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

Bachelor Thesis

Qualitative Parameters for Modifying Animated Character Gesture Motion

submitted by

Pascal Pohl

submitted March 31, 2010

Advisors: Dr. Michael Kipp and Dr. Alexis Heloir

First Reviewer: Dr. Michael Kipp
Second Reviewer: Prof. Dr. Dr. h.c. mult. Wolfgang Wahlster

Contents

Abstract
Embodied agents are a powerful instrument to enhance interactivity and appeal of multimodal interfaces. However, creating gesture animation requires high effort and expertise especially when a specific, individual style is desired. In my thesis, I present a method to vary gesture animation with a set of qualitative parameters. These parameters allow the users of EMBR, a real time character animation system, to apply different styles to gestures. The names of the parameters give a clear idea, what changes they will invoke. The parameters are intuitively to use and provide the creation of a wide range of styles. To modify a gesture, parameters are assigned to the motion inside the EMBRScript document containing the gesture description for EMBR. I explain my set of parameters and their influence on a gesture and describe the changes and implementations I had to realize for both EMBR and EMBRScript. With the help of my qualitative parameters the user can now in an easy way emphasize crucial parts of a gesture or generate gestures that reflect the particular personality of a character, leaving the actual gesture definition untouched.

Acknowledgments

First of all, I would like to thank Michael Kipp and Alexis Heloir for being very helpful and inspiring advisors. Thanks as well to the EMBOTS research group and the DFKI for providing the opportunity to work on this projekt. The work and research for my thesis was very interesting and would not have been possible without the help of my advisors and friends. Finally I would like to thank my parents and my girlfriend Alexandra for their support and patience during my studies.

1. Introduction

1.1. Motivation

The power of computer hardware is developing fast and both costs and size are getting smaller and smaller, more and more systems are included in our everyday life. The interest in realistic virtual characters is raising since more and more interactive systems are getting developed. The high realism of nowadays computer graphics need convincing behavior of the characters for a wide range of application including 3D games, multimodal interfaces and computer animated films.

To animate a virtual character or object, a skeleton structure, consisting of a chain of single joints, is often used to move the visible hull of the character. The hull follows the movements of the skeleton, stretching it allows detailed modifications (e.g. facial expressions), but the main motion is caused by the skeleton manipulation. The skeleton is animated by specifying rotation and position for every joint in the kinematic chain (kinematic chain means the subset of joints used to posture the skeleton).

There are several methods to generate the data needed to specify the skeleton posture through the whole gesture, important techniques are key frame animation and motion capture.

Key frame animation is the most similar method to traditional hand-drawn cartoons. The animator defines so-called key frames containing the important postures of the motion, for theses frames the animator specifies position and rotation for every manipulated joint either by hand or by mathematical methods like *inverse kinematics*(IK)[1]. The missing in-between frames are generated automatically by an animation engine by interpolating between two consecutive key poses.

Motion capture means the process of recording actors performing gestures and motions. The recorded informations are used to generate the animation data (again rotation and position of joints). Frame by frame the virtual skeleton is specified by recorded data.

Both methods use animation data in terms of 3D rotations and translations for every joint. Specifying directly this data by hand would mean a lot of work even for every key frame. So additional methods and algorithms are used to reduce the amount of data and raise the level of gesture description.

The highest level to reach would be just to formulate imperatives like "Drink a coffee". But for most character animation engines it is not possible to transform such phrases to the low-level data needed. It is not possible to transform the intention of "drinking something" to a plan of several actions (crab the cup, raise it to the moth), and further

[1] Using IK it is sufficient to specify a target point in space and a joint that should reach this point (see 2.2.1)

to skeleton postures. The user has do define a more concrete behavior for its character, there are several languages settled all on different levels between both extrema.

Human gestures are highly complex, every gesture is influenced by many several aspects, depending from physical constitution, mood and situation of the character. Embodied character animation engines try to generate convincing and natural motion that is influenced by theses expressivity aspects. The animation of different characters or the simulation of divers moods and feelings implies slightly chances in the low-level animation data.

With respect to usability defining and adapting a complete gesture to different characters, a bridge between the low level data to a more conceptual and user-friendly description simplifies this adaptation. A set of intuitive and understandable parameters provides the user a tool to tune the expressivity of the gesture. A general setting for a complete gesture avoids adjusting key frame by key frame.

The Embodied Agents Behavior Realizer (EMBR) tool is a real time animation engine for interactive embodied agents developed by the Embodied Agents Research Group at the DFKI. EMBR produces character animations based on the EMBR script language (EMBRScript). EMBRScript provides a low-level control on key frame level and gives the user additionally the possibility to add finer and more detailed animation definitions when needed.

EMBRScript provides a control layer to specify motion and behavior of the character, but it is not possible to tune expressivity by meaningful and understandable parameters. In my Bachelor Thesis I will develop a set of such qualitative parameters that allow the user to adapt a neutral gesture specified in a EMBRScript document to the desired appearance and situation. Different setups should result in believable and realistic variations of the gesture.

The set of applicable modifications should be able to generate a wide range of unique styles and to emphasis important gestures. The user has to get a clear idea what changes a parameter applies to the neutral gesture and how he can design the desired style.

1.2. Goal Description

Within my bachelor thesis I will enrich EMBR by a compact and intuitive set of qualitative parameters. These parameter should allow the user to tweak the expressivity of a gesture and the characteristics of the virtual agent performing.

To reach this, I have to accomplish the following goals:

- Get an overview of similar work for embodied agents and compare how expressivity is generated and controlled in these models.
- Develop a set of qualitative parameters, easy to understand and to use. The parameters should cover all aspects of expressivity and lead to believable modifications.
- Enrich EMBRScript giving the user the possibility to use parameters, I need to extend the language and introduce the definition of the parameters in EMBRScript.

- Decide where the expressivity modifications occur, by re-adjusting the input script, in a "pre-processing" step of the input data, inside the animation process or as a post-processing of skeleton postures.
- Implement all necessary classes and functions for the EMBR engine, so that the parameters influence the motion generation process.
- Demonstrate that the use of parameters leads to good results for a test suite of motion

2. Background

2.1. Related Work

In this chapter I give an overview of related topics, what means are used to generate expressivity in animation films and other agent animation engines. I introduce different languages used to describe actions performed by virtual agents.

2.1.1. Principles of Traditional Animation

Already in 2D hand drawn animations drawers thought about how they can focus the audience on important points of action, or how to express different styles of motion (e.g. forces, weights and). In Principles of Traditional Animation [12] Lasseter describes a set of principles used to create life full and human like characters and explains how to apply them to 3D computer animation:

- *Squash and Stretch* - Moving objects causes the shape to squash and stretch, it expresses rigidity and mass of an object
- *Timing* - The number of frames between two key points defines weight and size of the object.
- *Anticipation* - The preparation for the next action.
- *Staging* - The audience is focussed on a single idea of the animation.
- *Follow Through and Overlapping Action* - Finishing one motion and the transition to the next action.
- *Straight Ahead Action and Pose-To-Pose Action* - Two different methods of motion planning.
- *Slow In and Out* - Density of in-between frames before and after key points
- *Arcs* - The path of the motion is described by arcs to archive natural movements.
- *Exaggeration* - The design, shape and action of characters can highlight the crucial idea of the animation.
- *Secondary action* - The action of an object caused by another action.
- *Appeal* - Design and action of objects attract the attention of the audience

The transfer of these principles to modern character animation yields to ideas how expressivity can be modified by animation engines. Current engines use these methods to influence the appearance of motion and to shape characters. This paper inspired my work since it demonstrates and explains several techniques where to influence the animation process and with what means. Principles I realized for EMBR are the *Anticipation*,*Follow Through* for emphasizing the power and meaning of motions. My implementation of *Arcs* and *Timing* allows the user to choose between straight motion paths and soft curves, between fast, jerky and slow, smooth movements.

2.1.2. The Behavior Markup Language

The idea behind the *Behavior Markup Language* (BML), described in [11] and [17] was to unify the interface of various models for Embodied Conversational Agents (ECA). Nearly every available animation engine has its own behavior description language, and above these languages are several layers of description between the low level input data for the engine and a clear high level behavior description by the user.

Since BML is designed to be a uniform description language, Kopp et al. introduced as well the SAIBA framework (Situation, Agent, Intention Behavior, Animation) for the generation of natural multimodal animation. This framework splits the generation of multimodal behavior in three modules: (1) formulation of the intent as highest layer (2) planning of the behavior, the multimodal realization of the intent from (1), and (3)the realization of the planned behavior. The BML settles between the last two modules and describes (non)verbal behavior independent from the particular way of animation engine used.

BML is an XML based language that specifies a sequence of behaviors that should be realized by the underlying animation engine. These behavior tags coordinate gesture, gaze, speech, head and body motion by combining tags to blocks, inside such a block. Figure 2.4 shows a simple behavior block. Behaviors are split into several categories,

```
<bml>
   <head id="head1"  type="shake" amount="0.5"/>
   <face id="face1" type="smile" amount="1.0"/>
   <torso id="torso1" type="bend" amount="0.3/>
</bml>
```

Figure 2.1.: An example of a simple BML block

inside these categories it is possible to further define several attributes (either a sub-type attribute or several type-specific attributes. These attributes describe visual appearance and movement dynamics of the behavior to add expressivity to the behavior. Table 2.1 describes the main behavior categories of BML.

Temporal and conditional synchronization points allow temporal alignment of the particular behaviors. Conditions and events can be used to trigger behaviors, inserting a

BML Element	Description
`<head>`	Head movement (not including the eyes), like nodding shaking orienting to a specified angle.
`<torso>`	Orientation and movement of spine and shoulder (e.g bend, straight).
`<face>`	Facial expressions, subtypes are eyebrow, eyelid, mouth.
`<gaze>`	Movement of the eyes, neck and head direction, defines the direction the character is looking in.
`<body>`	Full body motions, types include overall orientation, position and posture.
`<legs>`	Movements dedicated to the lower part of the body, pelvis, hip, legs including knees, toes and angles.
`<gesture>`	Coordinated movements of arms and hands, like pointing, reaching, emphasizing beats, depicting or signaling.
`<speech>`	Verbal and paraverbal behavior, including the words to be spoken and pauses.
`<lips>`	Control of lip shapes, visualizing the spoken phonemes

Table 2.1.: Main behavior categories of the Behavior Markup Language.

