Saarland University
Faculty of Natural Sciences and Technology I
Department of Computer Science.

Master Thesis

Sensor-Based Interactive Editing of 3D Motion Paths for Character Animation

by Frederic Raber

submitted on .11.2012

Supervisor: Prof. Dr. Michael Kipp
Advisor: Quan Nguyen

Reviewers: Prof. Dr. Michael Kipp, Prof. Dr. Antonio Krüger

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, d. _____ _____

Abstract

I propose a gesture-based interaction technique called MotionBender for editing character animation data using a motion sensor (Kinect) and a simple click controller. For the visualization of motion I use motion paths, often used for showing e.g. the movement of the hand through space over time, and allow the user to directly "bend" the 3D motion path with his/her hands and twist it into the right shape. In a comparative evaluation with a mouse-based interface I found that subjects using my technique were significantly faster. Moreover, with my technique, subject movement was more coordinated, i.e. movement was done in all three dimensions in parallel, and the participants preferred my technique in a post-experiment questionnaire. I also found a gender effect: male users both like the gesture interaction better and achieve better performance.

Table of contents

1 Introduction . . . 6

1.1 Motivation . . . 6
1.2 Goals . . . 7
1.3 Approach . . . 8
1.4 Outline . . . 8

2 Character Animation . . . 9

2.1 Background . . . 9
2.2 Related work . . . 10

3 MotionBender: Workflow . . . 15

3.1 Technical terms . . . 15
3.2 Workflow overview . . . 15
3.3 Workflow of the edit phase . . . 17
Load BVH . . . 18
Joint & mode selection . . . 18
Point selection . . . 18
Point moving . . . 18
Generation of the movement . . . 18

4 Concepts & Implementation . . . 20

4.1 Motion paths . . . 20
4.2 Neighbour function . . . 21
4.3 Edit modes and parameters . . . 22
4.4 Input system . . . 23
4.5 Arm control . . . 26
4.6 Export of the generated motion . . . 27

5 Technical details . . . 28

5.1 Hardware . . . 28
Sensor-based control devices . . . 29
Two Mouses in parallel . . . 31
5.2 Software . . . 33
5.2.1 Post-Processing software for the sensor device . . . 33
5.2.2 Visualisation software for Motion capture data . . . 35
5.3 Motion capture file formats . . . 36
5.3.1 The BVH file format . . . 36
Header section . . . 36
Body section . . . 37
5.3.2 Other file formats . . . 37
Acclaim . . . 37
COLLADA . . . 38

6 Evaluation . . . 39

6.1 Methods . . . 40
6.2 Procedure . . . 42
6.2.1 Instruction . . . 42
6.2.2 Training . . . 43

6.2.3 Test . . . 43
6.3 Measures and results . . . 44
6.3.1 Recorded values . . . 44
6.3.2 Results . . . 44
6.4 Discussion . . . 46

7 Conclusion . . . 47

7.1 Summary . . . 47
7.2 Future work . . . 47

8 Appendix . . . 49

8.1 Install instructions for Kinect and the SDK . . . 49
8.2 Kinect hardware sender . . . 49
8.3 XInput skripts . . . 49
8.4 Questionnaire . . . 51
8.5 Evaluation handout (German) . . . 54

Bibliography . . . 55

1 Introduction

Character animation plays an important role in the science and also in everyday life. The central part of virtual character interfaces and avatars for sign language are the animated characters, that supply the interface between the human and the machine. Because of this fact, a good animation is important for these domains. A similar field is the gestural and behavioural analysis. For being able to conduct such an analysis, the movement of the character has to be tracked, and stored afterwards. For animated movies like Dreamworks' "Shrek", a lot of animations need to be created, post-processed and synchronized to each other, so the movie appears realistic in the end.

The common workflow for creating these animations at this point in time is quite complex, and needs some training for the animators in advance, to be able to effectively control the programs used. My thesis presents a character animation system, which is tailored especially for untrained users, so they can directly start with creating an animation, needing only few or no training at all in advance. This process should be improved by using modern sensor-based technology, trying to give a more intuitive manipulation method to the user. The next sub-chapters will give a deeper view on the motivation for my work, line up the goals that I want to meet with my implementation, and give a raw outline of the approach that I used in my work.

1.1 Motivation

One method to create character animations is the straight-ahead keyframed approach. This means that the animation is created manually by the animator. This is done using virtual tools like extrusion or lassos, which can be virtually wrapped around the model to get it into shape. The main screen of such an animation tool is shown in Figure 1.

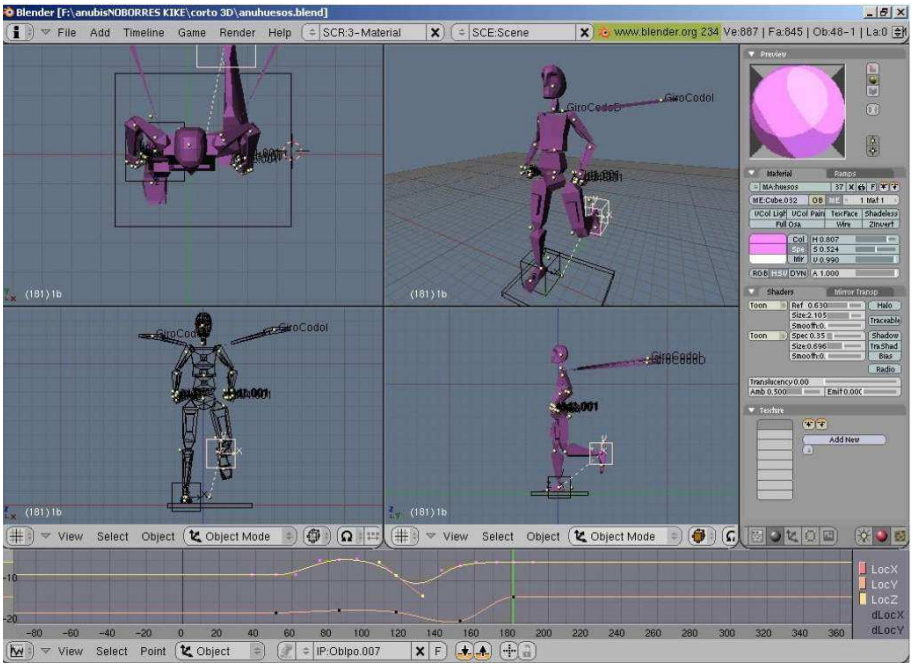

Figure 1. Main screen of blender

There are several views on the scene, from different perspectives and different elements of the skeleton. For example, instead of the model itself, also the bone and joint geometry or the polygon grid can be displayed. For animating a model, the user can move the joints to some other position by mouse. Blender can also calculate some intermediate keyframes in a limited manner. The techniques which are used by Blender are mostly indirect and virtual, which makes it hard to learn for beginners. As one can see on the screenshot, there are a lot of buttons, sliders and menus. For a person who has not been thaught all of their functions, it is impossible to start using it correctly. Because of the high variety of tools and functionalities which are supplied by the software, it needs a long training phase in order to know which tool can be used if the animation or the model needs to be changed to a specific look. Even then, the abstract behaviour of the tools makes it hard to foresee what happens when the tool is actually used, and which mouse movements cause which action. An additional problem is that there is a three-dimensional animation which has to be changed with a two-dimensional mouse device. Therefore, the third dimension has to be "emulated", for example by using the scroll wheel for movements in the depth axis. This increases the abstractness and the training time for a new user. There are some more-dimensional input devices like 3D-mice used. Studies have shown that they can improve the workflow. But for getting used to these devices, additional training time is needed. Chapter 5.1 goes into deeper detail about the different input devices. On the other hand, these tools are precise and deliver a maximum flexibility to the user, as he has to animate everything himself, exactly how he wants it to look like. There is no mandatory need for additional hardware, the tools can be used directly with conventional mice.

My implementation tries to keep these advantages, and to improve the usability by first providing a simpler and more intuitive user interface, and second allow a direct manipulation method with advanced input devices, to get a user-friendly interface design. In the next part of the chapter, I will define the goals of my work more precise.

1.2 Goals

To say it in one sentence, I want to create a character animation editor, which can be used not only by professionals, but also by novices. Although the interface should be more simple than in the tools that are used nowadays, this should not decrease the power and flexibility in that special domain. The implementation should have the following attributes:

Animation tool. My tool is not meant to be a universal tool for all kinds of 3D character animation, but it should be designed especially for the interactive post-processing of Character animations. Other additional functionalities are not supported

Intuitive. The editor should support direct manipulation methods, so the user gets a good feeling for the effect of his actions. In the ideal case, the modeling and behaviour of the tool works exactly like in a real world environment, so that the user does not need to learn specific movements. He acts like he is used to in the real world, and the software does exactly the corresponding action. The editing techniques should not be as abstract as with the tools in standard software, so the user does not need to first understand the sense and effect of each tool.

Efficient. Although a user-friendly design needs a reducing of complexity and therefore reducing the variety of functionalities, the program should still be as powerful as a standard user interface. The creation and modification of animations should be possible at least as quick as with common tools, maybe even faster.

Flexible. The user should be able to control the behavior of every single joint seperately.

Joy of use. Users should enjoy working with this technique, and not be frustrated by a complex and counter-intuitive workflow.

Now that the specific goals of my work are clear, the next sub-chapter will give some overview on the approach that I used to reach my goals, the overall workflow and the most important concepts that I used.

1.3 Approach

The implementation project of my solution is be called "MotionBender", it is an editor for motion-based character animation, by bending the motion trajectory. MotionBender works in two phases: The recording and the editing phase. First, a motion is performed in front of the screen and recorded at the same time. A view sensor is capturing the movements of the user, and allows the recording by a special software on the attached PC. In the second step, this motion can be edited, which is the main part of my thesis.

For my work, I took a different approach than the straight-ahead method. My implementation is based on an *interpolation-based* method. The user does not define the complete motion, but he gives several constraints on the motion to be created, and the software tries to create the corresponding motion regarding to that given constraints. My implementation gives *motion curves* to the user for describing the motion. These curves are three-dimensional curves. There is exactly one motion curve for each joint in the skeleton, describing the movement of that joint during the animation. The user can "grab" this line at specific points and twist it into the shape that he needs. The software adapts the original motion so that the new curve is matched by the character's body part. The fourth part of the chapter will now give a short overview on the upcoming parts of the thesis and its contents.

1.4 Outline

The later chapters of my thesis start first with a part about needed background information, covering Character animation and its applications on electronic hardware. Related work will be presented, which I used as an inspiration for my implementation. In detail, four related articles will be presented, covering different techniques of interpolation-based character animation for different domains. Advantages, drawbacks and the corellation to my work will be discussed.

The third chapter is the first chapter describing my tool *MotionBender*. The overall workflow for recording and editing an animation is described. The editing process is described in detail. Deeper information on the used concepts like motion curves, the storage formats as well as implementation details for the different parts of the software are given in chapter five. The used hardware, its alternatives and the software needed in order to use it, is also discussed there.

I conducted a user study, trying to solve the question whether or not the new sensor-based interface is better than a conventional interface using mice. The sixth chapter describes the setup, analysis and the outcome of my experiment, followed by a discussion of the results. The last chapter reviews the achievements of my thesis, leading to a discussion of the work. The thesis ends with a conclusion and some proposals for further work.

2 Character Animation

The most central aspect in my work is Character Animation. Here I will first give some general background about the topic and different approaches to solve the problem. The second part discusses current work in this area, based on the mentioned approaches. The chapter ends with a comparison of the related work, as well as a matching of the goals set in Chapter 1.

2.1 Background

In general, character animation is a specialized area of the animation process concerning the animation of one or more characters featured in an animated work. It is artistically unique from other animation in that it involves the creation of apparent thought and emotion in addition to physical action. I have a special form of this character animation, called *computer animation*[8]. There, this character is created using 3D computer graphics.

An often used approach is the keyframe method, where the character animator creates the keyframes "by hand". For achieving this, the software uses "virtual tools" or metaphors, which can be used by the animator to modify the scene. One tool implementing this approach is for example Blender[1]. A detailed description of Blender (see Figure 1) and its functionalities can be found in [22].

The technique is powerful and universal, which results in animations that look smooth and natural. On the other hand, it needs a lot of training in advance to be able to use the software in an efficient way. The tools are quite abstract, their effect on the model and the scene is hard to understand in the beginning. The editing techniques and the user interfaces are complex and counterintuitive for unskilled users. There are 4 windows for different views on the scene, which makes it hard for novices to understand the actual look of the model. The toolbox on the righthand side consists of a lot of buttons and sliders, whose utility and effect on the scene is not clear at all to an untrained user. This is why it needs a lot of practise and also ability to be able to use such a tool. Even then, it is still time-intensive to create a complete animation, or even a whole animation movie of more than one hour runtime.

Another approach is the *interactive* method. The author is directly controlling the movement of a virtual character by an input device, or by some visualisation of the movement. For example, he creates or modifies a trajectory, describing the character's movement. [15] give some more background information about the different approaches, as well as an alternative interactive approach.

The advantage of interactive editing is the higher intuitivity, which makes it easier especially for novices. The control methods are designed easier, as in [15]. In this paper, the movement of the character is controlled only by the mouse. The character is following the movement of the mouse. There are no tools or metaphors which need to be understood first like in earlier tools. The user can directly start using the software, as there is less or no training needed at all. On the other hand, the user does not have the control over the generated animation and every single frame of it, as most of the frames are generated automatically by the software. If the generated result is not the desired one, it is hard to change the animation to the correct result.

The next section will discuss several implementations and techniques of this method, and discuss their advantages and drawbacks, regarding my special goals for this work.

1. http://www.blender.org/

2.2 Related work

There are several papers and articles covering the topic of Character Animation. A first article on this topic is *Motion warping* by Andrew Witkin and Zoran Popovic [24], written in 1995. His approach also works in two steps. He takes an existing animation, and allows some after-editing on it. The central concept which he uses is a warp function as explained in [24]. A warp function supplies mainly two functionalities: The movement of the character can either be shifted, e.g. moved to another place in the scene, or it can be scaled. The latter case does not mean that the animated figure is re-sized, but the movement of some body parts can be changed so they look more or less distinct. Figure 2 illustrates the behaviour of the animation upon a changing warp curve.

