor to enhance its processing speed and make it usable in real-time applications. This speed-up version of PHF is Fast Persistent Feature Histograms (FPHF) **[44].** This paper also proposes a new sample consensus based method to calculate a coarse alignment, the SAmple Consensus Initial Alignment (SAC-IA). All those algorithms are implemented into PCL, their author being active member of the PCL project.

Although the FPHF algorithm is a speed-up version of PHF, the computation time is still too long to imagine a real time reconstruction. Indeed, the computation of those features can take up to 30 seconds for large point clouds. Thus, if using FPHF for registration, users will first have to take a series of capture. Then these captures will be automatically aligned afterward.

During the test for this feature, we used the same procedure as the one used for testing manual alignment in previous chapter. The scanned person stood on a turnable chair and is rotated by 90 degree after a series of captures done at different heights. The software successfully aligned point cloud captures from the same side **(fig.5.3 and fig.5.4)**. However, it was not able to align point cloud of different side together **(fig.5.4 and 5.5)**. Indeed, point clouds corresponding to the same side have strong overlap whereas there is a very small overlap between point clouds from different sides, making it very difficult for the program to find correspondences.

A major problem encountered when attempting offline alignment is the fact that one will realize that one of the point clouds is corrupted due to a slight movement of the model. This is not noticeable until we try to align this point cloud and thus all the previous alignments have been done in vain as the scan has to be started from the beginning. Offline alignment is suitable when scanning static object but not person as they can't stand completely immobile for a long time. To avoid this, real time reconstruction should be envisaged.

5.2. Real time reconstruction

The previous approach, which used 3D features for coarse alignment, can't be imagined for real time due to the amount of time needed to process feature for each point in a cloud. To speed up the coarse registration step, one approach could be to reduce the search space for matching features to two dimensions by using the Kinect colour map. Real time estimation of a camera position using feature extracted from video stream is called SLAM (Simultaneous Localization and Mapping). A lot of work have been done on it in the computer vision and robotic community. Its main use is to allow robots or other autonomous devices to perceive their environment. We will review different SLAM techniques. But first, we will try an approach which is not a SLAM technique as we try to use the Kinect skeleton tracking ability.

5.2.1 Skeleton Tracking

When we enumerated the different possibilities offered by the OpenNi API, we mentioned that the skeleton tracking feature can be used for tracking the camera pose. We Also mentioned that the OpenNI API allow for segmentation of the tracked user from the rest of the environment. Thus our approach was to estimate the transformation between the current captured skeleton and a reference skeleton captured at the beginning of the capture using SVD (singular value decomposition) transformation estimation. To do so, we placed the model on a turnable chair which will be turned in front of the Kinect when the Kinect stayed stationary. But this approach has a drawback. To capture fine detail, the Kinect must be held around 1 or 1.5 meters from the scanned subject. But at this distance, only a small part of the scanned subject is seen by the Kinect and thus the skeleton cannot be tracked. When using this approach, we have to sacrifice the level of detail.

Unfortunately, this approach was not successful as the skeleton tracking is efficient only when the tracked user can be seen from the front. When tracking from the side, the skeleton became unreliable and ended up at completely wrong position (**fig.5.1)**. The obtained coarse alignment can be seen on **fig. 5.2**. This alignment is too poor to be refined with ICP. Thus we will explore other methods of alignment.