`<wait>` tag suspends the behavior until a condition is met or an event arrives, additional time-out definition to fall-back can be specified to return to normal behavior after a certain time span. Temporal points split a behavior into six parts, yielding seven synchronization points: Start, Ready, Stroke-start, Stroke-end, Relax, End (see Fig. 2.2

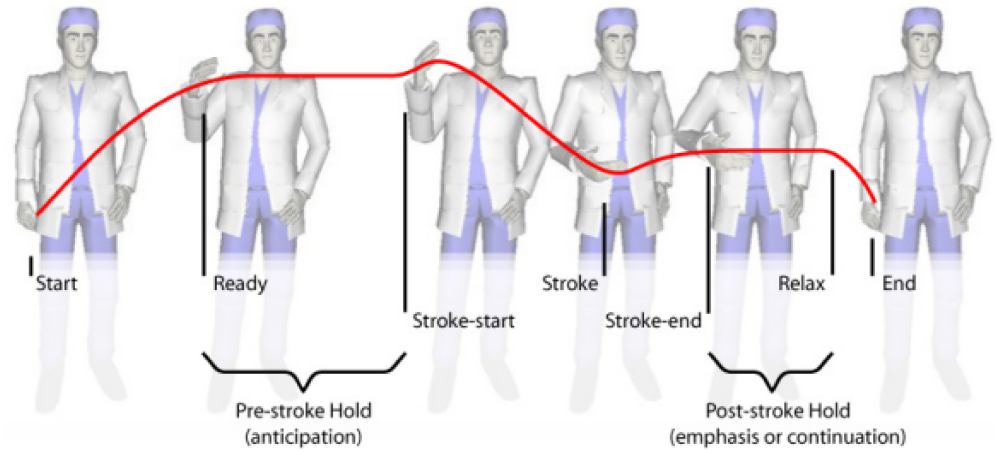

Figure 2.2.: The synchronization points of a BML gesture (taken form [11])

BML allows also users without much experience in character animation or programming to express multimodal behavior. But a BML description of a gesture is not a

concrete specified motion understand by animation engines. The description includes tags like "bend", "sit", "point", the underlying engine must be able to interpret this tags, or a parser must be used to transform it into low level input data.

EMBRScript is settled below the BML approach, it also allows to invoke prerecorded or predefined motion sequences (e.g. autonomous behavior like breathing or pose targets like hand shapes). To archive for example a pointing gesture the user has to specify target points for the hand, its orientation and a index finger hand shape. But the access to target coordinates gives the user the possibility to realize every pose he wants.

2.1.3. GESTYLE language

A markup language for ECAs considering style annotations is the GESTYLE language developed by Ruttkay and Noot [18]. GESTYLE allows to describe gestures involving face, arms and hands together with spoken text.

It uses a hierarchical structure to specify complex gestures. On the lowest, atomic level, so-called *basic gesture* represent the smallest part of a gesture. Basic gestures are for example nod, head-shaking, one handed beats. A *gesture expression* combines several basic gestures, it supports parallel or sequential definitions. A *meaning* has a label on a higher level, like "be sad", "take turn in a conversation" and is mapped to one or more gesture expression representing. All mappings are stored in a *style dictionary*, every style has its own dictionary.

Additional elements of GESTYLE are *manner definitions* specifying motion characteristics (e.g. smooth, angular) and *modality usages* denoting preferences for the choosing of certain modalities (e.g. less or more head movement). The concrete style of an ECA is finally specified in a *style declaration*. A style is declared by a combination of several style dictionaries (e.g male + German + young + athletic) together with a manner definition and a modality usage element, both optional. So it is possible to combine the same text and meaning tags with different style declarations yielding to distinct gestures.

After reading a style declaration for a ECA, a combined style dictionary is build. This dictionary contains all meaning mappings from every style. If meanings appear in more then one dictionary, a weight specified together with the style declaration decides what meaning is chosen.

Modifier tags (manner definitions and modality usage) can affect the modified style dictionary dynamically while parsing the gesture meanings. Either the situation or mood of the ECA can change through a conversation, or circumstance could restrict usage of an arm (because it is carrying an object).

The meaning are then resolved to gesture expression and further to sequences of basic gestures. This data is then used to generate low-level animation data for an underlying animation system.

The GESTYLE language gives the possibility to arrange and combine gestures out of small gesture bits. The definition of style dictionaries organizes them under stylistic aspects. Style declarations describe the characteristic of an ECA and choose the corresponding gestures constructs. For EMBR I have also to find a way to specify the characteristic of agents in an reasonable way. The change of such a definition should

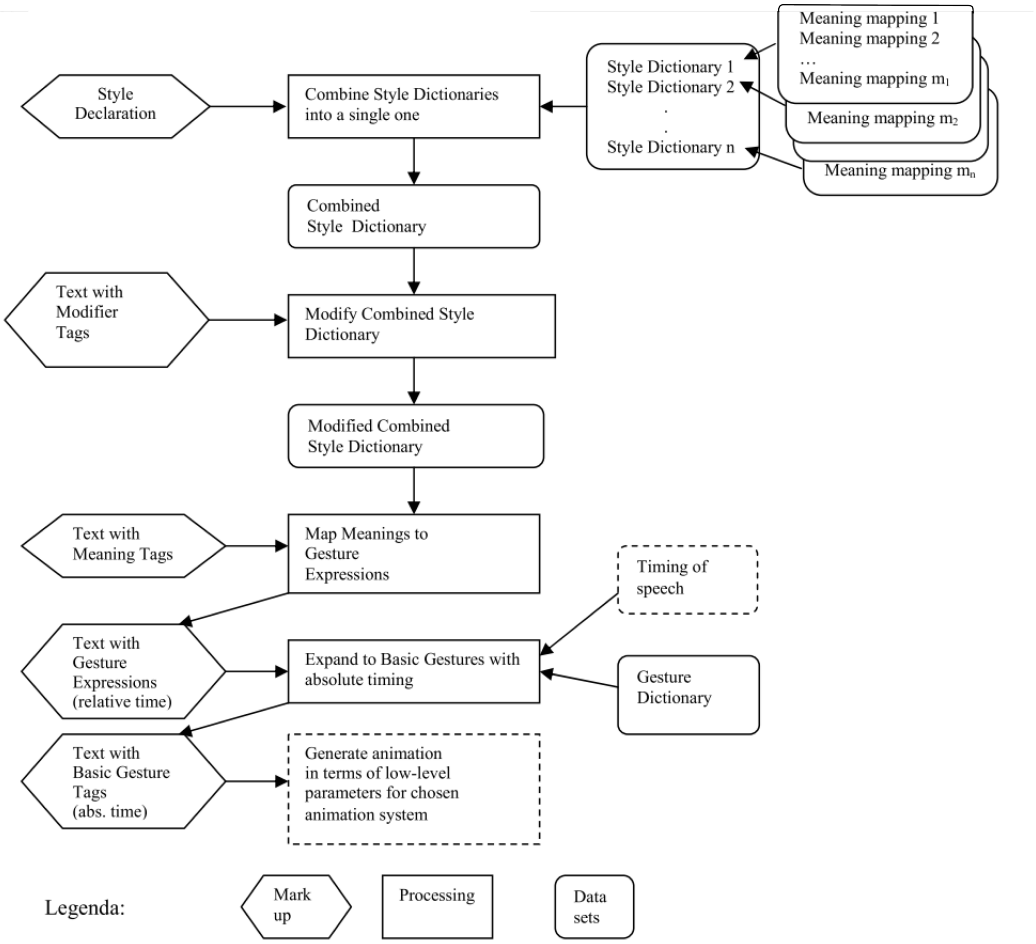

Figure 2.3.: Interpretation process of GESTYLE (taken form [18])

then be sufficient to adapt the gesture to the new conditions.

2.1.4. Parameter-based Character Animation

In this section I introduce the two papers that mainly influenced my work. Both models are so called parameter-based animation engines, which means that the user has the possibility to influence the animation process by a set of parameters, controlling the expressivity of the character.

Approach by Hartmann, Mancini and Pelachaud

A model developed by Hartmann, Mancini and Pelachaud [6], provides a behavior realizer, from now an called HPM which synthesis embodied conversational agent's gestures based on a set of input parameters. HPM consists of different tools to produce animations, it interprets markup texts in APML (APML, a Markup Language for Believable

Behavior Generation) with annotations [4].

```
<performative type="announce">
  <rheme>
    Whatever works for
    <emphasis x-pitchaccent="Hstar" deictic="you" intensity="0.4">you</emphasis>
    <boundary type="LH"/>
    thats for
    <emphasis x-pitchaccent="LplusHstar" intensity="0.4">you</emphasis>
    <boundary type="LL"/>
  </rheme>
</performative>
```

Figure 2.4.: Example for an APML script fragment

The realizer is articulated around the following components :

- *TurnPlanner*: extracts tags from the APML script
- *GesturePlanner*: selects appropriate gesture prototypes to match with the text and schedules rest phases
- *MotorPlanner*: calculates the necessary angles and timing for key frames
- *Interpolators*: generate in-betweens, according to MotorPlanner output

Attribute	Description	Implementation
Overall Activation	quantity of movement during the animation	selects a subset of the animation library, influences choice of abstract gestures and duration of rest phases
Spatial Extent	amount of space used by movements	influences the IK targets in the MotorPlanner and leads to different angles
Temporal Extent	duration of motion segment	modifies frame timing in the GesturePlanner
Fluidity	degree of continuity and smoothness	modifies the control of the interpolation splines
Power	dynamic properties of the movement	affects hand shape and interpolation splines parameters
Repetition	tendency to repeat specific movements	repetition of (arm-)strokes into a single motion

Table 2.2.: Expressive parameters provided by the HPM engine

The attributes shown in Table 2.2 are combined and mapped to qualitative labels (e.g. abrupt, smooth). Assigning one of these labels to a motion segment modifies

parameter settings to the corresponding values and affects the gesture computation in the HPM engine. The spatial modification scales the gestures according to a reference point lying at the agent's solar plexus. It is possible to scale the gesture by a different factor in every direction (horizontal, vertical and frontal). For the new target point a IK algorithm computes the modified posture. I also implemented this for EMBR but I decided to find another method since I realized that it is really necessary to give the user the possibility to tweak all three dimensions. My goal was it to come up with a simple parameter, the method introduced in the HPM model must be set up with care since otherwise strokes or deictic gesture could be destroyed. Figure 2.5 shows the result for the maximal spatial settings.

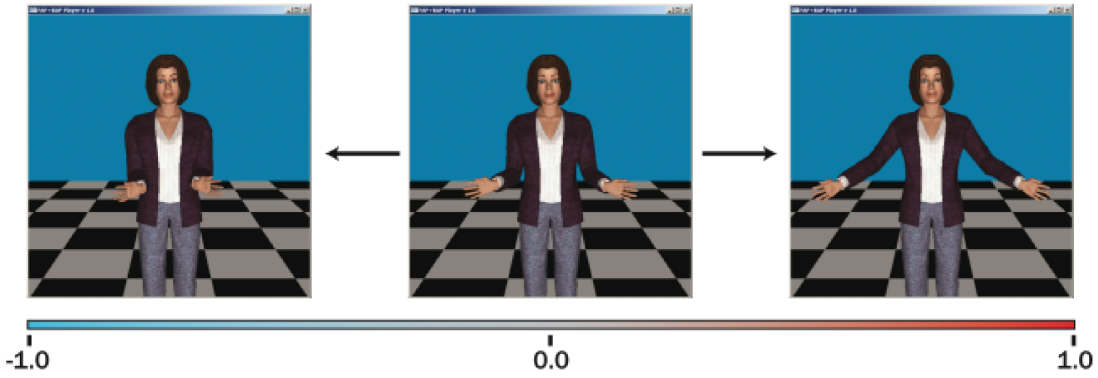

Figure 2.5.: Three settings of the spatial parameter for the HPM model: (left) minimal extent -1.0, (middle) neutral extent 0.0, and (right) maximal extent 1.0

The temporal modification follows the theory of Fitts' Law[1]. With the help of a modified version of this law, the duration to complete each segment from key point to key point is computed. The temporal extent parameter influences the calculation, for positive values the motion is accelerated by shorting the duration, otherwise for negative values the duration is increased. My method for EMBR is quiet the same, but I refer the new calculated time not to Fitts' Law but (since EMBR provides a global timing) to the neutral duration of the gesture.

