Utrecht University

Department of Computer Science

Master Thesis Game & Media Technology

Real-time Laughter on Virtual Characters

Author:
Jordi van Duijn

Supervisor:
Dr. Ir. Arjan Egges

August 25, 2014

Acknowledgments

I would like to thank Dr. Ir. Arjan Egges for giving me a scientific basis in the field of computer animation and for being my supervisor throughout my master thesis project. Next, I would like to thank Sybren A. Stüvel MSc for always giving a helping hand at times when the programming did not go very smoothly. Also, I would like to thank Harro van Duijn for convincingly laughing in his high quality microphone to provide me with an excellent sample of human laughter. Lastly, I would like to thank everyone from the Blender community for developing Blender, which I have used extensively throughout this 'hilarious' research.

Abstract

For virtual characters to behave and communicate in a human-like manner, they should not only be able to communicate verbally, but also show and communicate with non-verbal emotions or actions like laughter. Getting a virtual character to laugh in a natural-looking way is a challenging task, because it does not only involve smiling but also typical motions of the rest of the body. This becomes even more challenging if the character is simulated in *real-time* and is not acting autonomously but is reacting directly to input signals like sound or a video feed. Previous approaches that work in real-time have focused on detecting laughter from sound and/or videos and converting the input signals to features of laughter in the face, but none of these include full body motions, which are equally important to a natural-looking laughter simulation. Using prerecorded or live sound of laughter as input, we directly drive synthesized breathing and facial animation and introduce *laughing energy* to select predefined full body animations that match in intensity to the input laughter from the sound. Using our method, it is possible to simulate natural-looking laughter on virtual characters, directly responding to input signals like sound, in applications like games or any other real-time application that involves virtual characters, contributing to more human-like behavior and a more lively interaction.

Keywords: computer animation, laughter, synthesis, real-time

Contents

Chapter 1

Introduction

Although smiling and laughter are often not taken very seriously because of their fun nature, they are very important non verbal communication methods when it comes to human interaction. Genuine smiles and honest laughter can make conversations more cheerful and at ease while poorly executed fake smiles and fake laughter can make people feel like they don't want to be communicated with. Because of this, developers of games in which communication and interaction of players to players or players to agents is a vital part of the game, try to incorporate emotional states like laughter. Figure 1.1 shows two characters from the game *League of Legends* [1] in a laughing state, activated by the user. Artists made sure that the laughing animation of the body looks convincing, but the game does not support facial animation so the laughter looks rather strange from up close.

Figure 1.1: Two characters from the game *League of Legends* in a laughing state. Note that the faces are static and do not match with the state of the body.

Figure 1.2 is a screenshot taken from a cutscene from the game *Watch Dogs* [2]. In contrast to the previous example, this laughter includes facial expressions, which makes it look more realistic (although this is hard to see on a still image), but since the laughter occurs in a cutscene, the user has no influence on it whatsoever.

Figure 1.2: A cutscene from the game *Watch Dogs* with a kid that is laughing while he is getting tickled by his mother.

So laughter is present in games, but it is usually scripted in advance and/or does not respond to user interaction in a dynamic way. The goal of this thesis is to find out if motions and actions typical to laughter can be simulated on a virtual character in a natural-looking way by controlling it in real-time using input signals like sound. For this idea to be used in games or other applications that involve virtual characters (like avatar simulations) it is important that the input signals can be processed and used in the simulation in *real-time*. Laughter on virtual characters produced in real-time, reacting directly and dynamically to the input of the user will add a new dimension to the interaction with and the perception of the characters.

In this thesis, we present a method that achieves natural-looking laughter in real-time, using prerecorded or live audio signals from a laughing person. After discussing related work in chapter 2, chapters 3 and 4 will elaborate on how we use the audio of a laughing person to simulate laughter on virtual characters. Subsequently, we will take a look at the implementation and results in chapter 5 and finish with some conclusions on the possibilities, limitations and future work in chapter 6.

When it comes to computer animation, illustrations are sometimes inadequate to motivate certain choices or demonstrate results. This is why, during this report, we will refer to the videos from the following playlist [http://youtube.com/playlist?list=PL9j77vsM9-ulTzsEL9VlcZTbQabwDWMUJ] to demonstrate certain parts of our system.

Chapter 2

Background

In this chapter, the background and related work of our research will be discussed. First we will discuss the psychology of laughter in section 2.1, after which we will go into detail about the related work on laughter on virtual characters in section 2.2. After this, different techniques used to animate virtual characters and how they relate to our research are discussed in 2.3, finishing with the motivation and goal of this research in section 2.4.

2.1 The psychology of laughter

Laughter is a conspicuous but frequently overlooked human phenomenon. It is not only essential in human interaction in numerous ways [3], it is also known to have healthy effects, and especially to be a great medicine against stress [4]. The study of laughter and its effects on the body, from a psychological and physiological perspective is called *gelotology*. For us, it is important to have a look at studies of laughter to find out how and why people laugh in order to simulate the laughter of our characters in a natural-looking way. For this purpose, Ruch and Ekman [5] provide an excellent overview of a variety of studies on the different aspects of laughter, showing what is known about laughter in terms of respiration, vocalization, facial action, and body movement and they attempt to illustrate the mechanisms of laughter and define its elements.

Laughter is a very spontaneous and lively activity, so merely reading words about laughter will not give an adequate impression of how it actually looks like when people are laughing. To learn more about how people behave and how they move while they are laughing, we have watched close to an hour of videos showing laughing people, which was not only highly entertaining and very catchy, but together with the results of Ruch and Ekman revealed the following characteristics of the motions and actions of humans during laughter that are significant for our research:

1. From the videos, it is clear that almost all people tend to shake their upper body while they are laughing. Ruch and Ekman explained that this is caused by a forced breathing pattern during laughter which is provoked by numerous exclamations of *'ha'* (or anything similar).

2. The videos show that when people start to laugh, they put up their laugh-

ing face but during the laughter their mimic barely changes. This is not literally mentioned in the work of Ruch and Ekman, but in their definition of a *laughter bout* they mention that during the *offset* of the laughter bout, a long-lasting smile fades out slowly. The only two noticeable changes in the face while people are laughing are the opening and closing of the eyes and the opening and closing of the mouth, which is according to Ruch and Ekman caused by the fact that people inhale and exhale a lot of air during laughter, and opening the mouth makes this easier.

3. The most characteristic and yet most diverse part of human laughter is the motion of the whole body. This diversity is caused simply by the fact that every individual has their own specific way of laughing. This also becomes clear from the long list of body movements that are typical to laughter mentioned in the work of Ruch and Ekman. Furthermore, the videos showed that the way that people move their body is highly dependent on their surroundings; people who are sitting move different from standing people and people also tend to support themselves to any object (wall, table, other people) that is close by. There are a couple of typical motions though, that return quite often in the videos, like people covering their face with their hands, people trampling their feet on the ground, the classic knee-slapper, and some others.

2.2 Laughter on virtual characters

Expressing emotions is a vital part of human interaction. Emotions can be expressed through gestures and rarely come without a specific facial expression which becomes clear from the work of Ekman and Friesen [6], showing the vast variety of facial expressions and the feelings that come with them, both from the people that are expressing the emotions as well as the ones perceiving them.