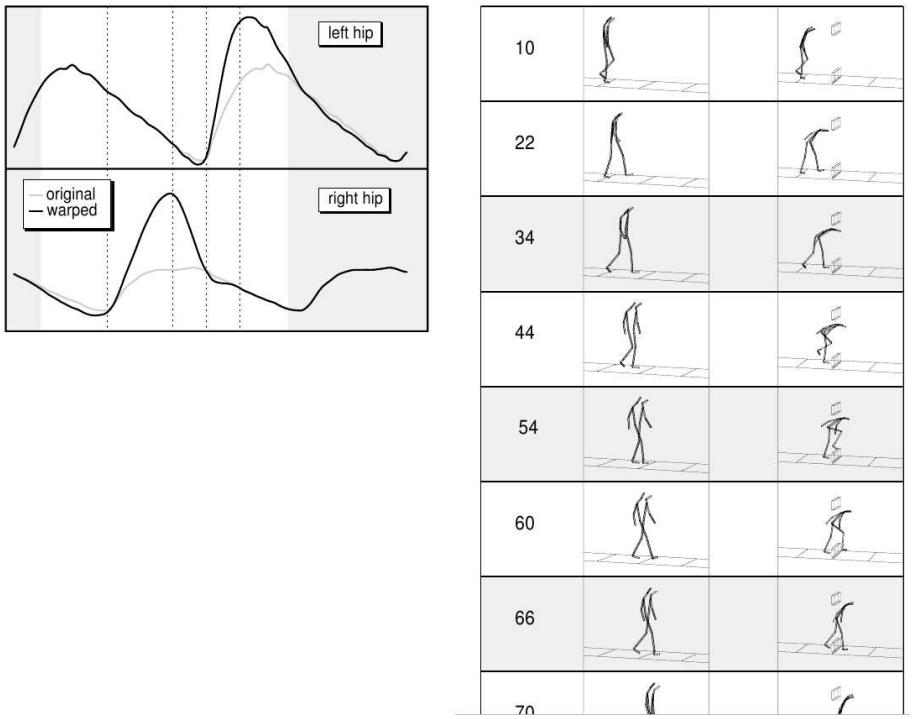

Figure 2. Warp curve

The lefthand side of the picture shows the movement of the left and right hip over time. The grey line denotes the original, the black line the warped movement. The corresponding motions to the curves are presented on the table at the right side. The leftmost column denotes the frame number, followed by the original motion, and the warped motion at the rightmost column. The motion has been scaled at two positions in the middle of the animation. It was scaled in both cases, to look more distinct, so that the hips are moved higher during the walking process of the character.

These methods have all been integrated into the SoftImage UI, which is a standard interface similar to the one of the Blender open source rendering engine. That results in a more user-friendly user interface than the standard version of such motion editing programs. What animators change most of the time, is making the the animation to look more or less distinct or to move it to another position, which is exactly what this new functionality provides to the user. Still, the user interface is nearly the same as in the original version, which is confusing and unclear to the standard user. In addition to this, the concept and effect of a warp-function is not known at all to a non-proficient user. This can lead to even more problems than with the traditional method. By using interpolation, the user looses some control over the generated scene, as he can not edit every single frame by hand. He has to change the parameters of the warp function in a trial-and-error method in order to reach the desired result, if this can be

reached by a warp function at all. In my work, I re-used this idea of shifting and scaling a character animation, but in a different style. Table 1 summarizes which of my goals were met by Witkin's work.

	A. Witkin et al. '95
Efficient	
Intuitive	
Flexible	
Joy of use	2D-Mouse
Relation to my work	Post-Process via shift/scale

Table 1. Reached goals of Witkin et al. '95

The interface is faster than the conventional method, as the warp function generates almost the complete animation for the skilled user. But still, it is hard to learn the usage of the software, maybe even harder than without this additional tool in the UI. There can be one warped curve for every joint in the skeleton, which gives the maximum flexibility in adaption of body parts. The control is done with tradition input devices. As we can see, there are some unmet goals for my special domain, where especially the unintuitive UI is a more serious problem. The next related work tries to tackle this problem by a more intuitive, direct and interactive manipulation method, still using the mouse as the control device.

In *Layered Performance Animation with Correlation Maps* by Neff and Kim [15], no prerecorded motion is needed. The whole animation is created from scratch during each session. The movement of the character and its joints is done by the mouse, whose positioning data is processed by a correlation map. There is one map entry for each body part of the skeleton. Corresponding to these map parameters, the body parts are moved according to the mouse movement. The user can select one or more body parts which should be animated in every edit step. For example if the hand and a leg are selected to be modified, then the XY-movements of the user's mouse are directly mapped to a 3D-movement of the character's hand and leg.

This technique is more intuitive and user-friendly than the previous one, the only thing the user has to do is to move the mouse, and the character will act accordingly. Another advantage is the direct manipulation, which allows the user to see directly the effect of his acts. By this, he can easily learn the interface by trial and error. Animation is possible in several steps. The user can for example first animate a hand of the character, and afterwards the legs or the other hand. This gives the user more flexibility, as well as the possibility to split the animation process into several parts, and to concentrate on only one body part in each step.

On the other hand, this complex system of correlation maps has first to be set up and tuned in order to function correctly. Another problem is that every user has a different feeling for the movements, e.g. the maps have to be adapted for every single user in order to be effective. The interface for tuning the maps is not user-friendly at all. In the paper, the maps were tuned manually by the experiment leader. Still, the system is controlled by a conventional 2D-mouse. Using a device with more degrees of freedom, the usability could be improved, as also the maps get more input and can react better. The project is a proof-of-concept implementation, which does not allow every single body part to be selected and edited. Only the most important ones are supported at the moment.

To recap, this project is superior in the domain of intuitivity and user-friendliness, which is the biggest difference to the previous stated work. It is also fast to create an animation with this tool, maybe even more faster. The user can interactively edit the animation, and animate

also several body parts together, which leads to a saving of time. The flexibility is also given, as the body parts can be edited independingly. Still, there are no "advantaged" input devices used, which could further improve the workflow. My work re-uses the idea of mapping movements of input devices in an interactive, although not that sophisticated way.

	Neff et al. '09
Efficient	
Intuitive	
Flexible	
Joy of use	2D-Mouse
Relation to my work	Mapping method

Table 2. Reached goals of Neff et al.

Up to this point, all of the stated articles used the mouse as an input device. The next article, called *Motion Doodles: An Interface for Sketching Character Motion*[21] by M. Thorne et al., goes a totally different way. Like in the last work, the animation is created from scratch, and not only post-processed. The interface is "sketch-based" and uses a tablet PC as an input device. The user can draw a line on the tablet, characterizing the animation to be created. Out of this line, the program generates the animation, without any further user interaction. For achieving this, the line is segmented into several parts, and matched to a database. When the segment looks like a slope for example, the program generates a salto for this part of the animation, whereas it produces a normal walking for a straight line or a jump for a bow. The process is illustrated in Figure 3.

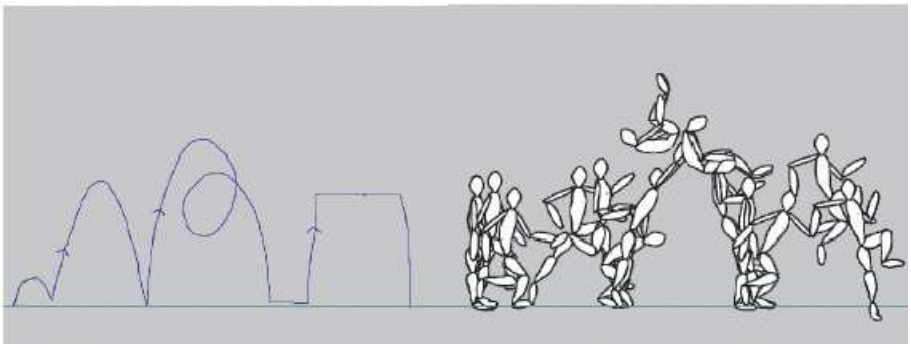

Figure 3. Motion curve and generated motion

This method of creating an animation is also very intuitive, as it is the usual method like people describe a persons movement, for example when gesticulating in a talk to another person. The whole animation with all its body parts is generated automatically, which saves a lot of detail work for the user. On the other hand, the generated motion is fixed and can not be corrected or reworked afterwards. If the generated motion is not the desired one, the user has to start new from scratch. The biggest drawback in my special domain is that only two-dimensional animations can be generated with this algorithm, altough it might also be adapted to a 3D-system.

To check the other goals which I set for my work, we have an intuitive, but also fast design of animations. All the user needs to do is to draw one line, describing the animation. The complete design of the animation and its details is done automatically. The interface is not at all flexible for adapting single body parts. There is no possibility to select single parts of the skeleton, or to draw a motion curve for a specific joint. The system uses a tablet PC, which is a kind of "advanced" single-touch device, but still not the maximum what can be used in input device technology. I integrated the idea of editing and describing a motion using a movement line into my project.

	Thorne et al. '04
Effiicient	
Intuitive	
Flexible	
Joy of use	Tablet
Relation to my work	"Motion line"

Table 3. Reached goals of Thorne et al.

Although this method is very user-friendly, there are still some problems in the flexibility, regarding the adaption of single body parts. The next work, called *Interactive Generation of Human Animation with Deformable Motion Models* by J. Min, tries to solve this drawback, reusing the idea of a motion trajectory.

Like in the first mentioned project, this implementation needs an existing animation, and allows only post-processing of this motion. The software works with 3D-movements, and therefore uses 3D-trajectories to visualize the character's movement. There is exactly one trajectory for each joint of the skeleton, representing the movement of that joint over time. The user can modify one or several trajectories. Upon these constraints given by the user, the software generates a motion, so that all the user-defined curves are matched. This is done by moving the other joints and body parts, on which no specific curves are defined by the user.

Unlike the previous work, every single joint can be animated seperately. As there can be several curves defined in one step, it is possible to perfectly match more than one curve in the animation. The user interface is still intuitive by using the idea of a motion curve. The automatic interpolation of the software saves a lot of time for the user. He does not need to take care of the explicit movement of every joint of the character, he only gives the constraints that are important for him. On the other hand, the modification of the curves is still done with a conventional mouse. There is no discussion about any other input devices, that could further improve the workflow.

To take a look on our checklist, we have again an intuitive device, using motion curves for the visualisation and editing of the motion. The procedure works fast, the user creates only the fixed constraints that he needs, without taking care about the movement of other parts of minor matter. Every single body part can be animated seperately, which gives a maximum flexibility for the user in these terms. Still, the used input device is nothing new and has some capabilities for an improvement. In my thesis, the movements are visualized in exactly the same way, also by seperate motion curves for seperate joints. In contrast to this implementation here, MotionBender does not allow the simultaneous change of several joints, which has to be conducted in several edit steps.

	J. Min '10
Efficient	
Intuitive	
Flexible	
Joy of use	2D-Mouse
Relation to my work	3D-trajectories and editing techniques

Table 4. Reached goals of Min

To come to a conclusion, there is a lot of related work in Character animation with several approaches. One approach which is intuitive, fast and user friendly is the technique of 3D motion trajectories. But still, there is no intuitive and enjoyable way to change or "bend" the trajectory. My work will be based on this technique, and try to improve it using more sophisticated input devices, like view sensors.

The following chapter will give an overview on the central workflow and usage of my tool. Details on the different concepts as well as their implementation will be given in later chapters. I conducted a user study on my software, which tries to solve whether the sensor-based control is superior to a conventional control mechanism, using mice. The last two chapters will deal with this user study, as well as a discussion and conclusion about the implementation and the different input devices.

3 MotionBender: Workflow

The most central part of my work and also my implementation is the workflow which the user has to go, from creating an animation from scratch until the editing of the just recorded motion. The chapter will start with some technical background information needed to understand the chapters later on. View sensors and BVH files will be explained briefly. The second chapter discusses the overall workflow, whereas the third part gives detailed information about the workflow of the editing tool which I implemented. The details of the concepts and background information about used techniques are given in later chapters.

3.1 Technical terms

This section will give a brief introduction to the background topics of this thesis. Details will be given in Chapter 5. Most of the workflow is controlled by a *view sensor*. The first question now is "*What is a sensor-based control*". An optical sensor tries to track the user's position and the position of his body parts. In the end, a complete skeleton, representing the user in front of the sensor, is generated. This data is sent to the application which is in use. With his own body's motions, the user can interact with the application by moving his hands, feet or any other body part. The effect of the motion depends on the application and its design. In my work, the user has an additional wireless mouse in his hands, which allows a faster selection than it would be possible using only the sensor.

Another important term used in my thesis is the BVH file format. A BVH file (BioVision hierarchical file) is a hierarchical file format used to store character animation data on the hard disk. My work uses this format to load and store character animations, as described in next chapters. Now that the important terms are clear, the next section will start describing the big picture of the workflow for my implementation of *MotionBender*.

3.2 Workflow overview

The overall workflow is divided into two steps, as Figure 4 describes. First, the motion is performed in front of a view sensor. This motion is recorded by a software tool, and stored inside a BVH motion capture file. My implementation loads this file, and allows a postprocessing of the motion.

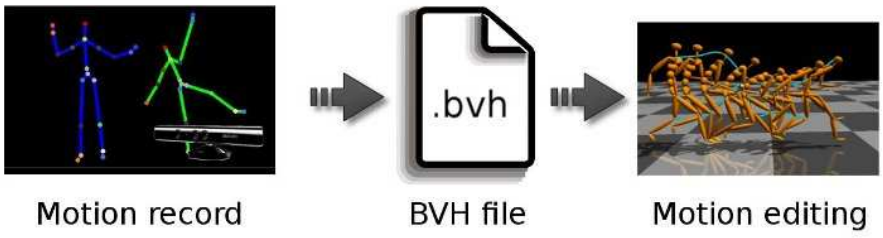

Figure 4. Workflow steps

The first step (the motion recording) is done by an external tool called "Brekel Kinect"[2]. Figure 5 shows the main screen of this tool. Brekel is connected to the view sensor via the OpenNI and NITE framework. Technical details on the frameworks are given in Chapter 5. In the lower left corner, the standard camera image is shown. The lower right picture is a visualisation of the depth information gathered by the sensor, plus the recognized person and its joints. In the main screen in the center, this joint information is projected on the optical image.

