Evaluation of motion retargeting using spacetime constraints

Master's Thesis

of

Bas Lommers
Student Number: 3441431

December 6, 2013

Supervisor:
Dr. J. Egges

Thesis number: ICA-3441431

Utrecht University
Faculty of Science
Department of Information and Computing Sciences
MSc programme - Game and Media Technology

Abstract

This thesis report describes the evaluation of the retargeting method "Retargetting Motion to New Characters" as introduced by Gleicher. Our description of the method shows more details than the original paper does. This report also grants more background information about differentiation a constraint function, using either divided differentiation or automatic differentiation. Furthermore the for the interpolation method, a B-spline, is explained more extensively. Explaining the range of influence of B-spline control points and calculating a point on the B-spline locally without evaluating the whole spline. Finally experiment have been executed on our own implementation of this method. The experiments evaluate the method on its performance with different kinds of constraints, different motions and different scalings. The resulting motions of those experiments are inspected visually on quality.

Contents

1 Introduction

Virtual characters have a large role in games, simulations, movies and commercials. Usually they perform motions in a virtual world. There are several ways animate those characters, but the most realistic results are still achieved through motion capture. There are several techniques for motion capture. The end result of motion capture usually is a motion defined in a root position and joint angles that change over time. Motion capture works well if the human actor has the same measurements as his virtual equivalent, but when the sizes of body parts start to change it fails. This can be seen on the left side of figure 1.1. By defining a motion in joint angles the positions of the body parts are implicitly defined by the character performing the motion. When this character changes the information about the body part positions get lost. The importance of joint angles and body part positions shows clearly when body parts start intersecting with the environment or each other. But for characters with a different size it is impossible to keep every body part on the same position. Deciding which positions are important and which are not is not trivial. Some body-part positions might have importance in global space and some body-part positions in character space, relative to another body part.

1.1 Motion retargeting definition

Retargeting is transferring a motion from one character to another as if one character is doing the same as the other character. Transferring the motion on to the same character in a different environment, like foot plants on a walking motion or placing the hand on the right place when picking up an object, could also be seen as retargeting, but this thesis won't focus on that. For a character to do the same as another character it is important to know the characteristics of the motion. It is hard to recognize those characteristics. For example if Pixar's Luxo walks it jumps, but if it would have wheels it would roll. The characteristic in this motion would be something like default movement. The need to recognize motion characteristics is reduced when the character is

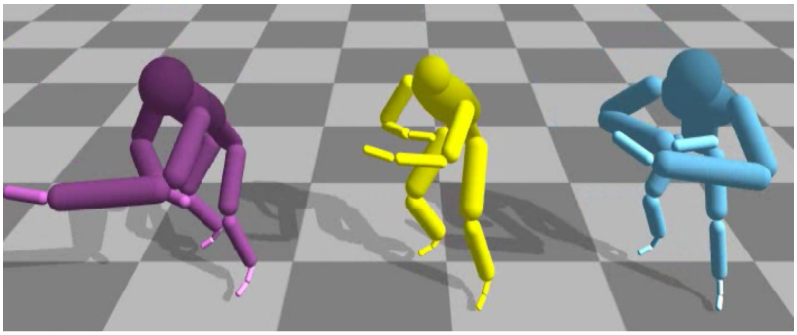

Figure 1.1: The middle character is the character with the same measurements as the actor the motion is recorded on, the left character has larger limbs and the motion is transferred by using the same joint angles. The right one uses a more advanced approach. (source:[4])

more similar, because then the problem is more about recognizing similarities between characters and transferring those. The motion characteristics are then implicitly transferred.

Transferring motions to characters with different morphologies can be useful in many cases. For example with retargeting motions, the motions can be applied to more characters and thus save expensive motion capture time in a studio. Retargeting could also be useful for characters that do not exist in the real world. Moreover the characters can also transform to another character during a motion, like humans transforming to a Hulk or werewolf. Another example application would be in games where dynamic character creation becomes more popular. Since characters are not known beforehand it is impossible to record all motions.

Retargeting can be applied in many different ways. The easiest application is to retarget a recorded motion to another character than the motion capture actor. Of course it is not possible to find a motion capture actor for any arbitrary character. Another application would be real time retargeting where an end user is able to create a character and existing recorded motions have to be applied. In cases of morphing characters retargeting can also be applied. With new input devices like the kinect retargeting gets yet another application namely converting movements from the player to the character in game. And in the future retargeting could possibly even be used to control robots.

1.2 Approach outline

This report describes the evaluation of an existing retargeting method, we redescribe this method with more details for better understand. Moreover we add a method for weighting constraint values. Evaluation is important to get a better understanding of the topic. The experiments done on the original method show little configuration parameters. We are going to evaluate how constraints should be defined and what effect constraints have on finding the solution. Before we go into detail we first discuss research done related to retargeting in Chapter 2. The method we use is limited to characters with skeletons with the same hierarchy but other bone lengths. We solve retargeting using spacetime constraints. The details about how the retargeting problem is translated to spacetime constraints are found in Chapter 3. This results in a constrained optimization problem. How the constrained optimization problem is solved is discussed in Chapter 4. Most optimization methods use a gradient to get to their solution. The gradient to our problem is a Jacobian matrix. The specifics of the Jacobian are discussed in Chapter 5. The final solution is an adjustment of the original motion. Instead of creating a totally new motion only the adjustment is created. To get a smooth adjustment a B-spline is used. This is discussed in Chapter 6. Chapter 7 discusses the evaluation of the method to find the strengths and weaknesses. It shows why and how the experiments are done and what the results are. Finally in Chapter 8 we draw the conclusion and makes suggestions for future work.

2 | Related work

This chapter discusses research related to solving the retargeting problem. There are different approaches to the retargeting problem. Some research focusses on retargeting in relation to the environment, others focus on specific body parts or specific constraints. Some focus on the "message" that is expressed by a motion. The multiple interpretations and the vagueness of "message" show the complexity of retargeting. Would humans with the same measurements do exactly the same if they are asked to mimic each other? Mimicking the same situation is different from performing the same action, performing the same action would be performing a motion that serves the same goal, while mimicking has more to do with trying to perform the same motion. A human can not even perform exactly the same motion, in term of joint angles, the same way twice because of external factors and muscle limits. Several kinds of constraints can be violated during a motion, there are physical constraints like collision, balance and force limits. There are also action constraints like: walking, picking up, waving. Style constraints are harder to define, as those are about this vague "message". Retargeting is usually split in a motion creation and motion playback. The motion creation sets the parameters for the motions, for example constraints. During motion playback those constraints are solved. The first section 2.1 explains some basic terminology used throughout the whole report. The second section 2.2 handles creating a motion. The next section 2.3 discussed what has been done on motion playback. The fourth section 2.4 discusses how other work on retargeting is validated. Next a conclusion is made in section 2.5, followed by a section 2.6 about the contribution made by this thesis.

2.1 Terminology

Before discussing other work in the field of retargeting, some general principals are explained. We are going to explain the following terms: execution time, spacetime constraints and inverse kinematics.

2.1.1 Execution time

In this section we discuss three terms that we use related to execution time. Execution time is the time it takes an algorithm to run. How this time is expressed exactly depends on the algorithm. Three basic indications are used: offline time, real time and online time. Offline time is the time before playing the motion, everything needed for the algorithm is available from the start. Real time is during the playing of the animation, but the motion and the environment are known beforehand, allowing the algorithm to look ahead in time. The last kind of time is online time, this is also during playtime of the animation but only data about the current time and past can be used by the algorithm. For example when there is a live feed of motion data from a capture device or input from the user.

2.1.2 Spacetime constraints

Spacetime constraints are introduced by Witkin and Kass [9]. Motions usually have to meet certain constraints, for example reach a point, avoid collision, be in balance. To create a realistic motion physical constraints on the acceleration have to be met. Trying solve those constraints while creating a realistic motion can be very complex. Solving those constraints on a frame by frame basis would make it hard to control limits on acceleration. Therefore spacetime constraints are introduced. Instead of only solving in space, which would result in snapping behaviour or complex constraint definitions, the constraints are also solved in time, so a motion can be adjusted in the time before the constraint becomes active, resulting in a smooth path matching the constraint.

2.1.3 Inverse kinematics

Inverse kinematics (IK) is a technique that can be applied on a chain of joints. The goal of IK is to find the parameters of the joints to get a certain configuration of a chain. Usually this configuration is giving by a position to be reached by the end of the chain. Although in animation usually only angular parameters can be set on the joints, translating joints can also exist. Inverse kinematics is the opposite to forward kinematics where the joint parameters are given and the configuration of the chain is calculated. Forward kinematics is trivial, but IK is not. Usually there are multiple solutions to an IK problem, for example with a IK chain from shoulder to the hand, there are various positions to place the elbow while the hand and shoulder remain in the same position. Various iterative techniques exist to solve IK, techniques that are often used are Cyclic Coordinate Descent (CCD) and a couple of Jacobian based methods, inverse-Jacobian and transpose Jacobian. A Jacobian is the matrix of partial derivatives. The Jacobian is discussed in Chapter 5. CCD threats the IK problem as a set of one degree-of-freedom (DOF) problems. CCD iterates over the DOF and minimizes the distance to the target for each DOF. It keeps iterating until a threshold distance to the goal is met. These are problems that can be solved analytically. For a rotational joint the goal is projected onto the plane of rotation then the angle between the current location and the target location can be calculated. The joint then rotates for this calculated angle. For a translational DOF the goal is projected onto the line of translation, the distance is then calculated and the joint translated for that distance. For Jacobian based methods the Jacobian of the forward kinematics function is calculated. The Jacobian now contains the amount of change needed for each DOF to get closer to the goal. This can then be used to move towards the goal.

2.2 Creating a motion

Naively transferring a motion to another character with different measurements, by just transferring the joint angles leads to a violation of constraints that are obvious to humans. The body of the character intersects with either itself or its environment, the motion does not longer serve the goal it is supposed to. Solving this traditionally means either adjusting joint angles or setting up IK-chains. Fortunately tools exist to assist an animator with retargeting. Most tools assist with editing an existing motion, but there are also tools that allow the animator to create a motion.

A very influential paper in retargeting is written by Gleicher [2]. Here the idea of spacetime constraints is applied to retargeting. The method is used to retarget motions in offline time. An animator sets constraints to the motion and validates if the result is as expected. If the result is not sufficient the constraints are adjusted to give a better result. This is done by evaluation all constraints as a single function with a the motion as a vector input and the constraint results as a vector output. The gradient for this function is used to find the motion that meets the constraints. Although the constraints are not bound to a specific scaling, they are validated for specific scalings.

A method for retargeting close interaction between a limited number of objects is introduced

Figure 2.1: A set of the same characters with different measurements (source:[4])

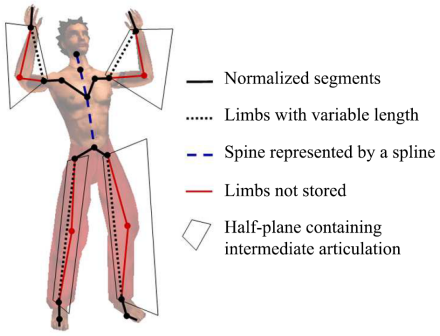

Figure 2.2: Character representation with half planes instead of the normal character representation with bones and joint angles. (source:[6])

by Ho et al.[4]. In motions where characters are close to other characters or the environment, like wrestling, dancing or getting in a car, the spatial relationships are important characteristics of the motion. Those characteristics are usually implicitly defined for one character. If the measurements of a character change those characteristics disappear. With a joint angle representation, the motion is made suitable for different morphologies by adding kinematic constraints. Those kinematic constraints take a lot of the animator's time to define. An automatic method for this would be very useful. Ho et al. introduce a new representation for spatial relations named the interaction mesh. This mesh connects important points in the motion together. Those important points are defined as joints of the characters and vertices of the environment. By tracking those spatial relations they can be preserved when automatically adjusting the motion to a character with different measurements.