Because expressing and perceiving emotions is so important for humans, adding emotional expressiveness to virtual characters gives an extra dimension to their interaction and is much preferred by users in cases of socially complex human-computer interactions such as education, rehabilitation and health scenarios [7]. Also in games, characters that show different kinds of emotional states and actions are developed to contribute to a more lively gameplay, especially in social games like *The Sims* [8] (Figure 2.1).

These emotions come across even livelier if they react to input signals like sound, video or motion sensors on the fly. In some cases even physics can have an effect on the visualization of emotions. A very nice example of this can be found in the work of van Tol and Egges [9], where they propose a method to control a realistic crying face in real-time. The tears that are generated are subject to external forces like gravity and collision, which contributes greatly to the realism of the simulation. Figure 2.2 shows several frames taken from a crying animation generated with their method, with a tear rolling down the cheek of the character.

The growing number of Embodied Conversational Agents (ECA's) developed both in research (e.g. [10][11]) and by the industry (e.g. [12][13]) also indicates that research on getting virtual characters to communicate in non-verbal ways is becoming more and more important. The state of the art ECA's already

Figure 2.1: Different emotional states from *The Sims* [8].

Figure 2.2: Several frames taken from a crying animation from [9].

look very realistic and show a variety of facial expressions and gestures, but still seem very robotic in their behavior. This is why Nijholt [14] argues that research on generating and interpreting facial expressions and on ECA's should be combined with research on emotional aspects like humor.

Although laughter is one of the most conspicuous and varying human emotion, there have been relatively few studies on laughter in the field of computer science. Fortunately, recent projects like *Incorporating Laughter into Human Avatar Interactions: Research and Experiments* (ILHAIRE) [15] try to help the scientific and industrial community to bridge the gap between knowledge on human laughter and its use by avatars, enabling sociable conversational agents capable of natural-looking and natural-sounding laughter. Previous studies on laughter in the field of computer science can be categorized in laughter detection from sound [16][17][18], detecting laughter types from whole body motion [19], synthesizing laughter sounds [20][21], and synthesizing laughter animations on virtual characters which is most relevant to our research and will be more extensively discussed in the next paragraph. The studies on laughter detection are not very relevant to our research because they focus mainly on detecting laughter, discriminating it from other sounds like speech, rather than extracting

features from the laughter itself which might be useful in laughter synthesis on virtual characters. Trouvain [22] summarizes the different terminologies used in studies of automatic laughter processing, as well as various ways to detect laughter types.

When it comes to synthesizing laughter on virtual characters, studies have focused mainly on the acoustics and the face. For example, Cosker and Edge [23] are able to distinguish between non-speech articulations like laughter, crying or sneezing and use it to drive facial animation. Their resulting facial expressions are not very convincing, which is most likely because they mainly focused on the acoustic part. Another example of laughter synthesis in the face is the work of Urbain et al. [24], where they present an audiovisual laughter machine, capable of recording the laughter of a user and responding to it with a virtual agent's laughter. This laughter is linked with the input laughter hoping that the initially forced laughter of the user will eventually turn into spontaneous laughter. One of the very few studies that do not focus on synthesizing facial details of laughter but instead focus on the body, is the work of DiLorenzo et al. [25], where they present a method to model anatomically inspired laughter on a torso using audio. Their results look very convincing, but unfortunately their physical model is computationally too heavy to run in real-time simulations and only includes the torso motion caused by breathing, omitting gestures that are typical to laughter.

2.3 Animating virtual characters

In order to simulate laughter on a virtual character, it needs to be animated. Several techniques have been developed to animate virtual characters, categorized in *procedural* and *data*-driven approaches. Animations can procedurally be generated using functions and models like physics-models [26][27]. Also in the field of genetic programming, research has been done on procedurally animating virtual characters [28][29]. Examples of animations that rely on data are ones that are *key-framed* by artists using techniques like [30] to speed up the animating process and ones that are generated using the data from motion capture systems [31][32]. Giang et al. [33] provide a detailed overview of the numerous approaches to simulating, controlling and displaying the realistic, real-time animation of virtual characters.

In our research, both procedural and data-driven animations have their advantages and disadvantages. Procedurally generated animations allow for control during the animation and can simulate real-time effects, directly responding to input signals. For us, this is very suitable for generating breathing animations and facial details. Unfortunately, typical full body laughter motions like 'slapping the leg' or 'rolling over floor' are too complex to control with any procedural method and have to rely on prerecorded animations acquired from for example motion capture data or in our case, key-framed animations.

2.3.1 Facial animation

As mentioned in section 2.1, Ruch and Ekman show that laughter not only induces motion on the body of a person, caused by breathing behavior or typical gestures, but even more so on the face in the form of facial expressions. This

means that a laughter simulation also needs facial animation, which is slightly different from the animation of the body of a virtual character. This is mainly because full body motion is based on joint rotation while facial details are expressed solely through deformations caused by the activation of dozens of tiny muscles (with the exception of jaw movement). The activation of these muscles and how this affects facial expressions is extensively discussed in another work from Ekman and Friesen [34], where they present the *Facial Action Coding System* (FACS), a technique for the measurement of facial movement. In practice, there are three ways to animate the face of a virtual character, as shown in Figure 2.3: moving around *vertex-groups*, interpolating between different predefined *shapes*, and skinning a physics based model relying on *muscle deformation*. Techniques that use the latter one [35][36] give the most realistic results, but are not yet (or hardly [37]) able to run in real-time. Moving vertex-groups gives direct control over the shape of the face, but makes it hard to create typical facial expressions. This is why, in our research we use shape interpolation to control the facial expressions of our character. Using predefined shapes allows for accurate facial expressions, and interpolating between them using simple interpolation technique is an easy task and keeps the simulation running in real-time.

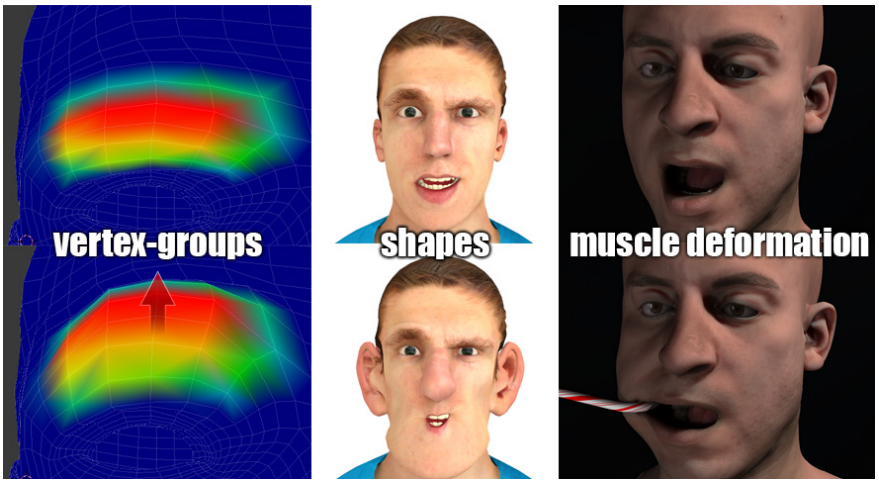

Figure 2.3: Different techniques to animate a face. From left to right: *vertex-groups* that allow for direct control (vertices that are colored red are influenced more by the upward motion than the ones that are blue), *shapes* that allow for accurate facial expressions or for example caricatures, and physics-based *muscle deformation* for physically accurate results and for example interaction with other objects. The images from the muscle deformation are screenshots taken from the work of [36].