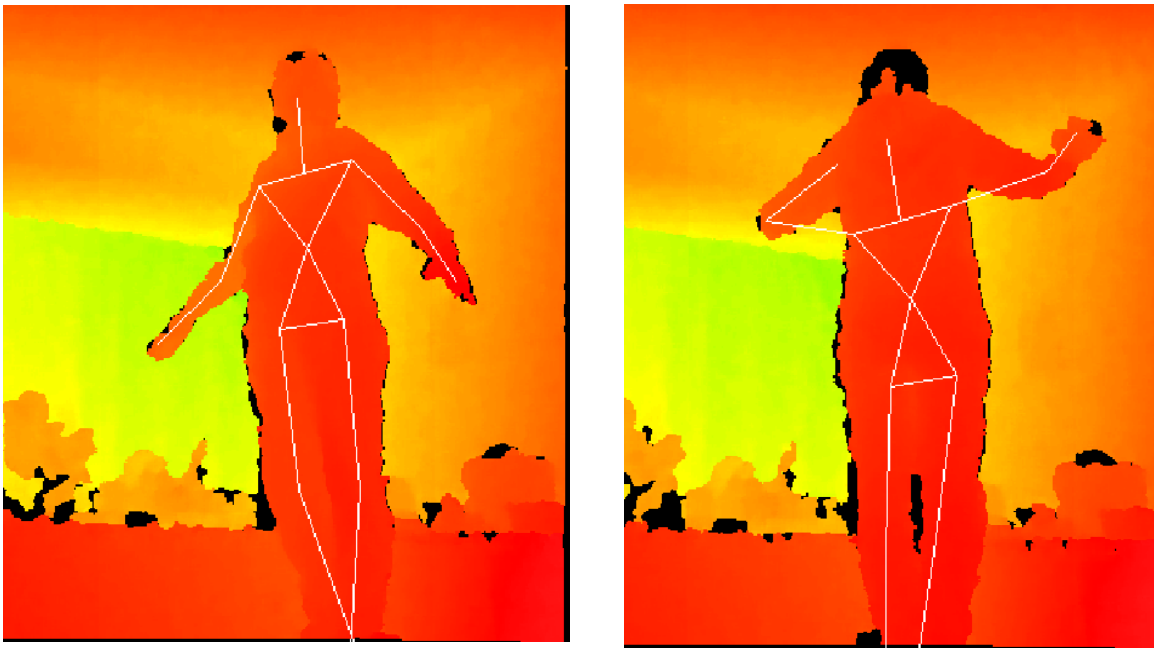


Figure 5.1 on the right the initial position of the skeleton. On the right, the skeleton after rotating the model

Figure 5.2 captured point cloud aligned using the skeleton position

5.2.2 AR marker

Our second approach was to use markers to track camera position and orientation as it is done in Augmented Reality application. In these applications, Computer generated content is superimposed over real-world, generally seen through a camera. To know the computer generated content should be placed, markers (figure) are tracked and their position relative to the camera is calculated. The inverse of the obtained transformation permits to know the camera position.

Unfortunately, this approach was unsuccessful. Indeed, if we place markers on the user, first those marker will appear later on the obtained model and more importantly, the markers have to be placed in the same position relative to each other which is obviously impossible as people will have different morphologies. The other approach is to place those markers on the environment around the scanned subject. The difficulty encountered here is to make sure that at least one marker is visible when capturing. Thus, several markers should be placed in the room and they must be printed in large format to allow recognition. Furthermore, a calibration step has to be done in order to calculate the relative positions between all markers.

This approach has been abandoned as it requires a complex set up from the user to place all markers whereas several computer vision algorithms allow for marker-less tracking. The most popular ones are the SIFT and SURF key point.

5.2.3 Image local feature: SIFT and SURF

SIFT (Scale-invariant feature transform) **[45]** and SURF (Speeded Up Robust Feature) **[46]** are detectors and descriptors for local features in image. They are used in various computer vision applications such as object recognition, 3D reconstruction, image stitching, video tracking, and match moving. These two algorithms detect key points into images and compute descriptor for each of them. The computed descriptor can then be used to match key points between two images and thus a transformation between those two images can be calculated. This permits to recognize objects or assemble different pictures of the same scene. Furthermore, even if the input is only 2D picture, it is possible to estimate the position of the key points in 3D space using reverse-projection calculation, thus allowing 3D reconstruction using picture taken from a different angle **[47] [48]**. However, with the Kinect, this conversion into 3D space, which can require some time, does not have to be done as the Kinect already provide the depth of each key points making possible to consider real time reconstruction.

Unfortunately, this key point matching is not perfect. Several false correspondences are generated. To sort out good and bad correspondences, a popular approach is to use RANSAC (RANdom SAmple Consensus) **[49],** an algorithm proposed by Fischler and Bolles. RANSAC is a model parameter estimator designed to sort out inliers and outliers for a particular model. In this case, correspondences are considered as inlier if the transformation between the two points of a correspondence permit to transform other points into their match. The SURF algorithm is partly inspired by the SIFT algorithm and is faster. A comparison between SIFT and SURF algorithm has been made by Luo Juan and Oubong Gwun **[50]**. With the two algorithms being highly parallelizable, GPU implementation can be found, notably in OpenCV for the SURF algorithm. This GPU implementation allows for real-time detection of feature point on a video stream.

PCL also features implementation of SIFT adapted to 3D point cloud. Unfortunately, this implementation was incomplete at the time of this project as it generated key points but no descriptors, thus making matching impossible.