The parameters power and fluidity are mainly realized by varying the interpolation curvature of the interpolators. Within the interpolators Kochanek-Bartels splines[2] [10] are used to compute in-between postures. These splines offer three parameters to tune the interpolation path: (1)tension, (2)bias and (3)continuity.

Fluidity controls parameter (3) which results either in edges or smooth curves in the key coordinates. It also acts on the retraction phrases and pauses between two gestures.

[1]Fitts' law predicts the time required to move a hand (or on the screen the mouse pointer) to a target area. The time needed to reach the area depends from its size and the distance of the starting point to it.

[2]A further description can be found in the description of my quaternion interpolation, since I use a modified version of Kochanek-Bartels splines

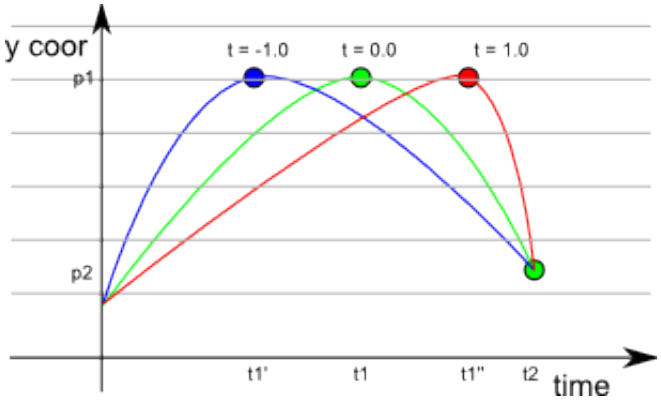

Figure 2.6.: The diagram shows the wrist path over time, the temporal extent settings influence the duration from p1 to p2. The end point remains fixed since it could be synchronized with other animation content (e.g. speech)

The power parameter controls tension and bias of the spline, tension varies the curve between straight linear paths and wide curves, the bias defines the passage of the key point, so it is possible to generate overshoot with the help of bias. Additional power features are, that for motions expressing high power, the hand contracts to a fist shape.

I also use Kochanek-Bartels splines for power and fluidity modification since they provide an easy way to influence in-between animation. In contrast to the HPM model, I interpolate the orientation quaternions and not the key poses, but the effects of the parameters tension, bias and continuity are the same.

EMOTE

The second approach for parameterized gesture modification I introduce is the EMOTE Model for Effort and Shape by Chi et al.[2]. It implements Laban principles from choreographic annotation in order to define a set of expressivity parameters. The Laban Movement Analysis [13] has five main components to describe the quality of body motion. The EMOTE model picks out the *effort* and *shape* components and uses their motion factors as attributes, defined between two opposite extremes. A overview is given in Table 2.3.

The Laban notation describes not the gesture itself but the qualitative aspects of it. So to create expressive surfacing gestures two inputs to EMOTE are necessary: First one is the parameter set for the qualitative attributes and second is a gesture. This gesture gets altered according to the (new) input values for the qualitative characteristic and consists of key frame data either from motion capture or IK.

Therefore the EMOTE approach post-processes an existing motion given by key poses, according to a parameter set, analytical IK-algorithms [16] compute elbow swivel angles and joint positions, if not specified in the given data. These original positions and rotations serve as starting point to re-calculate modified key postures.

Shape attributes influence arm and torso posture, joint orientations are changed to

Attribute	Description	Affecting gesture by
effort: space (*direct, indirect*)	attention to surrounding (focused or not)	choose of affected angle and path curvature
weight (*light, strong*)	sense of the impact of movement(without impact or powerful)	influence amount of wrist rotation and extension, timing and acceleration calculation
time (*sustained, sudden*)	lack or sense of urgency	number of frames between key points and key-frame-to-time mapping
flow (*free, bound*)	body tension and control	trajectory of path, affected joints, time and space use
shape: horizontal (*spreading, enclosing*)	open or closed arms	key point and angle modification in x,y-direction: adjusting angles of neck, spine, clavicle
vertical (*rising, sinking*)	affinity with weight, reaching upwards or stamping down	key point and angle modification in y,z-direction: angel of clavicles, pelvis rotation
sagittal (*advancing, retreating*)	affinity with time, reaching hands or avoiding a push	key point and angle modification in x,z- direction,angles of spine, neck and pelvis rotation

Table 2.3.: Attributes for the shape and effort elements of the EMOTE model

express different stance for the body. This rotations yield to either more open or closed arms (for the horizontal aspect of shape).

Whereas shape controls the movements by changing key poses, the effort parameters affect the execution of the gestures produced by the engine. Parameters define interpolation and add extra "flourishes": small changes in wrist bending, arm twist and displacement magnitudes. Same as the HPM model described above, a Kochanek-Bartels interpolation with parameters to adjust the curvature is used. Additionally it is possible to specify what joint should be used to perform the interpolation, the user can choice between end-effector position, joint rotations and elbow position.

The temporal modifications offer several parameters to invoke special temporal behavior. For normal motion a start and end in a rest is assumed, every movement has an acceleration and deceleration phase. In the timing functions the start and end points

of these phases are shifted according to the parameter input. Here the EMOTE approach uses temporal interpolation to create the traditional principle of anticipation and overshoot, backward movement in time results in backward joint motion in space. The same effects can be realized in EMBR be using a time warp function. Contrarily to anticipation and overshoot generated by the bias parameter of spatial interpolation, the joint traces exactly the same path as in the forward movement. Further it is possible to specify the overall number of in-between frames resulting in faster or slower motions.

2.1.5. Data driven approaches to Motion

The following models generate their animation modification by deducing expressivity changes from existing data, so I call them data driven approaches. For every style the engine is able to apply on a neutral motion, a sample motion is needed to gain the corresponding style setup.

Style Machines

The model presented by Matthew Brand and Aaron Hertzmann in [1] intra- and extrapolates given motion capture data. The given data set is used to synthesize new motion and resynthesize an existing motion according to a different style. Their framework automates editing and analyzing motion capture. In comparison with the approaches presented in Section 2.1.4, style machines needs no kinematic or dynamic model of the character to produce animations.

The process of animation synthesis has two steps, first a learning process, second the synthesizing process. In the learning phase existing data is analyzed and motions are split into small pieces (motion primitives, short sequences of frames), and a state-space representation of these motion primitives is created. They use a hidden Markov model (HHM) and add a style variable v that allows to input several styles to the learning process. The generated HHM contains all possible paths from one motion primitive to another, motion sequences are combined by loops in the HHM (see Figure 2.7). The HHM also represents the duration of a motion in different styles since a slow action yields to more frames and longer sub-sequences in the state-space model.

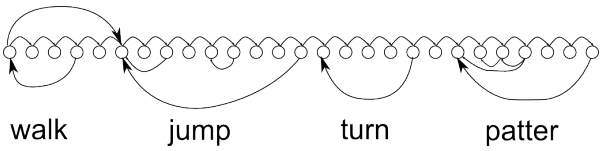

Figure 2.7.: State machine learned from input data, original motion is from left to right. High probability arcs show cycles of motion primitives.

After all input data is analyzed, split and transitions are computed, in the second phase, new motion is synthesized. This is done by searching and following a path on the HHM. The model computes to a motion path and a new style variable the maximum-

likelihood path in the HHM. The adaption of a new style parameter influences also the velocities of the performed motion primitives because its speed varies for different styles.

Using only motion capture and other pre-fabricated data to feed an animation engine limits the output animation of a "permutation of frames" from the input. It is possible either to replay the original sequence in a new style or to rearrange motion primitives according to the HHM. The interpolation step may produce some slightly new animations by blending between two frames postures that have not been connected in the original data, but it is not possible to create postures that are not included in the input data.

Although EMBR also supports the playback of pre-fabricated data, EMBR is rather supposed to generate new animation sequences then playback stored animation data. To use a self learning method for gesture modification would require a huge amount of data, the possibility to configure freely every desired key pose in EMBR makes it impossible to gain modification rules by data analysis in the way style machines does it.

Data-Driven Style Transformation for Gesturing Embodied Agents

In contrary to the parameter based approaches presented in Section 2.1.4 the style transformation method presented by Heloir et al. in [8] modifies existing motion data (either key-frame or motion capture data) according to transformation parameters that have been extracted from other existing captured animation sequence. In their paper they provide an example where different styles are transferred from a weather forecast animation (W neutral, angry, weary) to a train station announcer (T neutral).

Analogous to the style machine approach, style transformation can be decomposed into two stages: First stage, the transformation model is feed with motion data representing different styles. Second, an arbitrary motion sequence is adapted according to the styles and the corresponding modification setup (see Figure 2.8).

The learning phase analyzes sequence sets, a set consists of the same gesture performed in different styles. From this set the model learns style modifications for the given input styles. The transformation phase applies spatial modifications based on the principle component analysis (PCA), it is similar to the method by Egges et al. presented in [5]. Motion segments are re-timed by time-deformation profiles that consider timing of the learned style models.

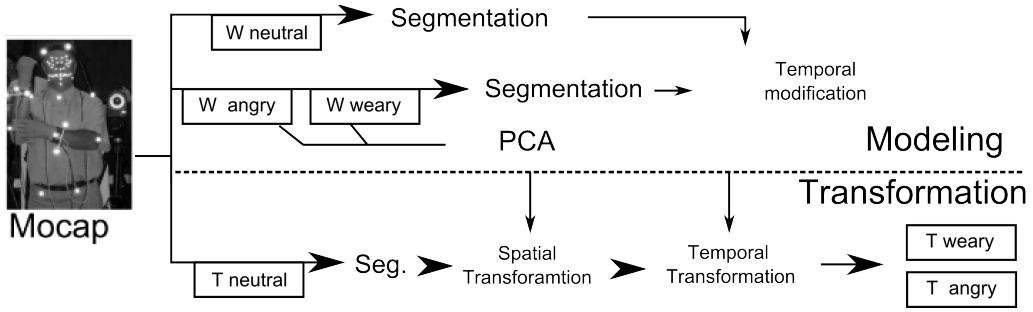

Figure 2.8.: Overview of the style transformation process

Both data-driven approaches, style machines and style transformation, differ in, that for the first, the motion data set is fixed, where for the second, it is possible to transform any arbitrary motion sequence. The style machine approach builds a close system out of the input data and all output is taken from that. The style transformation model uses the input data just to set up the modification models. The data that can get transformed to a learned style are independent from this set. But for every style configuration the system should be able to transfer, it needs a comparable motion in a neutral setting and in the style to learn.

This approach allows to apply style modification to arbitrary gesture sequence but it also works on recorded motion data. For EMBR, most gestures are not completely specified in terms of joint orientations but in constraints for every key pose. So following the style transformation principle would mean to first compute the neutral motion from the script and then transform this motion according to the style models. Therefor for EMBR it may be easier or more efficient to modify already the constraints instead of using post-processing methods.

2.2. Embodied Agents Behavior Realizer

In this section I introduce some techniques that are essential when talking about 3d computer animation. Then I present the Embodied Agents Behavior Realizer(EMBR) and the corresponding language EMBRScript. I will develop and implement my qualitative parameters for this engine, therefor I give an overview what EMBR was before I started my work.

