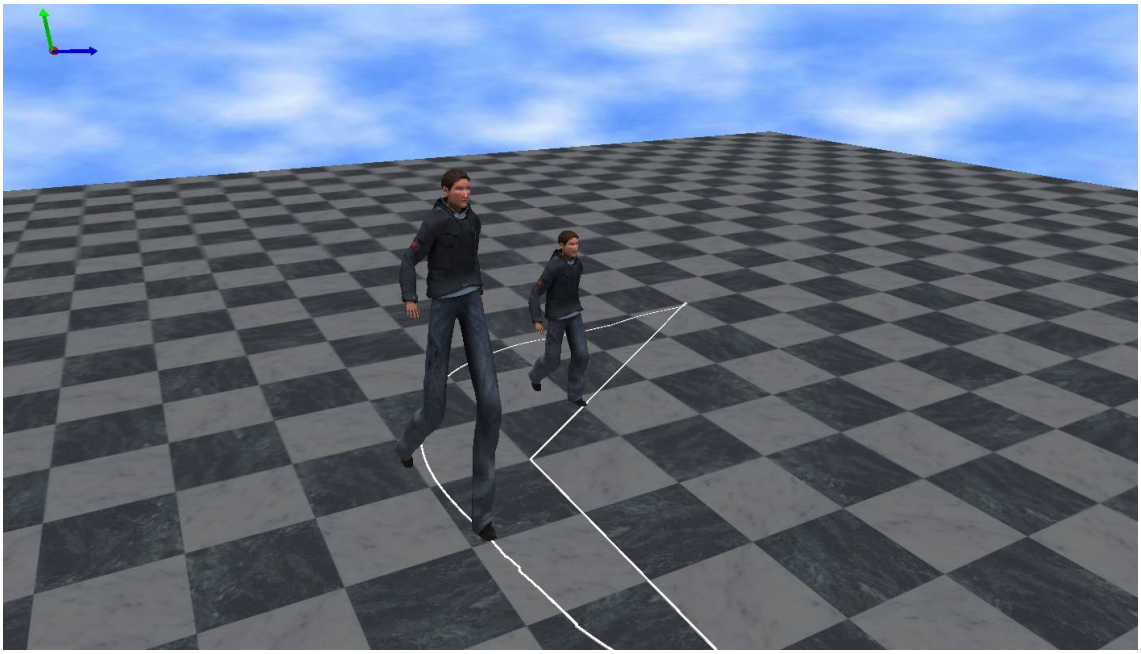

Synthesizing walking motions independent of limb length

(Working title was: Synthesizing walking motions with dynamic limb length adaptation)

Written by: Dennis Pullens

Supervised by: Dr. Ir. Arjan Egges

Thesis number: ICA-3692469

August 2013

Acknowledgments

We would like to thank everybody in the animation research group for their help, suggestions and feedback throughout the course of this project.

Synthesizing walking motions independent of limb length

Dennis Pullens (Stnr. 3692469)

Department of Game and Media Technology
University of Utrecht
Utrecht, The Netherlands
2013

ABSTRACT

Motion capture greatly improved the quality of animations, but the body proportions of the actor are not always equal to those of the virtual character. This paper presents a method to change the leg length of a character during an animation while preserving the motion capture data and without leaving visual artifacts. This results in a system capable of following a given path on a 2D plane in a 3D environment using a motion graph as input, while the legs of the character change. In order to plan the path with the change in leg length, a formula is introduced that estimates the preferred walking speed of a person solely depending on leg length. This formula is based on research from the field of biomechanics. When the resulting speed is applied to the character foot skating is introduced. This is removed using timewarp and a method called Morphology Independent Representation for constraint solving. The overall motion is smoothened using linear interpolation. The system is able to make a character smoothly follow a given path while the length of the legs are changed.

Table of Contents

Chapter 1

Introduction

The quality of animation used in applications, such as games and simulations, has become increasingly important over the years. The introduction of motion capture has made it possible to record animation played by actors, which produces realistic and high quality motions. One of the great drawbacks of this method is that an actor is required who can perform the task at hand. In games that are a real-life simulation, these actors are relatively easy to find. For example, the game studio responsible for developing the FIFA Soccer game hires actual football players and record moves and tricks so that they look realistic and professional (see figure 1.1), but there is a limit to what actors are capable of.

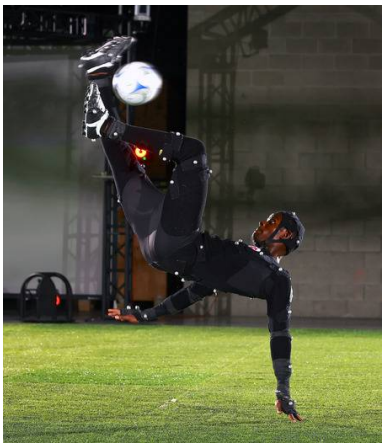

Fig. 1.1: Toronto's Maurice Edu during a motion capturing session for FIFA Soccer '09
Image source: http://lis3353.wikispaces.com/Motion+Capturing

In many games and simulations, characters are used with extreme skeleton structures (e.g. giants and dwarves) or even characters with dynamic structures (e.g. The Hulk, see figure 1.2). It is impossible to use actors for the animation of such characters. Our goal is to build a system that can animate characters independent of leg length, or even leg length that changes over time, without losing the quality of motion capture data.

Fig. 1.2: An image of the different stages of the Hulk-transformation
Image source: Deviant art page of RossHughes
(http://rosshughes.deviantart.com/art/Hulk-Transformation-260719300)

The main contribution of this research is a system that is able to modify limb length during animation created using a motion synthesizer. Our technique allows for limb length modification, while ensuring a natural motion without foot skating. Following a path using walking motions is a very common task in games and simulations. Usually some path planning algorithm (e.g. A* algorithm) creates a path that avoids obstacles. This system uses that path as input and creates a stream of motions that follow this path as accurate as possible with natural looking motions.

In chapter 2 we will describe related work. Other research in the fields of computer animation and biomechanics are referenced to get useful information about techniques that solve parts of our problem or compliments part of the solution. Chapter 3 explains our introduced method. Chapter 4 explains implementation details of our system. Chapter 5 shows the results of our method and implementation. In chapter 6 we will draw conclusions based on our results and discuss future work and improvements.

Chapter 2

Related Work

This chapter describes research that has been done in similar areas or research that compliments ours. Different aspects of animation are explained together with information about human walking. In section 2.1 basic information about animation is given followed by section 2.2 that gives an overview of different ways to represent motion. In section 2.3 several methods to create streams of motion are explored. Section 2.5 covers ways to manipulate motion. The field of biomechanics supplies information about different parameters that form a walking cycle, which is discussed in section 2.6.

2.1 Animation

Animation can be described as the rapid display of a sequence of static images to create an illusion of movement. It adds the dimension of time to an image. In computer animation, this definition is not far off. A character motion m is a function that at a specific time t returns a specific pose q. A variety of techniques exist to create these kind of functions. Procedural animation requires initial conditions and rules, such as forces that apply on a character or constraints, to automatically create a function that meets these rules. Keyframing requires an artist to define certain poses at certain times. During runtime, interpolation is used between these keyframes to create a continuous motion. Another technique uses artificial intelligence to create animation. The AI-algorithm requires a goal (e.g. walk from the current location to another location) and a model. Through many iterations the AI tries to accomplish the goal using the given model. If successful, the final result is a walking animation which can be stored. Motion capture is a technique where an actor or object is recorded in real life in order to animate a virtual model. Modern techniques use body markers and multiple cameras from different angles in order to determine the 3D location of each marker. The body pose of each frame is then stored.

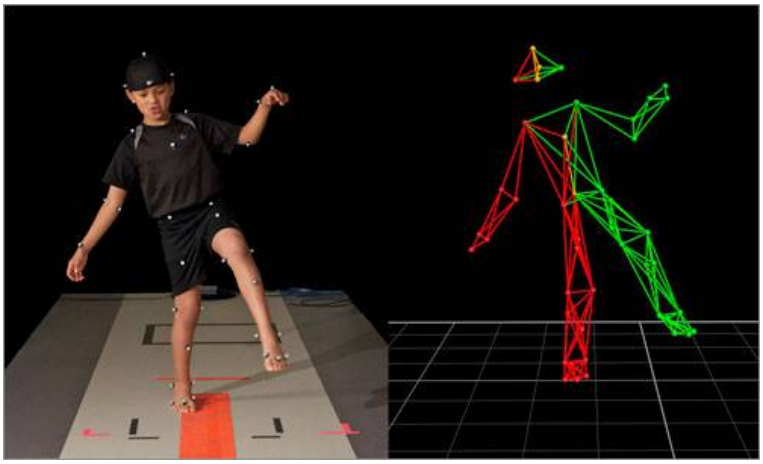

Fig. 2.1: On the left motion capture data is being recorded. On the right is the virtual playback of the motion capture data at the same frame.
Image source: http://wyss.harvard.edu/viewpage/267/ (Wyss institute)

2.2 Motion representation

There are several methods to store the skeletal representation of a single body pose. Some basic representations are point clouds and joint angle representation. A point cloud is a collection of different locations that could represent many different things (e.g. marker positions, bone positions). When a pose is represented by a point cloud, each position represents the position of a bone in the skeleton. Each pose q is represented by a single vector with locations and can be described as $q = (t_1, \ldots, t_n)$, where $t = (t_x, t_y, t_z)$ and n are the number of bones. A major drawback of such a representation is the scalability and the fitting of the skeleton. Each bone in this representation has a predefined length. When such a bone is scaled, all the bone positions affected by this scaling have to be recalculated, for each frame. This is computationally inefficient. Also, point clouds do not contain joint data. Each frame the pose needs to be mapped to the skeleton, which requires computation time for inverse kinematics solving.

The joint angle representation is a vector of joint angles and can be described as $q = (t_x, t_y, t_z, \theta_1, \ldots, \theta_n)$, where $t = (t_x, t_y, t_z)^T$ is the global root translation, each θ is the angle of a joint. Because it only stores information about the root position and the joint angles, and not specifically defined bone locations, it does not depend on bone lengths. A drawback of joint angle representation is that it is specific for one skeleton. When proportions change the whole animation needs adaptation in order to avoid problems such as foot skating.

The paper of Kulpa et al. [RK05] introduces a method of saving skeleton information that makes it easy to adapt animations to different characters with different morphologies. In addition, they introduce a low-cost real-time constraint solver.

Given a skeleton, the information is minimized into a new format. Classically, there are several kinematic chains that represent a skeleton, starting from the root to ends such as hands, feet and the head. The motion independent representation (also referred as MIR in this report) has these kinematic chains divided into three main parts:

- The normalized segments are composed of only one body segment (the hands, the feet, the clavicle and the scapula)
- The limbs with variable length that encode upper and lower limbs. The elbows and knees are not stored, their location will be calculated later on.
- The spine represented with a spline.

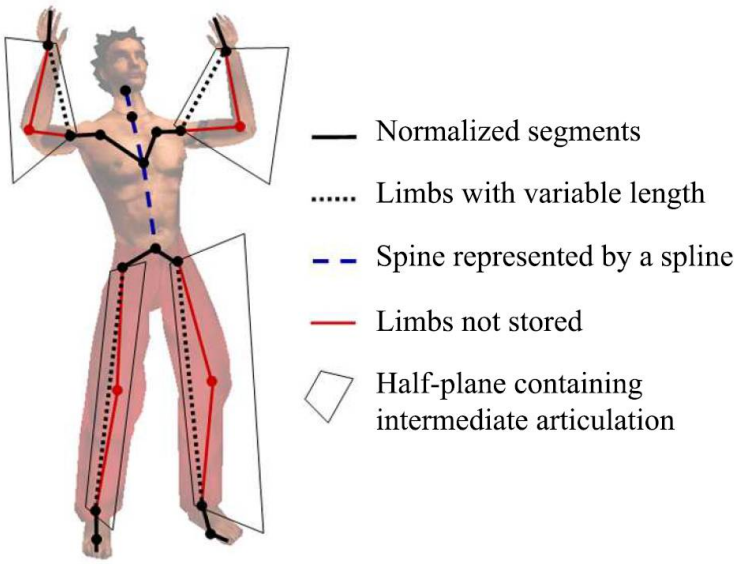

Fig. 2.2: Normalized representation of the skeleton
Image source: [RK05]

The limbs are stored using two data: a reference frame attached to either a hip or shoulder, and a scalar. Using the axes of the frame a half-plane can be created on which the articulation (elbow or knee) is located. The precise location of the articulations can be computed using the half-plane and the scalar, which represents the normalized

length of the limb. Because the representation uses a half-plane, there is a guarantee that the articulation will not ”switch” to the mirrored position of the plane.

Since the location of the articulations can be calculated using this representation, it is possible to give a location on which the wrist or ankle is constraint. This method then gives the location of the articulation. Using the location of the articulation and wrist or ankle, the skeleton can be repositioned to match these locations.

2.3 Motion concatenation

Motion is often a combination of several short clips. These clips can be created by an animator or recorded by a motion capture system. A problem with combining different clips, is that they don't always transition well. For example, when combining a part of a walking motion and a swimming motion, the animations are too different to create a smooth and natural transition. Much manual labor has to be done to pick motions that connect well to each other or transitions need to be created so that these two clips transition smoothly. To decrease the amount of labor, much research has been done to automatically pick these motions and create transitions.

Graph structures are widely used and much work has been done to improve them. Mizuguchi et al. [MM01] introduced ‘move trees‘ that became widely used. To create this structure, an artist has to hand pick animations and create transitions between them using tools. When one animation has transitions to multiple other animations, you get a graph. Since it required manual labor and requires skill from an artist, a good connectivity is not always guaranteed. Implementations often only reach relatively simple goals. An improved and automated model was created by Kovar et al. [LK02].

A motion graph is a graph structure consisting of nodes and edges. Each edge represents a segment of animation and each node represents a time in the animation where either the animation ended or can transition to another animation segment.

In order to locate the transition points, each frame of each animation is compared to one another. This is done by creating point clouds (see figure 2.3) in two user-defined window of frames; one bordered at the beginning of the first frame and the other by the end of the second frame. The distance is calculated by computing the total weighted sum of squared distances between two corresponding joints in these point clouds. Comparing each frame of one animation with each frame of the other forms a sampled 2D function

(see figure 2.4). The local minima are calculated and if they are below a given threshold a transition is created and a node added to the graph. This transition is created by first aligning the two motions by searching for the smallest possible error in the point clouds. Then the root is linearly interpolated and spherical linear interpolation is applied to the joints. During this interpolation blend weights are used to maintain continuity. At the inserted node the animation can either continue with the current animation, or switch to the other animation at the given time (see figure 2.5).

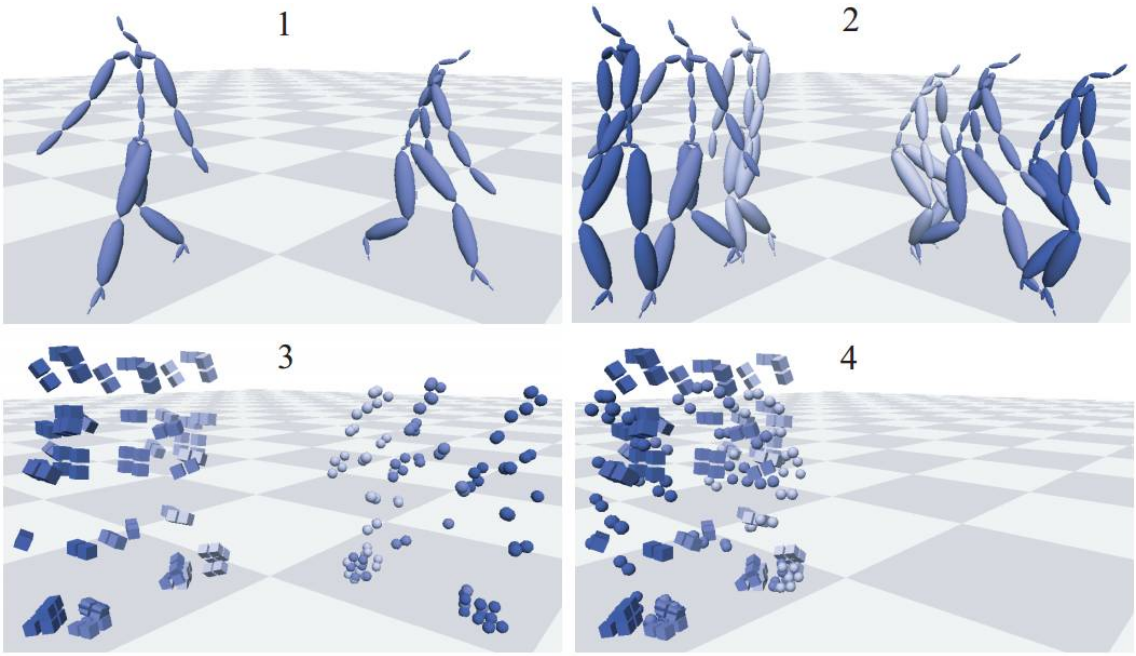

Fig. 2.3: To compute the distance between two frames of motion (1), local neighborhoods of frames are extracted (2) and converted into point clouds (3). The squared distance between corresponding points is then calculated through an optimization that factors out rotation about the vertical axis and translation in the floor plane (4).
Image source: http://pages.cs.wisc.edu/ kovar/thesis/thesis_mographs.pdf

As for the threshold; The lower this threshold, the smoother the transition will look, but the graph will most likely be better connected with a higher threshold.