With this method the spatial relationships are preserved automatically, so they do not need any animator interaction. Nevertheless additional constraints can be defined to adjust the motion. Moreover the user defined constraints there are also collision constraints that are generated by the system. The collision constraints also require for an iterative algorithm to be able to correctly adapt to occurring collisions.

Hecker et al. [3] introduce a generic motion definition. Their algorithm is designed for the game Spore, in this game the characters are created by the players. These characters can have varying morphologies. Skeletal key framing is a common method to animate characters in games, but for the different morphologies the skeleton is not known beforehand. An animator cannot create animations for any possible skeleton. Therefore the animations are defined in a generalized way and specialized at the time of playback.

Another method is proposed by Kulpa [6]. This method focusses on humanoid characters with different sizes, that have to be retargeted in real time in an interactive environment based on a small motion database. Other techniques generally use a large motion database or use a lot of computational power to calculate the desired result. Even if the computational issue of spacetime solutions could be overcome it is still hard to make spacetime solutions work in an interactive environment. The key to this technique is a new kind of pose representation as can be seen in fig. 2.2. In this representation the body is split into different parts that are stored in different ways. The body is a hierarchy of three different components: normalized segments, limbs with variable length and the spine. The normalized segments are segments that exist of one body part. Nothing special happens to these kind of segments since these are basically just a bone. The limbs

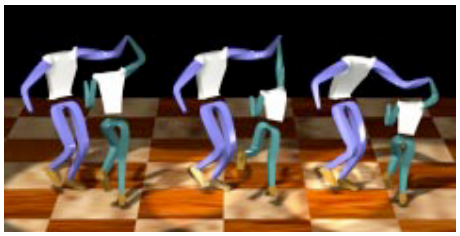

Figure 2.3: The motion of two dancers retargeted to resizing dancers. (source:[2])

are stored as normalized coordinates on an oriented half plane. Limbs are assumed to have three points, a begin, an end and a joint in the middle. The begin and end are on the edge of the half plane. The distance between the begin and the end point is given with a scalar indicating a percentage of the full leg length. The elbow position can be calculated with basic 2d geometry. The spine is represented by a spline. The spine of another character is fitted onto this spline. With this representation all sizes are relative thus the motion is implicitly retargeted. There is another part to retargeting, namely the resolving of constraints. This is done in the motion playback phase that is discussed in the next section.

2.3 Motion playback

In the motion playback phase the motion definition as defined by the animator is processed by an algorithm to create the actual motion. Depending on the method this motion playback can be executed in online time, real time or offline line. Gleicher [2] defines constraints as functions with a target value. Moreover an additional function is used as a target function. This target function gives the difference from the original motion, implying the motion has to look as similar as possible to the original scaled motion while meeting the constraints. The technique is focussed on retargeting motions to skeletons with the same hierarchy as the original motion, but with different bone lengths.

The interaction mesh introduced by Ho et al.[4] is created by applying tetrahedralization on the point cloud that is created from the joint positions and environment vertices. The interaction mesh and all the constraints are combined into one energy cost function that is minimized. For easy solving the interaction mesh is created with the joint positions instead of using the combination of joint angles to get to the position. This results in body parts that can change size and a roll rotation that is lost. This is resolved by adding constraints for the size of the bones and adding an additional point on the surface of body parts that can have a roll rotation. This is faster because it leads to a much sparser Jacobian matrix which is easier to solve. During the solving the bone length constraints are slowly increased from the original bone size to the target bone size.

An example of playback with the method created by Hecker et al.[3] is shown in fig 2.4. The playback is done with a special robust inverse kinematics solver which uses preconditioning to create natural poses. Bodies are introduced and used to define the characters instead of a skeletal representation. The main difference with the skeletal representation is that bodies are connected with a non-cyclic directed graph instead of a tree. This implies that a joint can have multiple parents. In the game, a player is not actually able to create loops but this limit was only for creating an easy interface. Bodies have certain capabilities like grasper, mouth, root, spine. The animations are still created by animators and for the animators the work flow remains the same. For walking animations a separate gait system is used. This system creates walking motions for characters with all the possible varying morphologies.

Kulpa et al.[6] annotated the motion with constraints, which indicate footsteps and other point or region constraints. These constraints are localized to specific body parts so that only the body parts that are influenced by this constraint are used for the calculation. Those constraints are

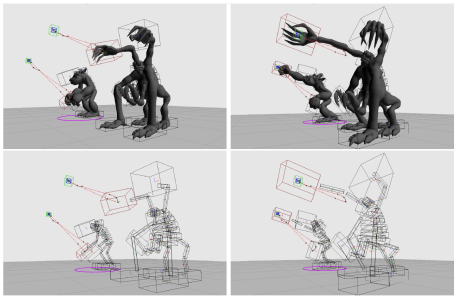

Figure 2.4: Two characters using the real-time motion retargeting, left in resting pose and right reaching for a point (source:[3])

solved with a modified Cyclic Coordinate Descent solver.

2.4 Validation

Generally it is hard to validate retargeting techniques. Many varying parameters exist of which some are unknown, for example not all possible goals for a motion are known and it is hard to generalize this. Another reason why retargeting is hard to validate is the limited case definition. Sometimes failure is part of the method, some characters just cannot perform tasks that other characters can. It is hard to define graceful failure. In this section we discuss some validation methods that are used.

In the paper by Ho et al.[4] different motions with close interactions are used for the experiments. Ho et al. evaluate the method on collision, non-colliding close interactions, interactive control and intra-body performance. Another part of the experiments is the evaluation of the computational costs. The method is claimed to be linear dependent on either the number of frames or the number of vertices in the interaction mesh. This is due to the local influence of the vertices. In the examples with two characters or a character in a simple environment, it takes 1 minute to solve 100 frames of animation on a Core i7 2.67 GHz CPU.

The method proposed by Hecker [3] is tested with no exact values. The qualitative tests are done evaluating the comments of players. It has not been stated in the paper whether the players are asked for their comments, what questions are asked and how many of the questioned players have responded. The comments of the players are said to be positive, but without any context about how the players are asked for their comments it is hard to put any value onto this statement. A plausible scenario could be the comments on the game forum are evaluated and only players who like the game post on there. Another form of testing is done by a team of animation testers. They validate the motions on various characters created by the players, but the criteria to which the motions are tested are not shown.

Kulpa et al. [6] set constraints to the feet, wrists and elbows. With those constraints three motions are created, a rest motion, a kick and a ward-off motion. Those motions are visually inspected. Extraordinary properties of the motions are discussed by the authors.

Most methods rely on perceptual evaluation of the resulting motion created by the implementation of the method. Most measurements like pose difference and distance to constraints are used by retargeting methods and are therefore hard to use for evaluation. We create an implementation of retargeting using spacetime constraint and perceptually evaluate the method. Although perceptual evaluations are hard to compare they do best match the goal of the method.

2.5 Conclusion

In the method introduced by Gleicher [2] the constraint function can be any kind of function which makes it very generic. Most constraints apply to space differences like the distance between a point and a body part, the distance between two body parts, the distance between the same point in time, but it could also be a realism function that could express realism in a value. The problem is approached as a constrained optimization problem. Constrained optimization problems are a well studied subject in informatics/mathematics. An advantage of this method is the generic way of defining constraints, because anything for which a target function can be made can be used as a constraint. A large disadvantage is the task of manually defining constraints. Although it is easier than finding people with the right measurements for mocap or key framing the whole motion, it is still a hassle to find the right constraints, at the right time. The downside of approaching the problem as being a constrained optimization problem is there is no solver that can gracefully fail when constraints are conflicting. Moreover it is hard to make such a solver if it is not defined in the constraints or optimization function what a graceful failure is. If there would be any case gracefully handling conflicting constraints it would have to be taken into account when defining the constraints.

For the method of Spatial Relationship Preserving Character Motion Adaptation, the interaction mesh together with some additional constraints achieves good results for all shown cases. The method could possibly also be used for online retargeting when more computational power is available. Nevertheless it is not proven that retargeting can always be expressed in matter of spatial relations to nearby points. Also the way points are chosen is not arbitrary. Currently characters' vertices are joints, but it could be the surface is important. The collision detection solves some problems that start occurring from choosing the joints as interaction mesh vertices. The method could potentially be used in interactive environments, since it is possible to perform the algorithm on a frame by frame base. Unfortunately the results of a frame by frame solution are not shown by the authors.

The morphology independent representation [6] is highly focussed on retargeting from humanoids to humanoids. Although it is possible to adjust the method to convert the morphology-independent representation to a totally different morphology it is not very likely that this will give good results. For retargeting different sizes of humanoids good results are achieved in the time needed. Spacetime methods do not even come close to the timing results achieved with this method. Nevertheless the results sometimes do look unrealistic. Retargeting methods also always need some kind of time component to start adjusting to the goal. This time component is now implicitly given by the motion that is meant for a specific purpose.

The method proposed by Hecker et al. [3] is successful at the task it was designed to perform. This method gets some advantages over other methods by changing the case of retargeting. First of all there is no ground truth motion to compare to. The characters are fictional and nobody knows how those fictional characters move. Also the animators are limited in creating motions, the animator is just searching for the right parameters that are sufficient for all characters. This could mean that character-specific motions and details are lost. Moreover the player of the game is shown the animation while designing the character and in that way steered towards designing good animatable characters. The retargeting also mainly happens in character space, which makes it hard to get inter-character animation. The method is also very discrete with the types for bodies and a separate gait system. This could lead to some motions or characters being impossible to create. Nevertheless all results are achieved in real time and the suspension of disbelieve is not broken with the animations on the characters. Also regeneration of the specialized motion from a generalized one is interesting. Saving only parameters that are necessary for the animation leads to a better understanding of motions.

2.6 Contribution

All of the current methods lack the detection of purpose of the motion. Most retargeting methods are actually just some kind of advanced inverse kinematics. They can reach or avoid points and areas. Such an advanced inverse kinematics technique is needed but there is also a need for some high level controller that instructs the lower level controller in terms of walk to the table until the object can be grabbed normally. Depending on the definition of the problem this is broader than retargeting alone. If the problem is only related to the character and not the environment this is broader than retargeting. If the environment and mood of the character are included in retargeting, pretty much the whole field of animation is subject to retargeting. Would any character from the real world ever perform a motion exactly the same way twice? For this report we evaluate the method by Gleicher [2] because of its generic way of defining constraints and resolving them. It is important to know the limitations of a generic method to know when more specialized methods or better generic methods are needed. It is also good to validate the generality of a generic method.

3 Retargeting using spacetime constraints

This chapter starts with the explanation of a motion definition in section 3.1, the next section 3.2 describes the method of retargeting using spacetime constraints as introduced by Gleicher [2]. Section 3.3 will explain the content of the parameter vector. The following section 3.4 explains the constraint function and how individual constraints can be merged into one function. The target function indicates the most desired pose is explained in section 3.5. In the final section 3.6 of this chapter the relation between different problem representations is shown. The retargeting problem then is translated to a mathematical problem. This problem can be seen as merely a function, but this function can be plotted and the problem can be viewed in its original context of being a motion. The retargeting problem as defined by Gleicher [2] is limited to transferring a motion on a character's skeleton to another character's skeleton with the same hierarchy but with different bone sizes. Limiting retargeting to characters with the same hierarchy has two reasons. First, it simplifies getting a good starting position to adjust the motion, which avoids solving a lot of constraints. Second, it makes it easy to measure the difference between the original motion and the retargeted motion. This difference metric is important for reducing the number of constraints on the motion.

3.1 Motion definition

Before going into detail about retargeting motions, we first define a motion. A motion is a change of morphology over time. Depending on the kind of motion different specific definitions are used. This explanation is limited to skeletal animation, but there are more kinds of animation like particle animation or lattice animation. Skeletal animation defines a skeleton to which a mesh of a character is bound. The morphology of the character is therefore expressed in posed of the skeleton. The skeleton is a hierarchy of joints connected by bones. All joints have a rotation and an offset from their parent joint. For the root joint this offset defines the character's position in space. The skeleton is defined as $S = (J_0, ..., J_n)$ with $J_0 = (p, r)$ and $J_i = (x, p, r)$ for any other i where S is the skeleton, J is a joint, i is the index of a joint, x is the reference to the parent joint, p is the offset position and r is the rotation. A motion on a skeleton is defined as a function that gives the root position and the joint rotations at a moment in time.

$$M_t = (p_{0,t}, r_{0,t}, ..., r_{n,t}) \tag{3.1}$$

where M_t is the pose of the character at moment in time t, $p_{0,t}$ is the position of the root joint and $r_{i,t}$ is the rotation of joint J_i at time t.