2.4 Motivation and Goal

As shown in sectioin 2.3 research has been done on simulating laughter on virtual characters. This research however, focuses mainly on the acoustic part and generates laughter only on parts of the body, while humans use both facial expressions as well as typical body motion to express laughter. Also, specific

breathing motions are very typical to laughter so in order to get a virtual character laughing in a natural-looking way, all of these parts have to be combined. Furthermore, generating emotions like laughter on virtual characters in real-time, responding directly to input signals, makes them much more human-like and adds a lot of liveliness to their interaction.

Given these motivations, the goal of our research is to create natural looking full body (body + face) laughter on a virtual character in real-time. The research done in [25] provides an excellent starting point for the motion of the torso caused by the breathing behavior during laughter. Although their method is too heavy to run in real-time, we can have a good look at their results to approximate the same effect in real-time. Also, inspired by the way they use the audio signal to drive the simulation of their torso, we use a similar signal in chapter 3 to drive the *smile-intensity* , the *opening and closing of the mouth*, and the motion caused by the *breathing* behavior. Furthermore, in chapter 4 we introduce *laughing energy* to play predefined full-body laughter animations from motion capture data, or in our case key-framed animations, that match the intensity of the laughter sound using the signal extracted from the audio. An overview of the system can be found in Figure 2.4. The implementation and results of our method are discussed in chapter 5 after which this report will be concluded with discussions on the possibilities, limitations and future work in chapter 6.

Figure 2.4: A schematic overview of the systems architecture.

Chapter 3

Audio driven animation

In this chapter, the process of converting the input laughter sound to a motion control signal that can be used to directly drive the breathing and facial animations of the virtual character will be discussed. First we will go into more detail about how raw input sound is converted to a motion control signal in section 3.1 after which we will show how this signal is used to get the character to show breathing motion in section 3.2 and how to put a smile on its face in section 3.3. The motion signal cannot be used to directly drive the full body animations. Instead, we use it to build up *laughing energy* that we use to select an appropriate full body animation from a list of predefined animations, which will be further explained in chapter 4.

3.1 Motion control signal from laughter sounds

We use either live or prerecorded sound of laughter to drive the laughing animations on our virtual character. When laughing, most people produce the loudest sound during their exclamations of *'ha'*. These exclamations are very typical to laughter and, due to the large amount of air that is inhaled and exhaled, are accompanied by chest and jaw movement. Moreover, the loudness of the sound also indicates how intense a person is laughing. The loudness of sound however, is just a subjective measure [38], therefore we will be using the physical strength of sound, namely the *amplitude*. So to exploit the properties of sound typical to laughter, we use the amplitude of the sound to create a motion control signal to directly drive breathing and facial animations and indicate how intense the person is laughing.

3.1.1 Amplitude

In order to get a usable motion control signal, the amplitude of the sound is mapped to the range [0,1]. However, amplitude has various definitions [39] and not all of them are suitable to be used for the motion control signal. Because our simulation runs with time-steps and at every time-step a window of sound has to be analyzed, the *Root Mean Square (RMS) amplitude* is most suitable to get the 'loudness' of the sound at a time-step. The RMS value of a set of values (in our case a chunk of sound at a time-step) is the square root of the

arithmetic mean of the squares of the original values [40]:

$$x_{rms} = \sqrt{\frac{1}{n}(x_1^2 + x_2^2 + \cdots + x_n^2)} \tag{3.1}$$

Now that we have a representative value for the strength of the sound, it is mapped to the range [0,1] so that it is clear that values close to 0 represent silence while values close to 1 represent loud laughter. This is done by keeping track of the minimum and maximum amplitude so far and using this to map the current amplitude to the range [0,1]. In pseudo-code, this would look like:

```
for every timestep:

    amp = getAmplitude(sound) #using the RMS amplitude
    if amp > maxAmp: maxAmp = amp
    if amp < minAmp: minAmp = amp

    motionSignal = (amp - minAmp) / (maxAmp - minAmp)
```

Note that this way of mapping the signal requires a calibration (short burst of sound in the microphone) at the start of the simulation to set an initial minimum and maximum amplitude. Both the minimum and maximum amplitudes are adjusted (if needed) at every time-step to make sure the mapping of the current amplitude stays within the range of [0,1]. Especially the maximum amplitude might not have been set correctly during the calibration and has to be adjusted as soon as the sound gets louder than previously measured.

The downside of using this way of mapping is that users might produce a loud sound very close to the microphone during the calibration, but start laughing farther away from the microphone. This will result in a situation where the maximum amplitude is never approached again while the user might be laughing very loud. To solve this, the minimum and maximum amplitude are respectively increased and decreased by a very small amount at every time-step as illustrated in Figure 3.1.

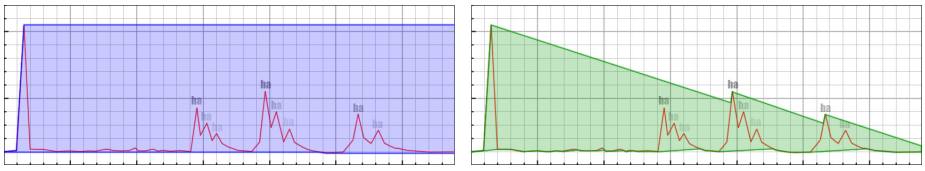

Figure 3.1: Two graphs showing the motion control signal (in red) with a high calibration peak at the start, but lower values during laughter. On the left side, the minimum and maximum amplitudes (blue area) are respectively never increased and decreased which results int he motion control signal during laughter to be mapped to values that are too low. On the right side this is solved by increasing and decreasing the minimum and maximum amplitude values with a fixed amount at every time-step. Note that this is a sketch to illustrate the idea, in reality the minimum and maximum amplitudes are adjusted with much smaller values so that the maximum amplitude will not get too low and the minimum amplitude will not get too high.

3.1.2 Rise and Fall

To add some simple control to how fast the motion control signal may rise and fall, desirable in cases of noise or distortion that cause extreme peak values, we set a *rise* and *fall* parameter to control how fast the signal may rise and how fast it may fall. The following equation shows how the motion control signal is adjusted by the rise and fall parameters:

$$finalMCS = \begin{cases} min(MCS, prevMCS + rise * dt) & \text{if } MCS > prevMCS \\ max(MCS, prevMCS - fall * dt) & \text{otherwise} \end{cases} \tag{3.2}$$

Where MCS stands for the motion control signal and prevMCS stands for the value of the motion control signal in the previous time-step. So whenever the MCS is larger than the previous MCS, the current MCS can only rise with a maximum of the rise value. Figure 3.2 shows how the motion control signal is modified by different rise and fall values.

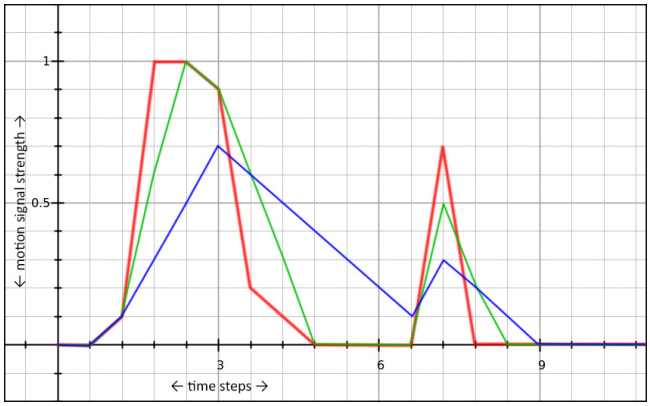

Figure 3.2: A graph showing the original motion control signal and two filtered ones using different *rise* and *fall* values. The red line shows the original motion control signal, the green line has a rise value of 0.5 and a fall value of 0.3, and the blue line has rise and fall values of respectively 0.2 and 0.1.