After the transitions are determined, the graph is pruned. This process removes dead ends and sinks. Dead ends are caused by the ends of animations that are not connected to any other segments. Sinks are nodes that only able to reach a small part of the graph. The resulting graph can generate long streams of motion from any given starting node.

A similar type of graph has been introduced by Arikan et al. [OA02]. Although the notation of the graph is slightly different, the ideas are the same. The two differ in

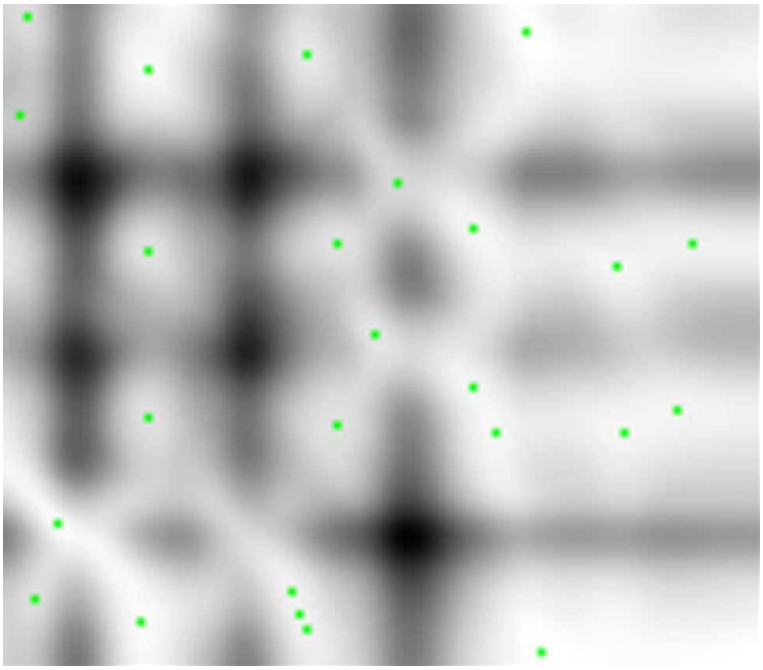

Fig. 2.4: An example error function for two motions. The entry at $(i; j)$ contains the error for making a transition from the i^{th} frame of the first motion to the j^{th} frame of the second. White values correspond to lower errors and black values to higher errors. The colored dots represent local minima.
Image source: [LK02]

goals and search methods. Arikan et al. focus on user restrains, especially space-time constrains, and use randomized search algorithms to extract motion. Kovar et al. focus on good connectivity for continuous streams of motion and less on meeting user requirements. Since both structures are very similar, both can achieve the same goal.

Heck et al. [RH07] create a graph based structure where transitions are made between parametric motion spaces instead of motion segments. A parametric motion space, which is represented by a node, is a collection of example motions which are being blend together depending on the input parameter. For example, if a boxing motion is required that hit a target at a specified location, the multiple examples are blended is such a manner that this goal is achieved. To be able to create a transition between motion spaces is a challenge, since theoretically you have an infinite number of possible body poses as end frame of the first and start frame of the second motion. Since everything in the graph is pre-calculated, online solutions are unavailable as well. Instead, each subspace is represented by an axis-aligned box that in their turn represent transition regions. Using these boxes, edges can be found and a graph can be constructed.

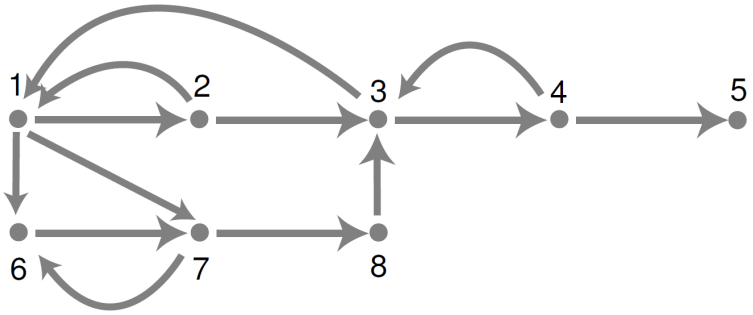

Fig. 2.5: A small motion graph. Notice that node 4 is a sink (it can only go back to node 3 or node 5) and node 5 is a dead end (there is no animation segment that node 5 can transition to)

Image source: [LK02]

We will use basic motion graphs in our implementation for two reasons. The first reason is that motion graphs are widely known, discussed and used since its publication. This makes it more likely that our method can be used in existing frameworks or compliment work of others. Secondly, motion graphs are very basic, allowing many methods and extensions to work with it. Also, methods such as parametric motion graphs have some features that are not suitable for our method. In the case of parametric motion graphs, one of the key features are the different parametric motion spaces. Since we focus on walking motions, the great advantage of this method is not used. There is also a conflict with the pre-calculated parametric boundaries and the change in morphology.

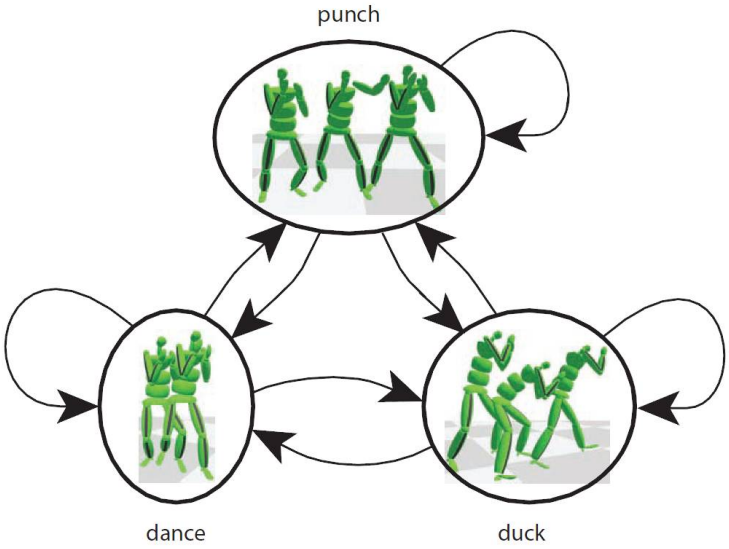

Fig. 2.6: An example of a parametric motion graph of boxing motions. Each node represent a specific motion with a wide range of possible parameters.
Image source: [RH07]

2.4 Path synthesis

Using a motion graph a stream of motions can be generated with natural and smooth transitions in between the different clips. This, however, does not yet give meaning to the stream of motions. Kovar et al. [LK02] present a framework where graph walks are extracted that conform to a user's specifications. The user supplies a scalar function $g(w, e)$ that evaluates the increase of error when an edge e is added to the existing path w. Additionally the user may specify some halting conditions that indicate when no additional edges should be added to the graph. Once a graph walk satisfies the halting condition it is complete. The goal of the search is to find the complete walk that has the least error of all the complete walks, or an error that is below a given threshold.

As specified in the introduction of this report, our goal is to find a walk w where the character follows a given path P as close as possible, where the given path is a list of 2D points which are connected by straight lines. This requires a scalar function that bases the error depending on how well the character follows this path. This is done by projecting the root onto the floor at each frame. This eventually forms a new path P' which can be compared to P. At the i^{th} frame of the walk w we first calculate $P'(s)$, which is the arc-distance from the start of P' to the current location of the root projected onto the floor. Then $P(s)$ is calculated, which is the point on P, whose arc-distance

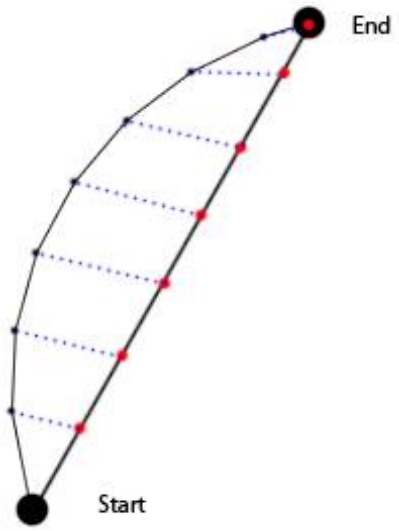

Fig. 2.7: This figure represents a walk. The thick line is the given path P and the thin curved line is the path of the projected root P'. At each frame of P', represented by the dots on the thin line, the distance is calculated between $P(s)$ and $P'(s)$. The blue dotted line represents the calculated distance at each frame. Note that when the arc-distance of P' exceeds the length of P, the last point of P is used as reference point.

from the start of P is equal to s. The error of a frame is equal to the squared distance between $P(s)$ and $P'(s)$. $g(w, e)$ is the total sum of these errors of current walk w when edge e is added:

$$g(w, e) = \sum_{i=1}^{n} \parallel P'(s(e)) - P(s(e)) \parallel^2$$

A graphic representation of this function is shown in figure 2.7.

The number of possible paths grows exponentially with the average size of a complete graph walk. There are a number of solutions to counter this problem for path synthesis:

- There is a user defined iteration length. Instead of comparing all the possible walks for the entire length of the path, they are divided into iterations. For each possible walk the error is calculated from the starting point of the previous iteration to the point where the arc-distance of P' reaches or exceeds the iteration length (see figure 2.8). The best walk of the iteration is then chosen and the next iteration

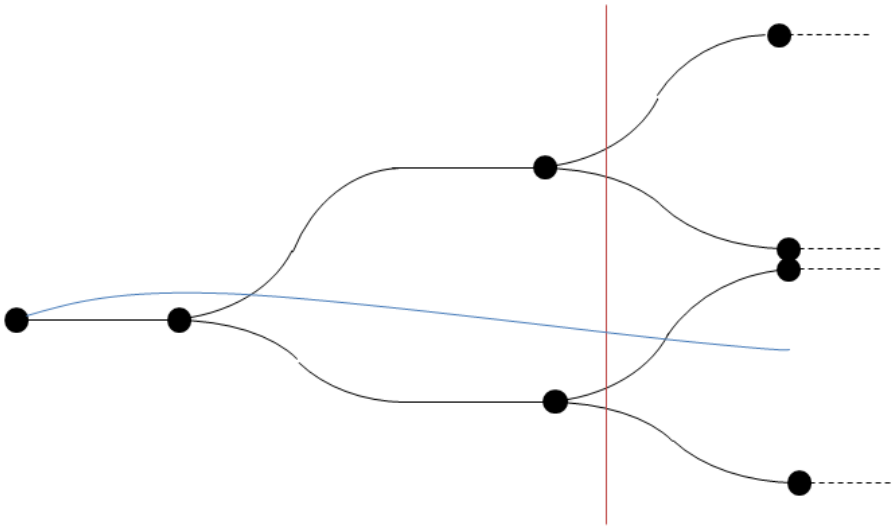

Fig. 2.8: In this image the blue line represents the path we want to follow, the black circles are the nodes and the black lines are the 2D root trajectories of the edges. The red line represents the iteration length. At this point the error of both walks are compared and the search continues at the best walk. In this case, the top branch would be ignored after the iteration.

starts. This can greatly decreases the possible number of walks, but can decrease the quality of the walk at the same time.

- There is a user defined threshold for the error function. The moment an iteration has an error function lower or equal to this given threshold, the current not-completed walk will be chosen and the next iteration starts (see figure 2.9).

- The children of a node are ordered greedily. The error of the segments that follow the current segment are calculated. Then these segments are ordered by lowest error, so that there is a bigger chance that the lowest error will be found first in that branch, making it unnecessary to search the other branches (see figure 2.10).

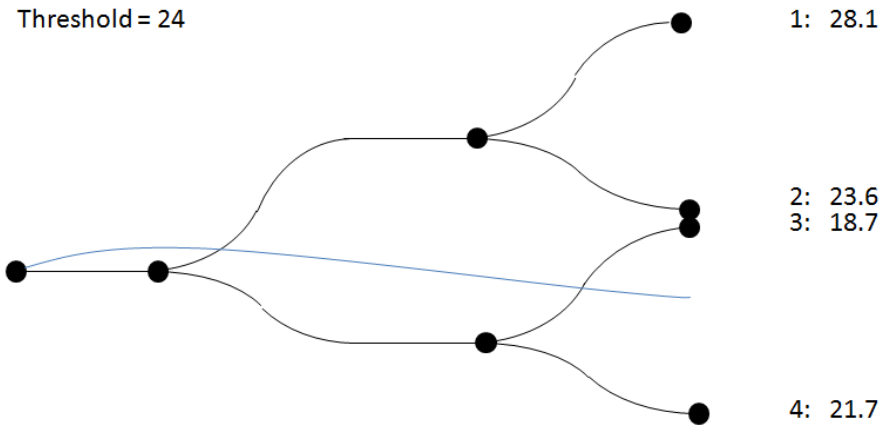

Fig. 2.9: The search continues until either a walk with an error equal or lower than the threshold is found or there are no walks left, in which case the best walk is selected. This image depicts a path search where the order is from top to bottom. The first walk does not meet the threshold and is ingored. The second walk does meet the threshold and is selected. This means that the bottom branch does not need to be searched, however, there could be a possible better walk in that bottom branch.

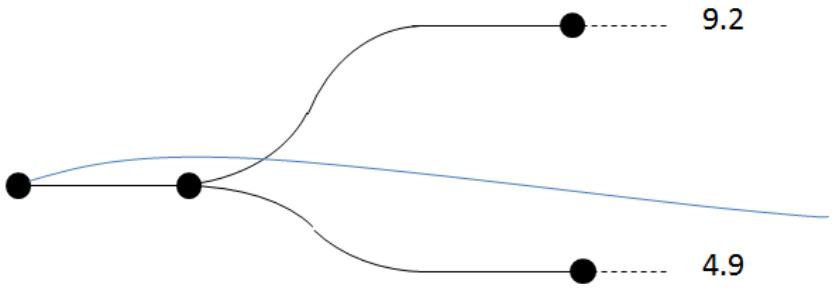

Fig. 2.10: At each node the error of the outgoing edges are calculated. The edges are ordered and searched from lowest error to largest error. This increases the chance of finding a walk that meets the threshold sooner than without greedy ordering.

2.5 Motion manipulation

There are many techniques that change the input motion in order to solve constraints. The problem our research focuses on is the reuse of motions on characters with different morphologies. An 'extreme' example is the playback of an animation on a completely different model, such as Hsieg et al. [MH05] propose. The example they show here is a human model that adjusts its pose so that it corresponds with that of a dog. Then the animation of the dog is played back on the human. We are interested in less extreme change in morphology. Glardon et al. [PG04] present a method capable of generating human walking motions with varying human length and walking speed. They use velocity and human length as input to calculate the walking frequency. Then they use their animation engine to calculate the phase update. Using this phase update the animation is time warped so that the character with the given leg length walks with the given velocity. Unfortunately they use both leg length and velocity as input, which is the relation we try to find. The use of time warp in their method is interesting, since it can alter the walking speed without breaking constraints.