2. http://www.brekel.com/?page_id=155

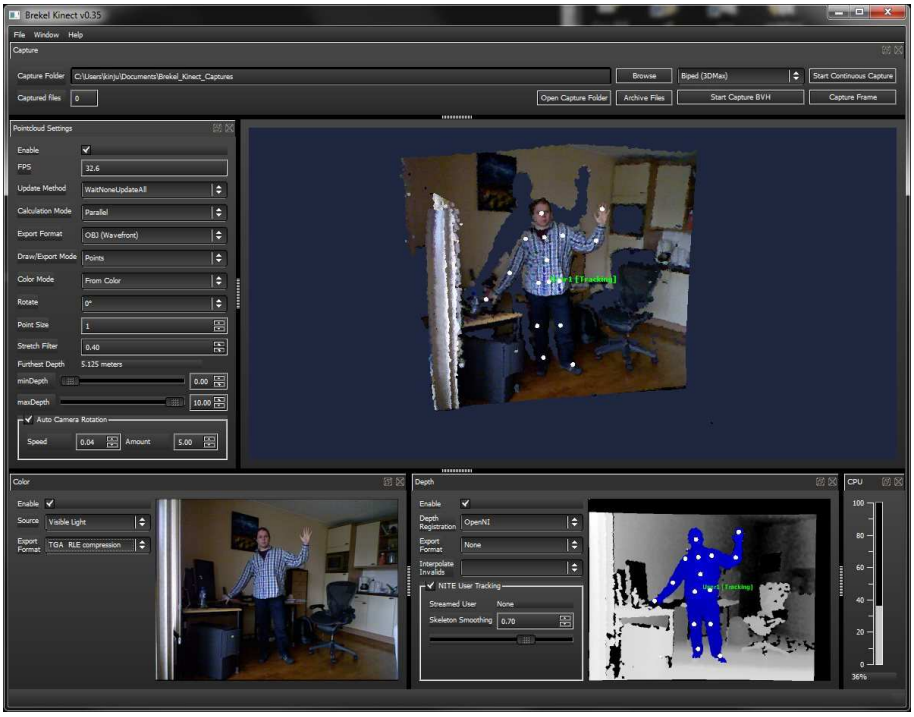

Figure 5. Mainscreen of Brekel Kinect

For using the program, the person first needs to be calibrated. Therefore, the user needs to stand in front of the camera in a psi-like pose for a certain time. When the calibration was succesful, Brekel shows the skeleton as an overlay in the main screen. For recording the BVH file, the button "Save as BVH" has to be clicked once.

The motion editing step is implemented by my software, called "MotionBender". This software reads the stored BVH file and allows the editing of the motion in again several phases. These phases will be treated in the oncoming chapter.

3.3 Workflow of the edit phase

Figure 6 illustrates the workflow of the edit-phase. The phases are implemented by the class "*KEController*", as decribed in Chapter 4. During each step, the user is informed about the actual phase via on-screen instructions.

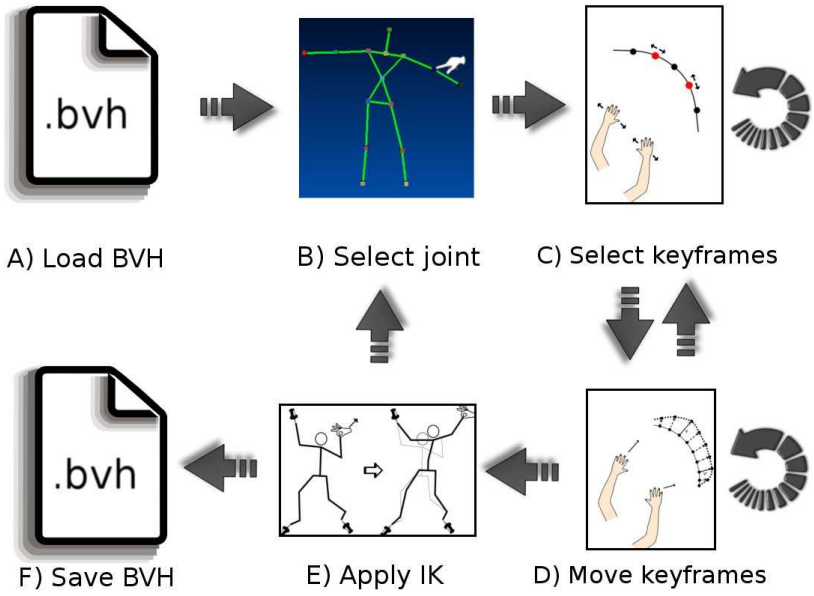

Figure 6. Workflow of the edit phase

The workflow starts with phase A) and continues to the next phases as indicated by the arrows. After selecting the joint which the user wants to modify (phase B), the main editing takes place in phases C and D, where the keyframes to move are selected, and finally moved to the desired position. This procedure of selecting and moving can be repeated infinitely many times until the desired result is reached. Afterwards, the character is automatically fitted to the new motion curve. The user has then the possibility to either continue with another joint or to store the new motion in a BVH file.

The interface in which this workflow is taking place is shown in Figure 7.

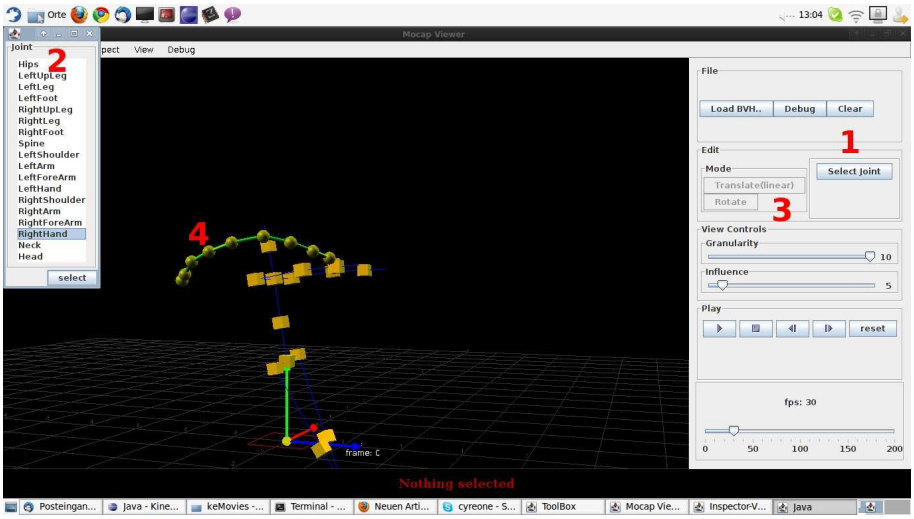

Figure 7. The main window of MotionBender

The middle of the window illustrates the skeleton of the character as well as the motion curve for the selected joint (the right hand in this example). The palette at the right contains several controls for playing the animation, positioning the camera as well as parameters for the editing, as explained in later sections.

Load BVH (A)

This phase is the first step which needs to be performed after startup. Here, the file created with Brekel Kinect can be selected and loaded. Upon this time, the character motion is displayed in the program and the motion can be played using the controls at the right.

Joint & mode selection (B)

The second step is to select the joint whose curve needs to be changed. After clicking on "Select joint" (right to 1. in the illustration), an additional window (3) is opened containing the joints which can be selected for editing. After the joint is selected, the corresponding motion curve (4) is displayed in the main screen. For starting the actual edit phase, the user clicks on one of the mode buttons (below 1. in the figure), depending on the desired edit mode. Details on edit modes can be found in 4.3.

Point selection (C)

Next thing to do is select the keyframes to move, e.g. the "balls" on the curve which the user wants to "grab" an move into the desired position. For this, three things need to be done. First, the hands need to be hold into a distance of about 1m. For starting the selection, the user clicks and holds the left mouse button. The two outmost balls are selected. By moving the hands together, other balls inbetween can be selected instead.

It is important to move the hands only horizontally together. Moving them vertically does not have any effect. Therefore, following the shape of the curve while selecting the balls, instead of moving the hands horizontally, can lead to unexpected results. For finishing the selection, the left mouse button is released. Upon this, the point selection remains fixed until the user starts a new selection phase. This procedure can be repeated as many times as needed to select the right keyframes. If the selection is okay, one can continue with the actual moving phase (D).

Point moving (D)

The same paradigm as for selection works for the moving of the points. When pressing and holding the right mouse button, the selected points are "attached" to the user's hands and follow the user's hands' movement. In addition to the "grabbed" points, the neighborring points are also moved. The distance and direction depends on a function, called the "neighbour function". Details on the used function and its implementation is given in Chapter 4.2. Just as the selection, this step can also be repeated as often as needed for getting the desired result.

Generation of the movement (E)

After the user has done its modifications, it is upon the software to modify the animation so that the newly created curve is matched. It is a nontrivial problem to reach a target within a skeleton, as not only the last joint, but also other joints inside the skeleton have to be moved to reach a target. This problem is called "inverse kinematics". Details on the technique and its implementation is given in Chapter 4.5.

After the program generated the new motion, the user can either continue with step B) or save the generated animation (F).

Save BVH (F)

At the end, the motion can be saved by clicking the corresponding button above 1) in Figure 7. This opens a dialog where the place to write the file can be chosen. The generation of the file out of the internal data structures is described in detail in Chapter 4.6.

Now that the overall workflow is clear, the oncoming chapter gives more details on the concepts and techniques used in the workflow.

4 Concepts & Implementation

The most central concept is the motion path, as described in the related work and [14]. The motion path visualises the movement of a joint over time. Figure 8 shows such a motion curve.

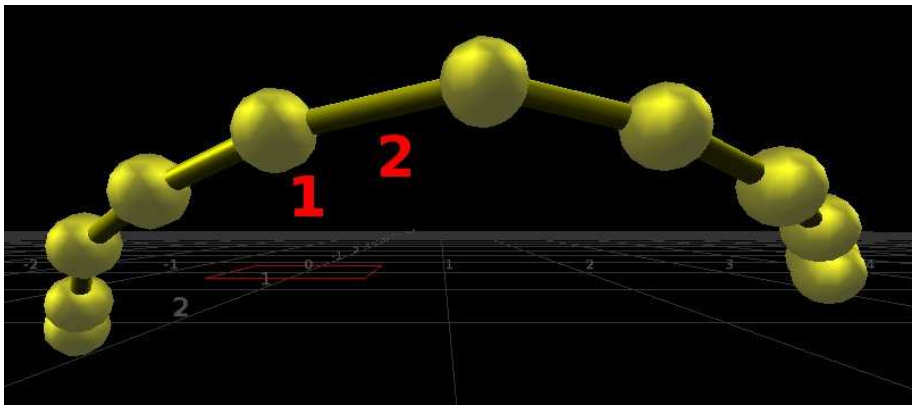

Figure 8. Motion curve

A curve consists out of connected balls (1). Every ball represents the position of the joint in a certain frame of the animation. For a better visualisation of the movement, consecutive frames are connected by a line (2). Details on the implementation are given in 4.1.

For editing this curve, a "neighbour function" (Chapter 4.2) is used, which also moves the neighbouring points in addition to the selected points. The neighbour function calculates the distance according to the direction and length of the original movement vector, whereas the size of the vector depends on the distance to the originally moved point. The actual behaviour of the neighbour function depends on the implementation, as described later.

There exist several parameters to customize the appearance of the motion path and the behaviour of the neighbour function (Chapter 4.3). With the granularity slider, the user can determine how much balls should be shown on the motion path. The higher the granularity, the fewer points are shown on the path. With this functionality, the user can get a better overview on the motion path, if the dots are very dense in the standard granularity setting. The intensity slider manipulates the behaviour of the above stated neighbour function. The higher the intensity, the more of the neighbouring points are withdrawn. The ongoing chapters will describe these mechanisms more detailed.

4.1 Motion paths

The concept of motion paths has been examined in several articles [12][14]. For every joint, there exist exactly one motion path describing the movement of the joint over time. Each frame in the animation is represented by one ball on the line.

In MotionBender, this concept is implemented by two classes: *WayPoint* and *WayTrace*. Each instance of *WayPoint* represents one point ("ball") on the line containing the graphical representation as well as the data on the corresponding frame. For example, the 3D-position of the curve (which is also the position of the joint in this certain frame) is also stored in the data structure. *WayPoint* offers functions for moving the point to a new location, for example when the position has beend modified by the user. Each WayPoint has a pointer to the next ball on the curve.

In addition to the ball, the class also contains the connection line to the next point on the path. Using the pointer to the next ball, the class computes the right position and angle out of the ball's coordinates and aligns the line so that it perfectly connects the two points. Also graphical properties like marking a point as selected, transparency, and color are directly set on the *WayPoint* class.

The *WayTrace* class represents the complete curve. It is mostly a container for a set of instances of *WayPoint*. Nearly all procedure calls to *WayTrace* like selection, or graphical properties are delegated to the contained WayPoints. If a movement is done by the user, the class is responsible to calculate for every WayPoint the actual movement vectors according to the *neighbour function* used (see next chapter). Then, this vector is applied to the WayPoints.

4.2 Neighbour function

As described in previous chapters, not only the points grabbed by the user are moved. Also points nearby are moved, according to the distance to the originally moved points. Figure 9 illustrates this effect. The WayTrace class calls an implementation of the *NeighbourFunction* interface. The Neighbourfunction distinguishes 3 different types of points. Let's say the points P_1 and P_2 have been moved by the user by vectors $\vec{V_1}$ and $\vec{V_2}$, respectively. Then we have these 3 different types of points on the curve:

1. Points between P_1 and P_2 are moved with an interpolated vector of $\vec{V_1}$ and $\vec{V_2}$. Points nearer to P_1 will have a higher rate of $\vec{V_1}$ in the resulting vector, and vice versa
2. Points outside the area of $P_1 - P_2$ are moved with the same direction as the vector of the nearest selected point. The length of the vector decreases with distance to the selected point.
3. Points which are more far away than M are not moved

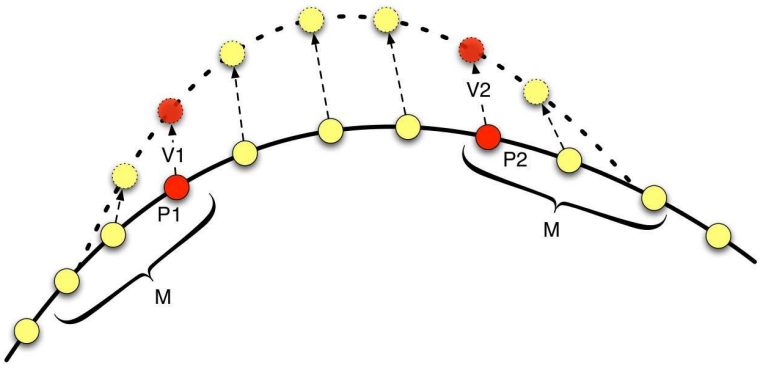

Figure 9. Effect of the neighbour function

On the implementation side, there is an interface *NeighbourFunction* and one implementing class called *LinearModifier*. The interface supplies two functions:

1. *getMovementVector* to calculate the movement vector for each point on the motion path, given the selected points and the original movement vectors
2. *setInfluence* for setting the *intensity parameter*, as described in 4.3. This parameter defines the number M of neighbouring points which should be moved within the originally moved point

My software uses the *LinearModifier* as an implementation. In this implementation, only a fixed number M of neighbouring points are influenced by the function. This number M is set by the intensity parameter. Upon this fixed number of points, the function outputs a vector of the same direction as the next original movement vector, with a linear decreasing length. The length is zero on all points more far away than M.