5.2.3.1 The Kinect RGBDemo

The Kinect has already been used for 3D reconstruction by using these image descriptors in several projects. The Kinect RGBDemo **[51]**, an open-source toolkit developed to help people start programing with the Kinect by demonstrating its different capabilities, has been made available on the OpenKinect project website. This software was developed by Nicolas Burrus from the Robotics Lab of the Univeristy Carlos III in Madrid. It provide several features including gesture recognition and skeleton tracking using Nite, people detection and localization, Export to ply format, Multiple Kinect ...etc. But the features we are interested in is the implementation of a freehand 3D scene reconstruction. The software uses SURF features to align point clouds and then store them as a surfel representation **[52].** Example of a room reconstruction using this toolkit can be seen in **fig.5.6**. Unfortunately, this reconstruction algorithm does not implement loop closure detection.

Loop closure Detection is a crucial element of reconstruction or SLAM algorithms. As the pose estimation is not perfect, there is always a small error between two pose estimations. Unfortunately, this error small error grows as it is accumulated over time. Thus, when a loop occur, which means the scanner returns to a pose it has already been before, the same element will appear at different position in the reconstructed scene due to the accumulated error. An example of this phenomenon can be seen on **fig. 5.7** where the same painting appear at two different positions.

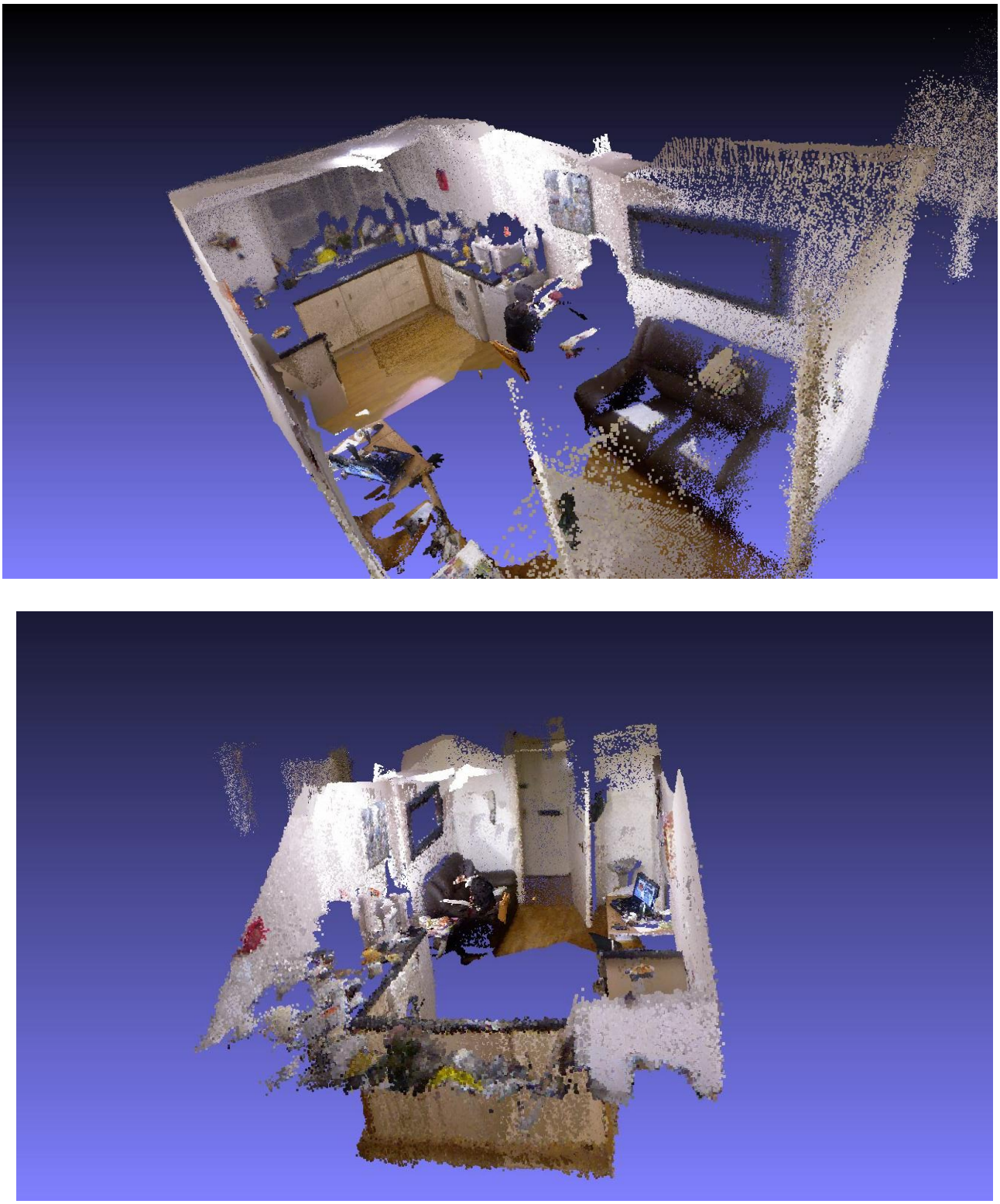


Figure 5.6 Example of a room reconstruction using the Kinect RGBD demo

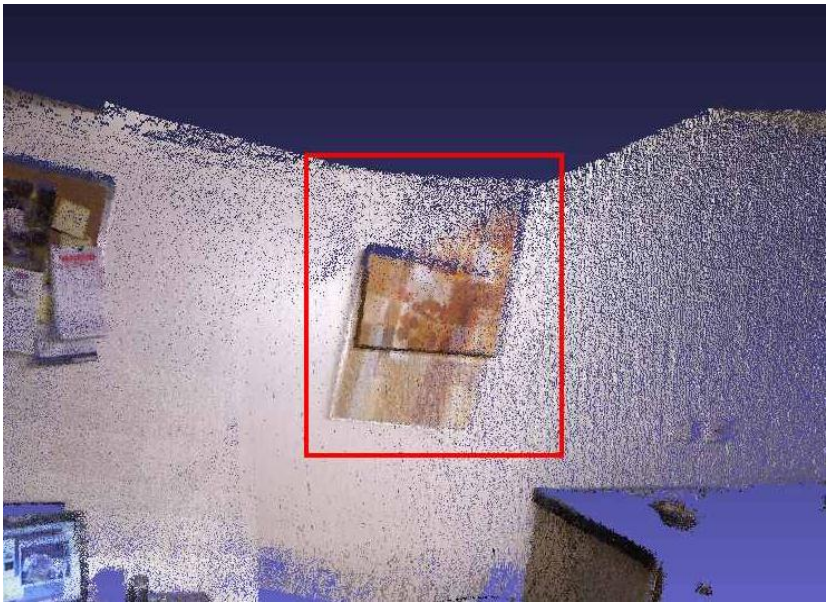


Figure 5.7 Missplacement of the data due to absence of loop closure handling

P Henry et al. **[53]** have developed a similar 3D reconstruction system for indoor environment using a Kinect. Using visual feature to estimate pose, they used a similar approach as the Kinect RGBDemo. The difference is that they used SIFT feature instead of SURF followed by a refinement using ICP. However, they integrated a loop closure detection and handling. They employed a graph structure to store transformations of key-frames. Loop closure is detected by matching the current frame to previous key-frames. When a loop closure is detected, transformations are applied along the graph to align the all map correctly.