Gleicher [Gle98] introduces a method to retarget motion to new characters with identical structures but different segment lengths. Their goal is to preserve details in the motion capture data while solving user defined constraints. In order to get good results, they introduce a space-time constraint solver. The great advantage of space-time solving is the ability to look forward and backwards in time to solve a constraint. This avoids snap-artifacts (an unrealistic change of bone positions in order to meet constraints) that inverse kinematic and other per-frame solvers often introduce. To solve the morphing of a character (the change in bone length during an animation), the scaling is taken into account in the initial estimation for each frame, where they try to find a global translation that fits the constraints best. This is similar to what we do in our solution as well.

The main reason we did not use retargetting was the fact that it was not real-time. Even though our path searching algorithm is offline, the entire playback including constraint solving is done online. This means that online implementations, such as real-time control over a character in a virtual environment, can be done relatively easily, while the same implementation using retargetting is impossible by default.

A whole other way of approaching walking animations are foot step driven methods. Given a set of footsteps a series of motions are chosen that match the footstep positions.

The Step Space by Van Basten et al. [BVB10] is such a method. They divide motion capture data into step segments, stored using 10 parameters. They use this representation to find the step animation that most closely resembles a given step. Because the foot step usually has a small error, inverse kinematics is used to precisely match the given footstep. Since different segments of motion have different speeds, time warping is used to smoothen the walk speed during the resulting animation. This is a technique we will use as well.

2.6 Relation between leg length and walk speed

When the leg length of a person increases, a change in walking speed is expected. We try to find a method where the preferred walking speed of a character is found, using only the length of the legs as input. The field that focuses on these problems is biomechanics. This field specializes itself in finding mechanical principles to living organisms, including humanoids ([McN05]).

To find a relationship between leg length and walk speed, information is needed that describes what a walking motion is and what it consists of. Boulic et al. [RB90] describe walking as: "By definition, walking is a form of locomotion in which the body's center of gravity moves alternately on the right side and the left side. At all times at least one foot is in contact with the floor and during a brief phase both feet are in the contact with this floor.". Figure 2.11 shows the temporal structure of the walking cycle with the main time and duration information.

This paper further describes how these, and more, of the walking characteristics can be calculated, eventually forming a human walking model. However, many of the input parameters are variables we are trying to find. But what becomes clear is that walking speed, just like any speed, is a function of distance and time:

$$v = fd$$

where v is walking speed, f is step frequency and d is step size. Thus if frequency and step size are known the walking speed can be calculated. However, the only input our method has is the length of the legs.

Hoyt et al. [DFH00] investigated the effects of limb length and running speed on time

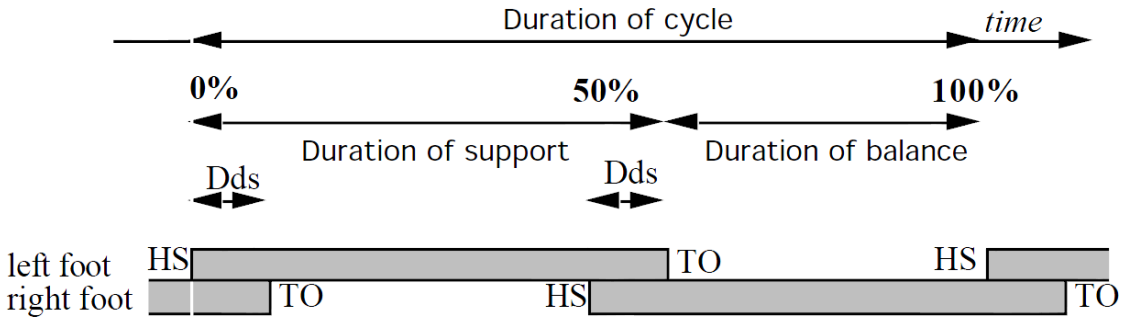

Fig. 2.11: Temporal structure of the walking cycle where Dc is duration of cycle, HS is heel strike, TO is toe off, Ds is duration of support (duration of contact with the floor), Db is duration of balance (duration of non-contact with the floor), Dds is duration of double support

Image source: [RB90]

of contact and step length. They use published data of 12 mammals, including humans, to analyze stepping behavior. One of their conclusions is that large animals, such as humans, have an average step length that is 25-30% shorter than their legs. This information is very useful, since it enables us to change the input of leg length into step size.

Much research has been done in the field of biomechanics concerning the frequency of the human walk. Grieve [Gri68] defined frequency based on empirical data as:

$$f = \alpha V'^{\beta}$$

where f is the number of complete cycles per minute, V' is the relative speed (stature/sec). The value α characterizes the individual and is between 61 and 65 for males and between 65 and 73 for females (according to the empirical data). The value β was about 0.58 according to the data, smaller for males and higher for females. This formula forms the base of many research papers. Kuo [Kuo01] uses different walking models to predict the preferred speed-step length relationship and uses this formula as a base truth, just like Bertram et al. [JB01] who observed individuals walking under different circumstances in order to find out which of these circumstances can predict the speed-frequency accurately.

Another interesting way of measuring the relationship between leg length and walking speed is to evaluate the effect of leg length and body mass on the gait speed and stride

length of infants. This was done by Rodriguez et al. [ER13]. Their study is interesting because the measurements are done over a period of six months while the leg length of the subject changed (the growth of an infant). Unfortunately no relation between leg length and stride length can be confirmed. It is also difficult to compare walking motion of infants to those of adults because the gait changes drastically until adulthood is reached.

2.7 Motivation

This chapter has shown that it is possible for a character to follow a path using a motion graph. Path synthesis makes use of the root trajectory of a character and depending on the difference between the given path and the trajectory a scalar function calculates an error. Normally, a root trajectory is easily created. However, when the leg length is different from the length during recording, the trajectory is changed. There is no existing method to calculate this trajectory, whose root projection on the ground is used during path synthesis, independent of leg length.

In the field of biomechanics much research involves human walking. Many different aspects of walking and the relationship between different aspects are defined, but a clear relationship between leg length and walking speed is not yet defined.

The lack of animation systems that can operate independent of morphology creates the need for one. Such a system should be able to create natural and smooth looking streams of motions for any given leg length. This requires a formula that calculates preferred walking speed depending on leg length.

Chapter 3

Synthesizing motion along a path

The goal of our research is to create a path synthesizer that takes leg scaling into account. Although the previous section discussed some aspects of walking, we still lack a formula that calculates the preferred walking speed of a person solely depending on his leg length. In section 3.1 we introduce an estimation of this formula. In section 3.2 our implementation of path synthesis is explained. When scaling, the root of the character needs a correction, this is covered in section 3.3. In section 3.4 solutions to different foot skating problems are given. The overall smoothening is discussed in section 3.5.

3.1 Preferred walking speed depending on leg length

A problem with the formula of Grieve, as discussed in section 2.6, and other formulas used in papers based on that formula, is that the increase in frequency and step length are based on characters with non-changing morphology. On average, the longer your legs, the greater the length of each stride. But since your legs need a longer time to travel, the step frequency decreases (Webb [Web96]). The formula by Grieve, however, indicate an increase in step frequency when the speed increases. This is likely because it is more manageable to increase frequency than it is to increase step length if the morphology remains the same.

In order to calculate the frequency, and therefore the preferred walking speed solely depending on leg length, we use a rough estimation in the form of a formula. The preferred walking speed of a normal-weight adult, according to Browning et al. [RB06], is about 5.0 km/h (or 1.39 m/s). Since the average human body length differs quite extremely we use the leg length of the average Dutch man (which corresponds to our recorded animation data) , which is 1.84 m. (Bogin et al. [BB10]), to calculate a base frequency based on the formula of Grieve (see section 2.6). This formula calculates the amount of full cycles per minute. One alteration will be the value of β, which will be

set to 0.43 based on the finding of Kuo ([Kuo01]) and the observation of Grieve that the value 0.58 was an average and should be lower on male subjects. This then makes:

$$f = 63(1.39/1.84)^{0.43}$$

Where 63 is the value that characterizes the individual and 1.39/1.84 is the relative speed (stature/sec). The frequency becomes 55.84 full cycles per minute, which is equal to 1.86 steps (half-cycles) per second. The estimation of the preferred walking speed formula should have this frequency as outcome with the leg length of 0.88 meter (1.84*0.479), where 1.84 is the average length of a Dutch male and 0.479 is based on the leg to body ratio according to Kreighbaum et al. [EK85].

The preferred walking speed will most probably decrease relatively when the leg length increases. This is because of several forces such as inertia and the forces acting on the hip torque (Kuo [Kuo01]). We estimate the frequency to be:

$$f = (0.143596l)^{-0.3}$$

where l is the length of the leg, measured from the hip to the ankle. The form of the formula was based on the knowledge of a decreasing frequency when the leg length increases. When the frequency is multiplied with the known step length, which is 25-30% less than the leg length, there should be an increase in walking speed, but a decreasing derivative. Note that this is a rough estimation and a more specific empirical study could drastically change the formula.

3.2 Path walk

In section 2.4 path synthesis was explained. The scalar function that calculates the error of a walk requires static leg length in order for it to work. One of the goals of this research is to change the leg length of a character during the animation. This requires an extension to the scalar function. The projected root at each frame, as seen in section 2.4, needs to be calculated according to the leg length and preferred walking speed. This changes the path and therefore the resulting stream of motions (see figure 3.1). A The new calculation for the root position becomes:

$$L_{root}(i) = L_{root}(i-1) + TnF_s$$

where $L_{root}(i)$ is the new root location at the i^{th} frame, $L_{root}(i-1)$ is the recalculated root location of the previous frame, T is the root translation of the non-scaled motion between frames i and $i-1$, n is a factor that counters the time warping (see section 3.4.1) and F_s is a scale factor calculated by dividing the preferred walking speed by the preferred walking speed of the base animation.

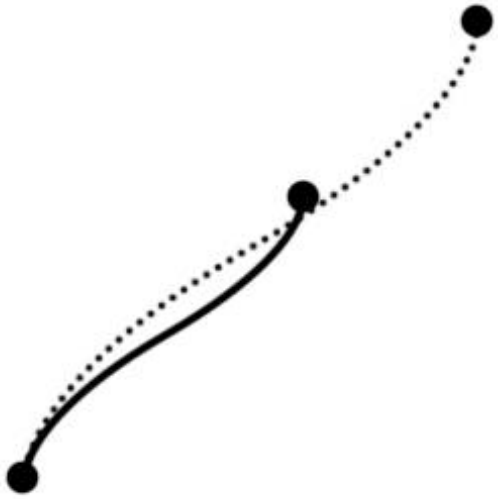

Fig. 3.1: This figure shows the change in the path when a motion is being scaled. The full line represents the motion before scaling and the dotted line represents the motion after scaling.

3.3 Y-correction

During runtime, the root position of the character needs to be recalculated each frame, otherwise the feet would go through the ground or hang mid-air, depending on the increase/decrease in leg length. There are two moments when the y-position is changed. First, at each frame the y-position of the root is translated according to the current scale. This is done by using the following formula:

$$L'^{y}_{root} = L^{y}_{root}(i) + h(b-1)$$

where L'^{y}_{root} is the new y-position of the root, $L^{y}_{root}(i)$ is the y-position of the root at the i^{th} frame, h is the largest height (i.e. y-difference) of the two legs measured from hip to foot (see figure 3.2) and b is the current scale of the legs in comparison to the base leg length. This basically scales the support leg and measures the height increase. This increase is then added to the root position, which then lifts the character by the same amount as the height increase of the support leg, which automatically means that the feet will not go through the ground or float mid-air.

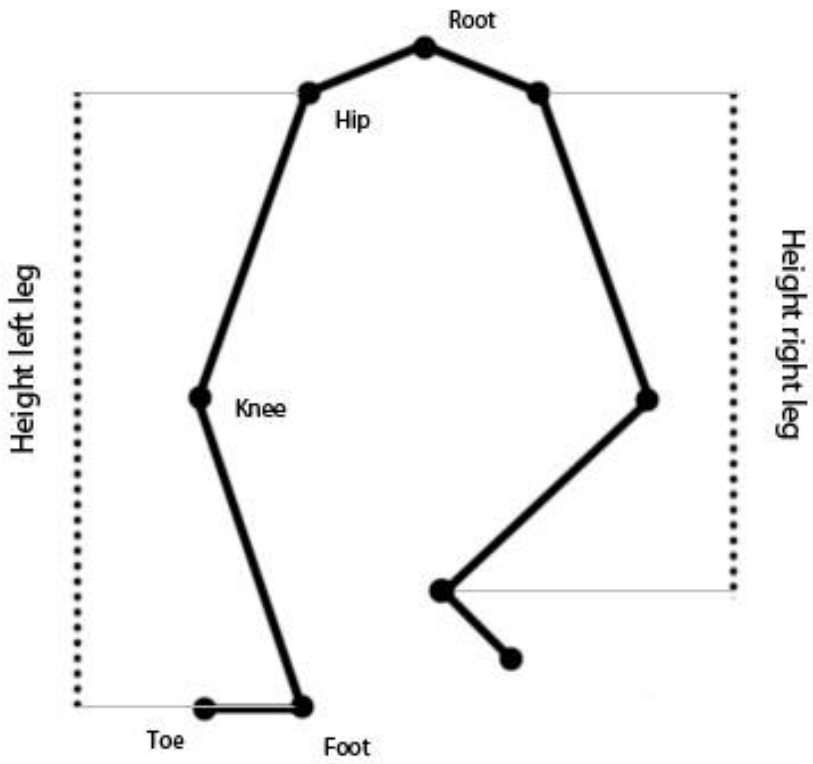

Fig. 3.2: An example of a pose where the y-difference of the left leg is larger than that of the right leg.

The second root position alteration is a correction that may be applied to the character

by the Morphology Independent Representation (Kulpa et al. [RK05]). When the legs are too short to reach the ground while a ground constraint is active, MIR moves the torso (i.e. the root) closer to the ground. This is a correction that is added to the previous y-correction discussed above. The actual y-position is a combination of both corrections.

3.4 Foot-skating

There are two cases of foot-skating that need to be solved. One is caused by the scale of the legs and the scaling of the speed not being a one on one ratio, while the second is caused by small errors in the motion capture data.

3.4.1 Time warp

When a motion is recorded, the actor has a certain average walking speed. When the length of the legs change, but the speed remains the same, the feet of the character will slide on the ground (i.e. foot skating). This foot-skating could be avoided by changing the speed by the same amount as the percentage increase in leg length. But, as the section 2.6 already showed, an increase in leg length does not have a linear increase in speed. In order to solve this problem, the ratio between the speed at which the animation plays and the speed at which the character walks is altered. This can be done by changing the speed of the animation while the speed of the character is set to his preferred walking speed. The following formula is used to calculate the new animation speed:

$$V_a = F_s/b$$

where V_a is the animation speed, F_s is a scale factor calculated by dividing the preferred walking speed by the preferred walking speed of the base animation and b is the current scale of the legs. The animation speed V_a is multiplied with the current timekey of the animation. This means that when, for example, you have an animation of 1 second, and V_a is 0.5, the animation will take 2 seconds, because the animation speed is halved.

Because the animation is played at a different animation speed during runtime, the average walking speed of the character is influenced. To compensate for this, a normalized time warp factor is calculated and multiplied with the translations during path synthesis and during runtime. For instance, when the animation is played at a slower rate, this factor will increase the translation, so that the walking speeds remains the same. The normalized time warp factor is calculated as follows:

$$n = 1/V_a$$

where n is the normalized timewarp factor and V_a is the animation speed.

3.4.2 Motion capture data cleanup

During the recording of motion capture data, the position of the body markers is determined on basis of different camera angles. There can be small errors each frame in locating these positions. In combination with small errors during the skeleton mapping, foot-skating occurs. When the character is scaled, this foot-skating can also be amplified and become more apparent. This section describes how the foot-skating is removed.

During the calculation of the path walk, the location of every joint can be calculated in world coordinates. The location for the left and right foot are stored in channels (basically a list of time keys and numbers). The moment a path walk has been calculated, the time periods where the feet touched the ground are calculated. This is done by getting the y-position Y of foot f at frame i, and the velocity V of the foot at frame i. If $Y(f, i,) \leq$ 'y-threshold' and $V(f, i) \leq$ 'velocity-threshold' the time key will be considered as active (Menardais et al. [SM04]) for that foot and stored in a new channel. When this is done for each foot for each time key there will be another iteration that calculates the average position of the foot during each active period. This is done as follows:

$$A_{foot} = \frac{\sum_{i=1}^{n} L_{foot}(i)}{n}$$

where A_{foot} is the average foot position, n is the number of frames of the time period and L_{foot} is the foot location of the i^{th} frame.