4.3 Edit modes and parameters

There are several properties and options which can be changed inside the MotionBender tool. First, there are two seperate edit modes implemented.

In the *translation* mode, the user can freely move the selected balls without any restrictions. The hands' movement is mapped directly to a movement of the points. It is a kind of "freestyle" mode. In the *rotation* mode, the program keeps the distance between the two selected points constant. The movement is limited to a single rotation, or can be moved together to another location. The user can not change the points' distance by mistake. The calculation of the rotation and the switching of the edit modes is done in the input system. The input system is designed by an external tool, which supports the interactive editing mode by a graphical user interface (see Chapter 4.4). Implementation details on the rotation mode and the concrete behaviour are also described there.

Besides of these fundamental edit modes, there exist two parameters for tuning the behaviour of the tool. They can be manipulated by the sliders on the righthand side of the main window, as show in Figure 10

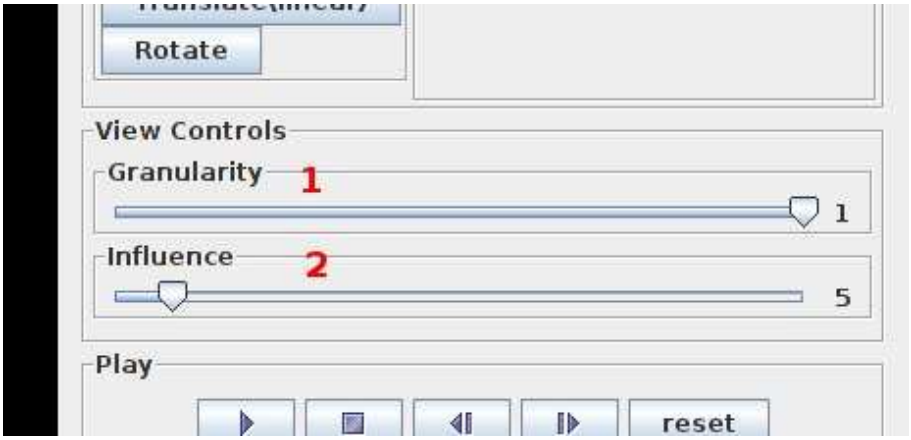

Figure 10. Edit parameters

The first parameter is the *intensity* parameter (marked as 1. in the illustration). As already stated before, this parameter allows the user to change the intensity of the neighbour function. In my implementation, this corresponds to the number of neighbouring points to the selected point, which should be influenced by the function. Using the slider, this parameter can be changed during run-time. The more the slider is dragged to the left, the fewer is the influence and the less neighbouring points are moved. Respectively, the more the slider is dragged to the right, the more points are influenced by the neighbour function. On the implementation side, this parameter is realized by a function call on the *NeighbourFunction* interface, whose implementation stores the value and uses it in its computations.

Another editable parameter is the *granularity* (number 2. in the illustration). The granularity influences the graphical representation of the curve. It determines the amount of balls shown on the motion path. Per default, every frame has its representing ball on the curve. This corresponds to a granularity of 1. If the granularity is increased, then some of the frames are not shown on the curve. For example a granularity of 5 means that only every fifth frame has its representing ball on the curve. All other frames are ignored on the graphical representation. But still, a modification on the curve is applied to *all* frames, not only the visible ones. The implementation of this parameter is done by a method inside the WayTrace class. The function manipulates the list of contained instances of WayPoint, according to the selected granularity.

4.4 Input system

The input system is designed within a data-flow diagram. A tool called "TouchFlow"[3] reads and executes this diagram. Touchflow has been constructed in association to a bachelor thesis ([19]) two years ago. After this, the project was further developed by Frederic Raber at DFKI. It is now an open-source project hostet at google code under the LGPL license. The tool reads the data from the needed input devices like mice or a Kinect device, and provides the rehashed data to the main application. The data-flow graph for the Kinect interface is shown in Figure 11.

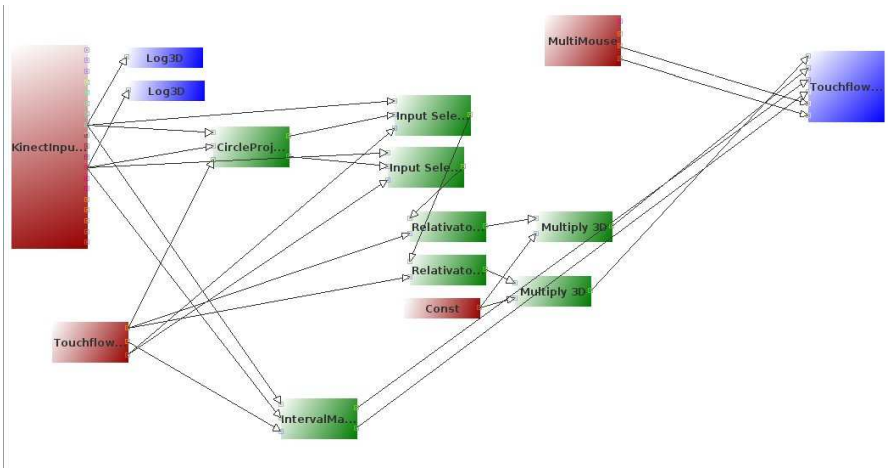

Figure 11. TouchFlow graph for the Kinect interface

In this graph, each of the coloured blocks represents a "module". A module is a unit which receives input data, modifies it, and passes it to the next module. The main application has access to exactly one module called *TouchflowOutput*, marked by the 1) in the illustration. The data which flows inside this module is received by the application, and processed afterwards. On the other hand, the *TouchflowInput* module (2) allows the application to send data inside the flow graph. With this module, the application is able to change the behaviour of the input system, for example when another edit mode is selected. This is achieved by the data flowing from the TouchflowOutput module to other modules inside the graph, which change their behaviour upon this new input.

Briefly, this input system works as follows:

- The module "Kinect" emits the position of the two hands
- The Kinect information is passed to the "IntervalMap" module at the bottom. This module is responsible for the selection phase. Depending on the hands' position, it outputs the index of the point on the curve which is selected, and again passes this information to the main application through the blue interface module
- The middle part is responsible for the moving of the points, corresponding to the selected edit mode. The selection of the edit mode is done by the *InputSelect* module. Depending on the input from the application, it either passes the unmodified hands' positions (translation mode), or it uses the modified data from the CircleProjection module (rotation mode)
- In both cases, the position is transformed into a relative position, relative to the starting position of the hands. This position is then passed to the main application
- In the upper section, the *MultiMouse* module is responsible for detecting the mouse clicks. As stated before, the selection and moving of the points takes place when the left or right mouse button is pressed down. The MultiMouse module detects this, and sends the button states directly to the main application, without any further modification

3. http://code.google.com/p/touchflow

The next subsection describes in detail which modules have been implemented, and how they work in detail.

Implemented Touchflow modules

In detail, there have been several modules which have already been implemented by the Open-Source community. But still I needed some additional functionalities, which have been implemented in the following new modules:

Kinect

- **Task:** Generates the positioning information from the view sensor
- **Output:** Two double3D determining left and right hand position

CircleProjection

- **Task:** Modifies two given input positions, so that their distance is equal to the distance of the reference points, as illustrated in Figure 12.

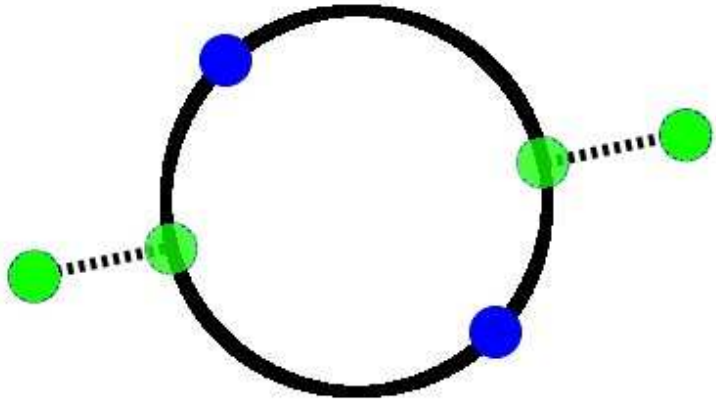

Figure 12. Functional principle of the CircleProjection

In the picture above, the blue points are the reference points. The input positions (green) are projected onto the circle (black) defined by the two reference points, so that the green points have the same distance to each other as the blue ones do

- **Input:** Two double3D positions (POS1 and POS2) to project, one double input (CALIBRATE) for switching calibration phase on(1) and off (0). The reference points are set as follows:
  1. Set CALIBRATE to 1
  2. The positions at POS1 and POS2 are now set as the reference points
  3. Set CALIBRATE to 0

- **Output:** Two double3D positions, determining the projected positions

InputSelect

- **Task:** Outputs one of the two input values depending on the value of the "select" input
- **Input:** Two double3D positions, one double value for selection
- **Output:** double3D containing the selected position, which is the first position if "select" is 0, and the second position otherwise

Relativator

- **Task:** Outputs the position, relative to a reference position, as described in Figure 13.

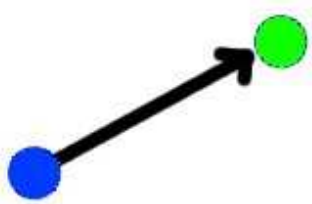

Figure 13. Functional principle of the relativator module

If the blue point is the reference position, and the green one the input position, then the output is the difference between those two positions, determined by the black arrow in the figure.

- **Input:** Double3d point, which needs to be transformed to a relative position, double for setting the reference point. The reference point is set like in the CircleProjection module.
- **Output:** Double3d, determining the relative position

Multiply3D

- **Task:** Multiplies a given position by a scalar
- **Input:** Double3D determining the position to multiply, double determining the number to multiply with
- **Output:** Double3D, the multiplied position

IntervalMap

- **Task:** Takes the hands' position, and maps it to a selected point
- **Input:** Two double3D, the hands' positions
- **Output:** Two double, the indices of the points to be selected

4.5 Arm control

After every edit phase, the actual arm movement for the new motion curve is automatically generated by the program. It is not a trivial problem to reach a given point with a body part. Let's say you want to reach a glass with your hand. In most cases, it is not enough to only move your wrist to reach the target, as the radius of movement is limited. You also have to move your elbow, maybe even the complete arm for being able to reach the point. But the movement of the arm or the elbow influences the position of the wrist, too. This leads to a combinatorial problem.

In computer science, this problem is called the problem of *inverse kinematics* (IK). There are several technical terms used in this context:

- The position that has to be reached, is called *end position*
- The joint which needs to reach that position, is the *end-effector*
- The list of joints which are allowed to be modified in order to reach that position, is named *kinematic chain*

Formally, for a given end position of a joint (end-effector), inverse kinematics calculates the positioning of all other joints (kinematic chain), so that the end-effector reaches the end position.

My implementation uses the technique of cyclic coordinate descent (CCD). CCD was first mentioned in the Master thesis of Chris Welman [23] in 1993 and is a cyclic heuristic algorithm. Its workflow is shown in Figure 14. The picture has been extracted from a master thesis by Quan Nguyen [16].

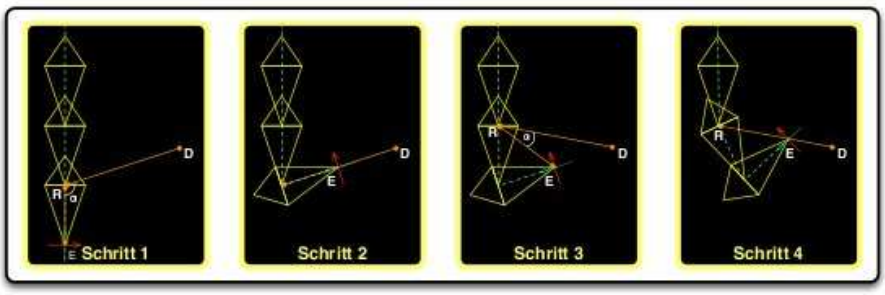

Figure 14. Workflow of CCD

The general workflow is as follows:

- In the first step, the first joint in the kinematic chain (usually the joint next to the end-effector) is rotated, so that the distance to the end-position is minimized
- In the second step, the same is done with the second joint in the kinematic chain, leaving the rotation of the joint in the first step unchanged
- This procedure is done with every joint in the kinematic chain
- When all joints in the list are processed, this procedure is repeated until either the end-position, or a certain number of cycles is reached

Note that not every end-position can be reached, as the skeleton root must not be moved to another location. For example, some points can be too far away to be reached by the skeleton. In this case, the algorithm terminates after the given amount of maximum cycles and returns an error code. Another important term in CCD are the *joint weights*. Instead of rotating the joint directly into the direction of the end position, it can also be rotated only a part of the angle into the right direction. This prevents the algorithm of acting too fast in one step and to overshoot the mark. A weight of 1 means that the full angle is applied, whereas for example a weight of 0.6 states that only 60% of the full angle should be used.

My project reuses a CCD implementation of another project called *IVEE*. IVEE was developed by Michael Kipp at DFKI. I reused the data structures, and adapted them so they work with my internal data structures. In my implementation, after every edit step, the program iterates over all frames, and applies IK on the end-effector with the new end-position. The kinematic chain reaches from the end-effector until the last joint before the hip (root). After loading the BVH file, my implementation keeps the animation data of the file in the main memory. After IK is applied and the rotation angles are changed, the program updates this "BVH data" so it is always up-to-date. The advantage of this technique is that it is directly possible to export the animation to a BVH file. But this makes it also impossible to "undo" a modification, as the original data is overwritten in every step. Details will be given in the next chapter, which treats the export of the generated motion.

Technically, the IK functionality is implemented by its own static class *CCD*. The class takes a list of bones, representing the kinematic chain, and a float array of the same length determining the corresponding *joint weights*. The function then accesses and modifies the rotation matrices of the joints, and applies the steps of the algorithm as stated above.