3.2 Problem definition

All motions that we use for retargeting are skeletal animations, but the definition of the original motion is not important for the retargeting. When the bones on the original skeleton are sized to the target sizes, the angles on the joints remain the same. This method would be sufficient if the scaling is uniform and when there is no interaction with the world or the world is also scaled uniform to the character. Unfortunately scaling in such a matter is hardly useful. Therefore we look at the case where the world is not scaled accordingly and the skeleton is not scaled uniformly. In those cases the motion does not satisfy all constraints. For example the feet no longer touch the floor or penetrate the floor when standing. When the character grabs something it misses the object.

The solution for this problem is trying to resatisfy the constraints on the motion. If the user would be able to define any constraint and the algorithm would be able to satisfy any constraint the problem would be solved. But some constraints are very hard to define in a satisfiable way. For example style is hard to cover in a constraint therefore another demand is made from the system, namely satisfy the constraints but change as little as possible to the original motion.

Retargeting in space has several input parameters and result values. The input parameters consist of the original motion, constraints on that motion and the target scaling. The result is a new motion for the target skeleton. To solve the problem we need to make a proper definition. To explain the definition we are first going to forget the temporal aspect. We only consider space solving, so we want to match constraints on a pose. In the next section we add the temporal aspect and adjust the problem definition accordingly.

3.2.1 Space solving

The problem is defined as minimizing $g(x)$ constrained by $f(x) \diamond c$. The result of the function $g(x)$ is larger when the pose looks less like the original motion. The definition of the constraint is $f(x) \diamond c$ where $\diamond$ is one comparison operator of the set $\{=, \leq, \geq\}$. $f(x)$ represents a property of a pose and c defines the value of the property it is constrained to. For space solving x is a parameter vector describing the pose of a character. The final solution uses another kind of parameter vector as can be read in section 3.3. For space solving all constraints apply since they are not restricted to certain periods in time. Solving this problem means finding changes for x such that $f(x) \diamond c$ is true. It depends on the solver how these changes are found. It is best understood with a simple gradient-based solver like gradient descent. Solving this problem with gradient descent means finding the effect every value of x has on getting close to solve $f(x) \diamond c$. Getting closer to solve $f(x) \diamond c$ means getting a smaller difference between $f(x)$ and c if the constraint is violated. This matrix of changes is the Jacobian matrix is discussed in Chapter 5. Steepest descent will result in a vector x_{new} for which the multiplication with the Jacobian results in the negative constraint vector. By applying this x_{new} to your current x you will find x that is closer to your goal $x \leftarrow x + x_{new}$. If all functions were linear the solution would now have been found but this process has to be repeated because the gradient changes. When solving the pose with a parameter vector describing the joint angles, this means changing the joint angles to better solve the constraints.

3.2.2 Moving from space solving to spacetime solving

When including time it is not a matter of solving constraints multiple times for different timestamps. Gleicher [2] uses a B-spline to interpolate over time. The article does not state why the choice for a B-spline has been made other than saying they want a smooth motion. We are going to explain what impact the use of a B-spline has. More about the B-spline can be found in Chapter 6. The control points of a B-spline have effect over a timespan.

3.3 Parameter vector

The parameter vector describes the change that has to be made to the original motion $M = M_{ori} + M_{change}$. The parameter vector x describes M_{change}, this is similar to the normal motion definition as described earlier in section 3.1. The explicit definition of x is $x = (x_{0,0}, ..., x_{n,0}, x_{n,1}, ..., x_{m,n})$ describing all control points $x_{i,u}$ on B-spline x_i. A pose at time t is described as M_t. A single value from the vector describing the pose is $M_{t,i}$. Those values are defined as $M_{t,i} = bspline(x_i)_t$, where i is the index of the value. $x_i = (x_{0,i}, ..., x_{m,i})$ where m is the number of B-spline control points.

3.4 Constraint function

In this section this function f and the value it is compared to (c) are considered. A constraint is defined as a limit of the possibilities of the motion. That is a vague definition since it raises the question what are possibilities or even better what is one possibility and how do possibility limits group together to a constraint. Those vague definitions relate back to the mathematical definition $f(x) \diamond c$. One dimension can be seen as one possibility that is limited. And all limited possibilities are grouped by this function. So the constraint is a collection of all possibility limits. The function f calculates some property of the motion and compares this to a constant c. Several function definitions exist to create the same constraint. For the function to be the correct constraint function it only has to go through one point. The comparison operator only defines whether the result has to be on the smaller or bigger half space or on the point itself. Purely mathematically it does not matter if the function is discrete, continuous or has a limited order of continuity. Mathematically your function could be isViolatingConstraints(x) = 0 where the result is either 0 or 1. But computationally it might matter; no known solver will be happy with the latter function.

3.4.1 Different operators

The $\diamond$ in $f(x) \diamond c$ is actually a vector of operators. Each element is compared individually to keep the dimensions of the vector independent from the other dimensions. Some solvers like the conjugate gradient solvers always solve only $f(x) = c$. To also solve for comparison operators $\{\leq, \geq\}$ an active set[1] is used. When using an active set, the constraints that are already fulfilled are not applied. This means when the $>$ or $<$ constraints represent a space, the solution using only this constraint will be on the edge of the space. Unless the solver either finds intermediate solutions that pass the edge of the constraint or if other constraints require the solution to be somewhere else than on the edge of the solution space.

3.5 Target function

The target function $g(x)$ is the function that is minimized to make the motion look as similar as possible to the original motion. The definition for this function is taken from Gleicher[2]. This definition is:

$$g(q) = \frac{1}{2} qMq \tag{3.2}$$

where M is a diagonal matrix containing the weights and q is the parameter vector. This is a short notation for a common way of calculating a weighted difference

$$g(q) = \frac{1}{2} \sum w_i q_i^2 \tag{3.3}$$

This definition implies the target function can never be fully optimized unless the constraints are met by just transferring the pose from one skeleton to another. For every other situation the

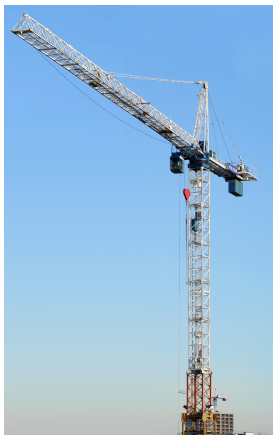

Figure 3.1: Example of a tower crane (source: Wikipedia, 2013)

parameter has to change and any parameter value other than zero will result in a value higher than zero.

3.6 Different problem representations

In this section we discuss different views of the problem. With our problem definition we have abstracted the retargeting problem to solving a mathematical problem. Since this problem definition is an equation it can be plotted in a graph to get another representation. All three of those definitions show different issues. To handle and judge the importance of those issues it is important to know how to relate those issues between the different representations. The parameter vector is a set of independent parameters and the constraint vector is a set of dependent parameters. This helps with picturing the shape that is created for a function plot. The space created by independent parameters is completely filled while for dependent parameters only the existing points get values and no new points are created. Purely mathematically this is not true. For example for $f(x) = \sqrt{x}$ there could be two solutions. Namely $\sqrt{4} = 2 \vee \sqrt{4} = -2$, but this does not matter in our case because we assume a function to always have one result. If cases with multiple answers should be evaluated all possible options should be evaluated separately as if every case only has one result.

3.6.1 One parameter to one constraint

In the previous sections we have discussed the relationship between the retargeting problem and the mathematical view on this problem. The graph view for the a normal motion is impossible to imagine because of its very high dimensionality, but very simple cases can be visualized and help with understanding the bigger case. First we start with a very simple example. We are going to only do space solving and we have a character, which in our example is a tower crane (See fig 3.1 for an example). For the sake of the example the crane has only one degree of freedom namely the rotation ($crane_{rot}$). The hook of the crane cannot move along the arm. For simplicity we take the length of the arm to be 1 unit. The case we want to solve is how to rotate the crane so an object can be placed on the road. A top-down view of the situation is illustrated in Figure 3.2.

In this case our parameter vector is $x = (crane_{rot})$ and our constraint function is $f(x) = sin(crane_{rot})$ and our constraint value is $c = road_{distance}$. This situation has two solutions as can be seen in figure 3.3.

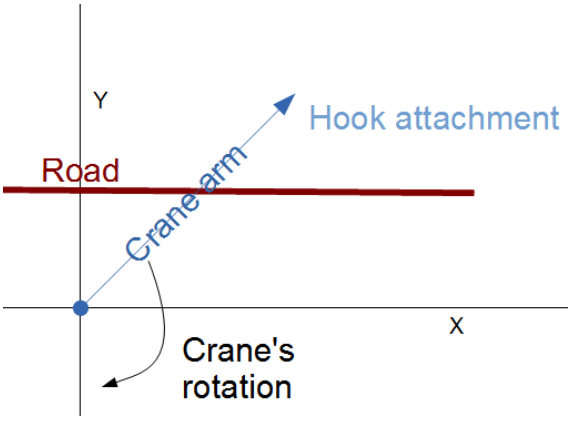

Figure 3.2: Situation sketch of the crane and the road

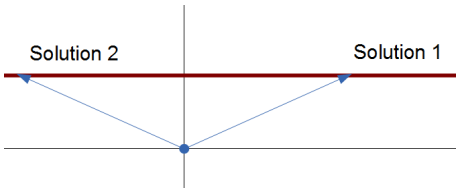

Figure 3.3: The two possible solutions

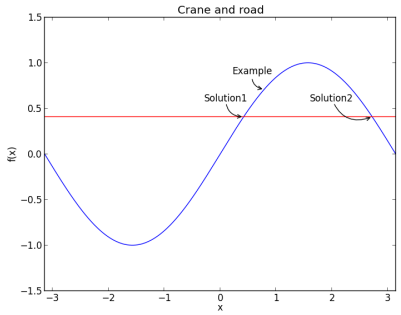

Figure 3.4: Graph of all possible configurations, this shows the relation between x and $f(x)$, the blue line and the result of $f(x)$ and the red line is c

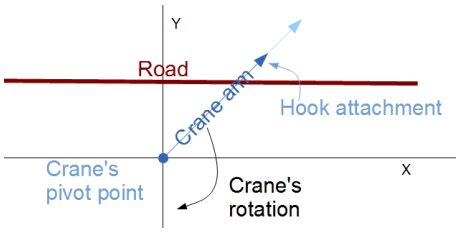

Figure 3.5: Crane with a hook that can move along the arm

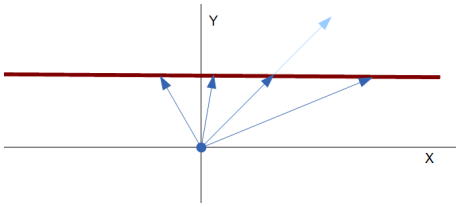

Figure 3.6: Some example solutions of the crane with a moving hook

3.6.2 Two parameters to one constraint

To get an idea of how a multiple dimensional input vector relates to a constraint we are going to make 3d plots. In the first example we are going to give the crane an extra dimension. The hook is now allowed to move along the arm. A schematic representation of this is shown in Fig 3.5. Contrary to the first example with one degree of freedom there is no longer a finite number of solutions. Some solutions are shown in figure 3.6. The corresponding 3d graph is shown in figure 3.7. In this graph the intersection between the red surface, represented as a grid (the constraint value) and the blue surface (the constraint function result) represents all solutions.