The average position is then stored in channels. At each keyframe during runtime the channels for each foot will be checked to see if there currently is contact with the floor. If so, the foot will be constrain to the calculated average position as discussed before. The forcing is done by using Morphology Independent Representation (MIR). At each frame a MIR-representation is created out of the current body pose. Then the corrected foot position is given as a constraint and using the MIR-representation the articulation (in this case the knee) is calculated using the bone lengths, half-plane and hip position. If there is no contact with the floor, the foot will not be constraint and the MIR will not be used.

Because of our y-correction, it is unlikely that the legs are incapable of reaching the ground. If there is a case when a foot constraint cannot be met because the leg is not capable of reaching its constraint, even after the torso correction, the constraint is ignored as a whole and the animation continues without it. The system will produce an error message to inform about the failure.

One drawback of this method is when a transition occurs during a support phase of a foot, a skipping-artifact may occur. This will produce two footstep constraints instead of one; one before the skipping and on after the skipping (see figure 3.3). This problem is not yet solved and is included in our future work.

Fig. 3.3: An image where the footstep is divided into two parts. The cubes are foot locations at each frame, the spheres are the footstep locations.

3.5 Smoothening

During the construction of the motion graph each frame of each animations is compared to each frame of each other animation. When the distance between two frames is below a given threshold a transition node is added. During the playback of a path walk one animation segment will proceed to the other in a fast manner. Depending on how alike these two frames are, the transition may contain artifacts where, for instance, the root rapidly moves a few centimeters to the side. In order to have a smooth motion transition, we use linear interpolation between the animation segments with a user-defined time window. From the start of the time window until the end the following formula will be used:

$$p_{new}(i) = p_1(i) * (1 - t) + p_2(i) * t$$

where p_{new} is the new 3D foot position at the i^{th} frame, $p_1(i)$ is the position of the current animation at the i^th frame, $p_2(i)$ is the position of the next animation at the i^th frame and t is the interpolation time (a value between 0 and 1, where 0 is the start of the time window and 1 is the end).

For us, linear interpolation was good enough for the small transition windows we used. Any other method could be used when desirable.

Chapter 4

Implementation

This section describes the implementation details of our system. It starts with an overview of our system in section 4.1. In section 4.2 we discuss the input and input parameters. Each important parameter is discussed and the limits of the values and the influence are discussed. In section 4.3 we cover the output of the system.

4.1 Setup

The method introduced and described in this report is implemented using the RAGE framework. RAGE is a framework provided and maintained by the Game and Media Technology staff of the University of Utrecht. The framework contains some basic animation methods (e.g. loading of animations, playback of animations) as well as some advanced methods. The RAGE framework is programmed using C++, but it can be controlled using Python scripts. This avoids compile times when making small changes to non-significant parts of the framework. For the 3D rendering RAGE uses the Ogre engine.

Pseudo-code for the high-level path synthesis and for the error calculation can be found in Appendix A. The pseudo-code is kept basic so it does not depend too much on our implementation.

Our system can be divided into two offline phases and an online phase. In the first offline phase we create our animations and motion graph. This information is stored so it can be used for multiple walks or even other purposes. The second offline phase includes the creation of the path walk and the calculation of the foot step positions. These calculations can take a while (see chapter 5) and are therefore stored so they can be played back at any time. During the online phase the walk is played back with the same parameters as the offline phase.

The first offline phase consists out of the following steps:

1. Collected motion capture data is put into separate animation files.
2. Desired animations are processed into a motion graph.

The second offline phase consists out of the following steps:

1. Given a path and thresholds, described in next section, the motion graph is searched for a path walk that satisfies these thresholds.
2. The footstep locations of the path walk and footstep periods are calculated.

This first step, where the path walk is created, also includes time warp calculations (see section 3.4.1). The root trajectory that needs to be calculated has to be the same as the root trajectory of the actual root trajectory during runtime. This is why this calculation needs to be calculated twice. The online phase uses the output of the offline phase and executes the path walk. The steps are:

1. The correct animation time t is calculated using timewarp so that the correct body pose q is extracted from the motion m.
2. The body pose is interpolated between previous and next animations, so that the overall animation is smoothened.
3. The leg length of the skeleton is increased or decreased so that it matches the given scale at the current time.
4. The root of the skeleton is corrected so that the legs and feet do not intersect the floor.
5. Morphology independent representation is used to retain the foot constraints at footstep periods.

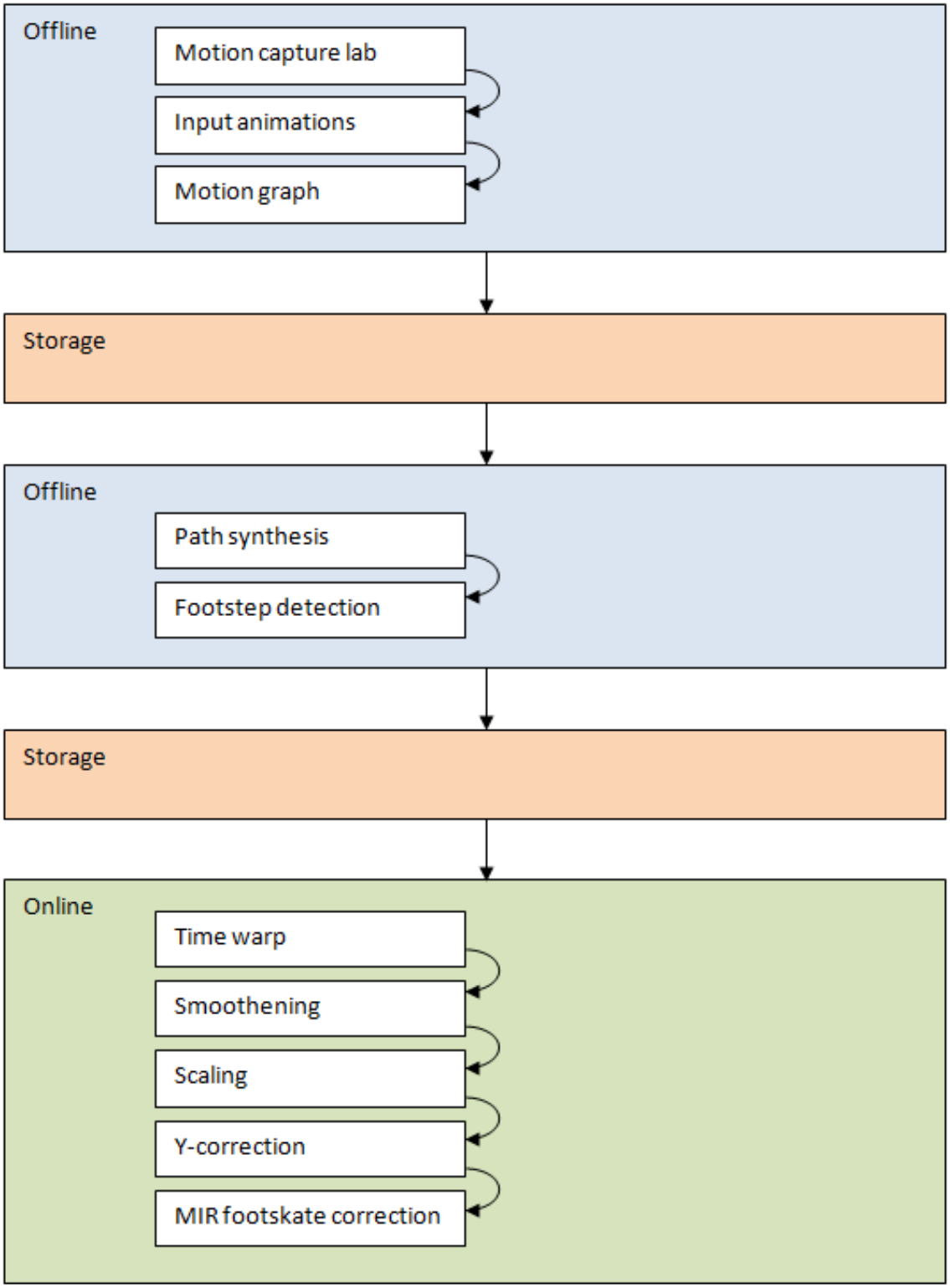

Fig. 4.1: General overview of the steps in our system

4.2 Input and parameters

There are several inputs and parameters a user can give to our system. In this section we will discuss the purpose, value range and influence of them all. The effect of some of the parameters can be seen in the results section.

anim_pairs This parameter contains the animations that will be used while constructing the motion graph. It is a list of paired animations. The user defines what animations should be compared with what other animations, which eventually builds the graph. Increasing the number of animations will enlarge the graph and therefore increase the computational time for the path walk. At the same time, more diversity in animations will increase the quality of the path walk, minimizing the total error. (Note that the motion graph will only be rebuild once the file 'motiongraph.sqlite' is not present in the 'Rage/scripts' folder.)

path The user may define a path which the character should follow. The parameter is a list of 2D points (x and z coordinates) on which we apply linear interpolation. A longer path increases computation time, but the increase is limited by the parameter 'iteration_length'.

iteration_length During the search of an acceptable path walk, the total length of the walk is divided into iterations. Each of these iterations has the length of this parameter. Once the current length of a walk is equal or greater than the iteration_length, the current walk will be compared to the best walk of the current iteration. Once the best walk of the iteration is found, the next iteration starts from the best iteration walk, until the total walk is completed.

scale_planning The user may define when the character changes its size and how long it takes to do so. This parameter is a list of three numbers. The first one indicates after what time during the walk the character should initiate the scaling. The second number represents the target scale. The third number represents at what time during the walk the scaling should be complete. Multiple scaling may take place during a walk. All the numbers should be larger than zero.

motion_graph_threshold This parameter represents the threshold for transitions during the construction of the motion graph. The lower the threshold, the smoother the transitions will look, but the graph will have worse connectivity and will be

smaller. The value should be between 0 (meaning the poses should be identical at a transition point) and the maximum possible number. If the threshold is too large, even motions that are not alike will have a transition point (e.g. a walking motion will transition in a mid-air jumping motion).

4.3 Output

After the algorithm is done, it generates several files in order to load the walk when the application is started. The following files are generated:

motiongraph.sqlite contains the complete motion graph. Once this file is removed, a new motion graph will be generated when the script is run again.

pathwalk.txt contains the last generated walk. Once this file is removed, a new walk will be generated when the script is run again. Note that all the 'pathwalk_'-files need to be removed in order to generate a new walk

pathwalk_footsteps.txt contains all the foot positions during the walk. Once a walk is loaded, the foot-skating will be solved using these positions.

pathwalk_visual.txt contains all the positions of the feet and root in order to give a visual representation of the complete walk. Only used for visual purposes.

Chapter 5

Results

In this section we will show the results of each individual part of our method. First we will show the performance of the path synthesis method in section 5.1, where the effect of each important parameter is shown. In section 5.2 the results for the foot skate solving is shown. Section 5.3 shows the results for the animation smoothening.

All experiments were executed on an desktop computer with an Intel Core i5-2500K 3.30 GHz quad-core processor and 16GB of RAM memory.

5.1 Motion graph

There are several parameters that influence the quality and computational performance of the path walk. In this section the influence of these parameters will be made clear. In order to compare the different parameters, the same path and same animations were used in these examples. One animation was used together with its own mirror-image (see figure 5.1). Because more animations drastically increased computational times, we limited them to these two (more on this can be found in the conclusion section). The path we chose can be seen on all the images, and the total length was 19.85 meter.

Because the change in leg length during an animation is interpolated, the results of an animation with changing leg length does not nearly conclude as much about the method as a full path walk with a higher or lower scale. For this reason no cases of leg length change during animation is mentioned in the results.

5.1.1 Scaling

The same path was calculated with three different scales: half the normal size, the normal size and one and a half times the normal size. The results of the 1.0 and the 1.5

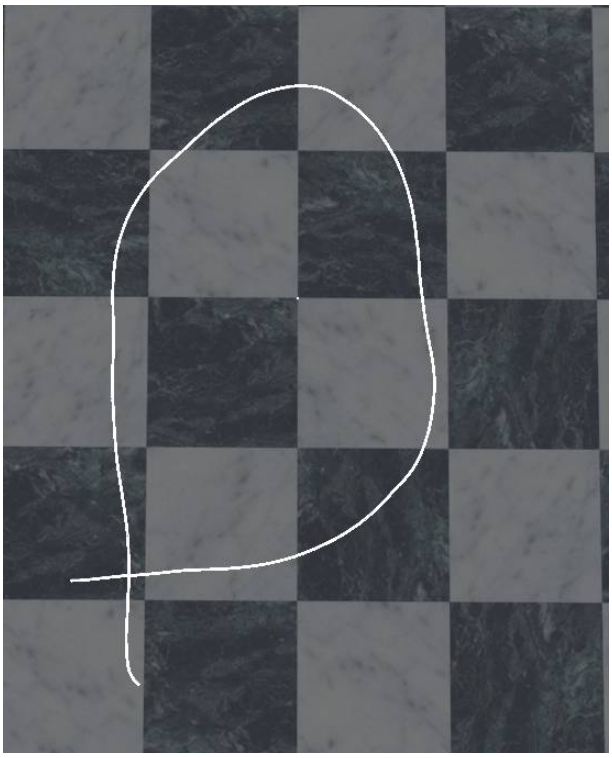

Fig. 5.1: The white line in this picture represents the root trajectory of the character during the base animation used in our path walks. A mirror of this animation was used as well, where the character first takes a left u-turn followed by a left turn.

scale can be seen in figures 5.7 and 5.11. One interesting observation is the first left turn in both images. Both turns use the same animation, but the scaled character has a greater arc, since his walking speed increased.