Both the Kinect RGBDemo and P Henry et al use a surfel representation to optimize storage of the scanned data. Surfel **[52]**, or surface elements, are an alternative to primitive representation of geometry. They are point primitive without explicit connectivity and thus are perfect for point cloud visualization. A surfel store information about location, orientation, patch size and colour. They are often used in project related to 3D reconstruction with Kinect as they permit to store information about the visibility of a given point and thus allow optimization by removing surfel which are not captured enough to provide sufficient information for its location.

5.2.3.2 Avatar reconstruction with the RGBDemo

Tests have been done to see if the freehand reconstruction implemented in the Kinect RGBDemo can deliver satisfying result for our purpose. First it was tried to capture the model by turning around him or her with the Kinect. Unfortunately, due to the absence of a loop closure handling, the accumulated error caused too big distortion on the generated model. Furthermore, this scanning has to be done very slowly and a quick movement of the Kinect would cause a major corruption in the reconstructed point cloud. Thus, with this software a complete scan in only one step is impossible. However, a good result can be obtained by scanning only one side of the model at a time. Thus in the end we still have to align the result manual. But now if we still turn the model at 90 degree, we have only 4 point clouds to align whereas before we had 16 point clouds to align. Even if it is not a fully automatic reconstruction, it is still a great improvement if the amount of work required is divided by 4. The result obtained can be seen on **fig.5.8.** Even if the hand reconstruction is still not perfect, it has a better quality than that in previous approach.