2.2.1. Character Animation Techniques

Inverse Kinematics

Virtual characters are animated by manipulating a skeleton consisting of a chain of joints. All joints are rigid and have a fixed length. The joints represent the bones of our virtual skeleton. At the connection points every joint has a number of degrees of freedom (DOF), limiting the directions of movement in this joint(e.g a shoulder has 3 DOF, an elbow only one). To define a posture for a skeleton, for every affected joint, its new translation and rotation is specified. But its is complicated to directly estimate the values for a desired pose (that is called *forward kinematics*, starting from the root, every joint is rotated until the last joint of the kinematic chain is reached).

But more often the goal is to find rotations yielding to a posture where a certain joint (end-effector) reaches a target point in space. *Inverse kinematics* (IK) algorithms provide the user the possibility to calculate these values from target coordinates. The algorithm underlies certain constraints, rotation angles for joints could be limited to guarantee the generation of natural postures, the user could also demand a particular rotation of some joints (e.g. the swivel angel between the body and the elbow, palm orientation and hand shape). IK algorithms are iterative and stop if the target is reached

or the displacement error is sufficient small. It as also possible, that the target is not reachable caused by the structure of the kinematic chain.

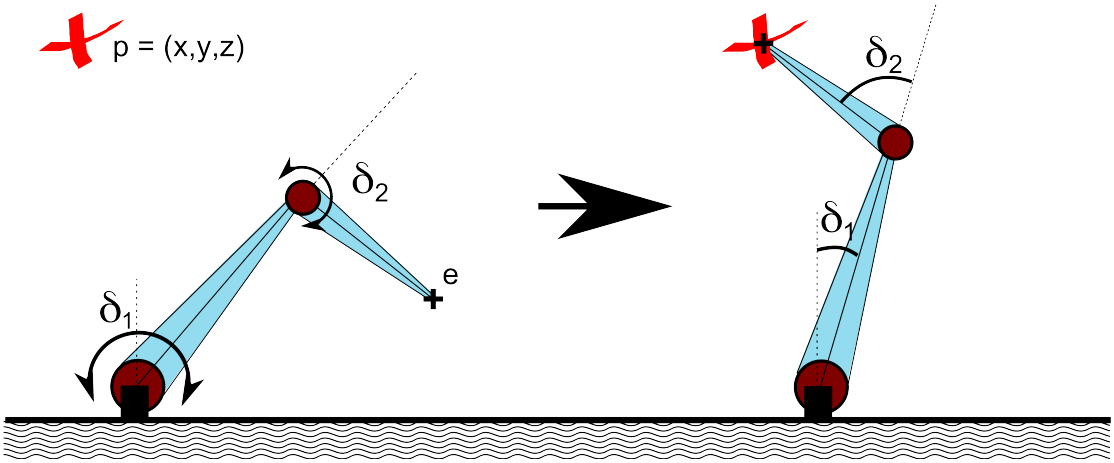

Figure 2.9.: Example for a simple kinematic chain with 2 joints, each with 1 DOF. The end-effector e should reach target p. The inverse kinematic algorithm computes both orientations δ_1 and δ_2 such that e lies in p.

Key Frames

An animation is generated by displaying several single pictures (frames) per second, where the moving objects change their position from frame to frame a tiny bit. Showing at least 12 frames per second creates the illusion of motion. A gesture sequence of one minute with a frame rate of 24 (what is the normal frame rate of a movie) already needs 1440 frames. It is not possible for a single user to define every particular frame posture by hand.

Animation with *key frames* means that the animator concentrates on the frames that contain the important and expressive postures for a gesture. So it is possible to reduce the amount of frames the user has to define significantly (every straight movement can be described by two key frames, the start and the end posture).

The missing in-between frames are computed by interpolation algorithm depending from at least two key frames. Every key frame has a global time point when the posture has to be reached by the character.

In-between Interpolation

Interpolation means that only a start and end posture is known and the corresponding timing, when every pose should be assumed. The interpolation engine calculates the in-between frames according to the current time point. There are several methods for calculating these in-betweens, the simplest approach is linear interpolation, which uses a straight line from start to end pose to translate the skeleton on.

If we want to know the position f_x of a joint at time x, we compute f with the help of the previous f_{prev} and the next position f_{next}:

$$f(x) = f_0 \frac{f_1 - f_2}{x_1 - x_0} * (x - x_0)$$

More complex cubic interpolation algorithms use more then two consecutive key frames. With the help of splines it is possible to generate interpolation curves instead of straight paths yielding to a more natural behavior of the character. Additional parameters allow to control that curvature and adapt it to the desired expressivity. In Section 4.2 I describe the parameterized interpolation method I implemented for the EMBR realizer.

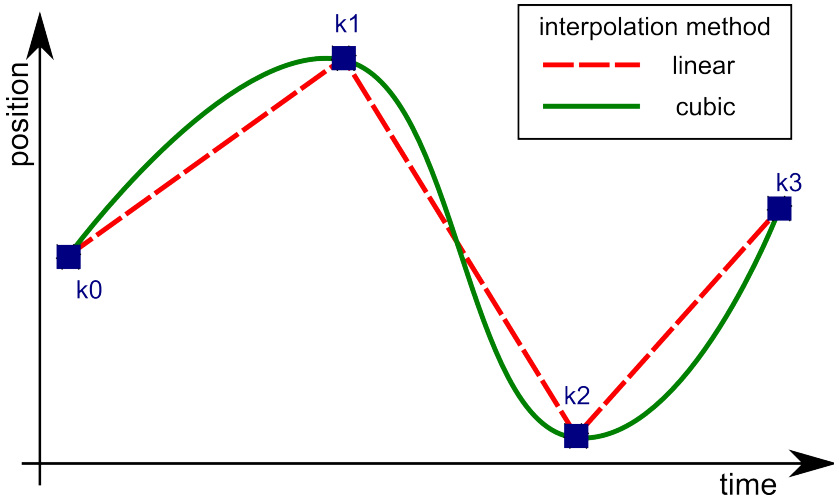

Figure 2.10.: Linear and cubic interpolation paths for joint position defined by key points $k0, k1, k2, k4$.

2.2.2. EMBRScript

A gesture for embodied agents can be described in various levels of detail. An abstract behavior language like BML (see Section 2.1.2) gives a good layer to express what the gesture looks like, it is possible to specify timing and synchronization of different gestures. For their realtime character animation engine EMBR [7], Heloir and Kipp introduce a new layer between the behavior description and the animation engine and call this layer *animation layer*. This layer hides the specifications of the realizer engine and allows the user to control the character by defining a series of *key poses*, using *global timing* and *absolute space*. The structure of the gesture description should be very simple without complicated and deeply nested specification items. On this layer they developed *EMBRScript*.

Every gesture in EMBRScript consists either of invoking a pre-fabricated animation clip with a global start time or of a sequence of *key poses*, each key poses has also a

global time point and describes the state of the character at this time value. An optional hold value causes the character to rest in this state for the given amount of time.

The user has the possibility to specify various *constraints* for every key pose, these constraints allow the use of four different methods of animation: skeleton manipulation (by Inverse kinematics or pre-calculated forward kinematic data (e.g. from 3D animation tools or motion capture data)), morph targets, shader and autonomous behavior. Here is an overview of the different constraint types and their use:

1. POSITION_CONSTRAINT: defines position of one specific joint by target coordinates. The user decides which body part (kinematic chain) should be used to reach the target. It is possible to add an extra offset to avoid collisions / overlapping with other limbs.

2. ORIENTATION_CONSTRAINT: Defines orientation for a single joint, the user choose which normal (x, y or z) he wants to define and assigns a target direction.

3. SWIVEL_CONSTRAINT: A swivel constraint can be used to limit the angle to a given value

4. LOOK_AT_CONSTRAINT: Determines gaze direction of the character.

5. POSE_TARGET: The user can assign different predefined hand shape models to a pose (such as open hands, a raise index finger or a fist) by referencing the corresponding pose target.

6. MORPH_TAGET: By morphing target, it is possible to influence the facial expression (smile, frowning).

7. SHADER_TARGET: Influence also facial appearance, can generate a blushing or paling of the skin.

Figure 2.11 show a short example for an EMBRScript file.

Gestures in EMBRScript are defined as a sequence of several key poses with a global starting time, a reference to a character which should perform the gesture. It is possible to split two-handed gestures into two separate key pose sequences, one for the left and one for the right hand. The reason for this is, that it is so easier to switch between two-handed and one-handed gestures. EMBR uses several channels to compute postures of particular body groups and blends them in the blender module.

EMBRScript already allows a lot of user control specifying gestures for embodied agents, but there are no parameters varying the overall expressivity of a gesture. It is already possible to affect timing by the means of the *TIME_WARP* functionality. The time warp influences the mapping between the current global time value and the time value used to interpolate two postures. If no time warp is defined, both values are the same and the motion is linear in time. The time warp controls temporal dynamics by either applying a tangent or exponential function to the global time value before interpolating. Fig. 2.12 demonstrate how *ease in* and/or *ease out* behavior can be modeled.

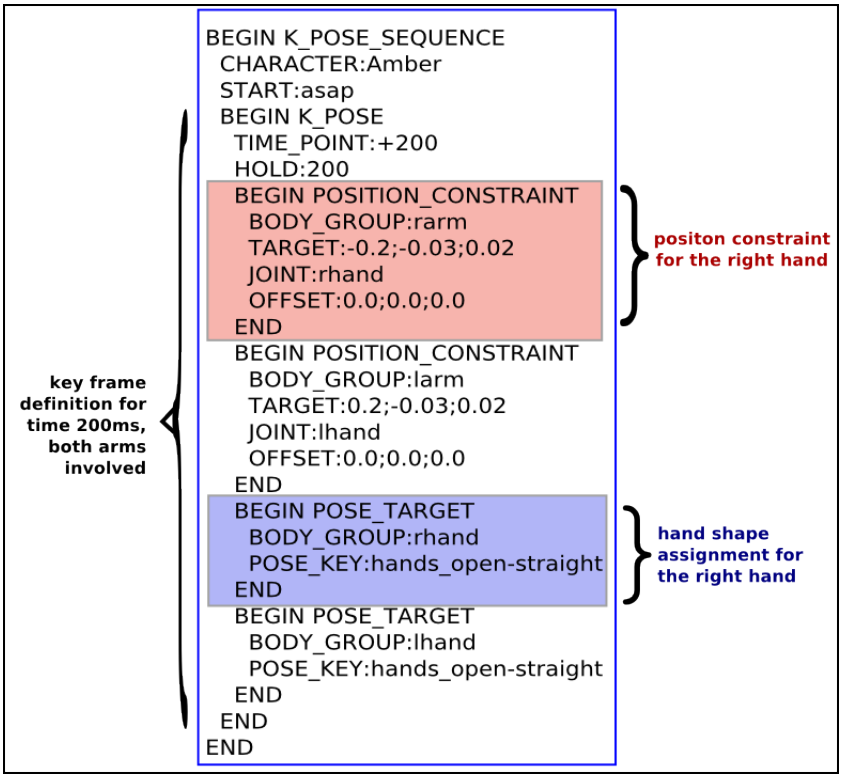

Figure 2.11.: Example for a EMBRScript, consisting only of one key pose, expressing a relaxed pose with both arms besides the body

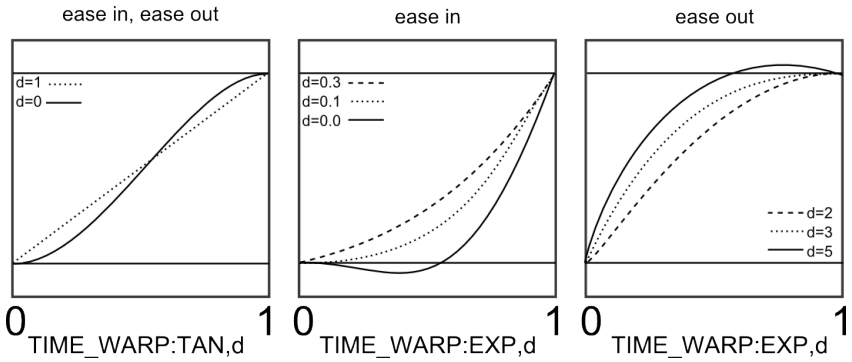

Figure 2.12.: Time warp profiles for ease in and/or ease out, either a tangent or exponential function is used, d models steepness.