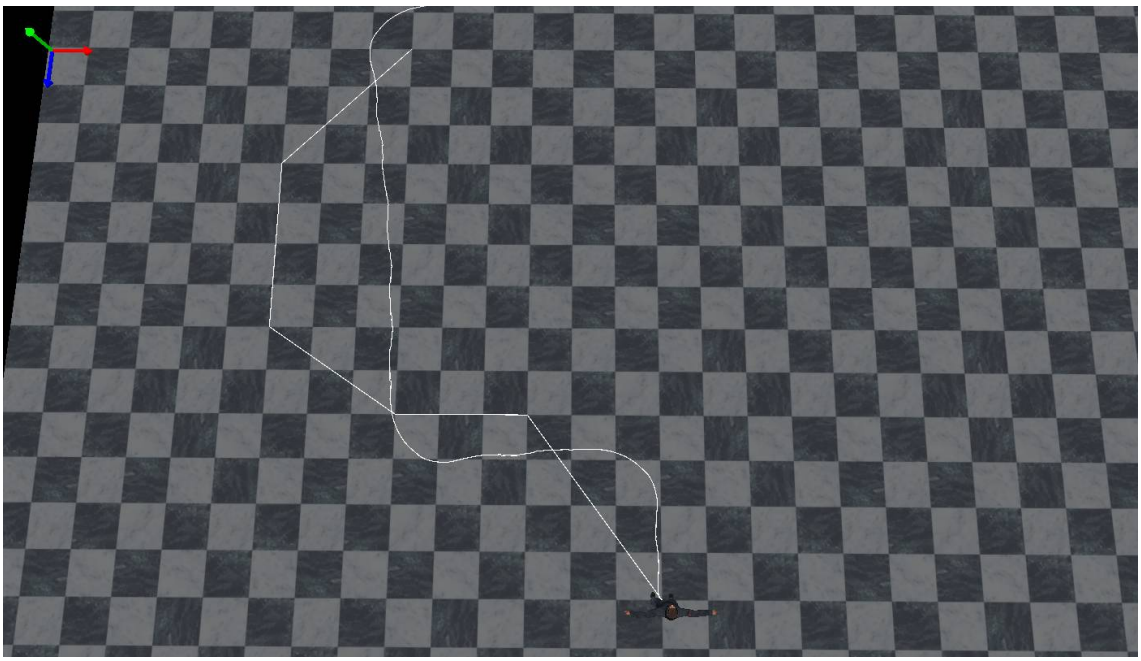

Fig. 5.2: Walk with character not scaled. Motion graph threshold: 0.006, iteration length: 10, iteration threshold: 0 (best possible path)

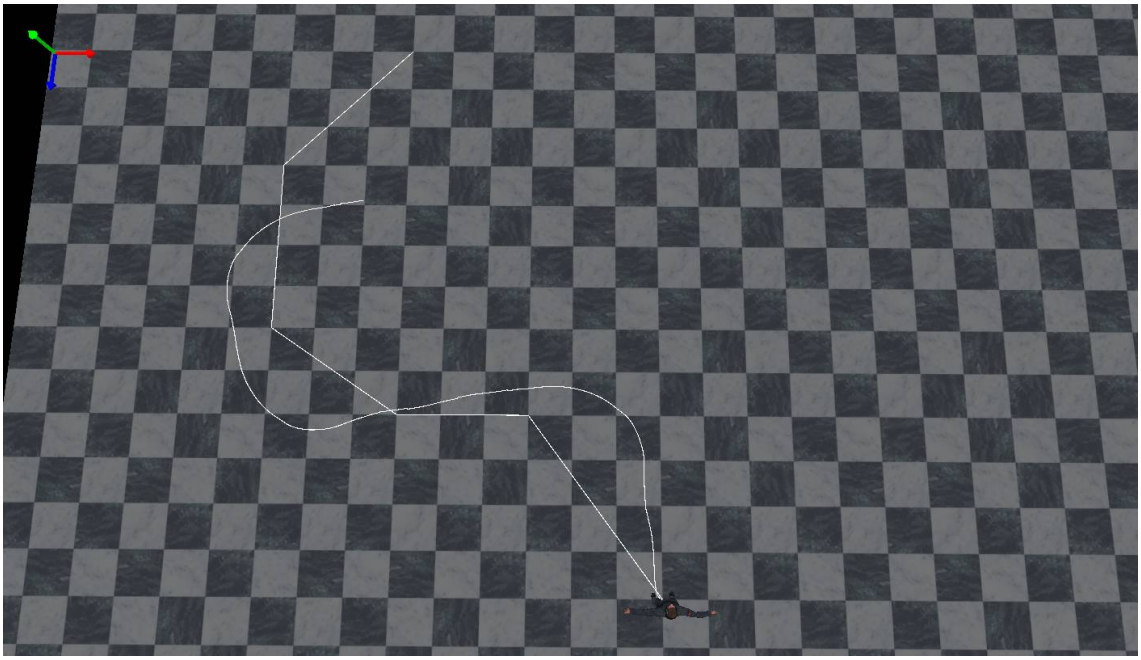

Fig. 5.3: Walk with character scaled on 1.5. Motion graph threshold: 0.006, iteration length: 10, iteration threshold: 0 (best possible path)

The computation times of these walks can be found in table 5.1. The search time of a path walk increases exponentially. This means that when a longer path is given by the user, it takes a longer time to compute a walk. When a characters leg length is increased, it has a larger walking speed. This means that it can travel a larger distance during an animation and that a path walk needs less animations to reach the path length. In the case of the 0.5 scale this effect is the other way around. It takes more animations to reach the path length, and therefore the search time increases drastically. The computation of the graph walk where the scale is 0.5 took longer than 7 hours. Unfortunately the program had to be terminated, but it still gives an indication about the increase in time.

Scale	Time (hh:mm:ss)
0.5	$>$ 7 hours
1.0	00:15:09
1.5	00:01:07

Table 5.1: The computational times of the path walks

5.1.2 Iteration length

In order to test the effect of iteration length on the path walk, several walks were created with different iteration lengths. The results can be seen from figures 5.4 to 5.11. The computation times can be seen in table 5.2. The total distance of the given path is 19.85 meter.

In our point of view, the overall results of the scaled version look worse than the non-scaled version. This is probably because the animations containing a bend have a greater radius when scaled. This makes it harder for the character to follow the path as specific points. The computation time, as seen before, greatly decreases when the character is scaled.

Scale	Iteration length (meter)	Error	Time (hh:mm:ss)
1.0	2.5	11238.20	00:00:06
1.0	5.0	3532.73	00:01:19
1.0	7.5	1869.43	00:02:02
1.0	10.0	1499.05	00:15:09
1.5	2.5	15428.80	00:00:01
1.5	5.0	3604.49	00:00:12
1.5	7.5	17192.10	00:00:30
1.5	10.0	2021.81	00:01:07

Table 5.2: The computational times of the path walks with different iteration lengths

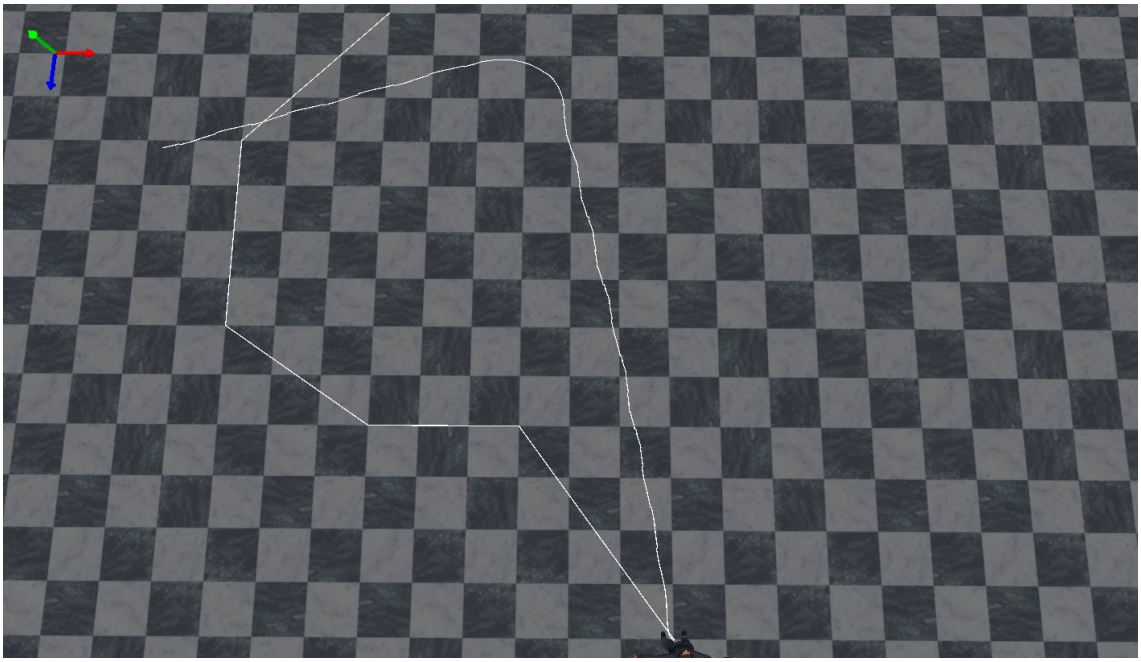

Fig. 5.4: Walk with character not scaled. Motion graph threshold: 0.006, iteration length: 2.5, iteration threshold: 0 (best possible path)

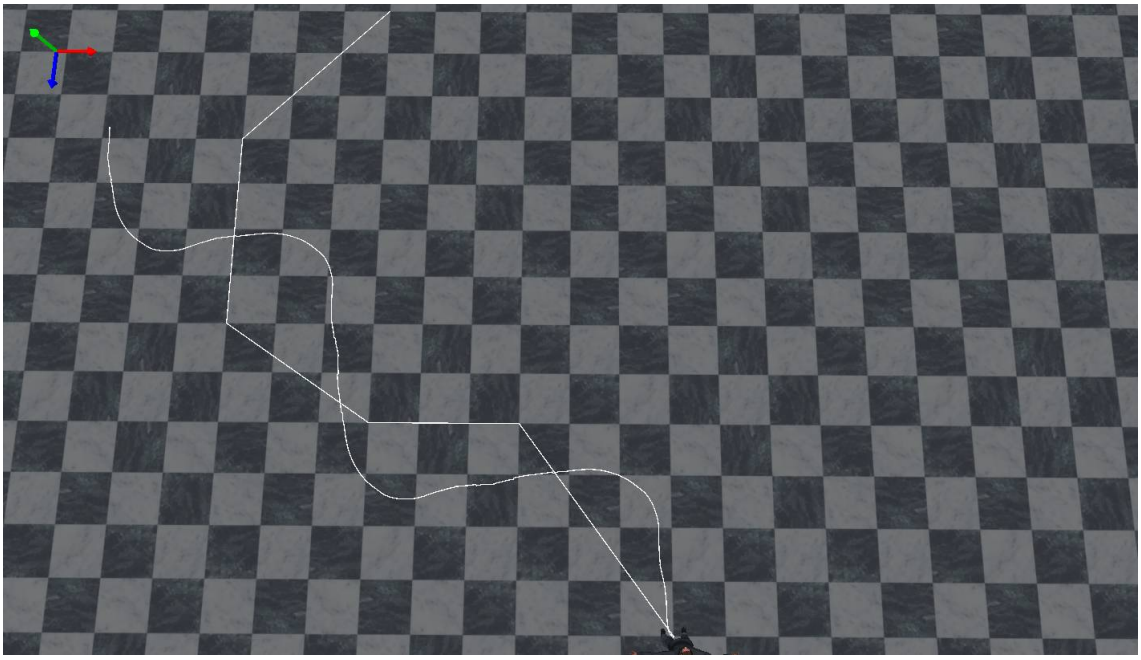

Fig. 5.5: Walk with character not scaled. Motion graph threshold: 0.006, iteration length: 5.0, iteration threshold: 0 (best possible path)

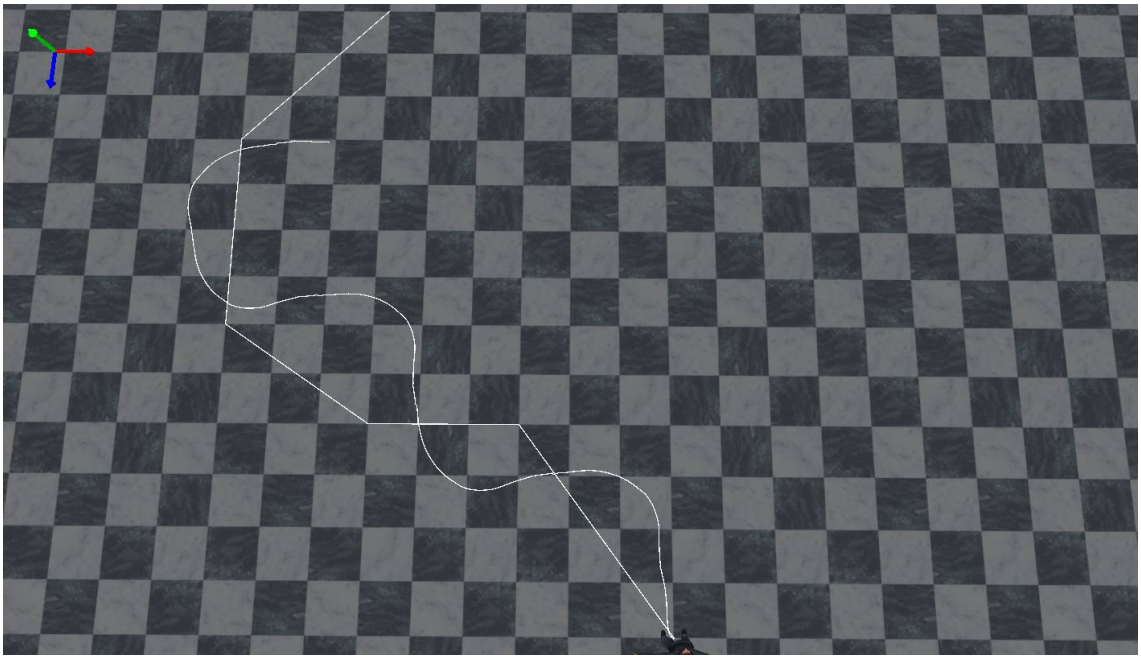

Fig. 5.6: Walk with character not scaled. Motion graph threshold: 0.006, iteration length: 7.5, iteration threshold: 0 (best possible path)

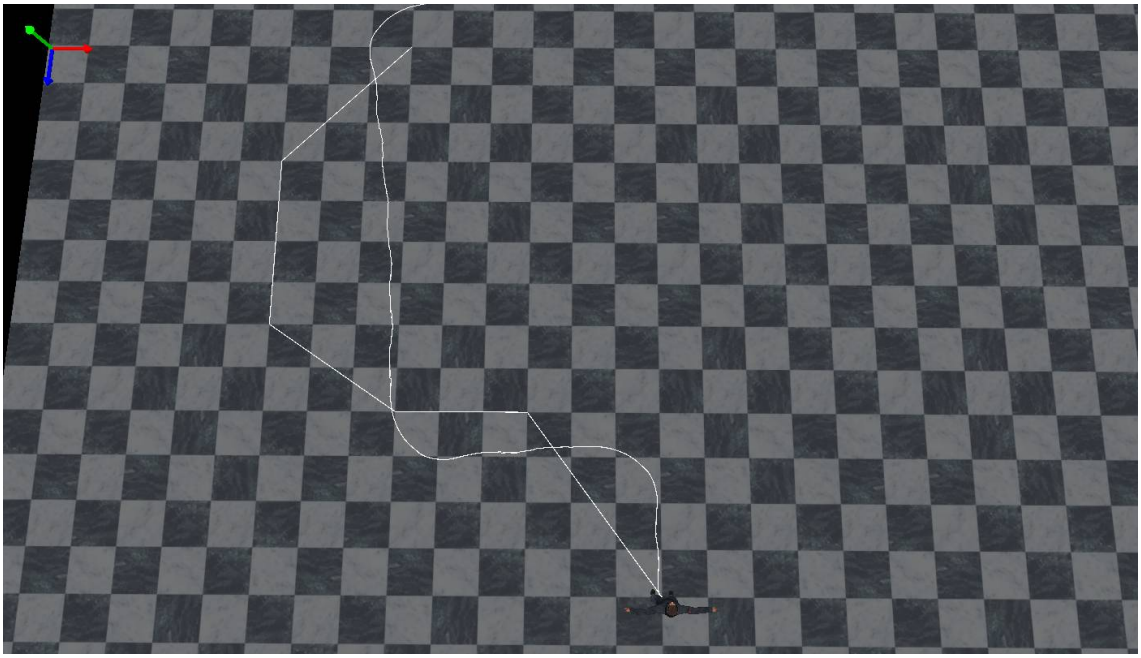

Fig. 5.7: Walk with character not scaled. Motion graph threshold: 0.006, iteration length: 10, iteration threshold: 0 (best possible path)

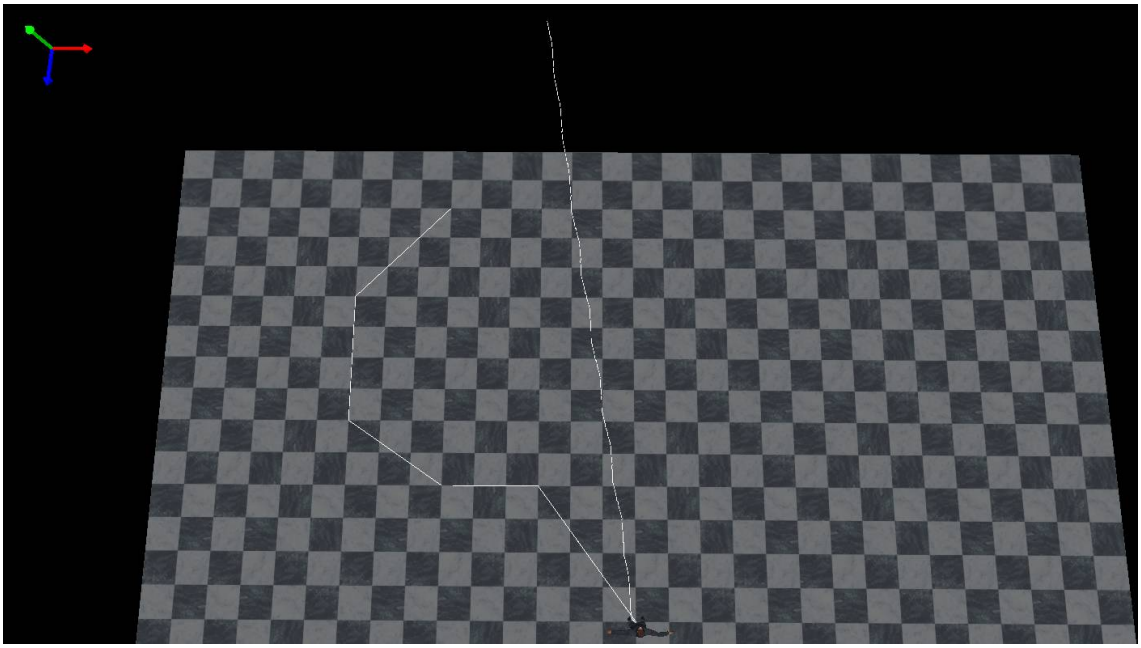

Fig. 5.8: Walk with character scaled by 1.5. Motion graph threshold: 0.006, iteration length: 2.5, iteration threshold: 0 (best possible path)