4.6 Export of the generated motion

After the user created the motion, he needs a functionality to store and export this motion to other programs for further usage. Therefore, MotionBender has the ability to export the character animation to a BVH file. In contrast to the loading of such a file, this functionality was missing in the original project and implemented by me afterwards.

Upon the successful loading of a BVH file, this raw animation data is kept in the memory. After every completed edit step, MotionBender updates these raw data out of the new rotation angles of the character. This makes sure that the raw data is always up-to-date, and can be directly exported. For converting the rotation data of the skeleton to BVH-conform animation data, I ported some functions of the *Horde3D* SDK[4] to java. The main goal of the function is to split the rotation about some arbitrary axis into three rotations about the three main axes (pitch, yaw and roll).

The functionality is implemented by a *BVHWriter* class. This class first takes the internal data structures, containing the actual motion capture data to be exported. This data is the BVH-affine data structure as described above. In the next step, the method *createFile* is called which iterates over these data structures, and accumulates the lines for the BVH files in a String variable. Finally, a call to *writeFile* writes this String to the given file.

This chapter described the implementation details, concepts and how these concepts are realized inside my application. The next chapter will now give some technical background information about the used hardware and software tools to get a deeper understanding of the components used for building my software project.

4. http://www.horde3d.org/

5 Technical details

The most important technical detail in my work are the sensor based devices. The first sub-chapter will discuss the functional principle of the device, the design of the drivers and the software development kit, as well as the software processing the raw input from the camera. The realisation of the alternative two-mice input system will be discussed.

My software implementation is based upon an existing project called "JMocap", developed by Michael Kipp at DFKI. The second sub-chapter will give details about the project, its current state and which additional functionalities had to be implemented by me in order to receive my software solution "MotionBender".

Finally, the third and last sub-chapter gives some background informations about the used motion capture file format, as well as alternative formats with their advantages and disadvantages. For a sensor-based interaction, we need two important things: First we need the hardware which is generating a depth-view snapshot of the scene containing the user, as well as the device driver which makes the sensor available to the application developer. This is described in the "hardware" chapter. Second, we need the software which takes this raw data and postprocesses it. There are two steps of post processing needed: First, the user's skeleton is reconstructed from the raw camera data (see 5.2.1). And second, this skeleton data is interpreted to the respective gestures by the MotionBender application, as already discussed in Chapter 4.4.

5.1 Hardware

Besides of using a view sensor like in my implementation, there are several other methods to control a character animation application. An approach which is used often at the moment is control by a single mouse. This type has several advantages:

- Nearly every computer has a mouse attached, the software can be installed and used directly on most machines
- Users are used to this "conventional" device, as they normally use it several hours every day. There is no need to understand and get a feeling for a new input scheme

But sensor-based devices have advantages over conventional devices:

- Unlike mice which have only 2 degrees of freedom (DOF), a depth sensor has 3 DOF. This allows the user to control a direct 3D-manipulation. The movement of the user can be directly mapped to a movement in the scene. On a mouse interface, this third DOF has to be interpolated, for example by using the scroll wheel
- By this direct and intuitive manipulation, it is easy for beginners to get used to the "new" interface. They can just act as they would do in the real world. Microsoft's games for the xBox Kinect are a good example, where you can play some sports games with exactly the same movements as in reality.

Besides of the conventional mice, there are mice with more DOF, called 3D-mice. The figure below shows such a device produced by "3DConnexion" .

Figure 15. 3D mouse, 3DConnexion Space Pilot [5]

This "Space Pilot" called 3D mouse has a cost of about 350€ currently, but there are also some slighty cheaper models at about 200€. The most important part is the cap in the middle of the device. It can be pushed, pulled, twisted or tilted to simultaneously pan, zoom and rotate a 3D-model in the environment for example. There is some recent work [4] which shows the leveraged efficiency of the workflow, when such an "advanced" mouse is used instead of a conventional one. Also the time to become a proficient user in this device is discussed in the article. To summarize their findings, it still needs about 1 week of intensive daily usage to be become proficient for most of the users. This last remaining problem could be solved by an intuitive sensor-based interface. The ongoing chapter will describe what in detail is a view sensor and which implementing hardware currently exists on the market. Pros and contras about the different types of devices are discussed.

Sensor-based control devices

For a sensor-based control device, the objective is to track the user's position and the position of his body parts. In the end, a complete skeleton, representing the user in front of the sensor, is generated and sent to an application. The user can control this application using his body, without holding or operating on any additional device. He controls the behaviour of the application moving his hands, feet or some other body part. There are several approaches to realise this idea of a non-intrusive control.

A first approach is *optical motion capture*, as illustrated in Figure 16. The central techniques are presented in [6], skeleton reconstruction is described in [10]. The user wears a suit with optical markers, marking the different body parts that need to be tracked (left picture). A webcam is recording the user inside the scene, and also the markers on his suit. A software can detect the markers on the images recorded by the webcam (second picture). Out of these marker positions in the picture, the software tries to calculate the 3D-position of the different markers, which corellate to the positions of the tracked body parts.

5. image source: http://www.techcinema.com/2011/12/3dconnexion-spacemouse-pro.html

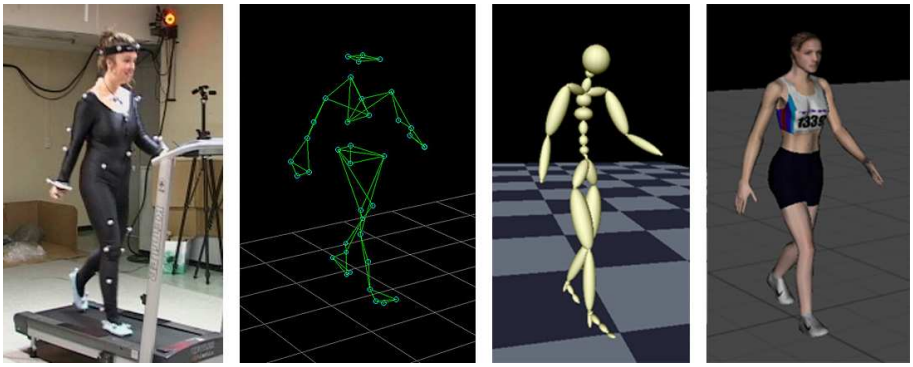

Figure 16. workflow of optical motion capture [6]

Another solution are *inertial sensors*. In contrast to the previous approach, no markers or cameras are needed. Instead, inertial motion sensors are placed on the body of the user. They can detect and measure accelerations and rotations performed by the user. In most cases, these sensors are integrated inside a wearable suit. Such a suit is shown in Figure 17.

Figure 17. Suit for inertial motion capture [7]

The movement detected by the sensors are sent to the PC via a wireless connection. Additional software can calculate the user's position and pose out of this raw sensor data. A short overview on how these calculations are done is given in [25]. The suit shown in Figure 17 is the XSens MVN[8]. These devices are quite costly, one MVN suit is about 40000€.

6. image source: lukemccann.wordpress.com

7. image source: http://www.xsens.com

8. http://www.xsens.com/en/general/mvn

A third and also markerless approach is the *time-of-flight camera*. This technique again works with view sensors. There is one device with two important components in it. An *infrared sender* sends out infrared light towards the scene. The *infrared sensor* is captchuring the light reflected by the scene. Out of the time-difference between sending and receiving the infrared rays, a software can calculate the time and length the light had to travel. This gives the distance to the next solid object at this certain point in the scene. Technical details are described in [5]. An implementation of this technique is the *Microsoft Kinect*[9] sensor. Kinect uses some special technique to distinguish which infrared-ray came from which position in the scene: Every ray sent out to the scene has a different pulse signature. When the ray is reflected by the scene, and received by the sensor, the software can recognize this signature and derive the scene position from it.

Originally, Kinect was meant to be proprietary hardware, only to be used in connection with an XBox360. There were no drivers or software development kits given for download by Microsoft. Today, there exist several SDKs for Windows and Linux. Also Microsoft published its own Kinect-SDK recently. My project supports both the Microsoft as well as the PrimeSense SDK. The latter one was published by the manufacturer of the Kinect sensor and one of the first official development kits. Details on this software will be given in 5.2.1. There are several pros and contras for both existing implementations. The MS version needs no calibration, the user is directly tracked. PrimeSense's version needs the user to stand in a specific pose for a certain amount of time, until tracking is active. Microsoft has integrated a fully automatic installer package, which needs no further configuration or other installed software in order to work. But still, the PrimeSense SDK provides a better and more precise tracking of the person and a higher stability. The latter fact was the reason why this SDK was used inside the evaluation of my program.

Out of the skeleton information given by the sensor software, I use both hands' position to control my application. All other information is discarded. Along to this control mechanism using Kinect, I present a second, alternative control method. In this method, two mice are used instead of two hands. Each mouse represents on of the user's hands.

Two Mouses in parallel

The conventional method in character animation is to use one mouse to modify the scene. Some of the professional users also use a 3D-mouse, as already stated in earlier chapters. Such a 3D-mouse is often combined with a conventional (2D-) mouse, where the user holds the 3D-mouse in his main hand. Recent work has shown that the parallel use of two input devices, one for each hand, can significantly improve the workflow[7][13]. In addition to the time saved by using two devices at one time, the cognitive load is also reduced. The user does not need to split the complex task into several steps. There is also some related work in manipulation in the 3D-environment [26] which tries to provide a more intuitive and natural user interface by using two mouse pointers at one time to rotate or manipulate 3D objects. Bimanual curve manipulation is discussed in [17]. The two-handed curve manipulation provided a lot shorter task completion time, because of both enhanced time motion cost and cognitive benefit. Another important aspect in this context is the parallelity of the actions. Especially when using two mice, users tend to use them sequential, e.g. first use one mouse, and afterwards the second mouse. There is nearly no benefit of using a second device if they are used in this way. The more optimal way is to use both hands at the same time. [3] discusses this problem and how the parallelism could be improved. A visual connection of compound movements increases the parallelism of the tasks, for example.

In my work, each mouse has the same task as one hand in the sensor. The left mouse substitudes the left hand, the right mouse the right hand. Figure 18 visualises the design of this interface.

9. http://www.xbox.com/de-DE/Kinect

Figure 18. Two-Mouse interface

Movements of the mouse are mapped to the X-Y plane. Movements in the Z-plane are emulated by the scroll wheel. Scrolling the wheel towards the screen means a movement of the point towards the screen, and vice versa. The control was technically realised by using the XInput toolkit. XInput allows the runtime test and configuration of input devices like mouse and keyboard via the command line. It is preinstalled on most *Linux* distributions. It makes it possible for me to create a second mouse cursor, attach a device to it and to gather the position data of both mouse pointers. This is done by several shell scripts, which can be found in the appendix (8.3). I have two scripts. One for creating the second pointer and attaching the second mouse to it, and for rolling back these operations. Second, the script which tracks the mouse position and sends the data via telnet to the destination host.

The next step after gathering the data from the input devices is the post-processing of this raw data and finally the presentation to the user. The next chapter describes these processing steps, and shortly introduces the software toolkit that served as a visualisation basis for my implementation.

5.2 Software

There are several software libraries and tools integrated in my project, which will be described here. For my task, I need several components. The chapter starts with the first one in the workflow, namely the post-processing software. This software takes the raw sensor data from the driver and generates the user pose and the position of his body parts out of it. In my approach the data is recorded and can be modified afterwards. For being able to modify a character animation, we first need to visualize it. The software which I re-used for this purpose is described in the second part of this chapter. The third component which is needed is the permanent storage of the generated motion. For this, I needed an implementation of the BVH file format. The last chapter discusses several important file formats as well as its advantages and drawbacks, and the reason why BVH is used in my implementation.

5.2.1 Post-Processing software for the sensor device

There are several libraries which can control the Kinect camera, which will be discussed here. They are not only different in the programming interface or the driver used to control the device, but also in the way the data is processed. The two main frameworks are the OpenNI and Nite framework, published by PrimeSense[10], and the Microsoft Kinect framework. The first was one of the earliest public implementations and the first official framework. The MS framework has been published recently, and contains the original algorithms used by the XBox to track the user.

The PrimeSense framework uses the "traditional" method, as described in [18][2]. In this method, the first step after startup is always a calibration phase, where the user has to stand in a special pose. For our framework, this is a "Psi" pose, as shown in Figure 19.

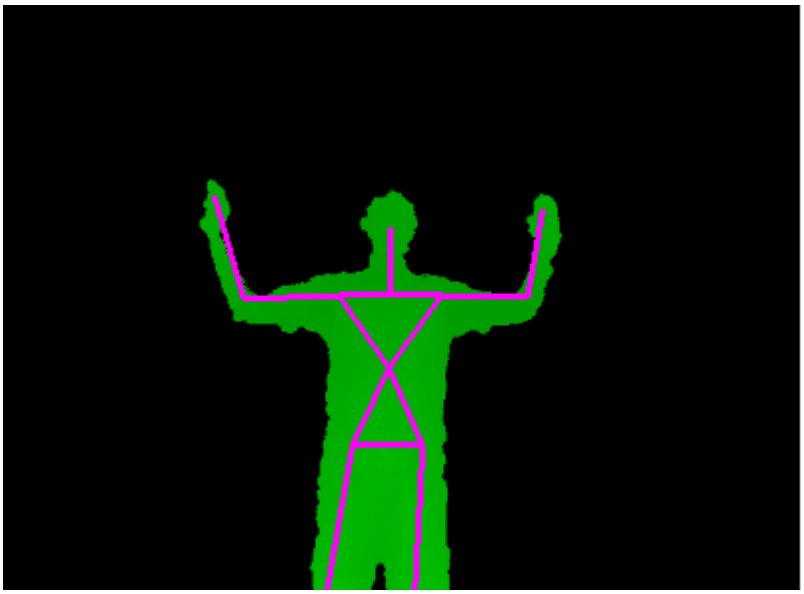

Figure 19. Psi pose for calibration

After some seconds, the software detects the user, which is visualized by the purple skeleton in the illustration. As this pose is known to the algorithm, it can match the depth information from the sensor to the corresponding body part, according to this "hardcoded" scheme. The software can find the first pose and position of the user in this way. After the tracking was succesful, the program is able to track the user.

10. http://www.primesense.com/

For tracking the user also in other poses after this initial step, the algorithm relies on one assumption: *Between two consecutive frames, the body parts can not move far away from the position in the last frame.* Therefore, the pixels of the new frame are assigned to the corresponding body parts, so that their distances stay minimal. In this way, every new frame is processed based on the informations of the last frames, especially the position and assignment of positions to the appropriate body parts.