3.6.3 One parameter to two constraints

To show how multiple constraints relate to one parameter we are going to try the point (solution1) of the first example. This point can be found by defining one constraint as the y-distance and the other as the x-distance. So $f(x)_1 = cos(x), f(x)_2 = sin(x)$ and $c = (point_x, point_y)$. The situation is illustrated in figure 3.8. A single value of c is a plane parallel to the dimension of the constraint function as we have seen in the previous section with one constraint and multiple inputs. With multiple constraints the shape is the intersection of all those planes.

3.6.4 Combining this to any dimensionality

Unfortunately the problems that usually occur are high dimension parameter vectors relating to high dimension constraint vectors. These cannot be intuitively plotted. But projections can be made. With those projections it is possible to get more intuition about how the function graph looks. With more intuition about the function graph it is easier to solve problems that occur during retargeting.

3.6.5 Local minima

This section handles local minima in the different views. In the graph view we see local minima as the lowest point in a "valley". The mathematical definition of a local minimum is $f(x^*) \leq f(x)$ when $|x^* - x| < \epsilon$, meaning there is no point x in the close surroundings of $x*$ which results in a lower result of $f(x)$. In the skeletal view a local minimum is a case where a constraint first has to

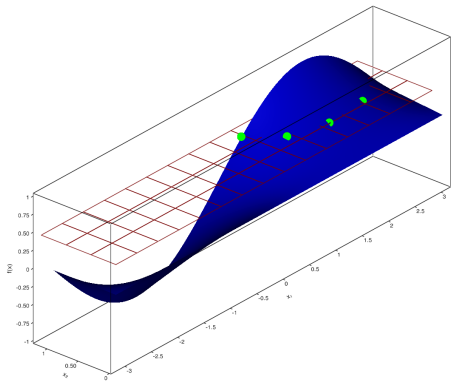

Figure 3.7: The graph of the crane picking something up from the road with a moving hook, the red grid is the surface representing the constraint value, the blue surface is the constraint function result and the green dots are the points from the example shown in Figure 3.6

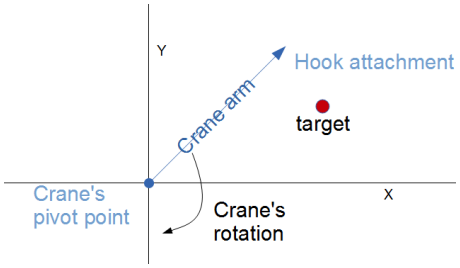

Figure 3.8: Situation sketch of the crane and a target

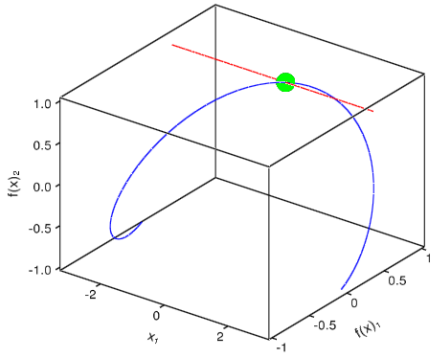

Figure 3.9: A plot of the crane reaching a point, the blue curve describes all constraint values for all possible inputs, the red line is the intersection of both constraint values.

result in a higher value ($f(x^*)_i < f(x)_i$ where i is the index to any value in the resulting vector) to eventually solve all constraints, for example when a joint first has to move away from a position to eventually reach it.

3.6.6 Unreachable constraints

Unreachable constraints are constraints for which there is no x where $f(x) = c$. There are two kind of unreachability, either the input is never able to create a shape matching a constraint or there is not input for which all constraints can be met. In the graph view this means that there is no area where all constraints' shapes intersect with the function shape. In the skeletal view it means no motion can be made where all constraints are met.

4 | Solving spacetime constraints

This chapter discusses how to compute the solution to the optimization problem stated in the previous chapter. Different methods exist to solve an optimization problem, but in practice none of those actually reach the optimal goal. Another issue with the current approach as discussed in the previous chapter is that the whole motion has to be solved at once. In this chapter we discuss a possibility to solve a motion in fixed steps in section 4.1. Furthermore we discuss other practical issues like setting weights for constraints in section 4.2 and thresholds for solving the optimization problem in section 4.3.

4.1 Solving in parts

In the optimization problem as stated in Section 3.2 the whole motion has to be solved at once. The space time optimization is performed over the whole motion, while it is not very likely that a change at the start of the motion will influence a constraint at the end of a motion. Doing spacetime optimization over the whole motion with a standard non-linear optimization library could already have major performance influences in simple cases. In a motion where some joint has to be on the original joint position during the whole motion, constraints are added for every fixed interval in the motion. Causing the constraint vector to grow over time. The parameter vector will also grow overtime since it contains all control points of the B-spline and those are also added in a fixed interval. The constraint vector defines the size of the resulting vector. Solving an optimization problem usually becomes easier with gradient information, but the gradient to this problem would have the width of the parameter-vector size and the height of the constraint vector size. Since both vectors grow linearly with the motion length in the simple example the gradient grows quadratically. The gradient is this big only because it is not derivable from the equation that the first control point does not effect the last constraint. With knowledge of the problem it is possible to reduce this quadratic complexity. For this we can use properties of the B-spline. A B-spline control point has influence over a limited time span and a time span is influenced by a limited number of B-spline control points. More about the influence of B-spline control points can be found in Chapter 6.

4.2 Weights

Different parts of the algorithm as described in Section 3.2 need weights. In this section the weights are split between weights needed for the optimization function and weights needed for the constraint function.

4.2.1 Optimization function

The first part is the optimization function. For this function, the influence of a degree of freedom on the pose is weighted. Changes in the root rotation have a large influence on the pose while a change in rotation of a finger bone is hardly noticeable. For this, the weight set proposed by

[8] is a good suggestion when used for joint angles. This is a weight set for blending motions although the requirements for blending are not exactly the same as for our optimization function. The requirement for blending is the transition between two motions. This requirement is good while we require the motion to look as similar as possible to the original motion. Difference measurements used in blending could also be useful for the optimization. A survey on motion blending is done by Basten et al. [7].

4.2.2 Constraint function

Another part that needs weighting is the result of the constraint function. Although in theory a constraint is always met no matter what the weighting is. In practice a constraint is always met within some threshold as will be discussed in the next section. For thresholds it is desirable if all constraints suffer equally from a change in the threshold. This kind of weighting can be fixed by a single weight set by the animator, but it is also possible to split this up into more weights. The first kind of weight is the weight to balance between units. Is the correction of 1 meter from a point constraint as important as 1 degree on the limit of a joint? For a degree of a joint the weight of a joint itself is also important. A weight set discussed for the optimization function can be used. Besides the weights for units there is also a correctional weight for a functional constraint. One dimension in the constraint function does not necessarily have to be one functional weight. By using multiple dimensions for one function constraint, it implicitly gets weighted higher. By adding all dimensions to a group and dividing all members of the group by the group count the weight is corrected. Although this is related to the weights of units, it is not the same. For example distance could be expressed in Manhattan distance or Euclidean distance that are both expressed in meters. Another kind of weight is the artist priority weight. For example the focus of the camera can improve the importance of a certain constraint or the colour of the feet attract attention raising the importance of constraints on the feet. The final kind of weight should be used for solving in parts, to correct for the influence of the control point at different points in time.

4.3 Thresholds

It is hard to find the exact solution for a constrained optimization problem on a computer. Thresholds can be applied to different parameters and measured in different ways. Thresholds in the algorithm can be applied to data, like input parameters or result values, but also be applied on meta data, like system time duration and number of iterations. Thresholds on meta data are usually focussed on getting guaranteed results while thresholds on the data are focussed on the quality of the results. In our implementation we use a threshold on the constraint values. Each value from the constraint vector is individually compared. The value is within the threshold if the difference between each constraint-vector value is smaller than a given threshold value. This threshold value is the same for all constraint values because the importance of constraints is equalized by the use of weights and because this equality is important for some solvers.

5 Constraint function differentiation

This chapter concentrates on the function differentiation. Function differentiation is important for solving equations numerically. The differential represents the slope of the function. This slope can be used for making better estimations of the next point closer to the goal. This chapter starts with the definition of a derivative in section 5.1. The next section 5.2 will be on calculating the Jacobian (the derivative of a vector-valued function). The different methods are discussed in sections 5.3, 5.4 and 5.5 that discuss divided differentiation, symbolic differentiation and automatic differentiation respectively.

5.1 Definition

This section shows the definition of a derivative and more specifically the definition of a Jacobian. The derivative for any function $f(x)$ is defined as $f'(x)$, this can also be notated as $\frac{\delta f}{\delta x}$. Although we keep using the first definition $f'(x)$ throughout this chapter, the latter definition is more useful for explaining the derivative. δ means the change in, so the derivative is the change in function result f divided by the change in input x. For a function with a single parameter and a single result this will result in a single value for the derivative, but for a vector valued function $f_{1,...,m}(x_1, ...x_n)$ with vector result the derivative is the Jacobian J, a matrix of partial derivatives.

$$J = \begin{bmatrix} \frac{f_1}{x_1} & ... & \frac{f_1}{x_n} \\ ... & ... & ... \\ \frac{f_m}{x_1} & ... & \frac{f_m}{x_n} \end{bmatrix} \tag{5.1}$$

One value in the matrix is the change of one input vector value on one result value. This is needed for every value in the input vector to every value in the output vector.

5.2 Calculating the Jacobian

This section discusses how to get the Jacobian of a function. There are multiple ways of differentiating a function. One of the ways is for a programmer to analytically differentiate a method to get the differential values of a function. This method is accurate, but it is tedious work and hard to maintain. Automated methods are symbolic differentiation, divided differentiation and automatic differentiation. Those methods do not require manual labour and are therefore more attractive. From those methods, divided differentiation can be seen as a black box method, only the input and output of a function have to be known to differentiate it. While the other two methods, symbolic differentiation and automatic differentiation, require knowledge about all parts of the function. There has been some discussion about whether or not automatic and symbolic differentiation are actually identical. In the paragraph about automatic differentiation we will

discuss what we believe sets automatic and symbolic differentiation apart. Others say automatic differentiation is the same as symbolic differentiation.

5.2.1 Differentiation rules

Differentiation rules are the set of rules that are used to get a derivative of a function. For each component of a formula there is a rule on how to translate that component to the derivative. The chain-rule defines how those rules can be applied recursively. This section will show a collection of the basic rules that can be used for differentiation.

Constant rule $(af(x))' = af'(x)$

Sum rule $(f(x) + g(x))' = f'(x) + g'(x)$

Product rule $(f(x)g(x))' = f'(x)g(x) + f(x)g'(x)$

Chain rule $(f(g(x))' = f'(g(x))g'(x)$

Power rule $(x^n)' = nx^{n-1}, n \neq 0$

Reciprocal rule $(\frac{1}{f(x)})' = \frac{f'(x)}{f(x)^2}$

Quotient rule $(\frac{f(x)}{g(x)})' = \frac{f'(x)g(x) - g'(x)f(x)}{g(x)^2}$

5.3 Divided differentiation

Divided differentiation is also called numerical differentiation. This technique is based on looking to a point before the point and a point passed the point and dividing the difference of the result by the different in. This is illustrated in Fig 5.1. So

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} \tag{5.2}$$

where h is a small step. If no rounding errors occur the accuracy of the differentiated function increases when h decreases. Unfortunately with smaller h rounding errors start getting larger. This could intuitively be explained by the influence of the rounding getting bigger with smaller values. With a single valued function divided differentiation takes two function evaluations. For vector valued functions this is different.