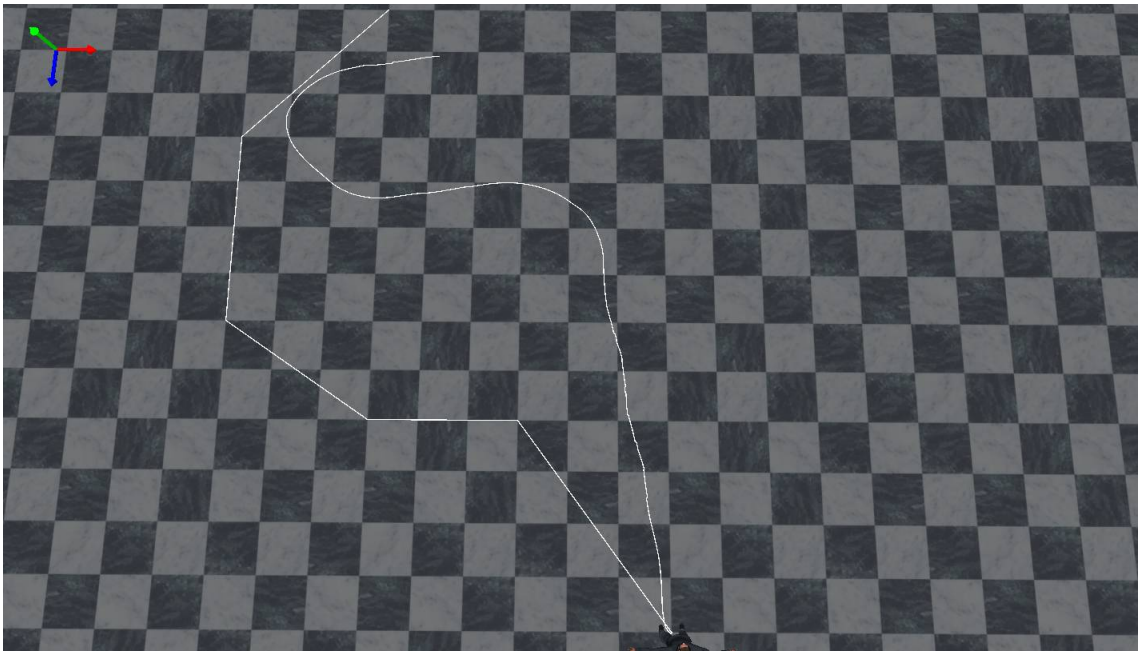

Fig. 5.9: Walk with character scaled by 1.5. Motion graph threshold: 0.006, iteration length: 5.0, iteration threshold: 0 (best possible path)

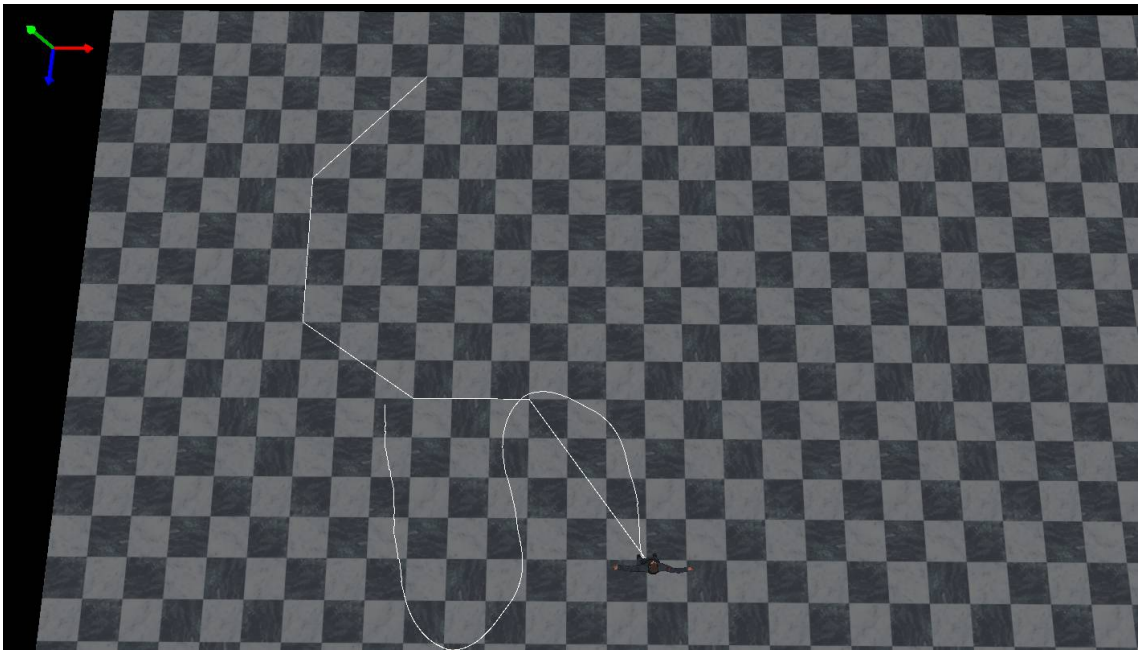

Fig. 5.10: Walk with character scaled by 1.5. Motion graph threshold: 0.006, iteration length: 7.5, iteration threshold: 0 (best possible path)

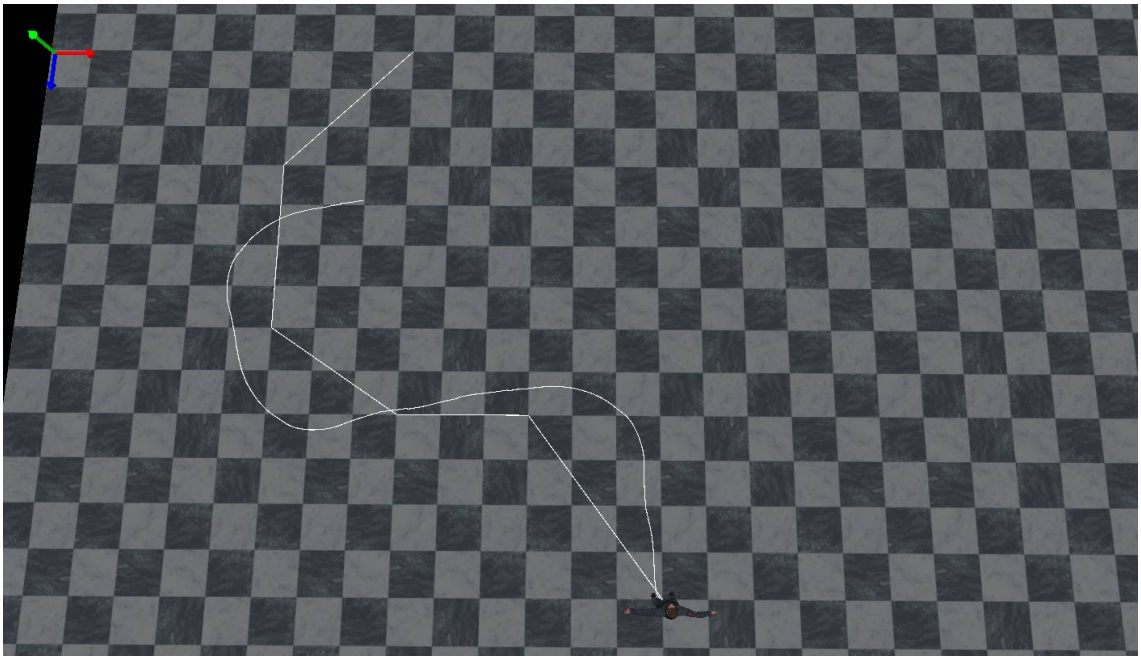

Fig. 5.11: Walk with character scaled by 1.5. Motion graph threshold: 0.006, iteration length: 10, iteration threshold: 0 (best possible path)

5.1.3 Iteration threshold

In order to test the effect of the iteration threshold on the path walk, several walks were created with different iteration thresholds. The results can be seen from figures 5.12 to 5.16. The computation times can be seen in table 5.3.

There are several interesting observations to be made. First of all, the difference between the iteration threshold of 3000 and that of 2500 show a significant increase in accuracy (decrease in error), the difference between 2500 and 2000 no increase and the difference between 2000 and 1500 show a decrease in accuracy. This is most probably a result of the iteration length. The error at the start of the path walk of the lower thresholds seem to be actually lower than those at higher thresholds, indicating that the first iteration successfully increase accuracy with a lower threshold. But the animations in the second iteration seem to be less suited for the remainder of the path. This is one of the drawbacks of the iteration length. Another observation is the increase in computation time. It is expected that for a better accuracy, more time is required and the results seem to support that expectation. The last observation are the duplicate outcomes of 2500 and 2000, and 1500 and 1000. This can be explained by the 2500 and 1500 thresholds already being below the 2000 and 1000 thresholds, therefore not changing the outcome. The slight differences in computation time are most likely due to different background processes on the computer.

Scale	Iteration threshold	Error	Time (hh:mm:ss)
1.0	3000	5231.03	00:01:24
1.0	2500	3309.00	00:03:26
1.0	2000	3309.00	00:03:32
1.0	1500	4037.92	00:08:54
1.0	1000	4037.92	00:08:52

Table 5.3: The computational times and error results of the path walks with different iteration thresholds

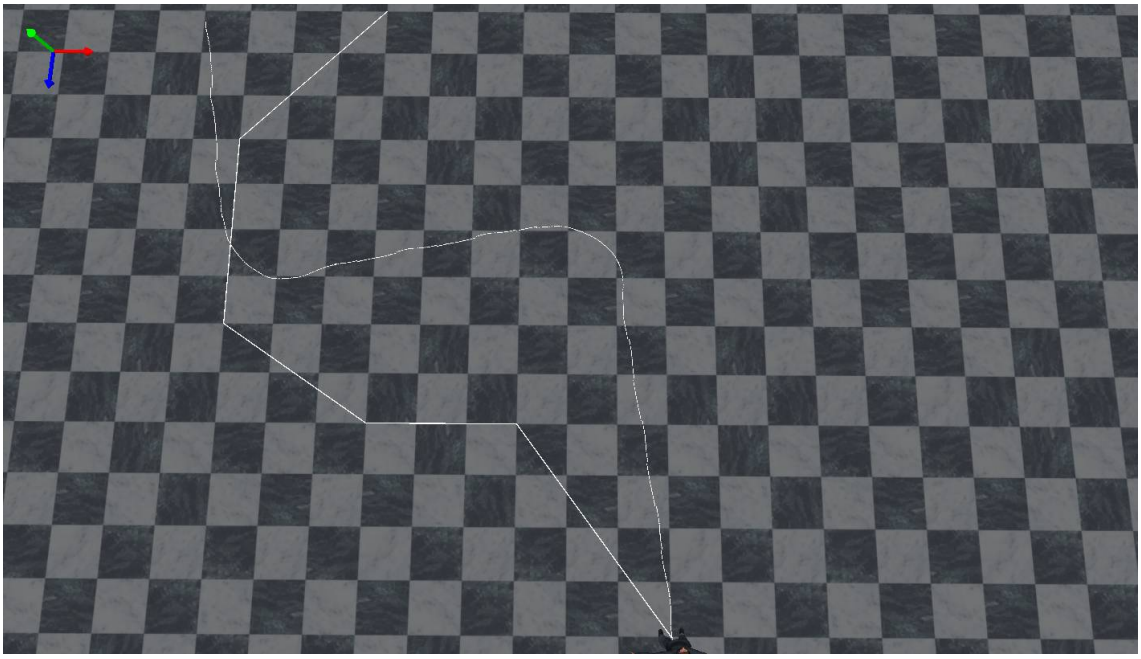

Fig. 5.12: Walk with character not scaled. Motion graph threshold: 0.006, iteration length: 10, iteration threshold: 3000

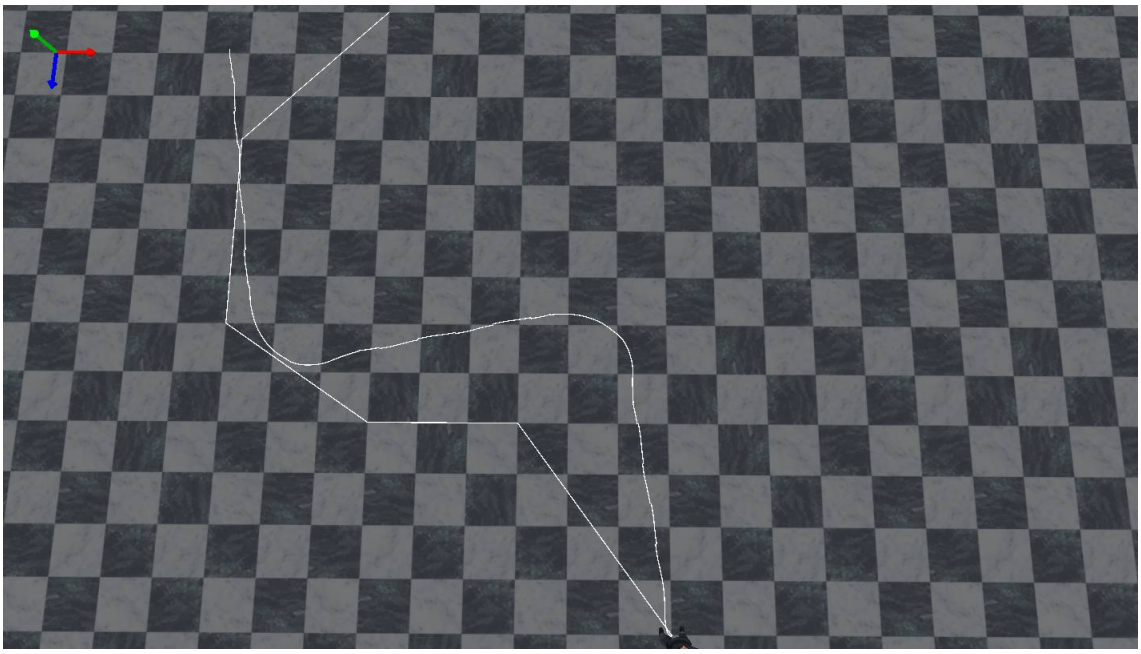

Fig. 5.13: Walk with character not scaled. Motion graph threshold: 0.006, iteration length: 10, iteration threshold: 2500