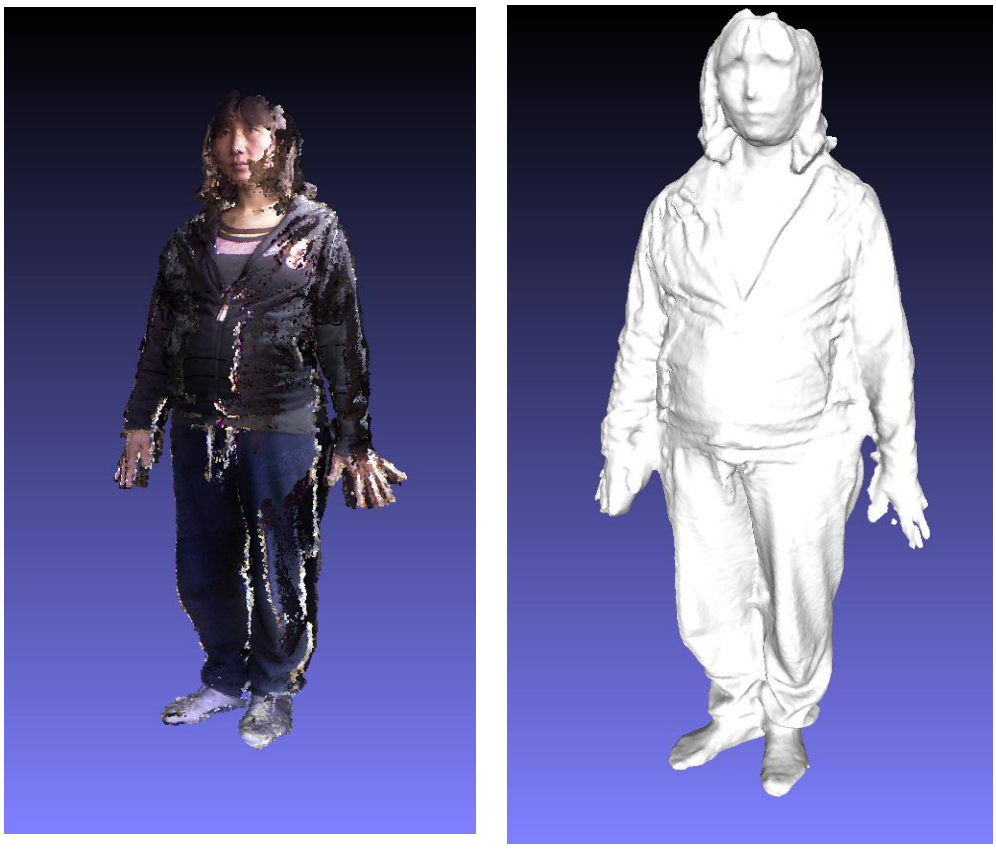


Figure 5.8 on left Aligned point clouds obtained using the Kinect RGBDemo. On right, the triangle mesh

5.3. Kinect Fusion

Microsoft recently released a video of a project called Kinect Fusion **[54] [55]** which displays very impressive real time 3D reconstruction. Not only the reconstruction is done at a very high frame-rate, the reconstructed scene is not made of aligned point cloud but consists of a triangle mesh generated in real time providing very high geometric detail as this mesh is constantly refined with the new capture. This allows for integration of an interaction of CG element into the scanned environment immediately during the scanning process, even allowing physic simulation.

However, the most stunning point of this reconstruction is the fact that the reconstruction is dynamic. Indeed, if we scan a table and there is an object on it at the beginning of the scan, after the object is removed and the table scanned again, the object will be removed from the reconstructed scene.

5.3.1 Technical overview

In contrast to reconstruction projects discussed previously, KinectFusion does not rely on visual features such as SURF or SIFT for the registration process. At each frame 3D point coordinate and their normal are calculated using a GPU implementation. Those points are stored to be reused for the next frame, where they are reconverted into image coordinate. A ray is then traced to find correspondences into the current 3D point cloud. Those correspondences are checked by measuring the Euclidean distances between them. The remaining correspondences are then used by a GPU version of ICP which calculates the transformation by minimizing the point-to-plane error metric defined as the sum of squared distances between each point in the current frame and the tangent plane at its corresponding point in the previous frame. The advantage of this technique is that the full depth map is used whereas, when using local feature (such as SIFT), only a small set of key points are used.

The transformed point are then stored using a volumetric representation which uses voxels. This representation has several advantages similar to the ones of surfel. It stores information relative to the uncertainty of the data, allows for fusion of data when scanning the same area and implicitly store surface information.

5.3.2 Problem solved by KinectFusion

The KinectFusion project shows that real-time accurate reconstruction is possible with the Kinect. It is adapted to our purpose as we can see on the video a person scanned resulting into a high quality model, similar as the one obtained in previous section, but without any colour or geometry error. The ease of use of this system allows anyone to easily reconstruct a posing model. This system not only solves our main problem which was reconstructing a person with a freehand scanning but it also solves another problems encountered.