But if the user wants to change expressivity of a gesture he has to work trough every key pose and also find a meaningful modification either of time point, coordinates or both to adapt every key posture. So within my thesis I will present a set of qualitative parameters, allowing to tune expressivity of a gesture, making it easier for the user

to create different characters or emphasize important parts of an animation sequence. Using qualitative parameters it should be sufficient to specify the modifications once and then only referring to these definitions from every affected key pose, instead of shifting target coordinates and time points by hand.

2.2.3. EMBR Engine

EMBR is a modular engine reading EMBRScript documents and generating corresponding embodied character animation in realtime. The engine calculates a skeleton posture for every time frame and sends it to an external renderer. The EMBR architecture consists of three parts:

1. **Motion factory:** In the animation pipeline, gestures are composed of various *motion segments*. Every *motion segment* represents an animation for one skeleton part and has an absolute start and end time. A motion segment manages the motion generation process, it keeps an instance of an specialized *actuator* that handles the different input formats (recorded animations, skeleton interpolation, morph targets, shader).

2. **Scheduler** The scheduler manages all motion segments created by the motion factory. It rearranges the segments in time and sends active segments (current time is between the start and stop time of the segment) to the pose blender, it adds weights to every segment allowing to slowly fade out from a segment (decreasing weight from 1 to 0). Segments that contain position constrains are marked with a `priority` tag. Expired segments are removed from the segment list.

3. **Pose blender** All segments that are passed to the pose blender get merged in the pose blender. This is done by weighted linear interpolation between the segments. Segments tagged with the `priority` tag override other segments completely to guarantee that the resulting pose from this segment is reached unchanged.

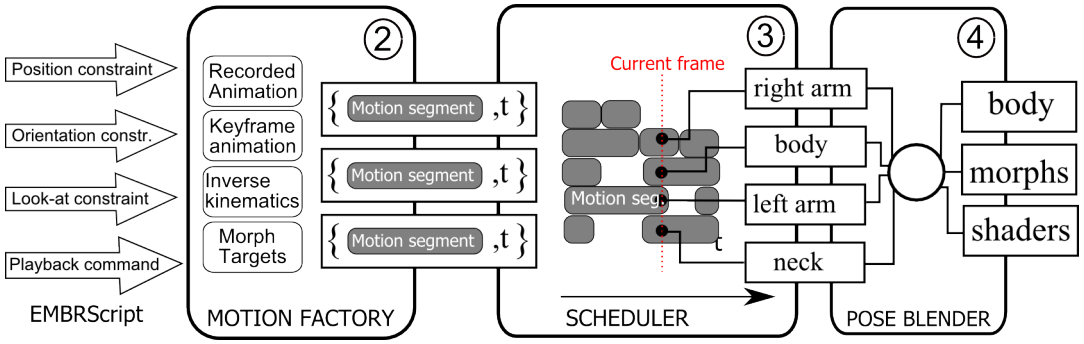

Figure 2.13.: Illustration of the modular concept of EMBR, taken from [7]

Figure 2.13 gives an overview, EMBR first parses the EMBRScript input document, containing commands to load data, set shader and position constraints. The motion

factory reads the time stamps of the constraints, creates motion segments and rearranges them according to their time stamps.

Motion segments represent animations for a part of the skeleton (a cinematic chain or *body group*) for a certain time span. It contains an *actuator* which computes the corresponding animation, each type of input data has its own actuator type.

The motion segments are sent to the scheduler which checks at regular intervals what segments are active or expired, deletes every expired segment and sends the set of active segments to the pose blender. The scheduler manages the segments based on global timing, the time value can be warped by the *time_warp* functionality.

The pose blender merges all incoming segments, interpolates postures and solves conflicts. Since EMBR itself contains no renderer, the computed skeleton is used in an 3D renderer like Panda3d[3].

[3]http://panda3d.org

3. Concepts

In this section, I introduce my set of qualitative parameters for modifying gestures generated with EMBR. I give an explanation what influence every parameter has on a gesture and how it is realized in terms of motion data modification. To have a better control what the parameters should emphasis and modify I present the concept of nucleus to affect the most expressive part of gesture with the modifications.

3.1. Expressivity Parameter Set

The new high-level parameters should capture all aspects of expressivity and explain itself so that the user knows what parameter he has to change to archive the desired expression. There are several characteristics, defining the style of a gesture on a high level description:

- amount of space, used for gesture
- time needed to reach key poses
- accelerations of limbs
- pauses and holds within a gesture
- path between consecutive key points
- additional motion, caused by higher accelerations, speed and power

Looking at the two described parameter-based models EMOTE and HPM in Section 2.1.4, we see that they provide a similar set of parameters to cover these aspects:

I have chosen four of the parameters also used by HPM, but implemented them in a different way. Each of the attributes is float-valued and defined over the interval $[-1, 1]$, where the zero point corresponds to the actions our generic agent without expressivity control would perform.

Here are the four qualitative parameters for EMBR:

- Temporal Extent: How fast/slow a movement is.
- Spatial Extent: Amplitude of movement.
- Fluidity: smoothness and continuity of overall movement.
- Power: Expressing force and engagement of the movement.

Expressivity	HPM	EMOTE	Trad. Principle
space used for gesturing	Spatial Extent	Shape, Effort (space)	Squash and Stretch, Slow In and Slow Out
timing, accelerations	Temporal Extent	Effort (time, weight)	Timing
trajectory, smoothness	Fluidity	Effort (flow, space)	Arcs, Slow In and Slow Out
weight, force	Power	Effort (weight ,flow)	Anticipation, Follow Trough, Arcs

Table 3.1.: Traditional Principles and their realization in EMOTE and GRETA

These parameters give a clear idea what they will change, a negative setting reduces the aspect in comparison to the neutral motion (e.g -1.0 for the spatial extent reduces the gesture amplitude) and a positive one increases it (e.g 0.7 for temporal extent results in a longer and slower movement).

3.1.1. Temporal Extent

The temporal extent parameter p_{temp} specifies the duration of the several gesture parts from key pose to key pose. A negative setting reduces the duration, a positive value increases the amount of time needed to reach the target point.

In EMBRScript all key poses provide a global time value t at which the pose must be reached. To modify the duration of a motion from current pose to the next following, my implementation shifts these time points. It calculates a delay in a logarithmic dependency to the original duration $d = t_{next} - t_{current}$ defined in the EMBRScript file and adds it to the absolute time points of the remaining key poses of the gesture:

$$delay = log(d) * p_{temp} * 1.5 t_{next} = delay + t_{next} \tag{3.1}$$

The logarithm is uses to avoid a too big offset. As we can see in from Equation 3.1, for negative temporal extent value, the delay is also negative and following time points are preponed.

Figure 3.1 shows the influence of a negative setting for the temporal extent parameter. The illustration shows two consecutive gestures A and B. Gesture A consists of five different poses, $a0$ to $a4$, the temporal modification is applied to two poses, $a2$ and $a3$. Both belong to the *nucleus* (see Section 3.2) of gesture A. The value of p_{temp} is negative, which means that the duration of the motion from $a1$ to $a2$ and from $a2$ to $a3$ is shortened. That means that the absolute time points of key pose $a2$ and $a3$ are preponed by the computed *delay* from Equation 3.1. One can see, that for $a3$ and $a4$ both delays sum up.

It can also be seen, that the delay is only applied to key poses belonging to gesture A, while B gets not touched. This is caused by the architecture of EMBR, the motion

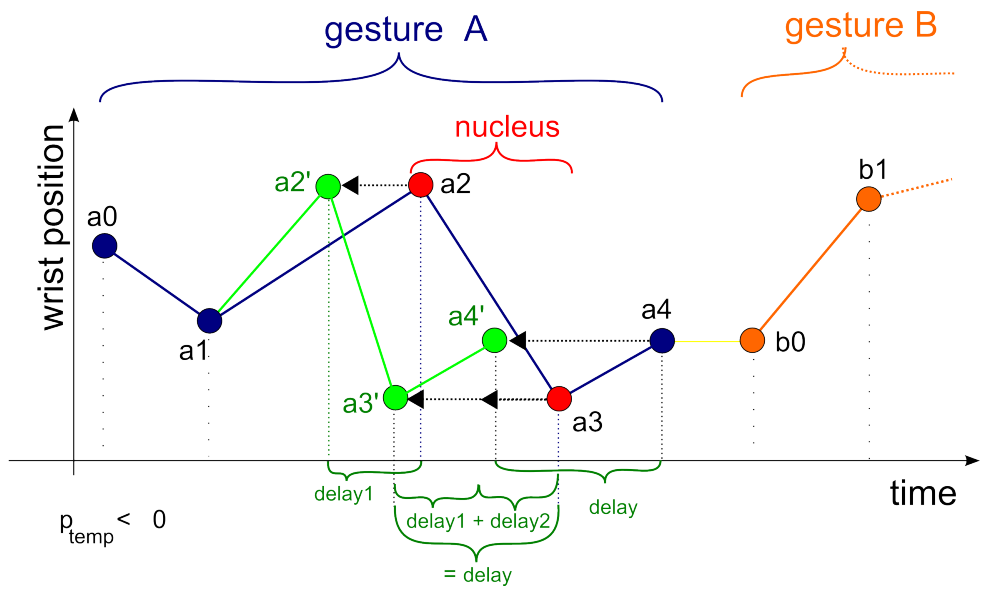

Figure 3.1.: Illustration of temporal modification for $p_{temp} < 0$, poses $a2$, $a3$ and $a4$ are preponed by the computed delay. Delays sum up towards the end of a gesture.

factory originally creates a separate motion segment for every pose specified in the EMBRScript document. At first, every motion from one key pose to another was separately stored in the motion segment pool and send to the EMBR pose blender when its absolute start time was reached.

To have the possibility to propagate a temporal offset to the remaining motion segments belonging to the same gesture, I introduced the concept of *Complex Motion Segments* (CMS) (see Section 4.1) combining all motion segments of one channel and one gesture.