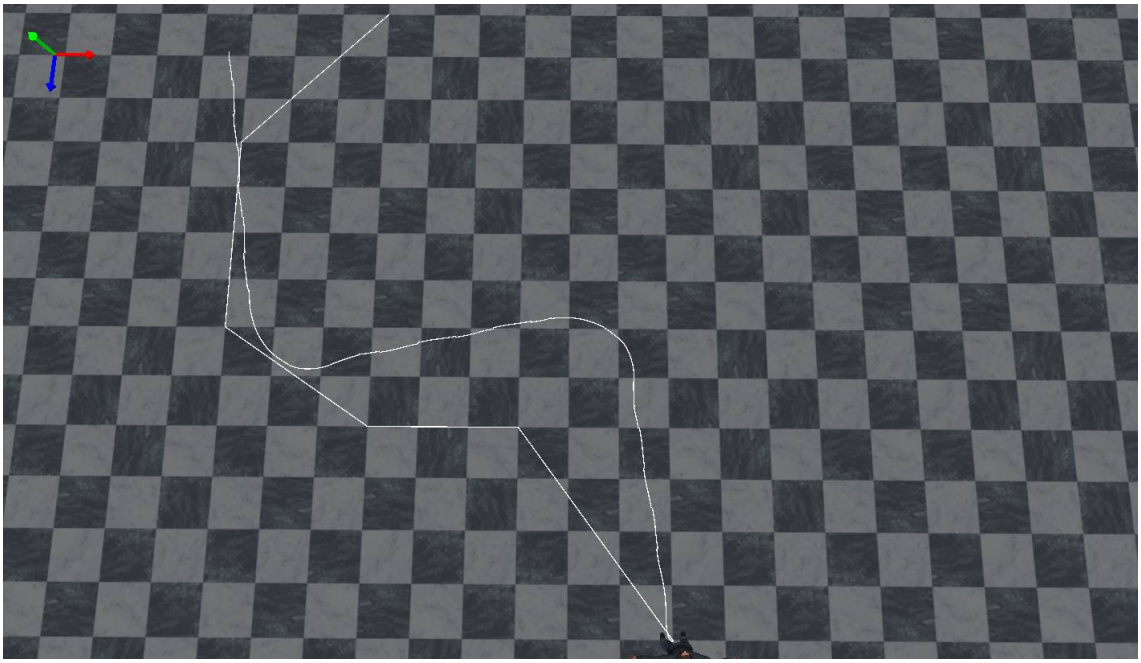

Fig. 5.14: Walk with character not scaled. Motion graph threshold: 0.006, iteration length: 10, iteration threshold: 2000

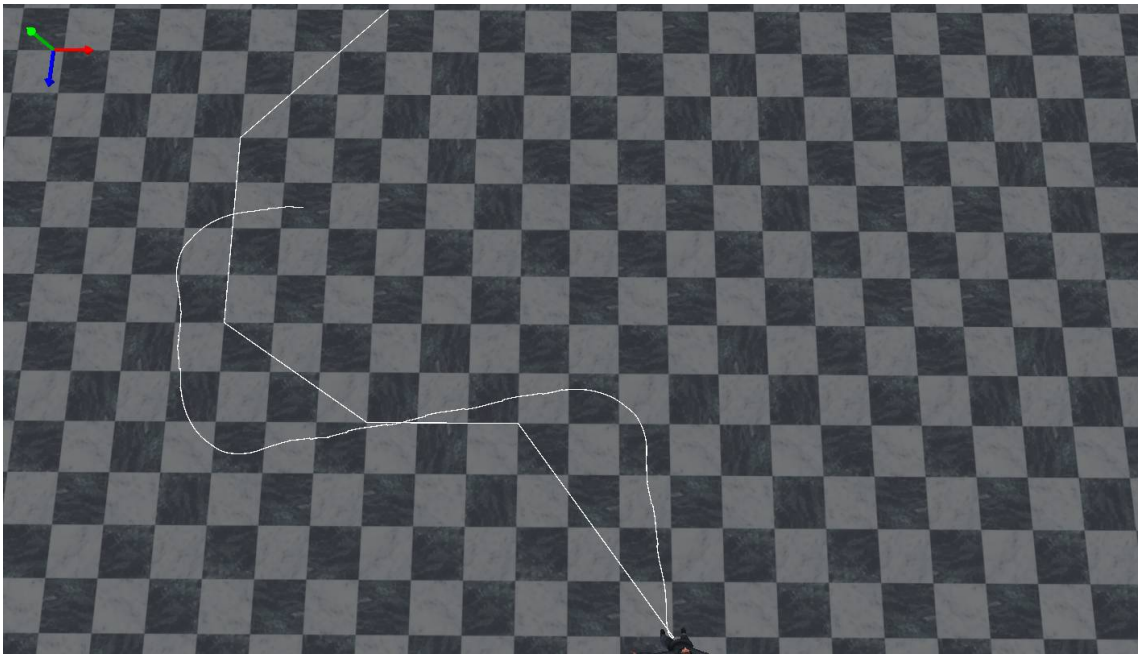

Fig. 5.15: Walk with character not scaled. Motion graph threshold: 0.006, iteration length: 10, iteration threshold: 1500

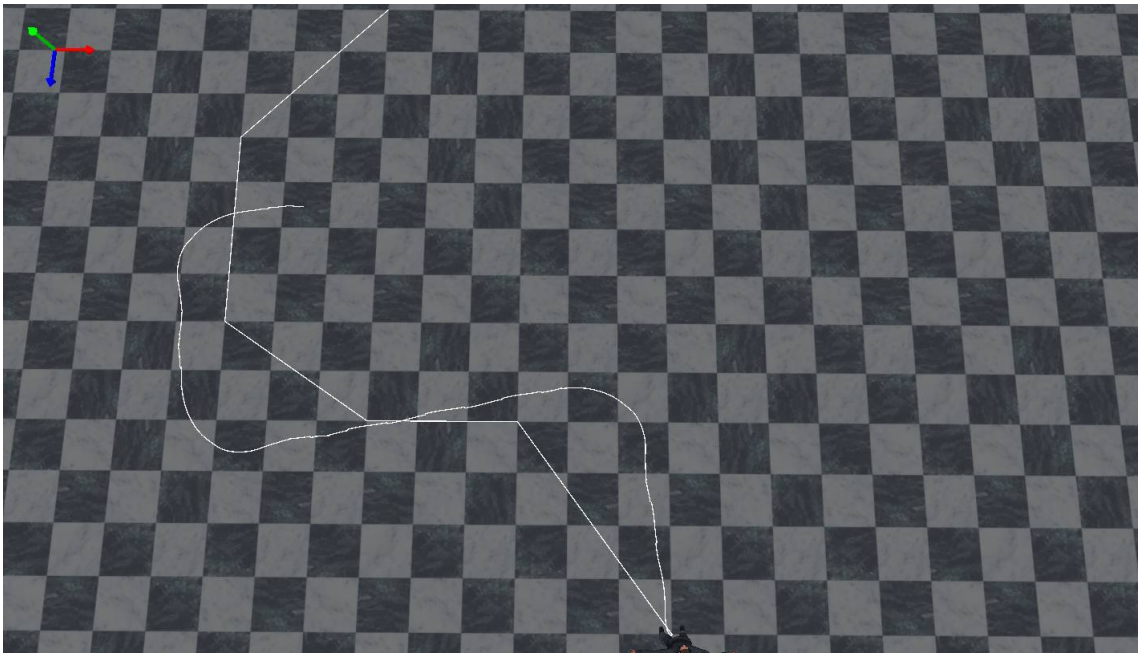

Fig. 5.16: Walk with character not scaled. Motion graph threshold: 0.006, iteration length: 10, iteration threshold: 1000

5.2 Foot-skating results

In figure 5.17 the path of the left foot of an arbitrary path is shown at the location where a footstep occurs. By using the technique described in 3.4.2 the average foot position during the footstep-period is calculated. The results of this is shown in 5.18. During runtime the morphology independent representation makes sure the foot stays in place. Figure 5.19 shows the path of the left foot during runtime using the MIR method.

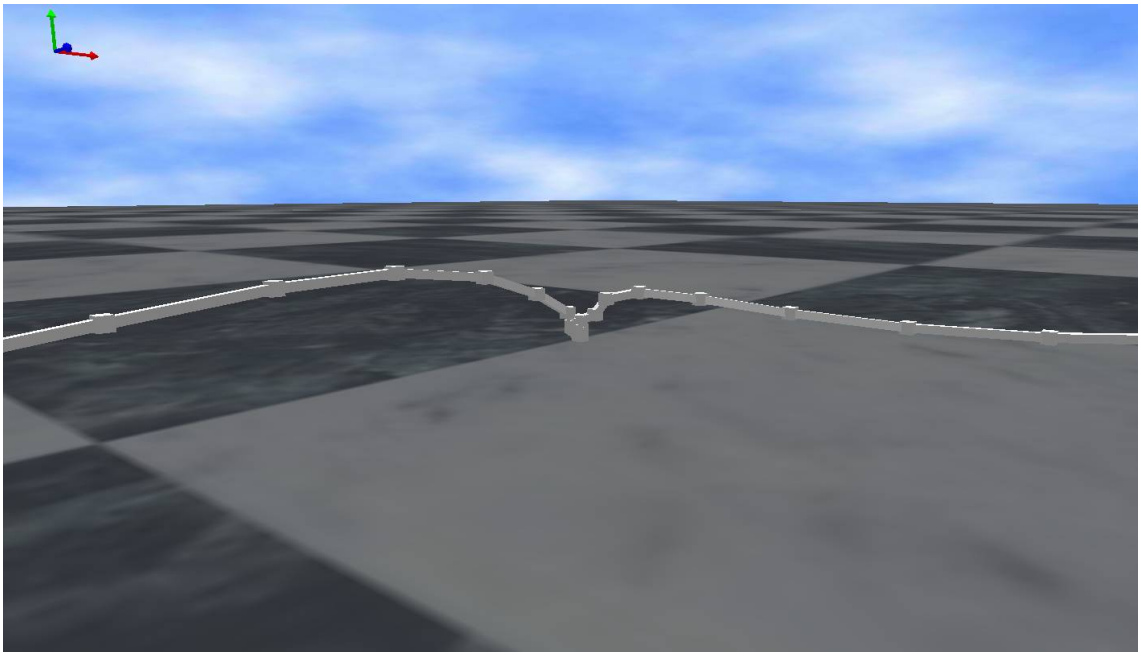

Fig. 5.17: Path of the left foot before correction

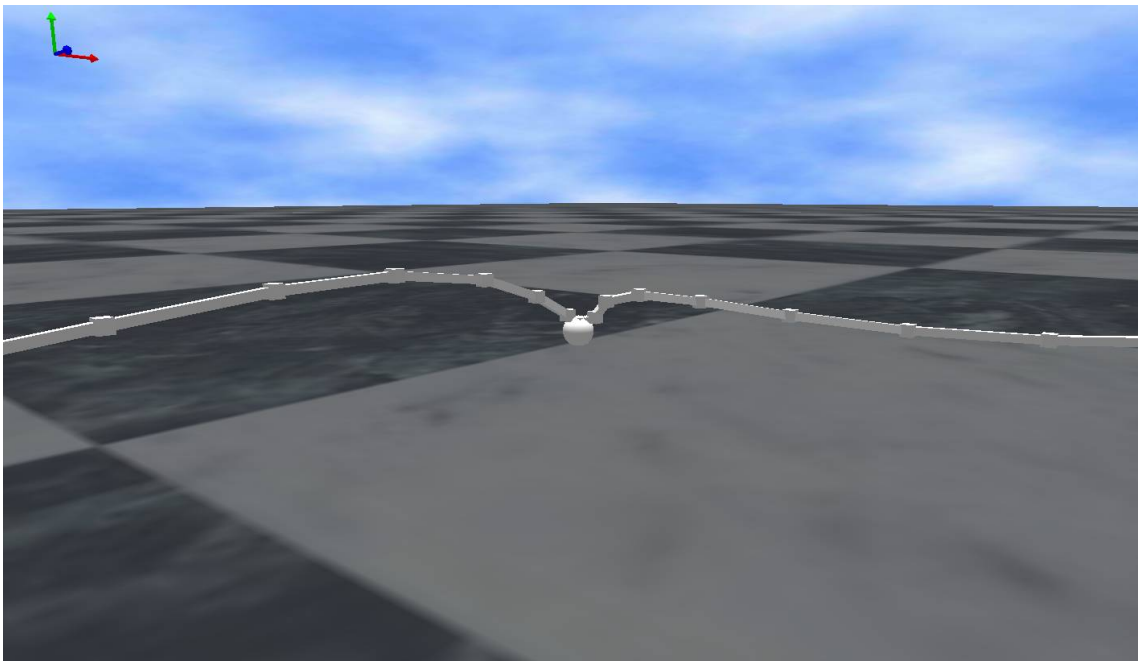

Fig. 5.18: Path of the left foot with the average position during the footstep. Apart from the detected footstep, the image is the same as figure 5.17.

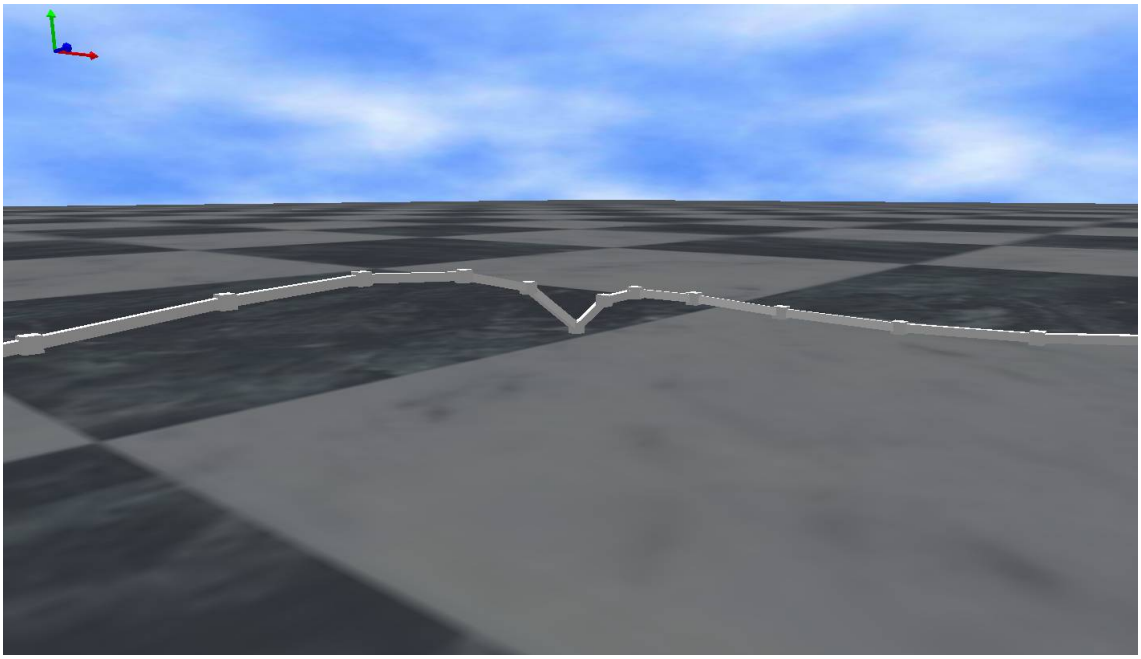

Fig. 5.19: Path of the left foot at runtime, with correction

5.3 Smoothening results

These are the results of the smoothening as discussed in 3.5. Figure 5.20 shows the path the root takes following an arbitrary path. This figure shows two obvious skips. After smoothening, as shown in 5.21, this skip is less obvious.