We saw that when using 3D scanner, an phenomenon called veil points appear provoking corruption in geometry and colour, which result in strip along with deformation at the border of one point cloud. As avatars are constructed from several capture, those stripe appear all over their bodies, spoiling the quality of the result. But this problem does not occur in the KinectFusion system as it constantly refines the generated mesh, correcting geometry and colour distortion.

Also, we saw from our previous scanning test that the hand reconstruction was a problem as it is hard for the model to keep hands completely immobile, provoking non rigid transformation in capture data, which are hard to handle without losing fine detail. However, as the KinectFusion Reconstruction is dynamic, error generated by a slight movement of hands will be corrected automatically.

Furthermore, the KinectFusion also solves the problem of segmenting the scanned person from the background. Indeed, it demonstrates segmentation capabilities. After having scanned an object, if this object is removed, the KinectFusion system is able to segment this object from the rest of the scene which remains static. Thus, after having been scanned, the model can step out of the kinect field of view and be segmented. This will avoid requiring users to manually attempt this task in an editing software such as Meshlab.

At the time of this project, Microsoft only released a video and a paper for the KinectFusion Project. No binaries or source code have been made available. Thus, it is impossible to test directly the KinectFusion technology with our project. However, the purpose of this project was not to deliver a system able to reconstruct automatically a person and generate an avatar but to see if available technologies allow such a system to be realizable, which have already been proved. In the last chapter, we will explore different ways to extend our system.

Chapter 6 Conclusions and Future Work

6.1. Conclusions

The hypothesis for this project was that the Kinect motion sensing input device can be used to allow games players to create realistic avatars of themselves. We believe that the research presented in this dissertation shows that this hypothesis is true, but with some limitations at present. In chapter two, we saw that using skeletal animation is advantageous for flexible 3D modeling and generating virtual characters. Thus we created a character generator using 3D rigged models for the <e-adventure> platform, which was previously using a character generator requiring the users to create a 2D sprite. This has made the <e-adventure> character generator far more accessible. However, a disadvantage is that the games player must now choose an avatar from the supplied library of models. We proceeded to research and develop a solution whereby the games player can create a completely new character whilst keeping the advantages of 3D skeletal animation. In chapter four we began to investigate using the Kinect for such a purpose. First, we looked at the available solutions for automatic rigging as without it, no avatar creation would be possible. We found an open-source solution for automatic rigging and then successfully produced a high detail 3D rigged model of a real person. However, the process used to reach this result was time-consuming and complex as it required manually aligning the obtained point cloud using editing software. Chapter five was therefore devoted to exploring the different available solutions for automatic reconstruction of point clouds. We discovered that Microsoft have developed a software application that is capable of efficiently reconstructing a scene where data points had been acquired using a Kinect. This software is not available to the community but it does demonstrate that in the future, high quality reconstruction will be possible for anyone with a Kinect. This work from Microsoft, plus the research presented in this dissertation, augurs well that creating an avatar that is a perfect replica of someone will soon be a reality. In the final section below, we explore possibilities to extend the functionality of such a system.

6.2. Future Work

6.2.1 Facial animations.

Chapter two discussed the well-known psychological problem of the uncanny valley. If a virtual character is highly realistic in terms of visual appearance, then there is more risk in that it will give an impression of not being alive if the animation is not good enough. Here body animation using a skeleton makes generated avatars look alive. However, the uncanny valley problem will appear if we focus on the avatar's face. Indeed, the Kinect allows acquisition of the fine geometrical detail of a face, which coupled with texture, gives a highly realistic result. Unfortunately, this face will be static and so we will fall into the uncanny valley. However, face feature recognition from a picture is now something that can be processed in real time by a computer. Thus it is easy to locate mouth, eye, jaws, etc. on the colour map and then to calculate their positions on the 3D avatar. A facial animation system can then be placed on the avatar. This has been attempted by M. Zollhofer et al. **[53]** who generated an animated 3D model of a face using a Kinect.

6.2.1 Motion capture for custom animations

We saw previously that when a studio scans a famous actor or sportsperson to create an accurate representation of them in a video game, they also use motion capture to make the virtual character's animation close to reality. We can now provide the game players with the possibility to create an accurate representation of themselves for the visual part. However, the Kinect can also allow us to go even further into avatar customization. Indeed, one of the feature faces that the Kinect has been designed for is motion capture. Thus, once we generate a rigged model, it is easy to animate it using motion capture. This would allow the game players to record their own animations.