But since it is possible to split one gesture into several parts, it is not possible at parsing time, to align the segments in time. The scheduler is supposed to control the correct gesture sequence according to the time stamps of the segment, the CMS has no information about the previous or following CMS.

Therefore a temporal modification can cause a pause after finishing the modified gesture since the overall duration of the motion is shortened for negative values or lead to an overlap with the subsequent gesture for positive values. Adapting the global timing of the consecutive gestures avoids these problems, but must be done at the moment by hand.

An additional possibility to change temporal behavior is the $TIME\_WARP$ functionality that was already included in EMBR before I started my work.

3.1.2. Spatial Extent

Spatial extent depends from the amount of space the character uses and the position of the limbs at each key pose. For lower values, the hands should be nearer to the body and motion paths should be shorter then for high extent values. Finding a satisfying approach for modeling this brought up several methods each with assets and drawbacks. I decided to use the *Bounding Box Approach* since it has shown most usefully for different kinds of gestures, whereas other approaches did not lead to convincing results for special geometric or symmetric gestures.

If the nucleus of a gesture contains of one single key point, it is not possible to compute a bounding box. In this case, I apply a simple scaling function, I shift the target point on a straight line though a point in front of the character's solar plexus $p_{reference}$ and the original target point k_i, I compute the new point k_i' as follows:

$$k_i' = k_i + ((k_i - p_{reference}) * 0.8 * p_{spatial} \tag{3.2}$$

The HPM approach defines spatial changes by increasing or decreasing bounds of the gesture sectors around the actor. For my concept I use a similar method. For gestures consisting of more then two key poses, its possible to compute the bounding box of the movement.

This bounding box is a cuboid, containing every limb position of the gesture nucleus. If we now want to scale the gesture, we can do this by stretching this box. Key points translate respective to the center of the box, towards or away from it. With the help of the bounding box, it is possible to preserve the geometric structure and symmetry of a gesture. For applying spatial modifications to a gesture we compute the bounding box of the original key points k_i. According to the value of the $spatialextentp_{spatial}$ this box is scaled together with the contained key coordinates around its center c:

$$k_i' = k_i + ((k_i - c) * 0.8 * p_{spatial} \tag{3.3}$$

Here is a list of other methods I thought about to use:

- Scaling around a central point is a very simple method, we define a point in front of the chest of the character as center of the gesture space and shift key coordinates on the connection line between the original position and the central point, influenced by the value of the SPATIAL_EXTENT parameter. But scaling every point with respect to that reference point expresses no information about the original motion since the central point is chosen independently from the key point. I use this method if the gesture consists only of one point. Here it this spatial shift is sufficient to enlarge or shrink the gesture.

- Following the trajectory is also possible to in- or decrease the amount of movement. We can use two or even three points to shift the key point target. We either compute the path between the point we are coming from and the target and just truncate or extend the path. This ensures, in comparison with the previous described method, that for instance the new motion path points in the same direction as before.

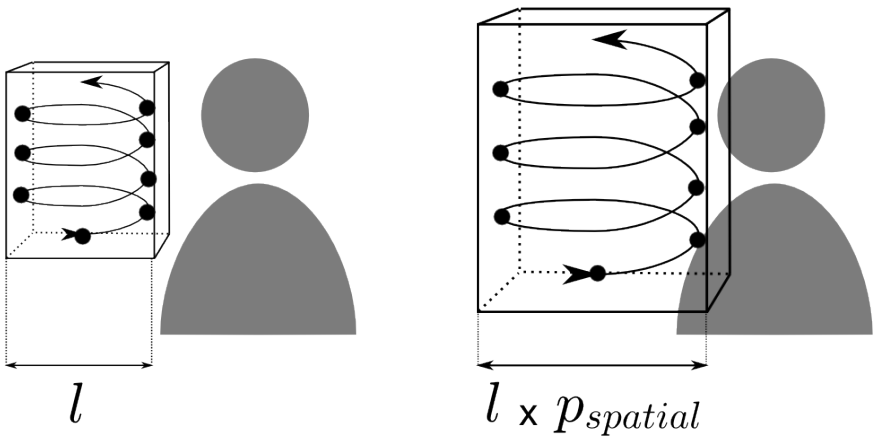

Figure 3.2.: All key points belonging to the gesture are transformed in a consistent way, the overall shape is preserved.

This method looks for the direction, the limb is coming from. If we have three points to compute the extent, we can also look to the direction of the next key point. We compute the vector between the previous and next point and shift the target on the orthogonal vector through the current point.

The drawback of this model is that for the first and last position we have no information about the previous (respectively the next) target point. This could destroy the appearance of gestures depicting geometry (e.g. a circle motion become an elliptic one).

- The EMOTE paper describes quite a different approach. Since they deal with motion capture data, the already have computed positions for every joint and affect now the joint angles of the posture. They move the position of the hand on a ellipse around the shoulder projection in three different dimensions: horizontal, vertical and sagittal.

  The spatial modification in EMOTE are not strictly directional but planar. A change in the horizontal plane modifies the joint position in forward/backward and left/right direction.

  This modification gives a more embracing or open-armed utterance to the gesture. Since EMBR deals also with motion generation from scripts and uses IK to perform gestures, we have to wait until the skeleton posture is computed before we can modify the target point. We would have to guaranty that although the timing constraints are meet.

3.1.3. Power

The power parameter p_{power} affects two aspects of motion, limb speeds and pauses, as well as paths. It varies the energy the agents uses to perform the gesture, a forceful

motion for a positiy value emphasizes the stroke phase of a gesture while negative values lead to loose motions.

For a positive power value it adds a preparation key pose to generate a preparative motion. This motion is just in the opposite direction as the stroke and has a length of $0.1 * p_{power}$*the length of the original motion. To stage the stroke it introduces an additional hold phase at this position with a duration of 300ms, the remaining motion is sped up by a factor $1.0 + (p_{power} * 0.2)$.

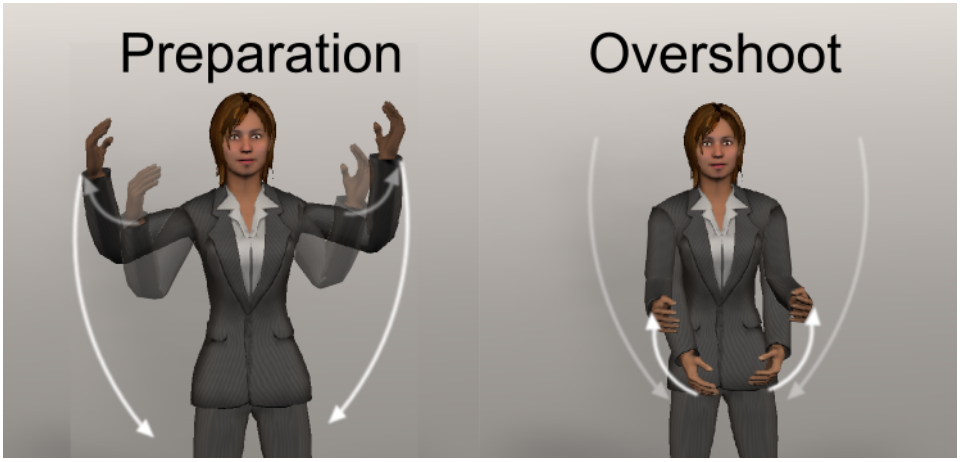

Figure 3.3.: Positive values for the power parameter add a preparation phase before the stroke and some overshot towards its end.

To express the high tension we interpolate with a parameter setting leading to nearly linear interpolation paths. This is reached by setting the tension parameter of the quaternion interpolation to p_{power}. This leads to straight paths for high power and wide curves for low power motion. Assigning p_{power} also to the bias parameter generates over- or undershoot at the end of strokes.

Both parameters together create then either straight overshooting path for a high amount of energy and loose, undershooting paths for low energy settings. The behavior of interpolation path is shown in Fig. 4.1 for tension and in Fig. 4.2 for bias in Section 4.2.

3.1.4. Fluidity

The fluidity parameter p_{fluid} affects pauses within the gesture and the continuity of the interpolation curve. A high fluidity setting shortens or removes hold phases by subtracting $300 * p_{fluid}$ms from existing pauses. It guarantees continuity of the interpolation by setting the continuity parameter to zero if p_{fluid} is equal or greater then zero.

A negative fluidity value introduces new hold phases, causing to rest the specified joint at every key post for $300 * p_{fluid}$ms . Setting the continuity parameter to $-p_{fluid}$ adds corners at the key points (see Fig. 3.4. The influence of the continuity parameter is shown in Fig. 4.3 in Section 4.2.

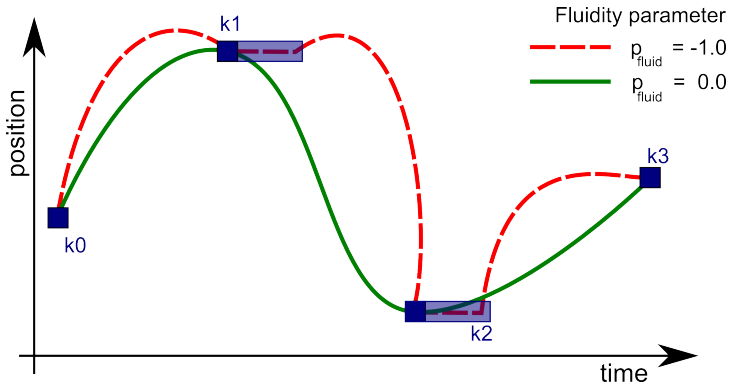

Figure 3.4.: A negative fluidity setting introduces new hold phases and leads to corners at the key point, whereas a neutral or positive setting guarantees continuity. The figure depicts the joint position k_n over time.

3.2. Nucleus

The parameters are not bind to a character (as it is done in [6]) but defined for a certain gesture. So it is possible to vary the expressivity within a sequence of gestures and change the style of the animation. So the animator has the possibility to highlight a part of a conversion or change the mood of the character through the animation.

To implement the parameters and to bind them to a gesture I decided to introduce the *nucleus* concept. This has two reasons. First, it is necessary to identify the most expressive part of a gesture, because in most cases only this part should be modified. Second, in EMBRScript a single gesture could span multiple gesture sequences (e.g. separate sequences for right and left arm). With the nucleus it is then possible to bind them together.

The nucleus concept bases on the gesture phases described by McNeill [14]and Kendon [9]. The expressivity of a gesture changes if the most meaningful part of a gesture is modified, therefore we bind these parts of the gesture to a nucleus associated with a parameter set. But there is no restriction to extend the nucleus to a complete gesture. Stylistic aspects that are determined by the nature of the agent (e.g. slow motions for an old agent) can by realized by binding all gestures perform by this agent to a specific nucleus, representing the corresponding modifiers.

3.2.1. Nucleus in EMBRScript

I added the NUCLEUS structure to the EMBRScript language, a NUCLEUS wraps all parameter settings and is identified by a unique NUCLEUS_ID. Within the standard definition of a gesture, it is then sufficient to bind all K_POSES belonging to the nucleus by referring to the corresponding nucleus over its ID. An example nucleus and its binding to a K_POSE is shown in Fig. 3.5.