A naive approach to translating from single valued to vector valued is

$$f'(x_{1..n})_{1..m} = \frac{f(x_{1..n} + I_{1..n}h)_{1..m} - f(x_{1..n} - I_{1..n}h)_{1..m}}{2h} \tag{5.3}$$

The result of this is actually a differential of a scalar by which x is multiplied. The proper differential is considerably more complex to define. First of all $f'(x_{1..n})_{1..m}$ can not be true, because the result is two dimensional as discussed before, so the right definition is: $f'(x_{1..n})_{1..m,1..n}$. Instead of adding and subtracting h from all values of x at once, h is only added to and subtracted from one value x_i per function evaluation. To describe this, we define h_i to be a vector of size n with all zeroes except for position i where the value is h. The final definition for the divided difference is

$$f'(x_{1..n})_{1..m,i} = \frac{f(x_{1..n} + h_i)_{1..m} - f(x_{1..n} - h_i)_{1..m}}{2h} \tag{5.4}$$

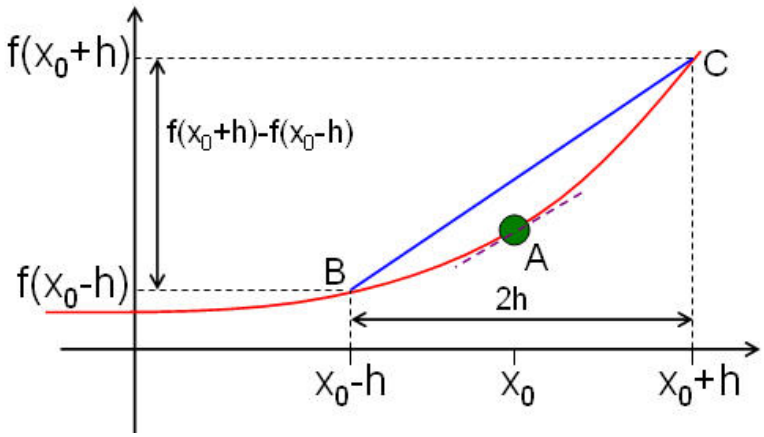

Figure 5.1: Illustration of divided differentiation (source: http://kineticmaths.com/index.php?title=Numerical_Differentiation, 2013)

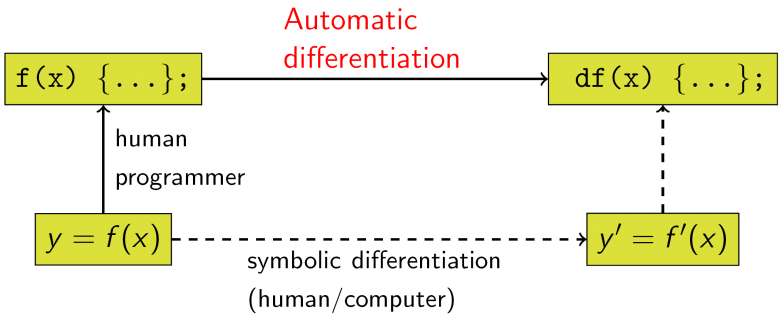

Figure 5.2: Difference between symbolic and automatic differentiation(source: Wikipedia, 2013)

Now the number of function evaluations is increased. Since the divided difference is done for every i and $1 \leq i \leq n$ there is a total of $2n$ function evaluations.

5.4 Symbolic differentiation

This paragraph shortly describes symbolic differentiation as a preparation for the comparison with automatic differentiation. Symbolic differentiation is sometimes used to describe the application of differentiation rules in general, while in other cases it refers to only this application by a computer. With symbolic differentiation the goal is to find the formula that gets the derivative instead of a function that provides the result of the derivative. Figure 5.2 makes an illustration of the relationship between symbolic and automatic differentiation. For the explanation of the difference it is important to make a difference between the mathematical definition and the programmatical definition. The mathematical definition is the definition how humans do math like all equations in this report. The programmatical definition is the definition in computer instructions. The difference between automatic differentiation and symbolic differentiation is although it is done on a computer the symbolic differentiation is done on the mathematical definition of a function, while the automatic differentiation is done on the programmatical definition. To illustrate symbolic differentiation we consider the following two statements:

$$f(x) = 5x^2 \tag{5.5}$$

$$g(y) = ysin(y) \tag{5.6}$$

If we want the derivative of $g(f(x))$ over x we have to first merge the two statements into one equation.

$$g(x) = 5x^2 sin(5x^2) \tag{5.7}$$

This equation is already bigger than the first two separately. The application of all the differentiation rules as described in section 5.2.1 results in:

$$g'(x) = 10xsin(5x^2) + 5x^2 cos(5x^2)10x \tag{5.8}$$

This statement has already grown compared to the two individual statement from the start. When there is another function that is using g all of the statement are reused multiple times again. Resulting in an exponential growth. Of course these statements can be simplified again, but this is not always very trivial.

5.5 Automatic differentiation

Automatic differentiation (autodiff) is differentiation based on the chain rule as described in section 5.2.1, but as said before it is different from symbolic differentiation. Autodiff benefits from the property of computers that they execute a sequence of instructions. This is exploited because every part of a differential equation results in a value. For the chain rule those values are used instead of the whole statement. This can be done by source code transformation or operator overloading. Source code transformation analyses the code as text and writes new functions. Operator overloading is rewriting the algebraic operators to use the differentiation rules, and rewriting primary functions, like sin and cos, to use their derived variant. Usually this is done by using another type for differentiation and changing the operators for that type. The following is an example to show how autodiff works. For equation 5.5 and 5.6 in code this is:

```
double f(double x)
{
    double a = x * x;
    return 5*a;
}

double g(double y)
{
    double b = sin(y);
    return y*b;
}
```

through code transformation this will become

```
autodiffval f_d(autodiffval x)
{
    autodiffval a;
    a.val = x.val * x.val;
    a.val_d = x.val * x.val_d + x.val_d * x.val;
    autodiffval c;
    c = 5 * a;
    c.val_d = 5 * a.val_d;
    return c;
}

autodiffval g_d(autodiffval y)
{
    autodiffval b;
```

```
    b.val = sin(y.val);
    b.val_d = cos(y.val) * y.val_d;
    autodiffval d;
    d.val = y.val * b.val;
    d.val_d = b.val * y.val_d + b.val_d * y.val;
    return d;
}
```

The advantage is that every statement always results in two other statements: the normal computation and the derivative computation. This means the computational complexity of the derivative is the identical to that of the normal function. Another advantage is that this kind of transformation can be run over and over, generating higher order derivatives. The general rule of rewriting is best shown by showing a code-class for the operator overloading variation of autodiff:

```
class autodiffval
{
    double val;
    double val_d;
public:
    autodiffval(double val, double val_d) : val(val), val_d(val_d)
    {
    };

    autodiffval operator +(autodiffval other)
    {
        return autodiffval(val + other.val, val_d + other.val_d);
    }

    autodiffval operator -(autodiffval other)
    {
        return autodiffval(val - other.val, val_d - other.val_d);
    }

    autodiffval operator *(autodiffval other)
    {
        return autodiffval(val * other.val, val_d * other.val + val * other.val_d);
    }

    autodiffval operator /(autodiffval other)
    {
        double newval = val / other.val;
        double newval_d_upper = val_d * other.val - val * other.val_d;
        double newval_d_lower = other.val * other.val;
        double newval_d = newval_d_upper / newval_d_lower;
        return autodiffval(newval, newval_d);
    }
};
autodiffval sin(autodiffval param)
{
    return autodiffval(sin(param.val), cos(param.val) * param.val_d );
}

autodiffval any_function(autodiffval param)
{
    return autodiffval(any_function(param.val), any_function_d(param));
```

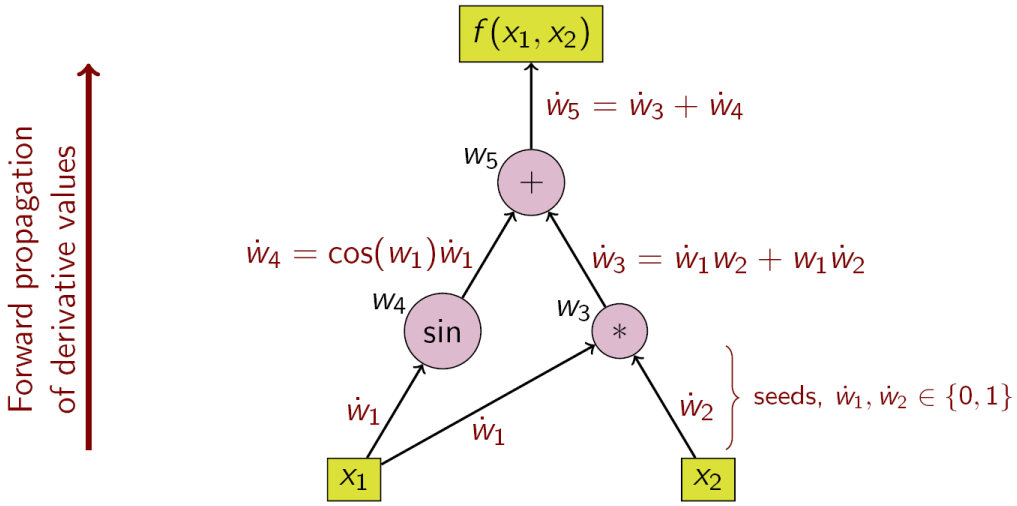

Figure 5.3: A computational graph of forward accumulation(source: Wikipedia, 2013)

}

The last example of any_function shows that it is even possible to use your own optimized derivatives for some function using autodiff. Until now we have discussed forward accumulation. Forward accumulation answers the question: "What effect does this independent (input) variables have on my dependent (output) variables?" There is also backward accumulation, for backward accumulation the question is: "What independent (input) variable have effect on this dependent (output) variable" Although the questions are different, the answer to them is the same for functions that only have one input and one output variable.

Backward accumulation

Backward accumulation is better approached with a computation graph. This directed graph shows what operations are done on the input variables to get the values of the output variables. See figure 5.3 for a computation graph with annotations for forward accumulation. The graph is traversed from the input variable towards the output variable. Each node in this graph is an operation and the incoming edges represent input values and the outgoing edges the output values. The start and end nodes can be seen as special assignment and read operations. The idea for backwards accumulation is to traverse backwards through the graph as shown in figure 5.4. Until now we have only discussed functions with one input variable and one output variable ($\mathbb{R}^1 \to \mathbb{R}^1$) but for functions with higher dimensions ($\mathbb{R}^n \to \mathbb{R}^m$) multiple sweeps have to be done. Since forward and backward accumulation start from either one input or one output node in the computation graph they only find the derivative for that parameter. To find the derivatives for all parameters multiple sweeps have to be done for either all input variables (forward accumulation) or for all output variables (backward accumulation). This relates to calculating rows or columns for the Jacobian. Each sweep calculates a row or column depending on the kind of accumulation. The cost of one sweep compared to a particular algorithm is not dependent on the number of parameters, but the number of sweeps is. Therefore forward accumulation should be used if $n < m$ and backward accumulation if $m < n$. The advantages of autodiff are it is accurate up to machine precision. It can be faster than divided differentiation, depending on the number of input and output variables. It is still possible to plug another differentiation algorithm in for a part of the code base. The disadvantage is the code has to be templated (at least for operator overloading). Making code ready for templating is not always easy when using external libraries.

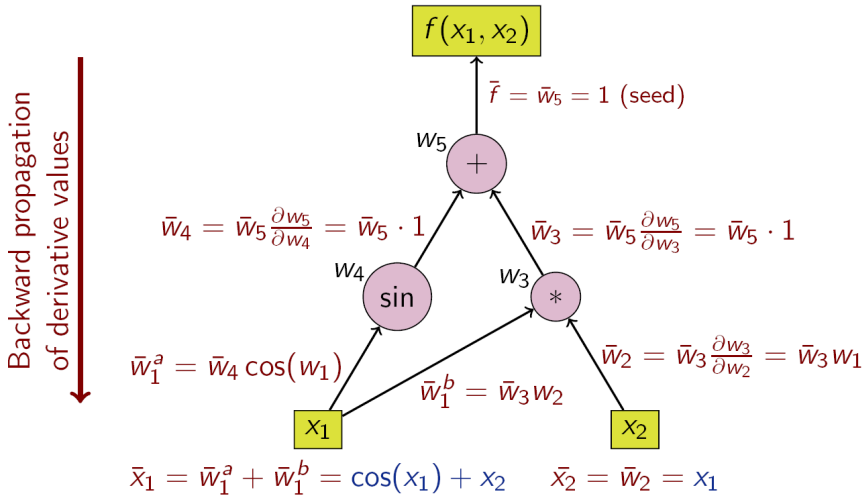

Figure 5.4: A computation graph of backward accumulation(source: Wikipedia, 2013)

5.6 Conclusion

This chapter has shown different methods of differentiation. Divided differentiation has lower accuracy than the two white box methods. Autodiff takes less computation time than the other methods and seems superior. Nevertheless divided differentiation could in practice be useful because of its black box properties. It could take a lot of programming time when the code has be transparent for an autodiff library. Creating a mathematical definition for the whole code base so it could be differentiated symbolically could also be very labour intensive. Most coding is not started from scratch and uses libraries, those do not provide all information needed for symbolic and automatic differentiation. Autodiff nevertheless provides options to use a user-specified definition of a differentiated function.