The resulting motion control signal, of which an example can be found in Figure 3.3, is now ready to be used to drive the motion of the torso caused by the breathing behavior during laughter (3.2) and facial animations (3.3) and indicate how intense the person is laughing so an appropriate full body animation (4.2) can be selected.

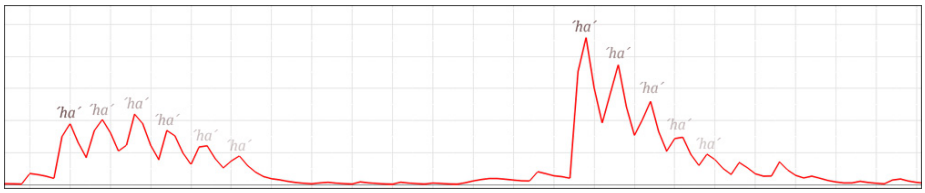

Figure 3.3: A typical example of a motion control signal extracted from the audio. Note that the peaks of the graph represent the typical *'ha'* exclamations during laughter.

3.2 Breathing animation

Filippelli et al. [41] showed that laughter is characterized by a sudden occurrence of repetitive respirations. This breathing behavior causes the lung volume and therefore the size of the chest and the abdomen to fluctuate greatly. Based on this, DiLorenzo et al. [25] simulated physically accurate laughter on a torso. Their results look very convincing and are superior to procedural approaches like weighted blending of shapes in a way that they can show the rich interplay between the subsystems of the torso (i.e. abdominal cavity, ribcage, clavicles, and spine). Unfortunately, their approach is computationally too heavy to run in real-time and can therefore not be used in our simulation. However, one could argue that the subtle details that create the visually convincing laughter with their method become less significant as soon as the virtual character is wearing somewhat baggy clothing like a T-shirt. In this case it would just be the larger motions (chest and belly expansion/contraction) that create the typical laughter motion caused by the breathing behavior. Furthermore, as soon as the virtual character also starts moving the rest of his body, it would become harder to notice subtle details around the torso. This is why for our simulation, we've chosen to use a simpler and real-time approach while making sure that the large typical motions caused by the breathing behavior during laughter are still clearly visible.

3.2.1 Shape interpolation

In order to simulate the motion of the chest and belly caused by the breathing behavior, we use *shape interpolation*, controlled by our motion control signal. How the shapes are controlled by the motion control signal is further explained in section 3.2.2. When it comes to animating facial expressions of virtual characters, shape interpolation is the most widely used method and consists of using a set of shapes (key facial expressions) to define a linear space of facial expressions. A shape is defined for every vertex as an offset to its position in the original mesh. This technique is used instead of skeleton-driven techniques because deformation in the face is not caused by joint rotation (besides the jaw) but by the contraction of dozens of tiny muscles. Also in our case, the chest and belly are not deformed due to joint rotation, but to the expansion and contraction caused by respiration, so shape interpolation is a logical choice to animate them.

Animating a face using shapes requires numerous shapes in order to show the rich variety of facial expressions. In our case however, the chest and belly

require only *one* shape on top of the base mesh. This is because the expansion and contraction of both the chest and belly are merely caused by increase and decrease of the lung volume, which can be considered as a symmetric and linear motion. The shapes that we used for our simulation are shown in Figure 3.4.

Note that the chest and belly only have a shape for their expanded state (when the character breathes in), while breathing out might cause the chest to contract to a smaller size than its rest state or breathing in really deep will expand the chest beyond the shape we created. This is solved by using linear *extrapolation* [42]. Figure 3.5 shows different states of the chest that can be created given an influence value using the base mesh (rest state), one shape that represents the expanded state, linear interpolation, and linear extrapolation.

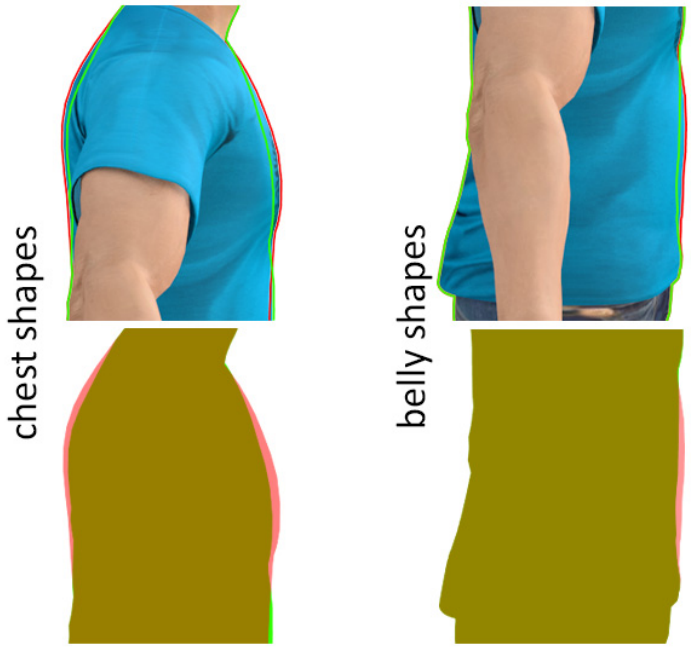

Figure 3.4: The different shapes of the chest and belly used for the breathing animation. The green lines and green shapes on the top and bottom of the figure show the 'rest' state of the chest and belly while the red ones show the shapes that represent the 'expanded' state.

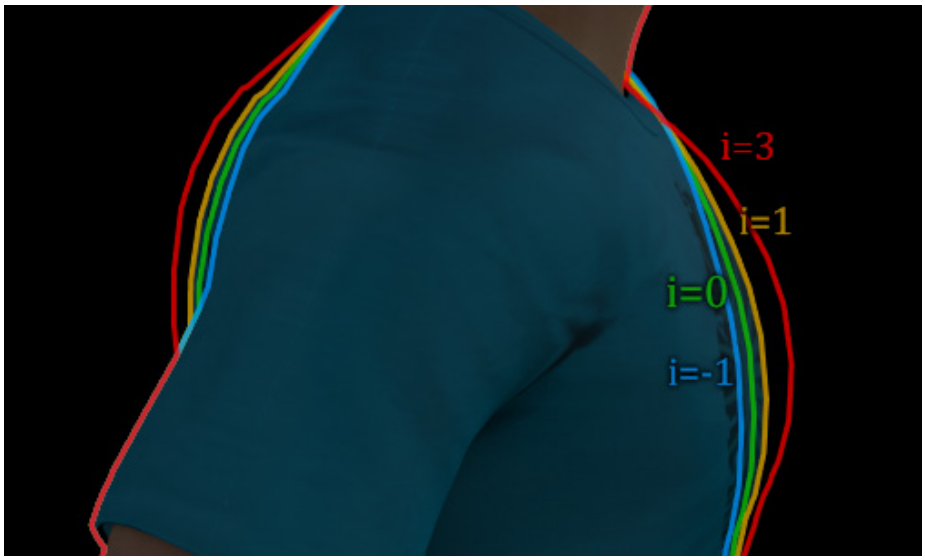

Figure 3.5: Different states of the chest. The green line shows the mesh in the rest state. The rest of the shapes are generated by linearly interpolating and extrapolating the shape that was created given an *influence (i)*. When the influence lays in the range [0,1] (0%-100%), the shape is linearly *interpolated* and as soon as the influence falls out of this range, linear *extrapolation* is applied. In this case, an influence of 1 (100%) implies that the lungs are full, 0 implies the rest state, -1 implies empty lungs, and the influence value of 3 is just to show that shapes can be exaggerated using linear extrapolation.