References

[1] R. O'Neill, *Digital Character Development: Theory and Practice*. Elsevier Science, 2008, p. 33.

[2] N. Ahmed, E. de Aguiar, C. Theobalt, M. Magnor, and H.-P. Seidel, "Automatic generation of personalized human avatars from multi-view video," in *Proceedings of the ACM symposium on Virtual reality software and technology*, 2005, pp. 257-260.

[3] M.-cruz Villa-uriol, M. Sainz, F. Kuester, and N. Bagherzadeh, "Automatic creation of three-dimensional avatars," in *in SPIE-IS&T Electronic Imaging*, 2003, pp. 14-25.

[4] "MASSIVE." [Online]. Available: http://www.massivesoftware.com/.

[5] "Virtual Worlds, Avatars, free 3D chat, online meetings - Second Life Official Site." [Online]. Available: http://secondlife.com/.

[6] C. Hecker, B. Raabe, R. W. Enslow, J. DeWeese, J. Maynard, and K. van Prooijen, "Real-time motion retargeting to highly varied user-created morphologies," in *ACM SIGGRAPH 2008 papers*, 2008, p. 27:1--27:11.

[7] M. Mori, "Uncanny valley," *Energy*, vol. 7, no. 4, 1970.

[8] R. Parent, *Computer animation: algorithms and techniques*. Morgan Kaufmann, 2007.

[9] J. E. Chadwick, D. R. Haumann, and R. E. Parent, "Layered construction for deformable animated characters," *SIGGRAPH Comput. Graph.*, vol. 23, no. 3, pp. 243-252, Jul. 1989.

[10] J. Lander, "On Creating Cool Real-Time 3D," *Gamasutra*. [Online]. Available: http://www.gamasutra.com/view/feature/3234/on_creating_cool_realtime_3d.php.

[11] K.-J. Choi and H.-S. Ko, "On-line Motion Retargetting," in *Proceedings of the 7th Pacific Conference on Computer Graphics and Applications*, 1999, p. 32--.

[12] S. O. Ead, "Designing a character avatar model for the Mermaids MMO."

[13] B. Flannery, "Video Game Education."

[14] "<e-UCM> The e-Learning Research Group at UCM." [Online]. Available: http://www.e-ucm.es/.

[15] "IEEE Standard for Learning Object Metadata." 2002.

[16] "Irrlicht Engine - A free open source 3d engine." [Online]. Available: irrlicht.sourceforge.net/.

[17] "OGRE – Open Source 3D Graphics Engine." [Online]. Available: http://www.ogre3d.org/.

[18] marc ten Bosch, "2D Skeletal Animation and Character Editor." [Online]. Available: http://marctenbosch.com/skeletal2d/.

[19] I. Baran and J. Popović, "Automatic rigging and animation of 3D characters," in *ACM SIGGRAPH 2007 papers*, 2007.

[20] J. Pan, X. Yang, X. Xie, P. Willis, and J. J. Zhang, "Automatic rigging for animation characters with 3D silhouette," *Comput. Animat. Virtual Worlds*, vol. 20, no. 2‐3, pp. 121-131, Jun. 2009.

[21] M. Poirier and E. Paquette, "Rig retargeting for 3D animation," in *Proceedings of Graphics Interface 2009*, 2009, pp. 103-110.

[22] N. D'Apuzzo, *Recent Advances in 3D Full Body Scanning with Applications to Fashion and Apparel*. .

[23] "Microsoft official Kinect Webpage Web." [Online]. Available: http://www.xbox.com/kinect.

[24] J. Giles, "Inside the race to hack the Kinect," *New Scientist*.

[25] "OpenKinect project webpage." [Online]. Available: http://openkinect.org/.

[26] "Primesence Nite webpage." [Online]. Available: www.primesense.com/nite.

[27] "OpenNI webpage." [Online]. Available: http://75.98.78.94/default.aspx.

[28] Y. Cui, S. Schuon, D. Chan, S. Thrun, C. Theobalt, and M. Informatik, "3D Shape Scanning with a Time-of-Flight Camera." .

[29] Y. Cui and D. Stricker, "3D shape scanning with a Kinect," in *ACM SIGGRAPH 2011 Posters*, 2011, p. 57:1--57:1.

[30] P. Breitkopf, H. Naceur, A. Rassineux, and P. Villon, "Moving least squares response surface approximation: Formulation and metal forming applications," *Comput. Struct.*, vol. 83, no. 17-18, pp. 1411-1428, Jun. 2005.