6 | Interpolating over time

This chapter discusses all aspects involved with interpolating over time. The chapter starts with the definition of the B-spline in section 6.1. The next two sections define two aspects needed to locally calculate a point on a B-spline. The first is the reference index to a control point in section 6.2 and the next section 6.3 shows the range of control points that have influence on a point on the B-spline. Those two sections are followed by a section 6.4 about the degrees of freedom of a B-spline and the range of influence of those. This chapter is closed by a motivation of the choice to use a B-spline for interpolation in section 6.5.

6.1 The definition of a B-spline

A B-spline, just like any other spline, is a 1-dimensional continuous shape. The values of this shape change over dimension t. The B-spline is described as a set of control points. Those control points have a positional value and a t-value. The B-spline also has a dimension n to indicate the number of control points involved for a point on the spline. When processed by the B-spline function they describe all points on the B-spline. The function definition is described as follows:

$$S(t) = \sum_{i=0}^{m-n-2} P_i b_{i,n}(t) \tag{6.1}$$

Where S is the function for the spline, t is a value in the t-dimension. i iterates over the control-point indices m. The B-spline dimension is described as n and P_i is the positional value. The weight of a positional point is given by the weight function b. The weight function is defined in the following two equations. The first equation defines the weight for the 0-dimension.

$$b_{j,0}(t) := \begin{cases} 1 & \text{if } t_j \leq t < t_{j+1} \\ 0 & \text{otherwise} \end{cases} \tag{6.2}$$

For all other dimensions the following is true:

$$b_{j,n}(t) := \frac{t - t_j}{t_{j+n} - t_j} b_{j,n-1}(t) + \frac{t_{j+n+1} - t}{t_{j+n+1} - t_{j+1}} b_{j+1,n-1}(t) \tag{6.3}$$

As can be seen from the equation the weight function is recursive for the B-spline dimension. In both equations t is the value for which the point on the spline is calculated. t_x is the t value for the control point on index x. j is the index of the control point vector and n is the dimension for which the weight is calculated.

6.2 Finding a knot vector index for a B-spline point

Usually in computer science the B-spline is evaluated by finding all points on the B-spline, while in this case we want to find the point on the B-spline for some t. Because of performance reasons

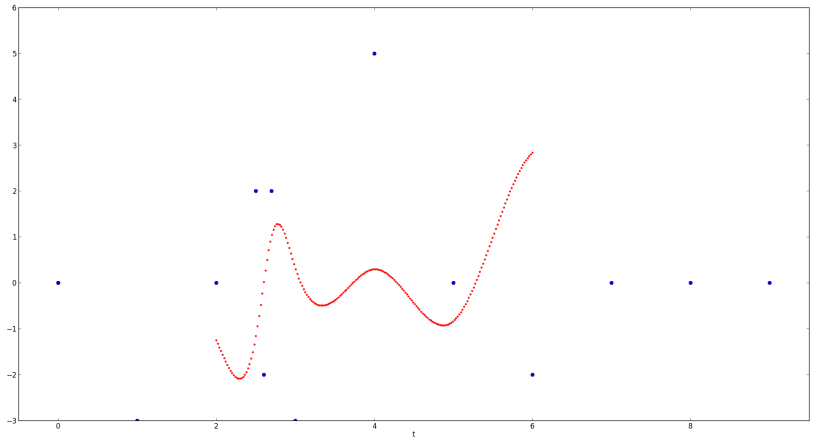

Figure 6.1: A B-spline with $(P,t)_i = (P_i, t_i)$

we do not want to evaluate all points on the spline to find the value at some t. In order to find the range of i for some t. So we need all i for which the weight $b_{i,n}(t) \neq 0$. For $b_{i,0}(t)$ it is clear i is not zero for $t_i \leq t < t_{i+1}$ as can be seen in equation 6.2. The knot vector can be binary searched for that i. We define this i which is defined by the knot vector i_o and define all other i relative to i_o. For $b_{i,n}(t)$ to return something other than zero at least $b_{i,n-1}(t)$ or $b_{i+1,n-1}(t)$ has to return something other than zero. Now we define the relationship between the functions of b with an n other than zero. $b_{x,n-1}$ will call both $b_{x,n-1}(t)$ and $b_{x+1,n-1}(t)$ x will only change through $b_{x+1,n-1}(t)$ and this can only happen n times since the limit on n is zero. So the maximum x in $b_{x,0}(t)$ to be called is $b_{i+n,0}(t)$ since only $b_{i_o,0}(t)$ will return something other than 0. $i_o - n$ will be the lowest i to return something other than zero. The lowest i in $b_{x,0}(t)$ is $b_{i,0}(t)$ when $b_{x,n-1}(t)$ calls itself recursively, any other combination will return i between $i_o - n$ and i_o.

6.3 Range of indices needed for B-spline point

So now we know the knots for which t has a value. But we do not know which knots have to be evaluated to know the value of $b_{i,n}(t), i_o - n \leq i \leq i_o$. For $b_{i,0}(t)$ only i and $i+1$ are evaluated. For $b_{i,n}(t)$ the evaluations of i are $t_j, t_{j+n}, t_{j+n+1}, t_{j+1}$ of those j is the lowest and $j+n+1$ is the highest. We do not have to worry about recursion any more because that is already done in defining the range of i. So the range of i to be evaluated is $i_o - n \leq i \leq i_o + n + 1$.

Since both P and t are accessed through i it is implied that a tuple $(P,t)_i$ could be made. This can be done but there are some oddities. The first one is there are more values for t than there are values of P that are used. This could easily be solved by just using 0 for P, or the last used value. The second oddity is that the resulting B-spline looks counter intuitive. Because the effective range of i selects control points with t_i smaller than t it looks like the effect is delayed. It will look more intuitive if $(P,t)_i = (P_{i+(n/2)}, t_i)$ because then half of the effective control points are positioned before the effective point and half are positioned behind it. This works best when n is uneven. Because the middle control point also gets the biggest weight it looks as if the control point is pulling the spline towards it. When using non-uniform B-splines the spacing looks more intuitive. See figures 6.1 and 6.2 for an example.

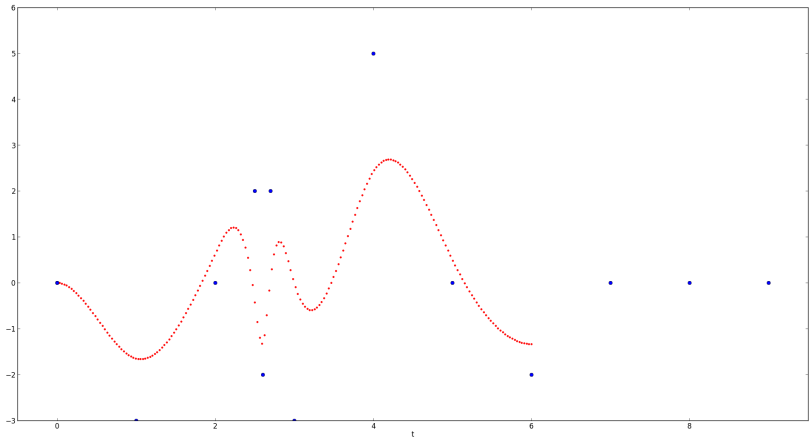

Figure 6.2: A B-spline with $(P,t)_i = (P_{i+(n/2)}, t_i)$

6.4 Degree of freedom and influence

This section is about the degrees of freedom of the B-spline and how this influences the number of individual constraints can be set on a time span.

The degrees of freedom of a B-spline do not increase with the dimensionality. Although multiple control points have effect on a time span, they also have effect on multiple moving time spans. This means when increasing the degrees of freedom on one time span it decreases on a timespan close to it. Nevertheless it seems impossible to create multiple minima/maxima within a single time span.

6.5 Choice for the B-spline

The B-spline is one of many ways to interpolate over time. We implemented this method because it was used by Gleicher [2], but it was not a thorough explanation. It stated that the choice for B-spline was made to control the frequency of the spline. But this can also be done with other kind of splines, an interpolation through points could be useful for giving more meaning to the values in the input vector. Next to that the weight function of the B-spline is very computationally expensive because of its recursive nature. In implementation described in the original paper, this is solved by using a uniform B-spline, for this kind of B-spline the weight function can be simplified.

7 | Experiments

In this chapter some experiments are set up to evaluate the produced motions. Retargeting can be done with a high variety of parameters. It is not trivial all of those parameter give equally qualitative results. This chapter starts with the experimentation plan in section 7.1. Here all the details of the experiment are covered. In the next section 7.2 the environment in which the experiments are run is discussed. Finally this chapter is closed in section 7.3 with the results that are obtained during the execution of the experiments.

7.1 Experimentation plan

This section shows the research questions that the experiment is going to answer. Moreover the hypotheses are shown and the variables of the experiment will be clarified.

7.1.1 Research questions and hypothesis

Using the method of retargeting using spacetime constraints, a framework is created to retargeted motions. This framework offers many possibilities. With the experiments we would like to find out what the performance of this framework is with different parameters. This section shows three different groups of research questions that are handled in the subsections. For each subsection the research questions are formulated and the are followed by the hypothesis for that question. The first section handles questions related to the constraint definition. The second section handles the performance for different kinds of motions and the last section handles different scalings.

Constraint definition

The first main question is: How should constraints be defined? This question is challenging to answer by experiments. That is why more specific questions are asked. What is the quality of the motion with a specific kind of constraint? We are going to ask this question for three kinds of constraints, a constraint where all the important joints have to be on the same position, what joints are important depends on the motion. So in the reaching motion which is discussed later we are going to set both the ankles and the wrists to be on the same position. For a walking motion only the feet are important. Another type of constraint is where the joint is constraint to the original position at an important moment. We are going to test this on the reaching motion and in this motion an important moment is the point where the joint at the farthest point during the reach. The final type of constraint is where the distance to the important moment is interpolated. It is expected that the smoothing supplied by the B-spline is not enough to cover the distance between the joint position in the scaled motion and the joint position in the original motion. Therefore we try to control the distance covered by the joint per time by setting distance constraints before the important moment. This distance is interpolated to zero over the approaching time. For different constraints it is hard to predict the quality based on the type of constraint. For the bone_original_position constraint the path of the bone will be smooth, but it will most likely not be the path that another character would take. There is a risk for snapping behaviour when using

the key_moment constraint. Although the B-spline guarantees a smooth path, the velocity could still be too high for it to be realistic. The interpolated distance constraint can control this speed, but the risk of this method is that the bone will always remain on the edge of region, which could lead to unnatural motion.

Motion performance

The next main question is: How does it perform with different motions? To answer this question we analyse two types of motions. One without global translation and one with global translation. In motions with global translation the speed of the character can be perceived as slow. While this problem does not occur for a motion without global translation. While maybe in a motion with global translation small changes could be less obvious. In the case of the different motions, the walking motion will be better solvable than the reaching motion. This is because the walking motion is only constrained at the feet while the reaching motion is constrained at the feet as well as the hands. When only the feet are constrained, it is most likely only the hips, legs and feet are adjusted and the upper body remains the same.