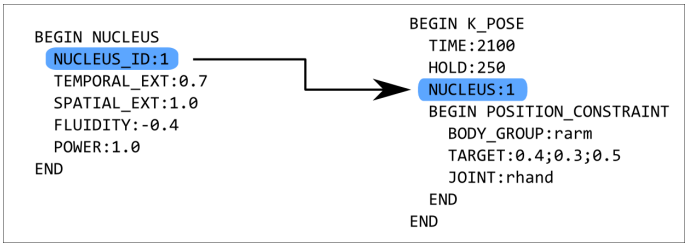

Figure 3.5.: The Nucleus embeds a set of expressivity parameters, it is bind to a K_POSE by ID

This design has the advantage that, if we want to apply a expressivity modification to a gesture, all we have to do are two modifications of the EMBRScript document: Fist, we define our nucleuses, each with its special parameter setup and an unique ID. Second, we add just one line to every key pose we want to be modified by the parameter settings of nucleus n: `NUCLEUS:$n$`and we are done.

3.2.2. Gesture Modifier and Nucleus Realization in the EMBR engine

I implemented the *gesture modifier* structure to bind parameter settings and modification functions together. For every nucleus definition in the script, one gesture modifier is created and manages the parameter settings. For every gesture nucleus that should be modified by these parameters (since its corresponding K_POSE is marked with the nucleus ID) the *motion factory* assigns a reference to the modifier object to the created motion segments.

The influence of the parameters affects two points of the animation generation process in EMBR. First, position constraints and global time point are adapted before the IK-algorithm computes the skeleton configuration for the key pose. Second the parameters *tension, bias* and *continuity* for the key pose interpolation are manipulated by the expressivity settings for *power* and *fluidity*. Figure 3.6 illustrates where in the animation pipeline (see Fig. 2.13) my implementation affects the gesture processing.

The gesture modifier provides a modification function *applyParameter* for every parameter of my expressivity set. The function checks for the corresponding parameter value and performs the gesture modification if the value is not zero. The modification is applied before the motion segments are sent to the *scheduler*. This avoids a superfluous IK computation of the neutral skeleton, the *pose blender* computes the skeleton configuration not until the start time of a motion segment is reached.

The influence of the expressivity parameters on the quaternion interpolation is described in Section 4.2.

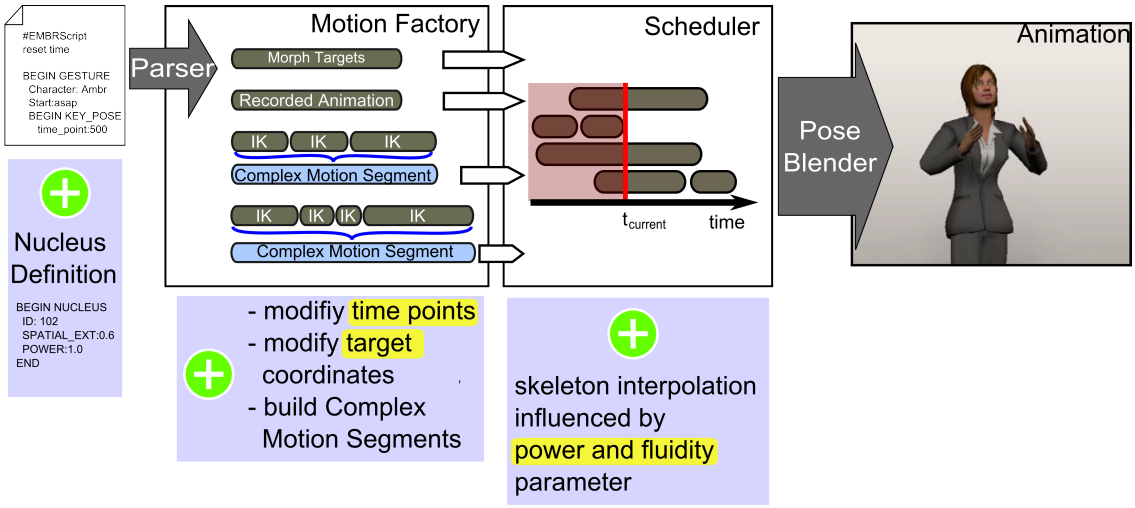

Figure 3.6.: Changes to the animation pipeline to realize gesture modifications: The EMBRScript document contains *nucleus* definitions and references from key poses to the nucleuses. The motion factory combines IK motion segments to complex motion segments, gesture modifier changes timing and shape of key poses. The interpolation process uses now quaternion interpolation with tension, bias and continuity control to compute the skeleton for the current time point.

4. Implementation Details

In this section, I give some further information on the structures and methods I needed to implement to realize gestural modifications.

4.1. Complex Motion Segments

Before I started my thesis, the scheduler in the EMBR engine managed the selection of active motion segments according to their respective absolute start and end times. The realization of my expressivity parameters requires information about preceding and subsequent motion segment for two reasons, first a modification of the global time points must propagate delays to following segments, second cubic interpolation method requires four key postures instead of two.

To gain the desired information I introduce the structure of *Complex Motion Segments* (CMS). A CMS filters key pose definitions that contain position and orientation constrains and binds the corresponding motion segments that belong to the same gesture and channel (kinematic chain) together to one CMS.

The absolute start and stop time of the CMS correspond to the start time of the first respectively the stop time of the last motion segment contained. Single motion segments loose their absolute time values and get a relative time to the position in the CMS.

The CMS has the same interface to the scheduler and the pose blender as a normal motion segment. When all small segments are combined to the CMS, it is send to the scheduler like a single motion segment just with one difference. Before it is send, it computes the skeleton configuration for all key poses within the CMS. Whereas EMBR computes skeleton postures for normal motion segments just in time (when it is send first time to the pose blender), for the cubic interpolation over a CMS, the information about postures is necessary that are not yet active and therefore not send to the pose blender.

4.2. Quaternion Interpolation with Tension, Bias and Continuity Control

Linear interpolation does not create convincing paths for limb motion and orientation, whereas cubic interpolation generates more natural curves. To be able to influence the interpolation path by my set of expressivity parameters, I transfered the parametric interpolation introduced by Kochanek et al. [10] [KB-Splines] to the squad interpolation algorithm by Shoemake [15].

The KB-Spline interpolation is based on the scaled sum of four Hermite interpolation functions. For calculating a pose P_s between key poses P_i and P_{i+1} with $s \in [0;1]$ the matrix form looks as follows:

$$P(s) = s \cdot h \cdot C = \begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_i \\ P_{i+1} \\ D_i \\ D_{i+1} \end{bmatrix} \tag{4.1}$$

The formulas for the derivatives D_i and D_{i+1} contain the parameters (t for tension, c for continuity and b for bias) and adapt the interpolation to the user input:

$$\begin{aligned} D_i =& \frac{(1-t)\cdot(1+c)\cdot(1+b)}{2} \cdot (P_i - P_{i-1}) + \frac{(1-t)\cdot(1-c)\cdot(1-b)}{2} \cdot (P_{i+1} - P_i) \\ D_{i+1} =& \frac{(1-t)\cdot(1-c)\cdot(1+b)}{2} \cdot (P_{i+1} - P_i) + \frac{(1-t)\cdot(1+c)\cdot(1-b)}{2} \cdot (P_{i+2} - P_{i+1}) \end{aligned} \tag{4.2}$$

Since our skeleton consists of joints each specified by a rotation in quaternion format, we do not want to interpolate just 3D vector coordinates as it is done in the model above. We use quaternions to describe the posture of the skeleton so we want also to interpolate over the orientation of the single joints. For interpolation between quaternions I use a modification of the *Squad* algorithm by Ken Shoemake [15] ([3]). The modified algorithm accepts also the three KB-Splines parameters.

The Squad algorithm uses a recursive slerp interpolation to compute in-betweens. It interpolates between two quaternions q_i and q_{i+1} with the help of two additional points a and b, $t \in [0;1]$. Shoemake defines squad as follows:

$$Squad(q_i, q_{i+1}, a, b, h) = Slerp(Slerp(q_i, q_{i+1}, t), Slerp(b_1, a_2, t), 2t(1-t)) \tag{4.3}$$

with

$$\begin{aligned} a &= q_i \cdot \exp\left(-\frac{\log(q_i^{-1} q_{i+1}) + \log(q_i^{-1} q_{i-1})}{4}\right) \\ b &= q_{i+1} \cdot \exp\left(-\frac{\log(q_{i+1}^{-1} q_{i+2}) + \log(q_{i+1}^{-1} q_i)}{4}\right) \end{aligned} \tag{4.4}$$

Having a look on the KB-Splines, we can influence the calculation of a and b by the 3 parameters. Whereas in the general squad model one has the same derivative for incoming and outgoing tangents, we now calculate two different tangents for the incoming and outgoing path T_0 and T_1:

$$\begin{aligned} T_i^0 =& \frac{(1-t)\cdot(1-c)\cdot(1-b)}{2}\cdot\log(q_i^{-1}q_{i+1}) + \frac{(1-t)\cdot(1+c)\cdot(1+b)}{2}\cdot\log(q_{i-1}^{-1}q_i) \\ T_i^1 =& \frac{(1-t)\cdot(1+c)\cdot(1-b)}{2}\cdot\log(q_i^{-1}q_{i+1}) + \frac{(1-t)\cdot(1-c)\cdot(1+b)}{2}\cdot\log(q_{i-1}^{-1}q_i) \end{aligned} \tag{4.5}$$

This can then be used when calculating a and b:

$$\begin{aligned} a &= q_i\cdot\exp\left(-\frac{T_i^0 - \log(q_i^{-1}q_{i+1})}{4}\right) \\ b &= q_{i+1}\cdot\exp\left(-\frac{\log(q_{i-1}^{-1}q_i) - T_i^1}{4}\right) \end{aligned} \tag{4.6}$$

The three parameters influence the following aspects of the path:

- *tension* defines how sharp the curve bends between the key points. For -1.0 it doubles the length of corresponding tangent vector leading to wide, unstressed curves, raising it to 1.0 it reduces its length to zero leading to linear interpolation behavior.
- *bias* specifies the direction of the path at the key points, for -1.0 it is completely determined by the incoming, for 1.0 by the outgoing tangent.
- *continuity* controls the angle between the incoming and outgoing tangents at key points. The continuity at the key points is preserved for a value of 0.0. Setting it to 1.0 or -1.0 the curve has acute corners to either the in- or outside of the path.

The Figures 4.1,4.2 and 4.3 demonstrate the influence of the parameters tension, bias and continuity on the quaternion interpolation path.

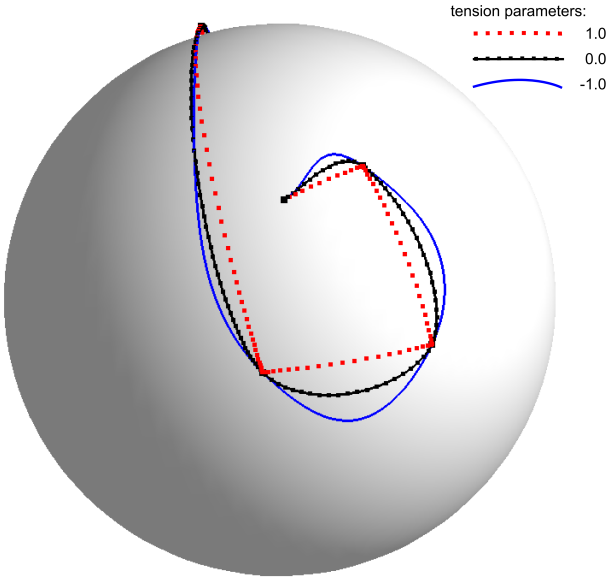

Figure 4.1.: Influence of the tension parameter on the quaternion interpolation spline visualized on a simplified SO(3) hypersphere.