[31] B. Steder, R. B. Rusu, K. Konolige, and W. Burgard, "Point feature extraction on 3D range scans taking into account object boundaries," in *ICRA*, 2011, pp. 2601-2608.

[32] M. J. Weiss, A., Hirshberg, D., Black, "Home 3D body scans from noisy image and range data," in *ICCV*, 2011.

[33] D. Anguelov, P. Srinivasan, D. Koller, S. Thrun, J. Rodgers, and J. Davis, "SCAPE: shape completion and animation of people," in *ACM SIGGRAPH 2005 Papers*, 2005, pp. 408-416.

[34] P. J. Besl and N. D. McKay, "A Method for Registration of 3-D Shapes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, no. 2, pp. 239-256, Feb. 1992.

[35] Z. Zhang, "Iterative point matching for registration of free-form curves and surfaces," *Int. J. Comput. Vision*, vol. 13, no. 2, pp. 119-152, Oct. 1994.

[36] "MeshLab Webpage." [Online]. Available: http://meshlab.sourceforge.net/.

[37] "The Digital Michelangelo Project." [Online]. Available: http://graphics.stanford.edu/projects/mich/.

[38] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[39] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proceedings of the fourth Eurographics symposium on Geometry processing*, 2006, pp. 61-70.

[40] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, "The Ball-Pivoting Algorithm for Surface Reconstruction," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 4, pp. 349-359, Oct. 1999.

[41] "Point Cloud Library." [Online]. Available: http://pointclouds.org/.

[42] B. Steder, R. B. Rusu, K. Konolige, and W. Burgard, "NARF : 3D Range Image Features for Object Recognition," in *October*, 2010.

[43] R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz, "Aligning point cloud views using persistent feature histograms," in *Proceedings of the 21st IEEERSJ International Conference on Intelligent Robots and Systems IROS Nice France*, Ieee, 2008, pp. 3384-3391.

[44] R. B. Rusu, N. Blodow, and M. Beetz, "Fast point feature histograms (FPFH) for 3D

registration," in *Proceedings of the 2009 IEEE international conference on Robotics and Automation*, 2009, pp. 1848-1853.

[45] D. G. Lowe, "Object Recognition from Local Scale-Invariant Features," in *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, 1999, p. 1150--.

[46] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-Up Robust Features (SURF)," *Comput. Vis. Image Underst.*, vol. 110, no. 3, pp. 346-359, Jun. 2008.

[47] M. Brown and D. G. Lowe, "Unsupervised 3D Object Recognition and Reconstruction in Unordered Datasets," in *Proceedings of the Fifth International Conference on 3-D Digital Imaging and Modeling*, 2005, pp. 56-63.

[48] M. S. L. Barazzetti, F. Remondino, "AUTOMATION IN 3D RECONSTRUCTION: RESULTS ON DIFFERENT KINDS OF CLOSE-RANGE BLOCKS."

[49] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381-395, Jun. 1981.

[50] L. Juan and O. Gwon, "A Comparison of SIFT, PCA-SIFT and SURF," *International Journal of Image Processing (IJIP)*, vol. 3, no. 4, pp. 143-152, 2009.

[51] "Kinect RGBDemo." [Online]. Available: http://nicolas.burrus.name/index.php/Research/KinectRgbDemoV6?from=Research.KinectRgbDemoV2. [Accessed: Oct-2011].

[52] H. Pfister, M. Zwicker, J. van Baar, and M. Gross, "Surfels: surface elements as rendering primitives," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 2000, pp. 335-342.

[53] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "Rgbd mapping: Using depth cameras for dense 3d modeling of indoor environments," in *In RGB-D: Advanced Reasoning with Depth Cameras Workshop in conjunction with RSS*, 2010.

[54] S. Izadi et al., "KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera," in *Proceedings of the 24th annual ACM symposium on User interface software and technology*, 2011, pp. 559-568.

[55] "KinectFusion Project Page." [Online]. Available: http://research.microsoft.com/en-us/projects/surfacerecon/.

Appendix A

Content of the CD

- Source code of the character generator developed for the <e-adventure> platform
  Read the README.txt file for compilation instruction and current issue of the program
  Character Generator folder
- 3D models generated (point clouds and meshes)
  3D models folder
- Pinocchio Auto-rigging software
  Read the README.txt file for instruction. Some model need the –rot option to be aligned correctly.
  Pinocchio folder
- Movies of the 3D model animated with Pinocchio
  Movies folder