Scaling limits

It is also not clear what the limits of the framework are. To what extent can a character be scaled and retargeted properly? Can anything be said about uniform scaling in relation to non-uniform scaling? This is going to be tested with the constraint to keep the joints at the same position during the whole motion. Keeping the joints on the same position could result in very different poses with great scaling. With smaller difference from the original motion the joints in the limbs can correct for the difference in position, but with larger scalings the torso has to be adjusted to correct for the difference between the original and the scaled joint position. This could lead to a different perception of the motion. The expectation is that as well the solvability as the naturalism of the motion will reduce as the difference in scaling increases. This can be explained by looking at the optimization function. A larger difference in scaling will result in larger differences in joint angles. Those joint angles result in larger changes in the parameter vector. The optimization function measures the quality of the motion by measuring the amount of change in the parameter vector. So the quality decreases when the difference in scaling increases. If larger changes have to be made to the joint angles in the parameter vector the gradient to the parameter vector is a bad prediction of the amount of change needed to reach the destination. Resulting in more iterations before reaching the destination.

7.1.2 Variables

In this section the variables of this experiment are discussed. First the controlled variables are discussed, which will remain constant during the experiment. The next section is a discussion of the input variables and finally the output variables.

Controlled variables

This section describes the controlled variables. The control parameter configuration, the threshold used for the solving and the sets for the input parameters are described here.

Control parameters configuration The control parameter configuration is the definition of the variables in the control parameter vector. This configuration has a big influence on the result of retargeting. Having too many parameters leads to a problem that is harder to solve. Too few parameters will lead to a bad quality or insolvablity. On the other hand the definition of the control parameter can also control the naturalism of the motion. A parameter that cannot have a value that exceed joint limits will prevent a motion from ever being in an impossible pose. The

parameter configuration that is used for the experiments defines the skeleton in Euler joint angles. All joints have three rotational degrees of freedom. The root is defined in three positional degrees of freedom. The joints that are used in the control parameter vector are: vl5, vl1, vt1, r_shoulder, r_elbow, l_shoulder, l_elbow, r_hip, r_knee, l_hip and l_knee.

Thresholds By using a threshold, the experiment is guaranteed to finish. The running time is limited to 24 hours and the constraint values all have to be between -1.5 and 1.5 cm. For distance constraints this means Euclidean distance and for position constraints this means distance per axis.

Motion set The motions we are going to use are a walking motion and a reaching motion.

Constraint set The constraints that we are going to use are: bone always on original position (bone_original_position), bone on original position at key moment (key_moment), bone distance smaller when closer to key moment (key_moment_interpolation). The first kind of set is simple, for a set of bones (depending on the motion) the bones have to be on the same position as they are with the original scaling. The second kind of constraint set keeps a bone at its original position at a key moment, key moments are footsteps and grasp moments. The last kind of constraint set linearly interpolates the distance over time towards the key moment. So additional constraints are added to the key_moment constraint making sure the bone is within a certain distance from the target bone. The distance for the key_moment_interpolation is 50 cm in 2 seconds.

Character scaling set We are going to use two kinds of scaling. The first is uniform scaling, which only influences the environment. The other is incremental scaling, where an additional scaling factor is applied on every level of the skeletal hierarchy. The incremental scaling is used to check for varying inter-body relationships. For the uniform scaling we are going to use the following scalings (0.5, 0.8, 1.1, 1.4, 1.8). For the non-uniform scaling the following scalings are used (0.7, 0.9, 1.1, 1.3).

Input variables

This section lists the input variables.

Constraints One of the constraints from the constraint set is used.

Motion One motion from the motion set is used.

Character scaling One form of scaling, either uniform or non-uniform is used.

Output variables

This section describes the output variables.

Video The tests ran for this experiment will result in a motion. To analyse this motion, the motion is rendered on a character and a video is made from this virtual character performing the motion.

7.2 Experiment setup

This section discusses how the experiments are executed. The experiments will be executed using our own implementation. This implementation is made in RAGE (Realtime Animation and Game Engine). This engine is C++ based and uses Ogre3d for rendering. Python is used for high level code. Most of the spacetime solver is written in C++. All data structures and functions are defined in C++. Python is only used as a datafile for the definition of constraints and scalings and to play the animation once it has been solved. After the definition of the constraints the only communication between python and C++ is to start the solving. NLopt [5] is used for the constrained non-linear optimization. This is running on an Intel core i7-3610QM 2.3 GHz CPU with 8 GB ram and a NVIDIA GeForce GT 650M on Windows 7. In table 7.1 the configurations are shown for the executed experiments.

A set of configurational parameters is used to configure the solver, in this paragraph those are discussed. Choices for those parameters are made because earlier test runs showed they performed best. NLopt is configured to use the LD_AUGLAG (Local Gradient-Based Augmented Lagrangian) algorithm. The bounds for the values are set from -360 to 360, this is to limit the rotations. In practice this also works for the translation of the root but this is in meters and none of the tested case take such a large area. The rotation limits could be tweaked more and be using the same values as the constraint functions for joint limits would. One NLopt iteration is limited to one hour which gave good results during test runs. Other stopping criteria are if the constraints have gone below a value of 0.015, since this gave good results while still able to solve during test runs. Since all constraints are position constraint and those are expressed in meters is means 1.5 cm. Furthermore the xtol_rel and ftol_rel are set to and arbitrary value of 1e-4. If either the optimization function(ftol_rel) or the parameter vector(xtol_rel) do not improve more than the set value the algorithm is allowed to stop, the exact meaning of this depends on the algorithm. The algorithm is allowed 24 NLopt iterations to allow it for a long time to solve while still begin able to run all experiments within a reasonable timespan. The interval between B-spline control points is 0.125 seconds as suggested in the paper by Gleicher. The interval between constraints is 0.13, so only one constraint of the same type exists per control point and rounding errors of there being more constraints per control point are prevented.

The clips recorded of the experiments are analysed. A general inspection of the motion is made, with the focus on the plausibility of the motion, rated by the author. The motion is rated as if the retargeted character has been asked to do the same as the original character did. Moreover the balance, penetration, body part path, acceleration and reaching the goal are inspected visually. The visual inspection is done because some properties are hard to express in discrete values and because of time constraints on the project, time invested in checking those point could also be invested in preventing them from occurring. Balance inspection is done visually, this is expressed in four different values: yes, plausible, unlikely and no. Yes means that there is not a doubt about whether it is in balance, plausible means it could be in balance, unlikely means it is not sure that it is not in balance but it does not seem like a natural balance pose and no is an obvious imbalance. Furthermore the moments of flight, when the character is not touching the ground, and the penetration with the ground and self penetration by the character are considered for rating the quality of the motion. The quality rating is depended on the depth of penetration or the height of the flight. The body part path test checks for implausible paths of body parts. For the acceleration test the author rates if the motion has implausible accelerations. The motion is also tested visually if the goal is reached, this means the hand reaches the right position in animation reach_low. For every tested motion it is mentioned whether it is solved within the 24 hours limit and if it did not solve what the remaining distance was. This distance should be zero for the motion to be solved. This distance is the sum of absolute differences from the threshold of the constraints.

id	scaling	uniform scaling	motion	constraints
01	0.5	true	reach_low	bone_original_position
02	0.8	true	reach_low	bone_original_position
03	1.1	true	reach_low	bone_original_position
04	1.4	true	reach_low	bone_original_position
05	1.8	true	reach_low	bone_original_position
06	0.7	false	reach_low	bone_original_position
07	0.9	false	reach_low	bone_original_position
08	1.1	false	reach_low	bone_original_position
09	1.3	false	reach_low	bone_original_position
10	1.4	true	part1	bone_original_position
11	1.4	true	reach_low	key_moment
12	1.4	true	reach_low	key_moment_interpolation

Table 7.1: Table showing the experiment configurations

id	difference	hours	balance	flight	mesh penetration	path	acceleration	goal
01	0.452511	24	plausible	yes	no	detour	head	no
02	0	1	plausible	yes	no	natural	natural	yes
03	0	1	yes	no	ground	natural	natural	yes
04	0.0121173	24	unlikely	no	hand->body/leg, ground	hips	hips	yes
05	0	21	unlikely	no	hand->body/leg, ground	hips	hips	yes
06	14.5786	24	plausible	yes	no	natural	natural	no
07	0.876182	24	no	no	no	natural	natural	no
08	0	1	no	no	hand->leg, ground	natural	natural	yes
09	0	1	unlikely	no	ground, hands->legs	elbow	elbow	yes
10	0	1	yes	no	no	natural	natural	yes
11	0	1	plausible	no	no	hand	touch	yes
12	6.63005	24	plausible	yes	ground	natural	natural	no

Table 7.2: Table with annotations while inspecting the experiment results, explanation of the values is done in the introduction of this chapter

7.3 Results

This section discusses the results of the experiments. First an overall analysis of the experiments is made. The section after that handles all individual motions and points out specifics per motion. Finally the research questions are answered with the results from previous sections.

7.3.1 Overall analysis

In Table 7.2 remarks are shown that are made during the inspection of the video clips. The motions either are in flight or have ground penetration. Very small penetration or flight, as even can be seen in the original motion is not annotated as such. Mesh penetration start occurring with up-scalings. The ground penetration even occurs when the retargeting is solved, this indicates that the configuration is not right. The configuration allows for 1.5 cm of difference from the original position while it is more than 1.5 cm. An example of ground penetration can be seen in Fig. 7.1. The body-part path is natural in most cases, except for the large scalings and the key_moment constraint. If a chain of joint angles that has to be adjusted passes a single limb the body path gets

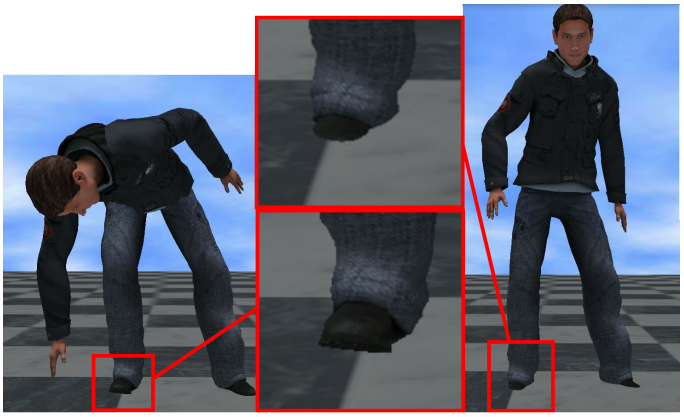

Figure 7.1: Two shots from experiment 03, (left) without ground penetration (right) with ground penetration

Figure 7.2: Unnatural hip position in experiment 04

Figure 7.3: The poses created in experiment 01 do no longer look like poses in the original motion (right)

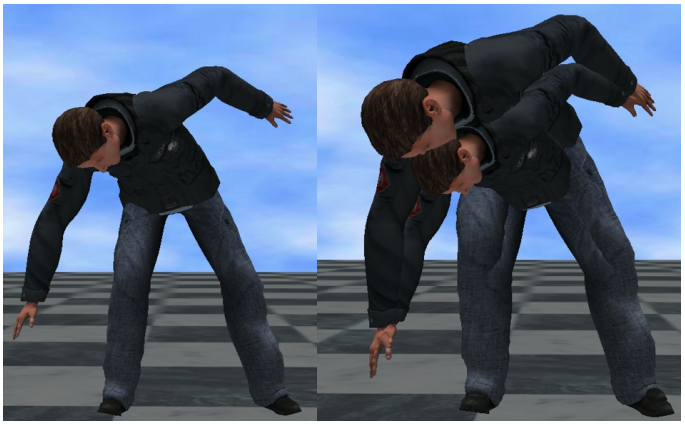

Figure 7.4: Good looking pose in experiment 02, (left) retargeted character (right) retargeted character with original, wrist and ankle positions are the same

unnatural. This is the case with the unnatural hips that are uniformly scaled with scalings: 1.4 and 1.8, as can be seen in Fig. 7.2. The fully stretched arm is not able to reach the goal position if the shoulder remains in the same position. With smaller scalings this has the effect of the character not being able to reach the goal but the body parts no not start to take odd paths. With the key_moment constraint there is a clear deviation from the path when the constraint gets active. The unnatural paths generally also have unnatural accelerations. The visual reaching of the goal is correlated with the solving, only for experiment 04 the goal seems reached while retargeting is not solved, but the remaining distance of 0.0121173 explains the visual misinterpretation. In almost all cases retargeting solves in one iteration or it won't solve at all. The only exception here is experiment 5 with 21 iterations. It is exceptional experiment 4 did not solve at all but experiment 5 did while it has a larger scaling. It is hard to explain this difference, it could have to do with another process taking resources while experiment 4 was running. Another point is the balance between the optimization function and the constraint function in NLopt is not well understood, maybe with a larger initial difference on the optimization function the optimization function is evaluated less. For the other experiments it seems like the solving time increases with the initial difference, but no statement can be made about this exact relation since it is either solved in an hour or not solved at all.