The advantages of this method are several:

- The method is very efficient, there is not much computing power needed to process the frames
- As the algorithm assigns the pixels so that the distance stays minimal, the results are very smooth and precise. There are no single frames with wrong information in-between, e.g. no "flickering"

But still, there are also two negative aspects:

- There is a calibration phase needed in the beginning. This can be problematic if users change often, for example in a public environment. This fact has also to be learned by the users first, before they are able to use the device.
- If the user or most of his body parts have left the field of view of the camera, they can hardly be tracked again after they re-entered the scene. This case can happen if some other person occludes the view of the sensor, or the user unintentionally leaves the scene while acting

The Microsoft framework goes another, more powerful through simple method [20][28]. In contradition to the traditional one, every frame is analysed independingly. The information about the position of the user and its joint is discarded in every frame, and built up from scratch. The core idea is that out of the depth distances of neighbouring points, one can infer the body part to which the point belongs to. In other words, the "skin structure" of the object is matched to known skin structures inside a database. This is a kind of machine learning algorithm. In detail, the algorithm was developed as follows: For each pixel x, a "feature" is calculated using the formula

$$f(x) = d\left(x + \frac{u}{d(x)}\right) - d\left(x + \frac{v}{d(x)}\right)$$

where (u,v) is a pair of displacement vectors, d(x) the depth at position x as displayed in Figure 20.

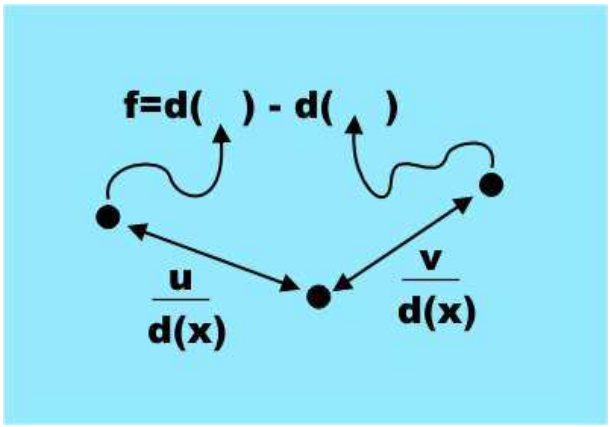

Figure 20. Feature calculation

Informally, this calculates the depth distance of two points nearby the considered point (x). Using a database containing depth images, where the body parts have been marked manually by a developer, the algorithm was trained in a machine learning process. This trained algorithm was then integrated into the XBox firmware, and later ported to the Microsoft Kinect SDK.

There are also several advantages and drawbacks on this alternative approach:

- The biggest improvement is that there is no calibration phase needed, as there is no "inductive" design which needs a first frame to rely upon. The user can be directly tracked after startup, without needing him to stand in a special pose
- This results in the ability to track lost body parts or complete persons, which have been occluded by some object or which left the scene. The software can directly find the person if he re-joins the scene
- The drawback about this method of independent frame analysis is that there is a higher chance of outliers and misdetected body parts, as there is no "stability constraint" like in the first approach, which tries to keep the movements as small as possible

In the end, both frameworks make the gathered information available to the developer via a pre-defined API, written in C++ with classes. As both of them only support C++ as a devleopment language, whereas my core application was made in Java, I created a small client which takes the position data, and sends it through a socket to the Java application.

The next step is to visualise this motion to the user. Therefore, the next part of the chapter talks about the motion visualisation software, which served as a basis for developing my application.

5.2.2 Visualisation software for Motion capture data

The core of my implementation is the *JMocap*[11] project. JMocap is a Java-based open source viewer for motion capture data and was developed by Michael Kipp at DFKI. A screenshot of the tool can be found in Figure 21.

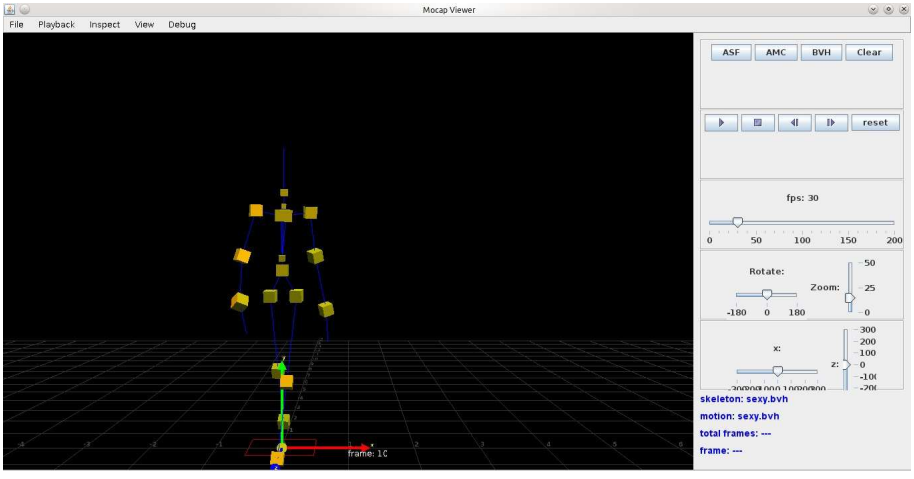

Figure 21. Screenshot of the JMocap tool

The user interface of JMocap is divided into two parts. In the area on the left side, the model is shown and animated. The right side contains several toolbars for loading an animation, starting the animation, and for changing the viewpoint on the model on the lefthand side.

11. http://www.michaelkipp.de/de/forschungsprojekte-und-publikationen/software/47-allgemein/91-jmocap

JMocap already brings some capabilities which I needed for my work:

- The loading of character animations in BVH and acclaim file format is supported
- The skeleton is visualised inside an edit-friendly surrounding, containing rasters and other visual fixpoints for an easier orientation in the 3D-environment
- The animation can be played
- There is a possibility to rotate and move the camera inside the scene, to have an optimal view on the character

I re-used this software and extended some new features:

- Visualisation of the *motion path*, as well as the needed functions for modifying this path
- Generation of a new motion (IK), following the new, edited motion path
- Saving of the generated motion, e.g. the export of the motion to a new motion capture file as described in 4.6
- JMocap came with a mouse-based control interface. I exchanged it to my sensor-based interface and an additonal interfaced based on two mice. The interfaces can be changed in the source code using a switch

The last part will discuss the possibilities for storing the motions. There are several file formats which are appropriate for this domain. I will show up the rough idea of the different most important motion capture file types, their file format and discuss the advantages and problems of them.

5.3 Motion capture file formats

After we created the motion that we need, we have to store this on a permanent storage, so it can be reloaded an reused at some time. I discuss three important file formats:

- BioVision Hierarchical Data (BVH, former BVA)
- Acclaim ASF/ACM
- COLLADA

First, I will give an overview on the BioVision file standard, its general structure and concepts. I present the other types based on their differences to BVH. The chapter closes with a very brief discussion of the different alternatives.

5.3.1 The BVH file format

A BVH file can be divided into two main parts. The first part of the file is the *header* section. This section contains the "static" bone structure. It describes which bones exist, which size they have, and how they are connected to each other. The second part is the *body* section. The actual animation data, describing the posture of each joint for every consecutive frame, is stored there.

Header section

In detail, the header contains a hierarchical structure, starting at a reference point. Figure 22 shows a sample of such a header. The reference point, called the *root* node, is usually the hip of the skeleton. Several other bones, the child bones, are attached to this root.

```
HIERARCHY
ROOT Hips
{
 OFFSET 4.811 -32.986 -146.140
 CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
 JOINT LeftUpLeg
 {
   OFFSET 10.668 2.303 -0.000
   CHANNELS 3 Zrotation Xrotation Yrotation
   JOINT LeftLeg
   {
     ...
   }
 }
 JOINT Spine
 {
  ...
 }
}
```

Figure 22. Sample of a BVH header

These are for example the right and left up leg and the spine. Each of these bones has several parameters. First, there is a fixed offset to the parent. This offset describes the length of the bone, as well as its initial position. Additionally, there is a list of rotation axes. For each of these axes, there will be a column in the body section, containing the actual rotation angle in every frame.

This scheme works recursively. Each of the stated bones can again have some children, which can then also have several children etc.

Body section

In contrast to the header section, which is a hierarchical structure, the body section does only contain a table of numbers without any structure. Each frame in the animation is represented by one row in the body section. There are as many frames in the animation, as there are rows in the body section. Each row consists of a set of numbers, seperated by spaces. Each number is one angle for a joint described in the header section. The order of the numbers is the same as the joint order in the header file.

5.3.2 Other file formats

Besides of this file type, there exist several other standards which are very common. The one which is most similar to the BVH file structure is the *Acclaim* file format.

Acclaim

The main difference to a BVH file is at first, that Acclaim seperates the header and the body section into two seperate files. This arises from the fact, that a skeleton, for example the skeleton of a specific user, is used in several animations, which are recorded and also played afterwards. By seperating the animation data from the skeleton description, the skeleton design can be re-used for other animations.

The second difference is that Acclaim support more header information. The header can also store information like

- Bone name
- Bone length
- Initial orientation
- Rotation constraints, for example limited rotation angles

Especially the last field can be helpful, as there are natural movement constraints for each human being. Violating these constraints will make animations look unnatural. With these built-in constraints, unnatural postures are impossible.

COLLADA

The third file format is COLLADA. In opposition to the two formats stated before, this one is an open standard. COLLADA is an "XML schema for exchanging digital assets among graphics applications". The offical specification can be found at [1]. The main goal was to make data exchange possible between graphics applications, so that every application supports the collada file format as an open standard. At this point in time, there exist several open-source libraries (COLLADA DOM, OpenCOLLADA, FCollada) that support developers in using the file standard. Most of the famous 3D modeling tools support COLLADA.

One of the special features of the file standard is that also physics are supported, and physical properties can be attached to every model such as

- Friction
- Collision volumes
- Physical constraints between physical objects
- Global physical properties like gravitation

As acclaim and COLLADA is not supported by the used motion record tool, and the additional bone information delivered no significant advantage in my special domain, I chose BVH as a file format.

So far for the implementation and the technical background, the ongoing chapter will describe an evaluation which has been conducted on my implementation. The main goal was to solve the question, whether my new sensor-based interface is superiour to mouse interfaces or not.

6 Evaluation

This section treats the evaluation of my software. First, the workflow of each single evaluation as well as the setup is described. Afterwards, the recorded metrics, the analysis and the outcome of the experiment is presented. The chapter ends with a discussion and a conclusion over the conducted experiment.

I implemented an alternative version of the tool where the user controls two mice. We used the mouse version as a control condition. Figure 23 shows subjects using the Kinect (left) and the mouse (right) interface. The experiment workflow is as follows: First, the user is given written and verbal instructions, on how the interface should be used. The functionality of the interface is demonstrated to the user, and can be practised by him within a limited time. Afterwards, the main test phase starts, where I record the data for my later analysis. The experiment ends with a short survey, where the user can express his own experiences with both input systems, regarding intuitivness, precision and performance.

Figure 23. Users in action

6.1 Methods

Participants

I had 21 persons in total. Two of them were taken out due to aborted experiments. We thus analyzed 19 persons. There were no restrictions on age, gender or profession on the subjects. Out of 19 candidates, I had 12 female and 9 male persons (Figure 24). The age is between 18 and 56 years (average age 24). Two of the subjects were left-handed, but use the mouse with their right hand. Participants were not paid.

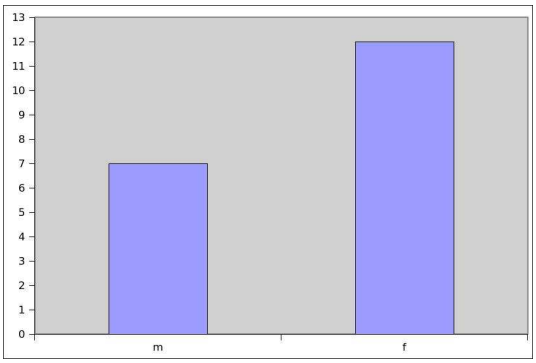

Figure 24. Gender distribution

The trials are not generated, but have been made using my software implementation. Half of the trials were done using Kinect, the other half using the mouse-interface. Every trial was created out of the default curve in only one edit step.

Apparatus

The hardware setup is shown in Figure 25. For the experiment, the user had to stand in front of a 22 inch display, in a distance of about two meters. For the mouse condition, subjects were sitting directly in front of the PC. The complete surface of the office desk was available for him to act upon with the mice.

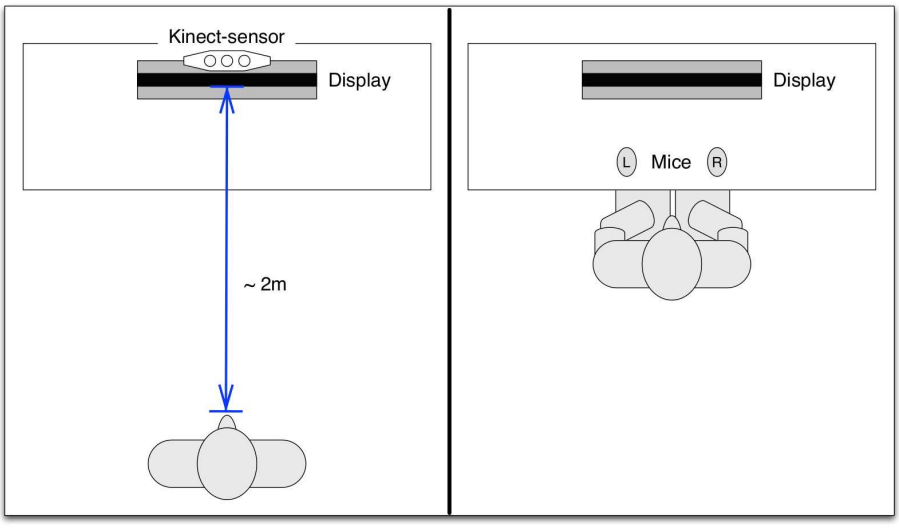

Figure 25. Hardware setup

Design

The evaluation confronts two possibilities of controlling my application. The first condition is the control by a combination of a sensor-based and a mouse interface, as described in previous chapters. The second condition is a two-mouse based control interface. There, each mouse substitudes one hand in the sensor-based interface, as described in a previous chapter. X-Y movement is controlled by moving the mouse, whereas movements in the Z-direction are achieved by rotating the scroll wheel. The two-mouse control is inspired by [12], where two mice are used for symmetric bimanual interaction.