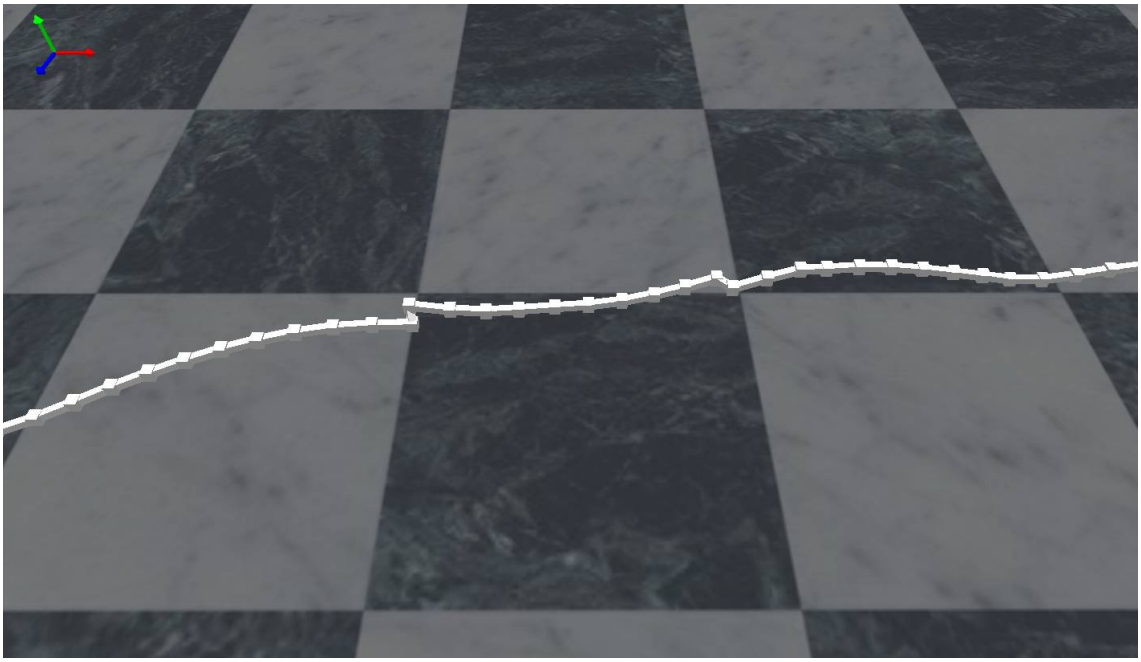

Fig. 5.20: Path of the root before smoothening

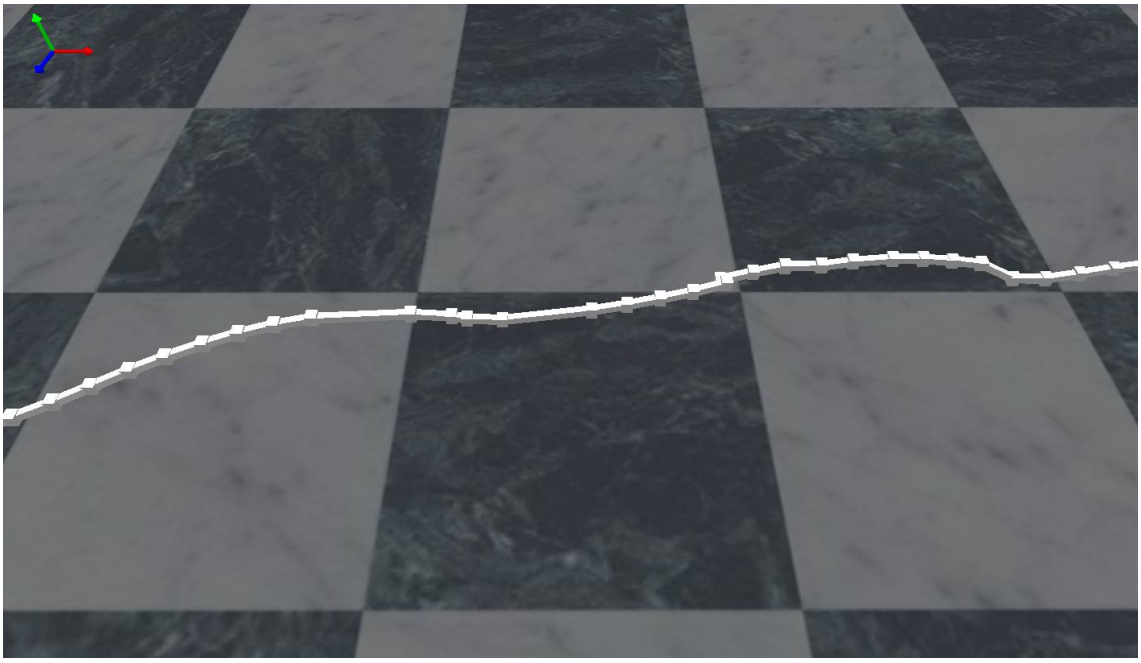

Fig. 5.21: Same path of the root after smoothening

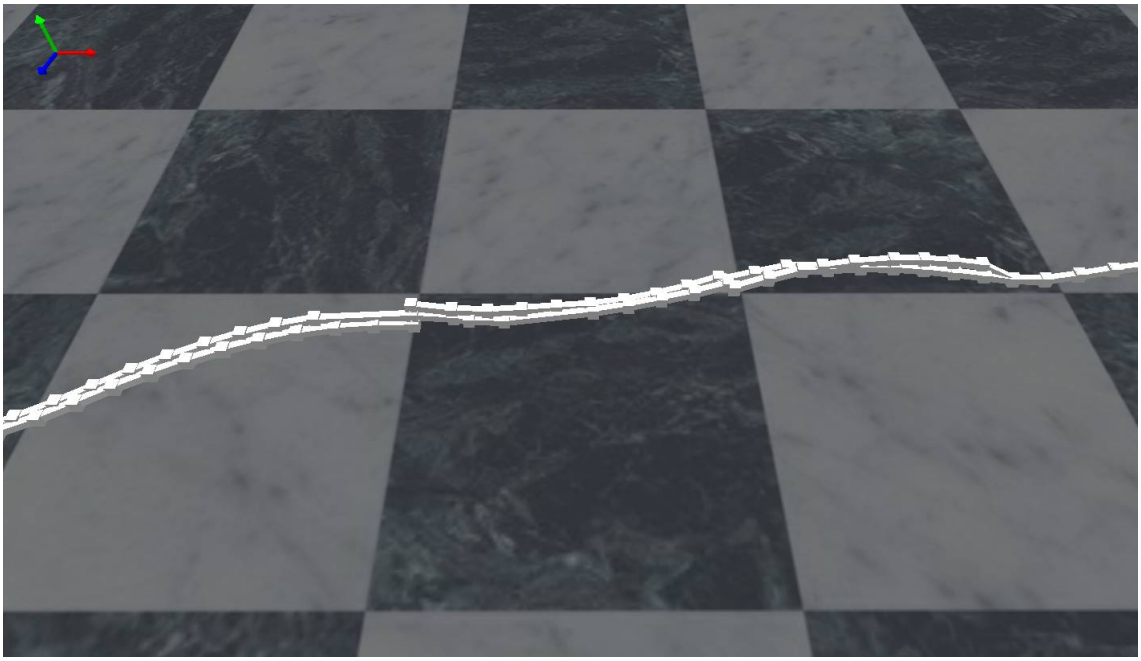

Fig. 5.22: Both paths combines to emphasize the results

Chapter 6

Conclusion and Future work

We have presented a method to create a path walk along a given path while the legs of the walking character changed. Additionally, methods to prevent (i.e. time warp) and actively solve (i.e. morphology independent representation) foot-skating as a result of scaling were implemented and the overall animation was smoothened.

The initial goal of our research was to explore the possibilities of morphology change during animations. We found that solving the change in leg length was an interesting problem on its own and we started investigating it further. We believe we achieved our goal to find a solution for path synthesis independent of leg length. Although we deem our research successful, there is some room for improvement.

Even though we thoroughly investigated the relationship of preferred walking speed and leg length in the field of biomechanics, there is little research that compliments this research question. Unfortunately our resources did not allow us to collect our own data to investigate this relationship. We made an estimation of what the formula could look like and used this estimation in our system. If future research is able to find a more accurate formula, this would be very easy to integrate in our system.

We explored the change in leg size during walking animation, but did not fully explore the possibilities of arm and torso increase. In case of walking, these changes do not have an impact, since there is no interaction between arms and the torso and the environment. But if this was the case, for example when a person that is reaching for an object on a high shelve, the morphology increase influences the animation. The morphology independent representation seems fit as a real-time space-time solver for such a task. The retargetting method of Gleicher ([Gle98]) is also capable of such a task, although not in real-time.

To deal with the scalability problem a motion graph has, we use iteration lengths and thresholds. A great drawback of iteration lengths is that it is possible for an iteration to have a small error, but that the first edges of the next iteration result in motions

with big errors. For instance, when the path that needs to be followed is a straight line, and the iteration resulted in a walk that follows this line accurately, it is possible that the first edges of the new iteration are a left and a right turn, which produce large errors. There are methods to evaluate the capability of a motion graph (e.g. Reitsma et al. [PR07]). Such an evaluation can make clear if a given motion graph can handle the given task. Animations or thresholds can be changed until a motion graph is good enough for the given task, but this then may result in worse computation times. This makes us question if data driven searches are a good solution.

For our research we chose a very basic approach concerning motion graphs and path synthesis. There is much research available that point out problems and offer solutions to this basic approach. [BVB11] by Van Basten et al. is an example of this. They point out two issues. The first one is that the distance metric, in this case the point cloud, might not always yield the best results. Van Basten et al. propose a set of guidelines to what distance metric is useful for which goals (e.g. the point cloud metric has least foot-skating result, but is slow and has much path deviation). The second thing they point out is that the root trajectory (i.e. pelvis-trajectory) of the character follows a sinusoid displacement in the horizontal as well as vertical direction the character is walking in. This means that the character is unable to follow straight lines and enforcing it will cause unnatural motion. They propose a method that generates better animations. These kind of improvements are interesting to look into and implement into our current system.

Although the foot correction method using morphology independent representation works to a certain extent, there are some drawbacks. First, even after smoothing the overall animation, there is sometimes an obvious 'snap' when a foot reaches the ground. This is because the constraint solving does not look at frames surrounding the footstep period. A second drawback is that a footstep period may sometimes be interrupted for a few frames. This can be caused by transitions in the motion graph or moments where the threshold for footstep detection are not met for one or two frames. Higher threshold could also solve this, but the 'snap' may become even more apparent since the time window of the footstep becomes longer. This is something that needs to be improved in the future.

With some adjustments our system should, in theory, be able to handle other forms of input than only a given path. Real time character control using a keyboard or joystick is possible when using our method in combination with the correct thresholds. Currently

our path synthesis and footstep detection are offline, but when the thresholds are low enough, real-time calculations can be done to steer a character in a certain direction.

Bibliography

[BB10] M.I. Varela-Silva B. Bogin. Leg length, body proportion, and health: A review with a note on beauty. *International Journal of Environmental Research and Public Health*, 7(3), 2010. 18

[BVB10] A. Egges B.J.H. Van Basten, P.W.A.M. Peeters. The stepspace: Example-based foot-driven motion synthesis. *Computer Animation and Virtual Worlds 2010*, 21, 2010. 15

[BVB11] R. Geraerts B.J.H. Van Basten, A. Egges. Combining path planners and motion graphs. *Journal Computer Animation and Virtual Worlds*, 22, 2011. 49

[DFH00] E.A. Cogger D. F. Hoyt, S.J. Wickler. Time of contact and step length: The effect of limb length, running speed, load carrying and incline. *The Journal of Experimental Biology*, 203:221–227, 2000. 15

[EK85] K.M. Barthels E. Kreighbaum. Biomechanics: a qualitative approach for studying human movement. 1985. 19

[ER13] P.L.P. Silva R.N. Kirkwoord M.C. Mancini E.B. Rodriguez, P.S.C. Chagas. Impact of leg length and body mass on the stride length and gait speed of infants with normal motor development: A longitudinal study. *Brazilian Journal of Physical Therapy*, 17(2):163–169, 2013. 17

[Gle98] M. Gleicher. Retargetting motion to new characters. *SIGGRAPH '98 Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, 1998. 14, 48

[Gri68] D. W. Grieve. Gait patterns and the speed of walking. *Biomedical engineering*, 3, 1968. 16

[JB01] A. Ruina J.E.A. Bertram. Multiple walking speed-frequency relations are predicted by constrained optimization. *Journal of theoretical Biology*, 209, 2001. 16

[Kuo01] A. Kuo. A simple model of bipedal walking predicts the preferred speed-step length relationship. *Journal of Biomechanical Engineering*, 123, 2001. 16, 19

[LK02] F. Pighin L. Kovar, M. Gleicher. Motion graphs. *SIGGRAPH '02 Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, 2002. 6, 8, 9, 10

[McN05] R. McNeill. Mechanics of animal movement. *Current Biology*, 15(16), 2005. 15

[MH05] M. Ouhyoung M. Hsieg, B. Chen. Motion retargetting and transition in different articulated figures. *CAD-CG '05 Proceedings of the Ninth International Conference on Computer Aided Design and Computer Graphics*, 2005. 14

[MM01] T. Calvert M. Mizuguchi, J. Buchanan. Data driven motion transitions for interactive games. *Eurographics 2001 Short Presentations*, 2(3), 2001. 6

[OA02] D.A. Forsyth O. Arikan. Interactive motion generation from examples. *SIGGRAPH '02 Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, 2002. 7

[PG04] D. Thalmann P. Glardon, R. Boulic. Pca-basedwalking engine using motion capture data. *CGI '04 Proceedings of the Computer Graphics International*, 2004. 14

[PR07] N.S. Pollard P.S.A. Reitsma. Evaluating motion graphs for character animation. *ACM Transactions on Graphics (TOG)*, 26 (4), 2007. 49

[RB90] D. Thalmann R. Boulic, N. Magnenat-Thalmann. A global human walking model with real-time kinematic personification. *The Visual Computer*, 6, 1990. 15, 16

[RB06] J.A. Herron R. Kram R.C. Browning, E.A. Baker. Effects of obesity and sex on the energetic cost and preferred speed of walking. *Journal of Applied Physiology*, 100(2), 2006. 18

[RH07] M. Gleicher R. Heck. Parametric motion graphs. *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, 2007. 8, 10

[RK05] B. Arnaldi R. Kulpa, F. Multon. Morphology-independent representation of motions for interactive human-like animation. *EUROGRAPHICS 2005*, 2005. 5, 22

[SM04] R. Kulpa B. Arnaldi S. Ménardais, F. Multon. Motion blending for real-time animation while accounting for the environment. *Proceedings of the Computer Graphics International*, 2004. 23

[Web96] D. Webb. Maximum walking speed and lower limb length in hominids. *American Journal of Physical Anthropology*, 101(4), 1996. 18

Chapter 7

Appendix A: Pseudocode

Below is the pseudo code, written in C++, for the global implementation of the path synthesis.

```
1  vector<Node> current_nodes;
2  while(! done){
3    //while there is still something to search for this iteration
4    while(current_nodes.size() != 0 && best_node.error > iteration_threshold
       )
5      //Search the children of the current node if:
6      //1. The current path length did not exceed the total or iteration
         path length (else, the current walk needs to finish)
7      //2. The error is less than the current error (if not, we already got
         a better path)
8      if(current_node.current_length < path_length
9      && current_node.current_iteration_length < iteration_length
10     && current_node.current_error < best_node.error){
11       vector<Node> children = get_children(current_node);
12       for each child in children{
13         calculate_error(child);
14       }
15       sort_children_greedily(children);
16       current_nodes.add(children)
17     }
18     else{
19       //If the current node is better than the best node, replace it
20       if(current_node.current_error < best_node.error)
21         best_node = current_node;
22     }
23   }
24   //Go to the next iteration unless we reached the final one
25   if(best_node.current_length >= path_length)
26     done = true;
27   else{
```

```
28      vector<Node> children = get_children(current_node);
29      for each child in children{
30        calculate_error(child);
31      }
32      sort_children_greedily(children);
33      current_nodes.add(children)
34    }
35  }
36  return best_node;
```

In the code above, a call is made to 'calculate_error'. The code for this method can be found below.

```
1  //Calculate the position and error of each keyframe
2  for each keyframe in current_node.keyframes{
3    //Calculate all the variables that we need
4    if(keyframe != current_node.start_keyframe)
5      current_node.total_duration = keyframe - previous_keyframe;
6    scale = calculate_current_scale(current_node.total_duration);
7    scale_factor = calculate_scale_multiplier(base_leg_length, scale);
8    timewarp_scale = scale_factor / scale;
9    normalized_scale = 1 / timewarp_scale;
10   original_position = choice.calculate_position(keyframe)
11
12   //Calculate the current position with the calculated variables
13   if(keyframe == current_node.start_keyframe){
14     current_node.previous_position = current_node.previous_node.
         previous_position;
15     current_node.previous_original_position = current_node.previous_node.
         previous_original_position;
16   }
17   current_translation = (((original_position - current_node.
       previous_original_position) * normalized_scale ) * scale_factor);
18   current_position = current_node.previous_position + current_translation;
19
20   //Update the current total length (2D)
21   current_node.current_length += current_translation;
22
23   //Calculate the error for this frame
24   path_position = calculate_path_distance(current_node.current_length);
25   distance = calculate_distance(current_position, path_position);
26   current_node.current_error += (distance * distance);
27
```

```
28    //Store information for the next keyframe
29    current_node.previous_position = current_position;
30    current_node.previous_original_position = original_position;
31  }
```