3.2.2 Driving the shapes

The influence of the shape shown in Figure 3.5 is controlled by the motion control signal. As shown in Figure 3.3, peaks in the motion control signal represent exclamations of *'ha'* that each come with a burst of exhalation. In the results of DiLorenzo et al. it is clearly visible that these exhalations decrease the lung volume which causes the chest to contract and it also shows that the belly slightly bulges out due to the motion of internal organs in the abdomen. Using this information, we connected the motion control signal to the influence of the chest and belly shapes in a way that high values of the motion control signal cause the chest to contract and the belly to slightly bulge out. Keeping in mind that the two shapes of the chest and belly represent their *expanded* state and that high values of the motion control signal result in *contraction* of the chest and *expansion* of the belly, the motion control signal controls their influences at a given time-step as follows:

$$chestInfluence(t) = 1 - MCS(t) \tag{3.3}$$

$$bellyInfluence(t) = MCS(t) \tag{3.4}$$

Since there is now a clear connection between the sound and the motion of the character the first signs of interactivity are already emerging. However, the chest and belly react too intensely to the sound, which creates a jittery look. In order to solve this, a smoothing pass has to be applied to the motion control

signal before it influences the shapes. Keeping in mind that we use live sound for our simulation, we can only use smoothing functions that use the current and/or previous values from the motion control signal. One way to create a smoother version of the motion control signal is to save the past x values and use their *weighted average* to control the influence of the shapes. In pseudo-code this would look like:

```
#create a queue and fill it with x zero's
smoothingQueue = Queue[x]
for every timestep:
   #add the current MCS value to the queue
   smoothingQueue.append(MCS)
   #remove the MCS value from the 'left' side of the queue
   smoothingQueue.popleft()
   #create the smoothed out MCS by computing the weighted average
   smoothMCS = smoothingQueue.getWeightedAverage()
```

The bigger the value of x, the smoother the motion control signal will become. However, this will also introduce a delay in the signal because only the previous values of x are taken into account.

Another way to smooth out the motion control signal is to use the rise and fall method used to create the motion control signal from the amplitude of the sound in section 3.1.2. This method is slightly less effective in creating a truly smooth version of the motion control signal because it does not take into acount previous values, but there is no more delay and since we can now control how fast the signal may rise and fall, we can eliminate the intensity of the sound that creates the jittery look that we mentioned before. The effectiveness of using the rise and fall method to tone down the intensity of the motion control signal that controls the influence of the shapes is demonstrated in Video 1 from the playlist.

3.3 Facial animation

The work of Ekman and Friesen shows how important facial expressions are for showing and conveying emotions. Naturally, this also holds for laughter. So for our virtual character to be able to laugh in a natural-looking way, laughter should also be simulated in his face, and again, for reasons mentioned earlier, this laughter should react to the input sound in real-time. Laughter in the face is expressed through *smiles*, in which a distinction can be made between voluntary insincere smiles (fake smiles), and involuntary genuine smiles, also called a *Duchenne smile* [5]. Ekman and Friesen have shown that their FACS can be used to distinguish these two types of smiles: fake smiles are created by merely contracting the *zygomatic major*, which raises (and pulls back) the corners of the mouth while a Duchenne smile also involves contracting the inferior part of the *orbicularis oculi*, which raises the cheeks, creating crow's feet around the eyes. Figure 3.6 shows examples of genuine and fake smiles on our character.

However, a Duchenne smile is not all that is going on in the face during laughter. Ruch and Ekman [5] argue that the more intense people are laughing, the more they open their mouth, so the large amount of respired air that comes with laughter can be inhaled and exhaled more easily. Besides this, also the

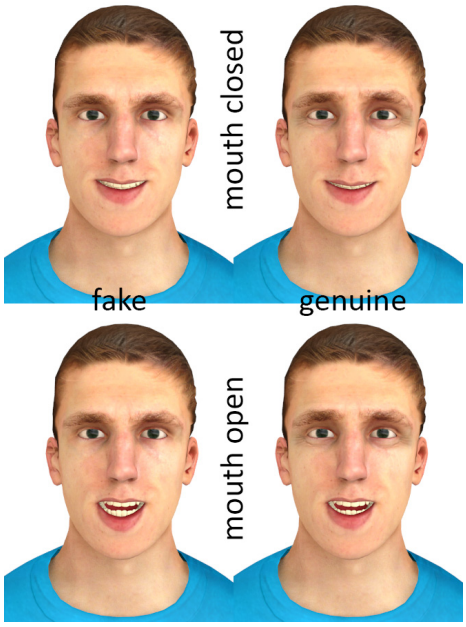

Figure 3.6: Examples of fake and genuine smiles. The left side shows fake smiles with the mouth open and closed while the right side shows genuine (or Duchenne) smiles. Notice the subtle difference around the eyes: the cheeks are raised a bit, which creates crowfeet.

opening and closing of the eyes has an effect on the overall expression of the face. In summary, laughter on the face can be expressed through combining a genuine *smile*, involving the contraction of two types of muscles in the face, opening and closing of the mouth by moving the *jaw*, and the opening and closing of the *eyes*. Because these three aspects of laughter are not necessarily synchronized, they are also treated separately in the simulation.

3.3.1 Smile

As mentioned in section 3.2.1, facial expressions on virtual characters are mostly generated using shape interpolation. Also in our case, this method is very effective because only one shape (on top of the base mesh) is needed to create a genuine smiling face. However, using only one shape will not allow for subtle changes within the smile (for example a slightly asymmetric smile), but these kinds of subtleties are hardly noticeable as soon as the character starts moving with his whole body. Moreover, using only one shape to express a smile simplifies the process of driving the smile with the motion control signal. Figure 3.7 shows the shape that was used to create the smile.

Similar to how the breathing motion is driven by a filtered motion control signal, the smile intensity (influence of the shape) is also controlled by a filtered version of the motion control signal. The only difference is that a lower rise and especially a lower fall value are required so the smile fades in and out slowly instead of appearing and disappearing in an instant. This is important, because from the videos that we have watched, we have learned that people

Figure 3.7: On the left the base mesh showing a neutral facial expression. On the right the shape that represents a smiling facial expression.

do not instantly put on or off their smiling face when they are laughing, but instead (albeit unintentionally) gradually start and stop smiling. The difference between using the raw motion control signal to drive the influence of the smile shape and using a smoothed out one applying the rise and fall method with low values is demonstrated in Video 2 from the playlist.