There are 12 trials, divided into two subsets of 6 items each. The subsets were shuffled in a controlled manner. Each subset will be processed by one condition. To avoid training effects on one condition, the first half of the subjects will start using the condition "Kinect" and continue with the "Mouse" condition, and vice versa for the second half of the subjects. Details on the creation and order of the trials will be given in later chapters.

Task

The user study tries to solve two central questions regarding my implementation:

1. **Which of the two conditions is in general more intuitive/leads to a faster result**
2. **Are there special groups, which profit more from the Kinect interface**

The task is to match a given trajectory to a target trajectory (Figure 26). Each subject uses both interface versions and has to match six trajectories for each interface. A trial is done if either the distance between given and target trajectory is below a certain threshold (successful match) or the maximum edit time (120 sec) has expired. In either case, the user directly continues with the next trial.

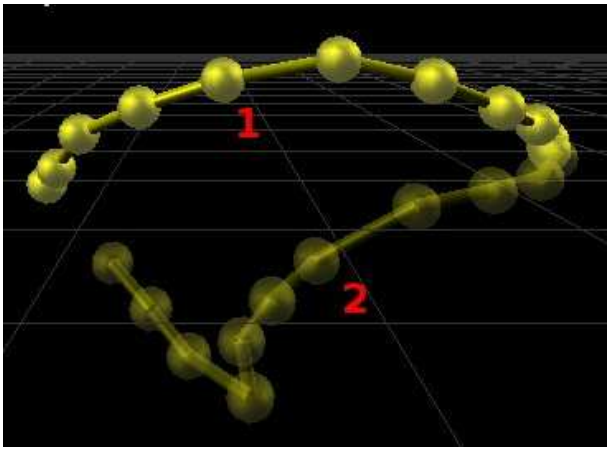

Figure 26. Motion trajectory

6.2 Procedure

The test procedure for all test candidates was as follows:

1. At the start of the experiment the subject reads the written instructions
2. After the subject finishes reading, the experimenter gives a demo of the interfaces and proposes hints on solution strategies
3. After a short question and answer session, the subject can practice on two sample trials, which will not be part of the main evaluation phase later. The sample trials and their order are always the same
4. The main evaluation begins, where the subject is presented 6 trials to be matched in the first interface condition
5. Phases 2-4 are repeated for the second condition
6. The experiment ends with a short two-page questionnaire, where the subject expresses his experiences with both interfaces in a multiple-choice form on the first page, and gives free feedback about problems and special experiences on the second page

The next chapter will give details about points 1 and 2. The second chapter handles the training phase (point 3). The last sub-chapter describes the main test-phase, as well as the generation and order of the trials given to the user.

6.2.1 Instruction

In the first phase of the experiment, the subject is given the instructions on how the experiment will be performed. First, he will be handed out a description describing the most important facts about the experiment. The handout can be found in the appendix, Chapter 8.5.

There are several hints given to the user. Each user gets exactly the same hints, which are the following:

- *Hold hands at shoulder height and stand behind the ceiling light, so the view sensor has an optimal tracking*
- *There are infinite many edit phases possible, not only one per trial*
- *The optimal strategy for matching a trial is as follows:*
    - *Match the curve in the top view (small window) first, then continue with the front view*
    - *All trials have been created in one edit step, so it is ideal to grab exactly these points which have been grabbed when I created the trial*
        - *These points are often the ones which are most far away from the default curve*
        - *Often, they are at the "peak" edges of the curve*
- *When selecting the points, the hands must be moved in a horizontal line to have an effect. The user must not follow the shape of the curve. Moving the hands vertically will not have any effect*
- *Using the two-mouse interface, the mice must not be lifted up at any time.*

After this first verbal instruction, there is a training phase in which the user should get a first feeling of the system, before the actual evaluation starts.

6.2.2 Training

After the overall picture should be clear to the subject, I do a sample myself, and explain the functionality in the meantime. For this, I load a test-curve (which will not be used in the main evaluation phase later) and go through the editing process step by step while explaining how it works in detail. Afterwards, the test person has the possibility to ask questions, which are be answered briefly. In the next step, the subject can do exactly two test-curves himself, and ask some final questions. Then, the main test phase starts.

6.2.3 Test

In the test phase, the user is given exactly 6 trials for each condition, and has to match the default curve to them. The trials are all of the same difficulty.

There are 12 trials $T_1...T_{12}$ used. These trials are seperated into two sub-sets A and B of 6 items each. Each subset is shuffled in a controlled manner (e.g. in a reversed order) to obtain the new sets A' and B'. So in the end the sets are structured as follows:

A. $T_1...T_6$

A'. $T_6...T_1$

B. $T_7...T_{12}$

B'. $T_{12}...T_7$

Each Subject is given the trial-groups in a different order, as Table 5 shows. $S_1...S_{12}$ denote the subjects, whereas C_1 and C_2 stand for the the used condition.

	C_1(Kinect)	C_2(Mice)
$S_1...S_3$	A	B
$S_4...S_6$	A'	B'
$S_7...S_9$	B	A
$S_{10}...S_{12}$	B'	A'

Table 5. Example for assignment of trial groups to subjects

After the test phase, the subject is given a short questionnaire, containing questions about his personal experiences regarding efficiency and intuitivity of both conditions. The questionnaire can be found in the appendix (Chapter 8.4).

6.3 Measures and results

This chapter describes the values that I tracked during my experiment, as well as the derived results. The following discussion chapter tries to interpret the results, coming to a conclusion about the experiment.

6.3.1 Recorded values

I computed several values for each of the two conditions in every experiment.
These values are:

Completion time. The average time which is needed to complete a trial

Mean squared error. The mean squared error of all points which have not been matched yet, formally $I = \frac{\sum (\|P_k - S_k\|)^2}{|P|}$, where P_i, S_i denote the points on the curve and the trial, respectively. This value is recorded to decide when a sample is matched, and the user should continue with the next one.

Coordination measure. For comparing the movements regarding coordination, I use the measure defined by Kipp/Nguyen [9]. Coordination measures whether movement is performed in all three dimensions in parallel (high value) or whether movement is sequentialized, i.e. performed along one dimension at a time (low value). Alternative coordination measures [3][27] are based on on a fixed optimal solution path. For my problem, the optimal solution may be counterintuitive and the various selection possibilities for selecting grab points further complicate matter. Therefore, such optimal-path metrics may yield misleading results. Kipp/Nguyen's measure is independent of optimal paths, taking only the raw movement vectors of the input device into account.

6.3.2 Results

Completion time

We used completion time to measure task performance. Figure 27 shows the average completion times over all participants and motion path samples. There is a clear difference between Kinect and Mouse.

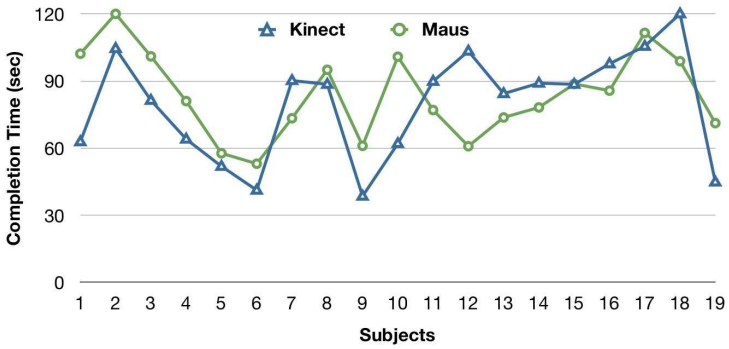

Figure 27. Completion time over all subjects

A paired t-test proved that subjects are significantly faster with the Kinect (M=79.36; SD=23.9) compared with the mouse (M=83.71; SD=19.02): t(18)=3.89, $p < 0.01$. For gender, I analyzed males and females separately. While the performance for the mouse was statistically equal, I found that, for the Kinect, male participants were much faster (M=65.24) than female ones (M=87.59) which I found to be highly significant (t(17), p-value<0.001).

I derived tests on the completion time of other sub-groups. I compared users, who like the Kinect interface more than the mouse, to the ones preferring the mouse, both with their favourite devices. Despite of a different mean completion time (M=60.5 for kinect group, M=70.77 for mouse group), I could not prove the result to be significant, due to a high variance.

Coordination

Using the above mentioned coordination measure [9] we compared the performance for the left and right hand separately over all participants and devices (see Figure 28). In both cases the Kinect interface encourages significantly higher coordination: For the mouse device the coordination values are almost zero (left hand 0.009, right hand 0.007), whereas the Kinect interface reaches a value of 0.29 for the left hand and 0.26 for the right hand.

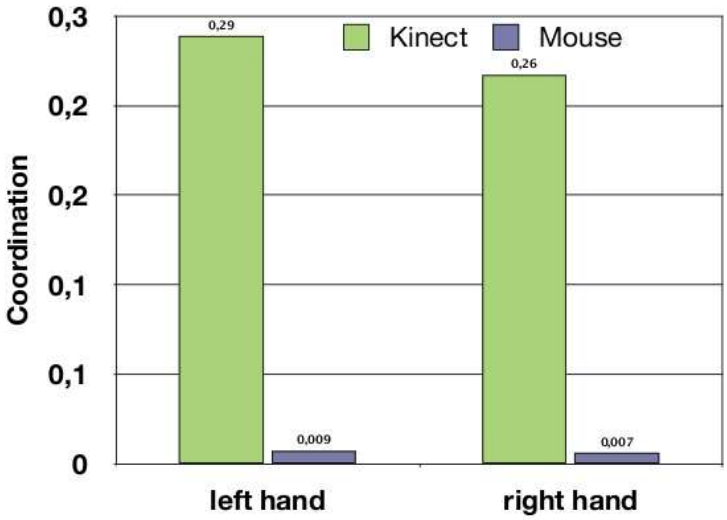

Figure 28. Coordination for left and right hand

Questionnaire

In a post-experiment questionnaire we asked the participants questions which discriminated between Kinect and mouse on a 5-point differential scale where the middle position was labeled both or equal. This questionnaire contained 17 questions regarding usability and user experience. For analysis, we transformed this to an interval between -2 for mouse and +2 for Kinect. We summed up ratings for each question over all subjects and computed a chi square statistic, comparing the ratings against the expected neutral value of zero. Only three questions reached significance, all three in favor of the Kinect interface.

- Which interface was easier to use? $(M = .21; \chi^2\,(2,n = 19) = 11.47; p < 0.005)$
- Which device was more fun to use? $(M = .84; \chi^2\,(2,n = 19) = 8.58; p < 0.05)$
- Which device do you prefer?$(M = .47; \chi^2\,(2,n = 19) = 11.47; p < 0.005)$

6.4 Discussion

The results have shown that our MotionBender interface is more efficient than the more conventional mouse interface. Performance was significantly better, movement was clearly more coordinated and participants preferred it over the mouse interface and enjoyed it more. I also found that male users seem to profit more from our gesture-based technique than female users. However, due to the low number of participants this may be an artifact and should be validated in future studies. Overall I reached all goals for our interface technique. The Kinect was more coordinated than mouse. For mouse, users mostly acted separately, so they stopped moving the mouse when adjusting the depth position by rotating the scroll wheel. Some found this limiting, others liked the fact that no unintentional movement in the z-plane was possible.

Comparing the left- and right-hand performance, the left hand was more coordinated than the right one in the mouse condition. I often observed that people are acting serially in matching the two selected points. They concentrated on the first (left), neglecting the second one initially. On the sensor-based interface, the difference is less. This might also be a result of the reduced cognitive difficulty with the direct-manipulation concept. Some problems emerged because subjects had problems to visually understand the 3D scene. Some suggested that a 3D stereoscopic view could reduce problems. However, Kulshreshth et al. [11] observed that a 3D view does not always result in a significantly higher performance, although users feel to act more efficient using stereoscopic view.

I had two views on the scene, a top- and a front-view. Some users were confused by these two different views, and temporary lost the orientation. In one interesting case, a subject had general problems with precise hand movements. She found the Kinect interface much easier to use because due to the whole-body motion some focus was taken away from hand movement precision.

7 Conclusion

7.1 Summary

We presented an interacton technique called MotionBender, suitable for character animation. We used the concept of motion paths for the visualization of the animation. This concept has been used in several other work. Still, there was no intuitive method to allow the user to change the motion path. I presented a new editing method for postprocessing the animation by grabbing and bending the motion path. For this, we used a combination of a motion sensor (Kinect) and a simple button controller (wireless mouse) as input devices. The technique was intended to be efficient, intuitive and enjoyable.

We validated our technique against a mouse-based interface and found MotionBender to be faster, easier to use and more enjoyable. We also found that our technique encouraged highly coordinated movement. A gender effect indicates that this technique particularly appeals to male users but this finding needs further study.

7.2 Future work

There are still several possible improvements on my work. First, I had only few subjects, namely 19. A good empiric evaluation would need some more different test subjects. I had a high variance on the test results, which made it difficult to obtain significant results, especially for subgroups. Every subject processed six trials with each condition, all of them were meant to have an equal difficulty. However, statistics suggest that some of them were harder than others, distorting the results. Such effects could be minimized with a higher number of trials for each candidate. Additionally, it might be an interesting observation, how the different techniques perform with samples in different difficulties. The sensor interface might pay off more with a higher difficulty level.

I implemented a sensor-based interface as well as a mouse-based interface as a reference. Another interesting interface design technique could be a multitouch-based control. Recent work [9] has shown that the usage of those devices can significantly improve performance in 3D-animation compared to a conventional interface. It would be interesting to compare Kinect and a multi-touch interface to each other, to find a “current best solution”.

My IK algorithm has currently no restriction on the rotation of the joints. In real world, not every joint angle is possible, due to physical restrictions. Animations generated with my tool can therefore look unnatural. The current algorithm could be enhanced so that restrictions on the joint angles can be set and managed by it. Some of the file formats like COLLADA (see 5.3) support rotation constraints as a joint property. The ideal implementation of MotionBender would be to load these properties, and pass them to the IK algorithm, which takes care of them during the modification process.

Another drawback of the IK algorithm is that it can only handle one joint position at the moment. The user can set the motion curve of exactly one joint, and all other joints will be aligned so that this curve is matched. There is no possibility to define the curve for two or more joints at a time, or to keep the curves from previous edit steps in the following steps. A modified IK algorithm, which is capable of handling more than one joint constraint, could solve this problem.