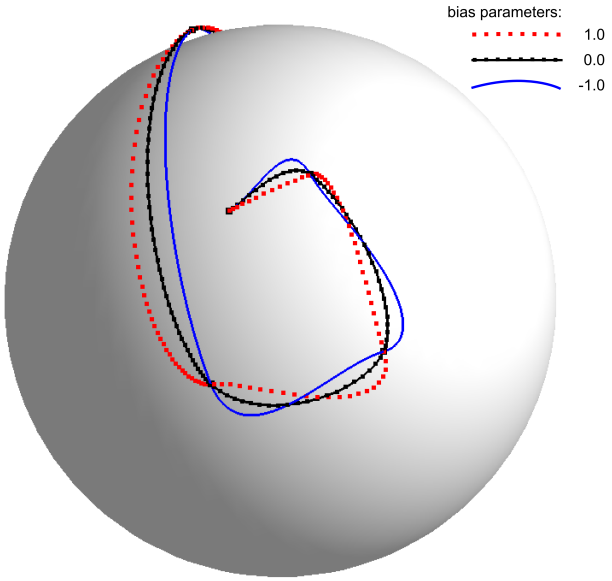

Figure 4.2.: Influence of the bias parameter on the quaternion interpolation spline visualized on a simplified SO(3) hypersphere.

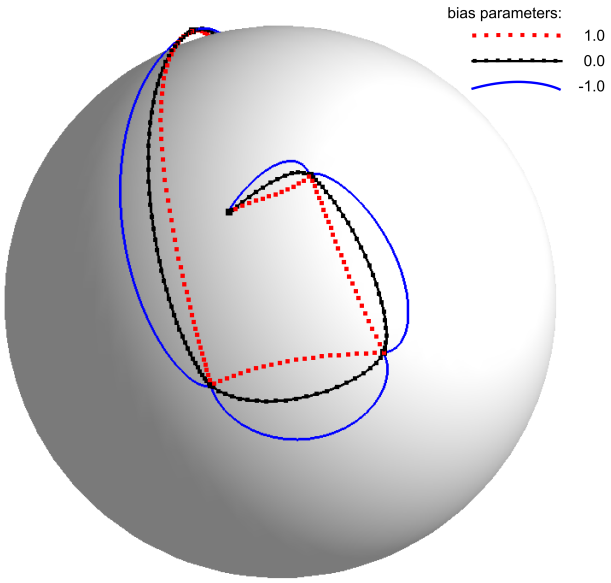

Figure 4.3.: Influence of the continuity parameter on the quaternion interpolation spline visualized on a simplified SO(3) hypersphere.

5. Conclusion

5.1. Summary

In my thesis I presented a new set of qualitative parameters for modifying animated character motion that is generated with EMBR. We have seen that human gestures are influenced by many aspects associated either with the personality or the situation of the character.

We got an overview of several approaches providing style adaption in different ways and saw that two main categories exist how to gain informations about a specific style. First, analyzing given example data sets where the user specifies what style the data represent to reproduce these styles later on. Second, generate different styles according to parameters determined by the user. We have seen that data driven approaches are often limited to some extent, depending from the given input. They can only reproduce styles that have been learned before. Parameterized models allow the user to adapt arbitrary gestures to any desired parameter combination.

We could observe that both approaches, HPM and EMOTE, provide a similar set of parameters for expressivity control. Developing the qualitative parameters for EMBR I also came up with the same parameters as HPM but with a different realization for EMBR. To vary the nuances of expressivity for EMBR we can now use four qualitative parameters *spatial extent*, *temporal extent*, *power* and *fluidity*.

My expressivity parameters affect the generation of animation by EMBR in three ways: First, the spatial extent influences the shape of the key poses and the power can introduce new poses to generate anticipation and overshoot. Second, the temporal extent controls the timing between the key poses and therefore the speed of the gesture. Third, both parameters power and fluidity affect the interpolation between the key poses and change hold phases within the gesture.

I developed a representation in EMBRScript that allows easily to change the style of pre-written gesture specifications by only adding a few lines to the document: Inside a nucleus definition, we can set the parameters to the values we need to reproduce the desired style. At least, we mark every key pose that should be modified by the corresponding nucleus ID.

I introduced three additional new concepts to EMBR and EMBRScript. With the adaption of the nucleus concept we have the possibility to emphasize the most expressive part of a gesture. But by stretching the nucleus over the complete gesture, we get an easy method to assign specific characteristics to the performing virtual agent. The new structure of the complex motion segments binds all key poses for one single body group together and organizes them in time, providing additional informations for the expressivity modification and interpolation. I implemented the new cubic interpolation

method for skeleton postures, the parameters tension, bias and continuity influence the curvature. It replaces the old linear interpolation for key poses and yields to more natural gestures since human motions are rarely linear in space. With my formulation of four qualitative parameters, based on the concept of nucleus, I believe to provide the user of EMBR a intuitive and effectively usable instrument to create quite natural styles and assign it to any arbitrary gesture.

5.2. Further Work

At the end, I want to give an outlook to possible further developments for EMBR. The biggest part of gestures generated by EMBR is specified by position and timing constraints in an EMBRScript document. Therefore, I concentrated my work on the implementation to modify these gestures by expressivity parameters. Since EMBR also supports the processing of pre-fabricated animation, it would be the next step to make qualitative modifications also available for the playback of existing animation data.

There exist several shader configurations for EMBR, it might be interesting, to combine special shader with certain parameter settings (e.g. blushing face) to improve the expressivity of a gesture. Another point of further development would be to include the use of morph targets and pre-defined pose targets. It then would be possible to enrich expressivity modifications by facial expressions and hand shapes (e.g. adding glare and fist to a powerful stroke). Another aspect of extension is the combination of parameter settings to a higher level attribute (e.g."tired" could combine a positive value for temporal extent, a negative value for power, spatial extent and fluidity).

A. Test Gestures Excerpt

In this appendix I list an excerpt of the gestures I used to develop and prove my qualitative parameters. I applied different parameter settings to the gestures and compared their influence and I believe that the parameters lead to convincing and believable gesture modifications. For the spatial parameters, this gestures showed the advantages and disadvantages of the different methods I described in Section 3.1. I used different types of gestures, to get a good representative cut through all possible gestures. Some gestures are iconic (e.g. "circle"), other emphasize a point of a discussion (e.g. "beat"), deictic gestures (e.g. "you and me", "wipe"). I also use gesture that only have one single point in their nucleus, this fact made a difference for the implementation of the parameters since their is only one point to refer.

A.1. Gestures with more then one key pose

Figure A.1.: A beat, used to emphasize a crucial point of a conversation.

Figure A.2.: A wipe with both hands from the inside to the outside.

Figure A.3.: The agent moves both hands down, as flinging away an object.

Figure A.4.: The character is waving with its right hand.

Figure A.5.: The agent performs a cup shape with its left hand.

Figure A.6.: The hands form a circle in front of the character.

Figure A.7.: The Character points at the audience and then at itself.

A.2. Gestures defined by a single key pose

Figure A.8.: Hands show a vertical distance

Figure A.9.: Hands show a horizontal distance

Figure A.10.: A raised index finger

Bibliography

[1] Matthew Brand and Aaron Hertzmann. Style machines. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 183–192, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[2] Diane Chi, Monica Costa, Liwei Zhao, and Norman Badler. The emote model for effort and shape. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 173–182, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[3] Erik B. Dam, Martin Koch, and Martin Lillholm. Quaternions, interpolation and animation. Technical report, 1998.

[4] B. DeCarolis, C. Pelachaud, I. Poggi, and M. Steedman. Apml, a mark-up language for believable behavior generation. In *Prendinger, H., Ishizuka, M., eds.: Life-Like Characters. Cognitive Technologies*. Springer, 2004.

[5] Arjan Egges and Nadia Magnenat-thalmann. Emotional communicative body animation for multiple characters. In *Proceedings of the First International Workshop on Crowd Simulation*, pages 31–40, 2005.

[6] B. Hartmann, M. Mancini, and C. Pelachaud. Implementing expressive gesture synthesis for embodied conversational agents. In *GW05*, pages 188–199, 2005.

[7] Alexis Heloir and Michael Kipp. Embr — a realtime animation engine for interactive embodied agents. In *IVA '09: Proceedings of the 9th International Conference on Intelligent Virtual Agents*, pages 393–404, Berlin, Heidelberg, 2009. Springer-Verlag.

[8] Alexis Heloir, Michael Kipp, Sylvie Gibet, and Nicolas Courty. Evaluating data-driven style transformation for gesturing embodied agents. In *IVA '08: Proceedings of the 8th international conference on Intelligent Virtual Agents*, pages 215–222, Berlin, Heidelberg, 2008. Springer-Verlag.

[9] Adam Kendon. *Gesture: Visible Action as Utterance*. Cambridge University Press, Cambridge, UK, 2004.

[10] Doris H. U. Kochanek and Richard H. Bartels. Interpolating splines with local tension, continuity, and bias control. *SIGGRAPH Comput. Graph.*, 18(3):33–41, 1984.

[11] Stefan Kopp, Brigitte Krenn, Stacy Marsella, Andrew N. Marshall, Catherine Pelachaud, Hannes Pirker, Kristinn R. Thrisson, and Hannes Vilhjlmsson. Towards a common framework for multimodal generation: The behavior markup language. In *INTERNATIONAL CONFERENCE ON INTELLIGENT VIRTUAL AGENTS*, pages 21–23, 2006.

[12] John Lasseter. Principles of traditional animation applied to 3d computer animation. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 35–44, New York, NY, USA, 1987. ACM.

[13] V. Maletic. *Body, Space, Expression: The Development of Rudolf Laban's Movement and Dance Concepts*. Walter De Gruyter Inc, 1987.

[14] D. McNeill. *Hand and Mind: What Gestures Reveal about Thought*. University of Chicago Press, Chicago, IL, 1992.

[15] Ken Shoemake. Quaternion calculus and fast animation, computer animation: 3-D motion specification and control, 1987.

[16] Deepak. Tolani. Analytic inverse kinematics techniques for anthropometric limbs /. *Thesis (Ph. D.)–University of Pennsylvania, 1998.*, 1998.

[17] Hannes Vilhjalmsson, Nathan Cantelmo, Justine Cassell, Nicolas E. Chafai, Maurizio Mancini, Stacy Marsella, Andrew N, Catherine Pelachaud, Zsofi Ruttkay, Kristinn R. Thrisson, Herwin Van, and Rick J. Van Der Werf. The behaviour markup language: recent developments and challenges. In *Proceedings of the Intelligence Virtual Agent Workshop*, 2007.

[18] Han Noot Zsfia, Han Noot, and Zsfia Ruttkay. The gestyle language. In *International workshop on gesture and sign language based humancomputer interaction*, Genova, Italy, 2003.