3.3.2 Jaw

As mentioned before, Ruch and Ekman [5] argue that the more intense people are laughing, the more they open their mouth, so the large amount of respired air that comes with laughter can be inhaled and exhaled more easily. This means that the jaw motion of a laughing person is mainly caused by the breathing behavior which means that it can also be directly controlled by the motion control signal (As shown in section 3.2). One shape for an open mouth was created of which the influence is again directly controlled by the motion control signal. Similar to how the smile intensity requires a filtered (smoothed) version of the motion control signal, the jaw motion also needs the motion control signal to be smoothed out a bit. The rise and fall parameters used for the jaw (as well as the ones used for the smile intensity and the chest movement) were manually fine-tuned until desirable results were achieved.

3.3.3 Eyes

From the videos that we have watched, we have learned that in most cases people tend to close their eyes when they are laughing intensely and open them again when they are more at ease. This would suggest that they can be controlled directly by the motion control signal in a similar way as the smile-intensity and the jaw motion. However, unlike the jaw and the smile, the opening and closing of the eyes also depends on the full body animation that is playing.

This becomes clear in for example the pointing animation that we use in our simulation (see Figure 3.8). Furthermore, we have seen cases where people open their eyes widely instead of closing them when they are laughing very intensely. So instead of driving the opening and closing of the eyes directly by the motion control signal, the eyes are opened and closed manually for each full body animation using a set of key-frames to indicate at what point in the animation the eyes should open and when they should close. This ensures that the eyes are never opened or closed at points in an animation where it would appear unnatural. Furthermore, because intense animations will be playing whenever the laughter sound is intense (explained in chapter 4), we can still indirectly make sure that the eyes are closed more often when the laughter sound is intense by setting key frames that close the eyes more often during intense full body animations.

Figure 3.8: During the 'point' animation, the character points at something that he finds funny. It appears weird if the character would have his eyes closed during the whole pointing part of this animation, because how would he know what he is pointing at?

The downside of key-framing the opening and closing of the eyes instead of controlling them directly with the motion control signal is that the animation of the eyes appears less responsive to the sound because they close and open at exactly the same time every time for each full body animation. Fortunately, people do not merely close their eyes because they are laughing, but also have to *blink* to prevent their eyes from desiccating. This eye-blinking adds some variety and randomness to the opening and closing of the eyes which breaks the pattern of the key-framed opening and closing of the eyes for each full body animation. Eye-blinking was implemented simply by creating a very short key-framed animation that closes and opens the eyes and playing it at a certain interval. The results of [43] show that people with normal eyes blink every 4 seconds with a standard deviation of 2 seconds, so for the interval of the eye-blinking a random value in the normal distribution of 4s with a standard deviation of 2s is picked. Before the eye-blinking animation is played, we check if the eyes are not already closed (or closing) as part of a full body animation to prevent conflicts in closing the eyes.

Chapter 4

Energy driven animation

We have discussed the parts of the simulation that can directly be controlled by the sound using a motion control signal extracted from the input audio, namely the motion caused by the breathing behavior and the facial animation. The last remaining part of the laughter simulation is the full body animation. People each have their own specific way of using their body while they are laughing, but what characterizes the full body motion of almost everyone (see section 2.1) are typical gestures like slapping the knee, covering the face, trampling the feet, and so on. Because there are so many different kinds of these gestures and because they highly depend on the posture and the surroundings of the person, it is very complicated to synthesize the full body animation, which means that it cannot directly be controlled using the motion control signal in the way that the breathing and facial animation is simulated.

Instead, we use a set of around 10 prerecorded or (in our case) key-framed animations that we play during the laughter simulation. The laughter intensity of these animations varies from very mild laughing motions like swaying back or front a bit to very intense motions like falling down on the ground, so all different intensities of laughter can be simulated. The intensity of these animations has to be synchronized with the intensity of the laughter sound in order to create a coherent simulation. Because our method uses live sound, we cannot simply take a look ahead in time to see what animation will fit the upcoming intensity of the laughter sound best. To overcome this problem and still be able to select full body animations that match the intensity of the laughter sound, we introduce *laughing energy*.

4.1 Laughing energy

The basic idea behind the laughing energy is that the motion control signal *produces* laughing energy and that the animations *consume* laughing energy. Each animation is manually assigned an amount of energy (corresponding to the visual intensity of the animation) which will be consumed from the laughing energy over the time that the animation is active. So if an animation of length l has been given an energy value of e, e energy will be consumed from the laughing energy over time l. This means that the more intense the motion control signal, the more laughing energy is created and the more intense the animation, the more is consumed (see Figure 4.1).

Now, if we try to keep the amount of laughing energy close to zero by selecting the animations in a 'smart' order, more intense animations will be played whenever the motion control signal gets more intense. This way, a clear connection is maintained between the intensity of the sound and the intensity of the animations.

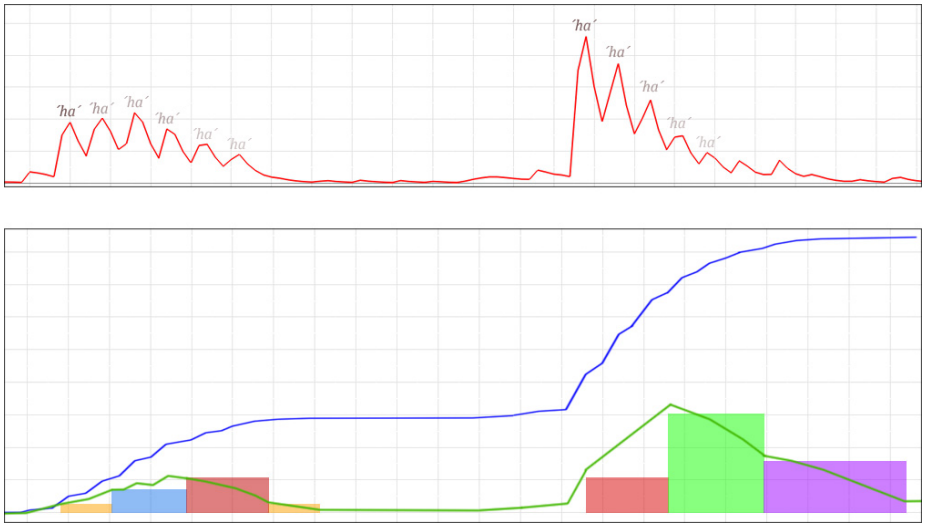

Figure 4.1: A sketch of the laughing energy process. The red line in the top graph represents the motion control signal. The blue line in the bottom graph represents the laughing energy when no energy is consumed by animations, this would correspond to the primitive of the motion control signal. The green line represents the actual laughing energy during an example simulation, it is increased by the motion control signal and simultaneously decreased by the animations. The animations are represented by the colored blocks. The height of these blocks stands for the intensity or *energy* of the animation while the width shows the *length* of the animation. Note that this is merely a sketch to illustrate the idea behind the laughing energy, it is not an exact case from a simulation.