The current implementation allows it to modifiy the motion curve, e.g. direction of the movement at each point in time. It is not possible to change the speed or the velocity of the animation. Such a “time-based” approach could be another improvement of my work. The workflow for such a functionality would be as follows: The shape of the curve is fixed during the complete edit step. The user selects two balls on the curve. In the next phase, these balls can be shifted on the fixed curve. Surrounding balls are also shifted, similar like with the neighbour function.

Another missing part is the motion recording function. The recording is currently done by another, external tool. The recorded motion is then imported into my application. The integration of such a recording functionality would have several advantages. At first, it is easier and more convenient for the user if he has to use and understand only one tool. Second, a record function makes it possible to re-record parts of the motion, or to add motion parts inside the existing motion. With the current solution, the complete animation has to be discarded and recorded newly from scratch.

8 Appendix

8.1 Install instructions for Kinect and the SDK

The only platform which is supported for interacting with Kinect, is Microsoft Windows. The official Mirosoft Kinect framework needs at least Windows7. The installation is quite easy, you can get a msi-file on the Microsoft hompage here[12]. The installer is doing the complete work.

The PrimeSense SDK requires only WindowsXP, but is harder to install. It consists of several components which need to be installed: The Sensor driver, the OpenNI framework, and the NITE framework. There are different versions for different CPU architectures. Installing only one of the components for the wrong architecture will make it impossible for the program to work correctly. Fortunately, there is an auto-installer available here[13], which does all the work.

8.2 Kinect hardware sender

The task of the Kinect sender is to gather the skeleton data from the Kinect device, and pass it to the main application by sending it over a socket. There are two versions of the Kinect sender which have been explored.

First, I used a sender based on the Microsoft SDK. This version has the advantage, that no calibration is needed, the user can directly start using the tool. But in practise, it came out that Microsoft's algorithm is quite unstable, and has some severe problems with changing illumination. Therefore, I switched to a version based on PrimeSense's Kinect SDK. In contrast to the former, this SDK needs a calibration phase, but works a lot more stable due to the different algorithm. The sender application is based on an implementation of Daniel Puschmann, a student employee at our chair. I added some skeleton information which has to be sent to my application, and changed the sending protocol to a form which is more convenient for my implementation.

8.3 XInput skripts

All the skripts posted here have been used in the evalution phase. They were executed on an Ubuntu 10.10 shell instance. For the evaluation, there are two scripts for the two tasks that need to be done: Create a second mouse pointer and send both pointers' data over socket to the application.

The skripts use the mouse's name or its id in order to identify the device. A list of all connected input devices, its names and ids can be achieved by executing `xinput list` on a shell console.

Skript for creating second mouse pointer

Xinput allows it to create a second input device group with its own mouse cursor. This group is created in the first line of the code by the create-master command, and named "master". The second command reattaches our second mouse device (named "2.4GHz 2way RF Receiver") to this newly created group

```
xinput create-master master
xinput reattach "2.4GHz 2way RF Receiver" "master pointer"
```

Figure 29. Source-code of script 1

Skript for sending

12. http://www.microsoft.com/en-us/kinectforwindows/
13. http://www.brekel.com/?page_id=160

For gathering the mouse position, the “test” function of XInput is used. This function prints out the mouse position as well as click events on the console, for a given input device. The name of the device is passed upon start of the script, denoted by the “$1”. This information is then send to localhost’s address 1235 by the telnet command. The “stdbuf” prefix before the xinput command makes sure that pipe buffering is disabled, and every message is directly passed to telnet. Without this prefix, the pipe would collect a block of messages, and then pass them to telnet in one go, which results in a juddering movement of the mice.

```
stdbuf -i0 -o0 -e0 xinput test $1|telnet localhost 1235
```

Figure 30. Source-code of the second skript

8.4 Questionnaire

Fragebogen

VP: _____

Sample-Folge:_____

1	Geschlecht	m O O w
2	Alter	
3	Beruf / Studienfach	

	Wie hoch schätzen Sie ihre Erfahrungen ...	
1	... im Umgang mit PCs allgemein?	gering O O O O O hoch
2	... im Bezug auf Bewegungssteuerungen (z.B. XBox mit Kinect)	gering O O O O O hoch
3	... im Bezug auf gleichzeitige Bedienung von 2 Mäusen	gering O O O O O hoch
4	... im Bezug auf 3D-Modellierungsprogramme (z.B. CAD-Programme, Blender, 3dsMax)	gering O O O O O hoch
5	... im Bezug auf 3D-Spiele	gering O O O O O hoch
6	Haben Sie Probleme mit 3D-sehen?	keine O O O O O stark
7	Sind Sie Links- oder Rechtshänder?	links O O rechts
8	Mit welcher Hand bedienen Sie die Maus?	links O O rechts

	Welches der Eingabegeräte ...	
1	... war nützlicher?	Kinect O O O O O Mäuse
2	... hat genau das getan, was ich erwartete?	Kinect O O O O O Mäuse
3	... erleichterte mir das Lösen der Aufgabe?	Kinect O O O O O Mäuse
4	... ermöglichte ein schnelleres Lösen der Aufgabe?	Kinect O O O O O Mäuse
5	... war einfacher zu handhaben?	Kinect O O O O O Mäuse
6	... benötigte den wenigsten Aufwand um die Aufgabe zu lösen?	Kinect O O O O O Mäuse
7	... war müheloser zu bedienen?	Kinect O O O O O Mäuse
8	... ermöglichte ein besseres Korrigieren von Fehlern?	Kinect O O O O O Mäuse
9	... war *leichter* zu erlernen?	Kinect O O O O O Mäuse
10	... war *schneller* zu erlernen?	Kinect O O O O O Mäuse
	... ermöglicht besser die Manipulation im 3D-Raum?	Kinect O O O O O Mäuse
11	Welche Bedienweise konnten Sie sich leichter merken?	Kinect O O O O O Mäuse
12	... ermöglichte nach kurzer Zeit einen sicheren Umgang.	Kinect O O O O O Mäuse
13	Mit welchem Gerät waren Sie zufriedener?	Kinect O O O O O Mäuse
14	Welches Gerät würden Sie für diese Aufgabe empfehlen?	Kinect O O O O O Mäuse
15	Welches Gerät machte mehr Spaß?	Kinect O O O O O Mäuse
16	Welches Gerät finden Sie besser?	Kinect O O O O O Mäuse

Falls Sie eines der Geräte besser fanden, warum?

Gab es besondere Probleme mit Kinect / Maus?

Sonstige Anmerkungen:

8.5 Evaluation handout (German)

Worum geht es in diesem Experiment?

- Wir haben einen drei-dimensionalen „Wurm“
- Dessen Form muss auf die der Vorlage angepasst werden. Dabei ist es wichtig, dass das Ergebnis möglichst präzise, aber auch die Editierungszeit möglichst kurz ist.
- Es gilt für jede Vorlage eine maximale Bearbeitungszeit von 120 sek.
- Während der Bearbeitung wird die Präzision am unteren Rand des Fensters angezeigt (sieht man gleich). In der linken unteren Ecke wird die verbleibende Bearbeitungszeit angezeigt. Die linke obere Ecke zeigt eine Draufsicht auf das geschehen.
- Eine Aufgabe ist beendet, wenn dieser Countdown abläuft, oder eine gewisse Präzision erreicht wurde.

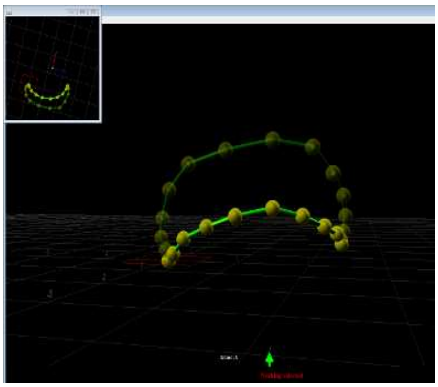

Figure 31. Ausgangssituation

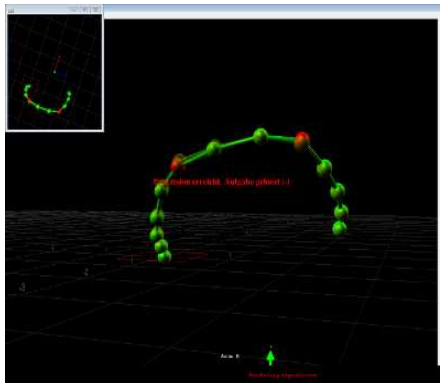

Figure 32. Mögliche Lösung

Zum Ablauf

Der gesamte Ablauf wird etwa 30+30min betragen. Er wird einmal mit der Steuerung per Kinect, und einmal mit Steuerung durch 2 Mäusen durchgeführt. Sobald Sie dieses Dokument zu Ende gelesen haben, wird die Vorgehensweise anhand einer Vorlage vorgeführt und erläutert. Im Anschluss daran haben Sie die Möglichkeit Fragen zu stellen und später selbst die Bedienung anhand mehrerer Vorlagen auszuprobieren.

Erst dann beginnt die eigentliche Testphase, in der Sie insgesamt 6 Vorlagen nachbilden müssen.

Zum Schluss folgt ein kurzer Fragebogen, auf dem sie auch weitere Anmerkungen vermerken können.

Viel Erfolg!

Bibliography

[1] COLLADA : Digital Asset Schema Release 1.4.1 Specification (2nd Edition), 2008.

[2] Dragomir Anguelov, Daphne Koller, Hoi cheung Pang, Praveen Srinivasan, and Sebastian Thrun. Recovering Articulated Object Models from 3D Range Data. In *IN PROC. UAI*, pages 18–26. AUAI Press, 2004.

[3] Ravin Balakrishnan and Ken Hinckley. Symmetric bimanual interaction. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '00, pages 33–40. ACM, 2000.

[4] 3Dconnexion GmbH. *Technology Assessment Group. The Economic Payback of 3D Mice for CAD Design Engineers*. 2008.

[5] S. Burak Gokturk, Hakan Yalcin, and Cyrus Bamji. A Time-Of-Flight Depth Sensor - System Description, Issues and Solutions. In *Proceedings of the 2004 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW'04) - Volume 03*, CVPRW '04, Washington, DC, USA, 2004. IEEE Computer Society.

[6] Gutemberg B. Guerra-filho. Optical Motion Capture: Theory and Implementation. *Journal of Theoretical and Applied Informatics (RITA)*, 12:61–89, 2005.

[7] Ken Hinckley, Randy Pausch, Dennis Proffitt, and Neal F. Kassell. Two-handed virtual manipulation. *ACM Transactions on Computer-Human Interaction*, 5:260–302, 1998.

[8] Jessica Hodgins James, James F. O'brien, and Robert E. Bodenheimer. Computer Animation. In *Encyclopedia of Computer Science*, pages 301–304.

[9] Michael Kipp and Quan Nguyen. Multitouch Puppetry: Creating Coordinated 3D Motion for an Articulated Arm. In *ACM International Conference on Interactive Tabletops and Surfaces*, ITS '10, pages 147–156. ACM, 2010.

[10] Adam G. Kirk, James F. O'Brien, and David A. Forsyth. Skeletal Parameter Estimation from Optical Motion Capture Data. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2 - Volume 02*, CVPR '05, pages 782–788, Washington, DC, USA, 2005. IEEE Computer Society.

[11] Arun Kulshreshth, Jonas Schild, and Joseph J. LaViola, Jr. Evaluating User Performance in 3D Stereo and Motion Enabled Video Games. In *Proceedings of the International Conference on the Foundations of Digital Games*, FDG '12, pages 33–40, New York, NY, USA, 2012. ACM.

[12] C. Latulipe and University of Waterloo. School of Computer Science. *A Symmetric Interaction Model for Bimanual Input*. Canadian theses. University of Waterloo, 2006.

[13] Andrea Leganchuk, Shumin Zhai, and William Buxton. Manual and Cognitive Benefits of Two-Handed Input: An Experimental Study. *ACM Trans. Comput.-Hum. Interact.*, 5(4):326–359, December 1998.

[14] Jianyuan Min, Yen-Lin Chen, and Jinxiang Chai. Interactive Generation of Human Animation with Deformable Motion Models. *ACM Trans. Graph.*, 29(1):9:1–9:12, December 2009.

[15] Michael Neff and Yejin Kim. Interactive Editing of Motion Style Using Drives and Correlations. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '09, pages 103–112. ACM, 2009.

[16] Q. Nguyen. *Werkzeuge zur IK-basierten Gestenannotation mit Hilfe eines 3D-Skeletts*. Saarland University, Saarbrücken, Saarland, GERMANY, 2009.

[17] Russell Owen, Gordon Kurtenbach, George Fitzmaurice, Thomas Baudel, and Bill Buxton. When it gets more difficult, use both hands: exploring bimanual curve manipulation. In *In GI 2005 Proceedings*, pages 17–24, 2005.

[18] Yuri Pekelny and Craig Gotsman. Articulated Object Reconstruction and Markerless Motion Capture from Depth Video. *Computer Graphics Forum*, 27(2):399–408, April 2008.

[19] F. Raber. *Ein Datenflusseditor zum Rapid- Prototyping von Multitouch-Anwendungen*. Saarland University, Saarbrücken, Saarland, GERMANY, 2010.

[20] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-Time Human Pose Recognition in Parts from Single Depth Images. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 1297–1304, Washington, DC, USA, 2011. IEEE Computer Society.

[21] Matthew Thorne, David Burke, and Michiel van de Panne. Motion Doodles: An Interface for Sketching Character Motion. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06, New York, NY, USA, 2006. ACM.

[22] Jason Van Gumster. *Blender For Dummies*. Wiley Publishing, Inc, Indianapolis, Indiana, 2009.

[23] C. Welman. *Inverse Kinematics and Geometric Constraints for Articulated Figure Manipulation*. 1993.

[24] Andrew Witkin and Zoran Popovic. Motion Warping. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '95, pages 105–108, New York, NY, USA, 1995. ACM.

[25] U xuan Tan, Student Member, Kalyana C. Veluvolu, Win Tun Latt, Cheng Yap Shee, Cameron N. Riviere, and Wei Tech Ang. Estimating Displacement of Periodic Motion With Inertial Sensors.

[26] Robert C. Zeleznik, Andrew S. Forsberg, and Paul S. Strauss. Two Pointer Input For 3D Interaction. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, pages 115–120, 1997.

[27] Shumin Zhai and Paul Milgram. Quantifying Coordination in Multiple DOF Movement and Its Application to Evaluating 6 DOF Input Devices. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '98, pages 320–327. ACM Press/Addison-Wesley Publishing Co., 1998.

[28] Zhengyou Zhang. A Flexible New Technique for Camera Calibration. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(11):1330–1334, November 2000.