Figure 7.5: Good looking pose in experiment 03, (left) retargeted character (right) retargeted character with original, wrist and ankle positions are the same

Figure 7.6: Although the arms in experiment 07 are wide the pose is not as bad as the similar pose in experiment 01 (see Fig. 7.3)

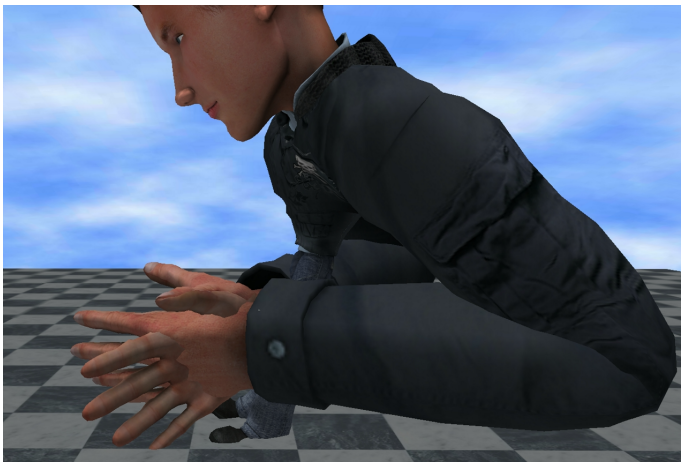

Figure 7.7: Imbalance in experiment 09

Figure 7.8: Strange elbow position and mesh penetration

Figure 7.9: Good looking result of a walking motion

Figure 7.10: The difference between the position of the original wrist and the scaled wrist, before the constraint becomes active, this distance is covered in 0.25 seconds

7.3.2 Per motion analysis

The properties that are handled now do not totally reflect the quality of the resulting motions. Therefore the motions are now handled on quality. Motion 01 does not differs very much from the original motion, it seems like a very expressive version of the motion (see Fig 7.3), furthermore the upper torso contains rotation which does not exist in the original motion. This is due to the cost of the rotation of the torso being less than adjusting the arms further. Motion 02 and 03 are very similar to the original motion as can be seen in Fig 7.4 and Fig 7.5, but in both motions the feet move when the upper body is moving. The oscillations created by the moving feet seems worse for motion 02. The similarity of the motion can be explained by the initial difference being smaller. The oscillation in the feet is the result of the translation of cost of translation the root being lower than adjusting joint angles. In motion 04 and 05 the hips move differently from the original motion. The hips move to the back and the front and to the left side as earlier was shown in Fig 7.2. Although there are changes between back and front it always tends to the right side of the character. The right hand is also the picking hand. This effect is worse with motion 05 where the scaling is larger. This effect is caused by the body having to correct for the distance between the shoulders and the feet. Although the scaling on motion 06 causes the character to look very differently from the original character, the motion itself remains more the same than with the small uniform scaling, but the difference from the solution is greater. This could have to do with the optimization function having a larger tribute to the final result than the constraints, since the constraints are not solved. Motion 07 looks like a plausible motion although it has the same problem as motion 01 where the arms are not really in a resting position but very wide, as can be seen in Fig 7.6. But in opposite to the motion 01 it has no strange oscillations. Although it does not reach the goal position the attempt to reach it looks good. It could be that strange oscillations start occurring in a later stage of the solving. That is why motion 01 is close to the goal and why motion 06 also has a better looking result. In motion 08 the oscillation when reaching looks small, but it is obviously off balance which makes it a very unbelievable motion. The non-uniform scaling changes the influence of a difference in rotation, causing it be off balance earlier. The character in motion 09 has very unbelievable proportions in this stage the constraints of keeping the hands and feet on the same position do not work any more. The constraints are too close to the body cause all kinds of mesh penetration. The character also leans backward while performing the motion just like motion 09 which makes it unlikely that it is balanced as show in Fig 7.7, but because of the large scale of the character the balance is no longer the main disturbing feature. The elbow also takes odd positions and there is a lot of mesh penetration occurring (see Fig 7.8). The walking motion 10 looks the best of all motions, this is because it has constraints only on the feet, a resulting pose is shown in Fig 7.9. It is noticeable that the steps taken by the character are too small but no strange artefacts occur. In motion 11 a discontinuity is noticeable when the constraint becomes active, but the rest of the motion is better than the others. The difference between the hand positions in the original character and scaled character is shown in Fig 7.10. No oscillation of the feet is noticeable. Motion 12 did not completely solve. Here the feet move while reaching.

7.3.3 Answering the research questions

With those results it is possible to answer the research questions. These questions are answered in the same way they are formulated. First we discuss the constraint definition, followed by the motion performance and finally the scaling limits.

Constraint definition

When comparing the different constraints (bone_original_position, key_moment, key_moment_interpolated) it seems the key_moment constraint works better, although the moment of touch is clearly visible the overall pose of the body is better. The key_moment_interpolated constraint was expect to perform better, because it was less constraint than bone_original_position and had to make

smaller steps per time step than key_moment, but nevertheless it was unable to solve while the other two were. This could indicate an error in the definition of this constraint, for as well the bone_original_position constraints as the key_moment constraints the constraint distance is always zero while with the key_moment_interpolated the distance is larger depending on time.

Motion performance

If we compare the walking motion with 1.4 uniform scaling to the reaching motion we can definitely say that the walking motion is easier to solve. As expected this has to do with only the foot constraint being applied, instead of as well the feet as the wrists. Only solving foot constraints is much easier than solving as well the hand constraints as the foot constraints. For the walking motion this is actually just a footstep solver and the scaling only influences the difference for the footsteps, while for the reaching motion the whole body has to be adjusted.

Scaling limits

Scaling seems to influence the solvability of the motion. Although scaling to a larger character seems to be easier to solve than solving to a smaller character. In the case of incremental scaling this effect shows best. This can be explained by the reachable space by the limbs. It is easier to solve a constraint with joint angles that only effect that constraint. If joint angles that influence multiple constraint have to be adjusted the balance has to be found between the constraints.

8 Conclusion

This chapter draws a conclusion from the results in section 8.1. It also discusses what is possible as future work in section 8.2.

8.1 Conclusion

In this research we have seen a method for retargeting and all aspects needed to implement it. The experiments show that using trivial constraints is not enough to do retargeting, being able to solve constraints does not solve the retargeting problem, defining the right constraints also requires attention. This report shows more detail on some components of retargeting using space time optimization. Using a set of predefined constraint type is easy to use for an animator, but because of the nature non-linear constraint optimization it is hard for an animator to fix errors if a motion is no longer able to solve. Although the focus has not really been on performance we have not been able to achieve results anywhere close to interactivity. This could be improved by a solver that can solve in fixed time steps, this is also needed for practical use since the memory usage grows quadratically with the length of the motion. In the experiments we have seen that for the smaller scalings 1.1 and 0.8 good results are obtained. These kind of scalings are reasonable in a usual character editor in games where the height of the character can be adjusted. Nevertheless if the wrists and feet were corrected with IK-chains on the arms and legs similar results would be achieved. Adjusting the motion with a single equation causes it to be globally optimized, but this also causes errors to be global as can be seen with the moving feet in the reaching phase.

We have chosen to work on this method because of its generality, and because everything can be solved if a constrained definition can be made. This generality also makes it more complex. There are many possible implementations and they do not all give the same result. The method is also low level, making it hard to implement functionalities like, take an additional step. Because the algorithm is low level maybe this generality is not needed, maybe at a low level a solver for a discrete set of options suffices.

8.2 Future work

This research has risen many new questions. In this section we will provide suggestions on improving the method as suggested by Gleicher [2] and suggestions are made on how retargeting can be used in real time or even online.

What is the best kind of interpolation to use? Currently the B-spline is used but as shown in section 6.5 this is not the most trivial choice. The B-spline is computationally heavy and causes the control vector to have values that have less meaning. Using a method of interpolation that passes through the control points would have understandable control point values. This could be important for understandability of the algorithm and providing feedback to the user.

Solving now happens for the entire motion at once, it would be better to solve for a fixed timespan, what is the best way to do this? To make this method real-time or maybe even online.

It is needed that solving can be done for fixed time spans. For this the impact of constraint should be researched. A start with this is already made in section 6.4, where we showed the influence of a control point on a B-spline is limited.

Solving for a fixed time span is just one of the steps that are needed for online retargeting. It is also hard to predict the upcoming constraints. With hindsight the method by Ho et al.[4] seems promising for this. Creating an interaction mesh for all object close to the character would provide for a generic constraint, it would still be unknown what the desire distance between those object is.

What constraints have to be defined? We have used simple point constraint on the end effectors, but better constraint definitions can probably be made. Another question is when do the constraints have to be applied? It would be useful to have an automated method to define the constraints. Although manually defining constraints is easier than key framing its still a labour intensive job. Moreover constraints cannot always be defined beforehand.

Research could also focus on defining different levels of retargeting. There are low level controllers to do the inverse kinematics part, but there should also be higher level controllers to decide if the character should take another step or take an additional action. Spacetime constraints as discussed in this thesis are best suitable for doing the low level part. They can solve many positional constraints and limits on joint angles etc., but defining constraints in such a way an additional step is taken to solve the constraint is very hard.

How do weights have to be set? In section 4.2 the different kinds of weights have been discussed but this does only offer a structure but no values. It is not trivial to define the weight balance between different angles. It is even harder to define a balance between distance and angles, because it depends on the constraint and other angles of other joints, so research should be done on what is the best way to incorporate the constraints in the weights for the parameter vector.

What is the best type of control vector to use? Currently the control vector consists of a limited set of joints which are configured with joint angles as stated in 3.3 but this is just one of many types. Other types of angle definitions can be used or other control vectors entirely. As seen in the related work some use a control vector based on particles with distance constraints. This splits the control vector in more individual components but requires more constraints solving. Maybe other control mechanisms exist that provide more independent controls.

Bibliography

[1] R. Fletcher. *Practical methods of optimization; (2nd ed.)*. Wiley-Interscience, New York, NY, USA, 1987.

[2] Michael Gleicher. Retargetting motion to new characters. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 33–42, New York, NY, USA, 1998. ACM.

[3] Chris Hecker, Bernd Raabe, Ryan W. Enslow, John DeWeese, Jordan Maynard, and Kees van Prooijen. Real-time motion retargeting to highly varied user-created morphologies. In *ACM SIGGRAPH 2008 papers*, SIGGRAPH '08, pages 27:1–27:11, New York, NY, USA, 2008. ACM.

[4] Edmond S. L. Ho, Taku Komura, and Chiew-Lan Tai. Spatial relationship preserving character motion adaptation. *ACM Trans. Graph.*, 29(4):33:1–33:8, July 2010.

[5] Steven G. Johnson. The nlopt nonlinear-optimization package. `http://ab-initio.mit.edu/nlopt`.

[6] Richard Kulpa, Franck Multon, and Bruno Arnaldi. Morphology-independent representation of motions for interactive human-like animation. In European Association for Computer Graphics, editor, *Eurographics*, Dublin,/Ireland, August 2005. M. Alexa, J. Marks. http://www.eg.org/.

[7] B. J. H. van Basten and A. Egges. Evaluating distance metrics for animation blending. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, FDG '09, pages 199–206, New York, NY, USA, 2009. ACM.

[8] Jing Wang and Bobby Bodenheimer. An evaluation of a cost metric for selecting transitions between motion segments. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '03, pages 232–238, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[9] Andrew Witkin and Michael Kass. Spacetime constraints. *SIGGRAPH Comput. Graph.*, 22(4):159–168, June 1988.