4.2 Selecting full body animations using laughing energy

In order to clarify the idea behind the laughing energy and how the full body animations are selected in a 'smart' order, we will first give a formal definition of the system and its constraints (similar to how problems are defined in *linear programming* [44]) after which it will be explained and motivated.

minimize:

$$L_t = L_{t-1} + MCS(t) - e(a)/l(a) \tag{4.1}$$

given constraints:

1. $e(a) < L_t$
2. $e(a) > e(a_{active}) * c$
3. $i(a) > I$

where:

- L_t = laughing energy at time-step t
- $MCS(t)$ = motion control signal value at time-step t
- $a \in A$
- a = the animation with which the laughing energy should be minimized
- A = list of predefined animations $[a_1 \dots a_n]$
- a_{active} = the animation that is currently playing (if any)
- $e(a)$ = energy of animation a
- $l(a)$ = length of animation a in time-steps
- $i(a)$ = the number of time-steps that a has been inactive
- c = constant factor to control how easy an animation can be overruled
- I = amount of time-steps that an animation has to be inactive before it can be played again

Equation (4.1) shows that we would like to minimize the laughing energy L_t. The equation is defined in the form of a *recurrence relation* [45], where the laughing energy at time-step t (L_t) builds on its value at the previous time-step $t-1$ (L_{t-1}) and $L_0 = 0$. The laughing energy is defined like this because the actual laughter simulation runs in time-steps, and the laughing builds on to its value from the previous time-step at every time-step, starting with a laughing energy of 0.

The only variable in equation (4.1) that we have control over is a, namely the animation that should be picked to minimize L_t. So at every time-step in the simulation, we check if an optimal animation can be played given the current level of laughing energy and the set of constraints, which will now be motivated:

1. $e(a) < L_t$. This constraint ensures that an animation can only be started when its energy value is lower than the current level of laughing energy, so that high-intensity animations will not be started whenever the laughing energy is low, because low laughing energy values correspond to low-intensity laughter from the audio.

2. $e(a) > e(a_{active}) * c$. In most cases, it is desirable to try and finish an animation before another one is started so no typical motions are cut off in the middle, which means that at most time-steps no animation is selected at all (this would correspond to an *infeasible* solution to the linear program). There are cases however where it is desirable to start an animation while another one is still playing. A typical case would be that a person starts laughing quietly and a corresponding quiet animation is started, but as soon as the quiet animation is started, the person starts laughing really loud. If we would wait for the quiet animation to finish while the person is laughing really loud, the laughing simulation would appear asynchronous, so in order to maintain responsiveness to the sound, a quiet animation sometimes has to be overruled by an intense animation. This is where constraint 2 comes into play, it allows for animations to be overruled by others when it is needed. The constant factor c was added to keep control over how easy an animation can be overruled. If for example $c = 2$, an animation can only be overruled by animations with energy values that are at least twice as large.

3. $i(a) > I$. This constraint was added to make sure that an animation cannot be played again shortly after it has finished (i.e. a has to be inactive for I time-steps before it can be played again), because these kind of repetitive patterns would appear unnatural in the simulation.

As mentioned at the motivation for constraint 2, at most time-steps, no animation is selected at all. However, as soon as there are one or more animations that meet the constraints, the animation with the highest energy value is started. This is because in the case of multiple candidate animations, the laughing energy is high enough for each of them (or they would not be a candidate); so to minimize it, playing the animation with the highest energy value is the best choice.

Given the three constraints and the definition of the laughing energy, the intensity of the full body animations will correspond to the intensity of the input laughter, as motivated below:

1. $L_t \approx 0$. The laughing energy is 0 at the start of the simulation and typically approximates 0 again when the input laughter has been quiet for a short (or longer) while. This is because when the audio is quiet, the motion control signal values will be very low, so the laughing energy will barely get increased, while if there was any laughing energy left, an animation will be started (if it meets the constraints) and consume the rest of the laughing energy (see region 1 in Figure 4.2). As soon as the laughing energy is close to 0, no animation can be started, because of the the constraint $e(a) < L$. This is desirable of course, because when a person stops laughing, no laughter animations should be played.

2. L_t is mild. The laughing energy typically has a mild level of energy when a person is laughing mildly and at the end of a more intense bout of laughter. When a person is laughing mildly, the motion control signal will also show mild values, so the laughing energy will only get increased slowly. While it is increased slowly, only mild-intensity animations will be played because the laughing energy will never reach values high enough

for the high-intensity animations ($e(a) < L$). This is clearly visible in region 2 in Figure 4.2. People tend to 'fade-out' the intensity of their laughter when they have been laughing very loudly. During this fade-out the laughing energy will be too low for high-intensity animations to be started ($e(a) < L$) so it will be the mild and low energy animations that will consume the laughing energy created during the fade-out. Region 3 in Figure 4.2 shows a typical fade-out of an intense bout of laughter.

3. L_t is large. The laughing energy can only get very high when a person is laughing really loud. This is because mild animations that are started at the beginning of an intense bout of laughter (because at that time they meet the requirements, while the laughing energy is still too low for the high-intensity animations ($e(a) < L$)) cannot keep up with the amount of laughing energy that is added by the high values of the motion control signal. The start of an intense bout of laughter is a typical case where the constraint $e(a) > e(a_{active}) * c$ comes into play, because the mild animations that were started should be overruled as soon as the laughing energy is getting too high to maintain responsiveness to the sound. Region 4 in Figure 4.2 shows a typical example of an intense bout of laughter where mild animations are overruled by more intense ones at the start of the bout and where the most intense animations get the laughing energy level back down during the peak of the bout.

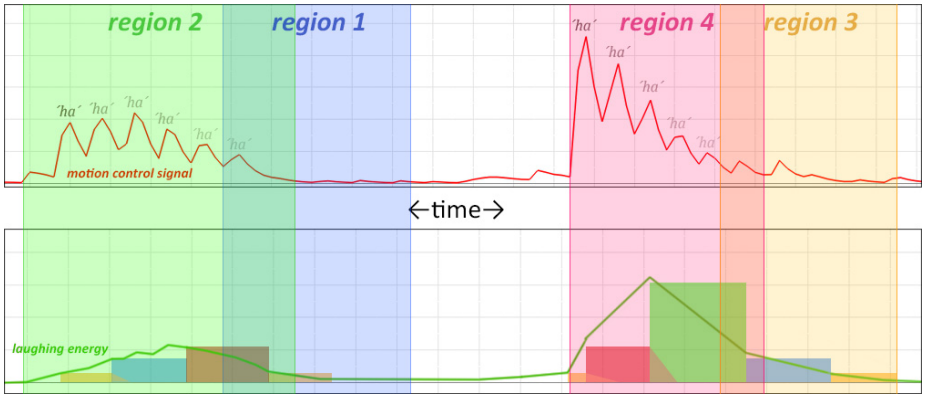

Figure 4.2: This sketch corresponds to the one shown in Figure 4.1. The different regions show typical situations during a laughter simulation. Region 1 shows an example of where the laughing energy gets close to 0. Region 2 shows a mild bout of laughter. Region 4 shows an intense bout of laughter, fading out in region 3. Note that the large green block, representing an intense animation, is started a bit too late to really match the sound. Unfortunately, this inevitable because we cannot take a look ahead in time to see if the person keeps laughing intensely or stops laughing in an instant. However, the constraint $e(a) > e(a_{active}) * c$ allows for animations to be overruled before they have finished, which shortens this delay.

The remaining constraint $i(a) > I$ was not included in the examples above because it is merely a rule to prevent animations from starting short after they have finished and it does not influence the minimization of the laughing energy very much if an animation is ruled out for a short while.

The laughing energy is formulated in the form of a *pseudo*-linear programming problem since the objective function is defined as a recurrence relation rather than a linear function. It was not formulated in the most formal way, defining the laughing energy as a sum of integrals, because the simulation runs in real-time and thus we do not know in advance how the integral will look like. Because the problem is defined in almost the same way it is implemented, the rest of the details of the laughing energy process have been left out and are discussed in the next chapter, as well as other matters that are specific to the implementation of the laughter simulation.

Chapter 5

Implementation

In this chapter, we will discuss what software was used to create the laughter simulation as well as implementation-specific issues that had to be tackled.

For the implementation of our method, we used *Blender* [46], a free and open source 3D animation suite. Because Blender is open source and supports scripting, it was very easy to extend it and add features so we could build a real-time laughter simulation. The complete modeling, texturing and visualization process of the character that was used, was also done in Blender, using Blender's wide variety of modeling and texturing tools and the build-in game engine. The code that controls the simulation is also handled by the game engine, where some parts of the code run at every time-step while other parts are only evaluated once as part of the initialization. Blender's game engine was without a doubt the most important piece of software that was used to create the laughter simulation.

5.1 Shape interpolation

In our simulation, we use shape interpolation for the facial animation and for the breathing animation of the chest and belly, as discussed in chapter 3. Blender has extensive functionality to handle different shapes for one mesh, including options to linearly interpolate and extrapolate these shapes. In Blender, these shapes are called *shapekeys*. Every mesh with shapekeys has a *base* shapekey which represents the original unchanged mesh and a number of shapekeys which each hold a different shape of this mesh as shown in Figure 5.1. Linear extrapolation allows the minimum and maximum influence to be respectively lower than zero and higher than one. This feature is used for the shapes of the face to allow for exaggerated facial poses.

The influence of each shape can be adjusted with a slider. However, these sliders cannot be accessed from within Blender's game engine where the simulation is running. Fortunately, these sliders can be animated by creating key frames with different values for the sliders at different moments in time, and in Blender every value that can be animated, can also be influenced by a *driver*. Basically, a driver is a link between two values so one value can be controlled by the other one. In our case we created bones and used the length of these bones to drive the influence of the shapekeys. The bones *can* be accessed from within the game engine, so by adjusting their lengths during the simulation, the shape

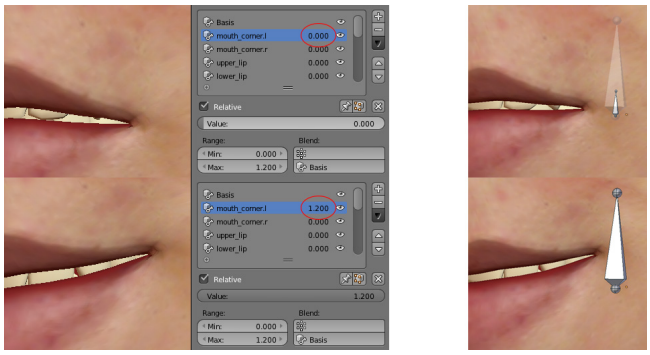

Figure 5.1: Shapekeys in action. On the left, the mouth-corner is directly controlled by the value of the shapekey while on the right, it is controlled by the length of a bone.

of the character can be adjusted, which is also shown in Figure 5.1.

5.2 Live sound

As input for the laughter simulation, either prerecorded or live sound of laughter can be used. Blender includes functionality to handle prerecorded audio files, but does not support the use of a microphone. To handle the microphone input stream, we extended Blender with *PortAudio* [47], a free, cross-platform, open-source, audio I/O library. To convert this input stream to a usable signal, we used *SoundAnalyse* [48], a module that provides functions to analyze sound chunks to detect amplitude and pitch. The next piece of code shows how the raw amplitude is processed in order to create a usable motion control signal to drive the simulation:

```
for every timestep:
   #extract the amplitude from the sound using PortAudio
   amplitude = getAmplitude(microphoneStream)

   #use maximum and minimum amplitude so far to map the amplitude to the range [0,1]
   if amplitude > maxAmp: maxAmp = amplitude
   if amplitude < minAmp: minAmp = amplitude
   motionSignal = (amplitude - minAmp) / (maxAmp - minAmp)

   #decrease maxAmp and increase minAmp by a fixed amount to handle changes in volume
   fallof = 0.001
   maxAmp -= fallof
   minAmp += fallof

   #make sure the motion control signal can not rise or fall faster than specified
   ampRise = 0.4, ampFall = 0.2
   if motionSignal > previousMotionSignal:
      motionSignal = min(motionSignal, previousMotionSignal + ampRise])
   else:
      motionSignal = max(motionSignal, previousMotionSignal - ampFall)

   #update previous motion control signal
```

```
23      previousMotionSignal = motionSignal
24      #tweak the motion control signal for better results
25      motionSignal = pow(motionSignal, 2)
```

As described in section 3.1.1, line numbers 6 to 8 show the mapping of the amplitude to the range [0,1] using the maximum and minimum amplitude so far. Also note that at line numbers 11 to 13, the maximum and minimum amplitudes are decreased and increased respectively to cope with changes in input volume, as explained in section 3.1.1.

Line numbers 16 to 20 demonstrate the use of the rise and fall parameters. Every time-step, the motion control signal is allowed to get increased by up to 0.4 and decreased by up to 0.2. The fall parameter is intentionally set to be lower than the rise parameter, because we noticed that the signal gave more natural-looking results when it had a minor 'fade-out'.

We noticed that using a linear mapping, high motion control signal values (0.6 ... 0.8) occurred too often, which resulted in high-intensity animations to be started too easily. To tone this down but maintain the range of [0,1], the signal is raised to the power of 2 at line number 25.

Using this implementation, the input sound is dynamically mapped to a motion control signal in the range [0,1], takes care of noisy input, and copes with changes in minimum/maximum input volume and is now ready to be used for the simulation.

5.3 Breathing and facial animations

As mentioned in section 3.2 and 5.1, we use different shapes of our mesh to control breathing and facial animations. These shapes are controlled by the length of bones which can be directly influenced from within Blender's game-engine. However, as mentioned in 3.2 and 3.3, the motion control signal needs to be smoothed out a bit for each shape to have a natural-looking effect. The next pieces of code show how this is done and what parameter values were used:

```
1   #this class is used to smooth and re-scale out the motion control signal
2   class Smoother:
3       #constructor with default parameters
4       def __init__(self, rise=0.5, fall=0.2, minE=0.0, maxE=1.0):
5
6           self.rise = rise
7           self.fall = fall
8           #minE and maxE are used to re-scale the motion control signal
9           #as shown in the smooth() function below
10          self.minE = minE
11          self.maxE = maxE
12          self.range = maxE - minE
13
14          #the output signal
15          self.e = 0
16
17      #input:  original motion control signal
18      #output: motion control signal subject to a rise and
19      #        fall parameter & scaled to [min, max]
```

```
20      def smooth(self, motionSignal):
21          #allow the signal to rise and fall only by a set amount
22          if motionSignal > self.e:
23              self.e = min(motionSignal, self.e + self.rise)
24          else:
25              self.e = max(motionSignal, self.e - self.fall)
26
27          scaled = self.e
28          #re-scale the motion control signal so that:
29          #every original value smaller than minE will be mapped to 0
30          if self.e < self.minE:
31              scaled = 0.0
32          #every original value greater than maxE will be mapped to 1
33          elif self.e > self.maxE:
34              scaled = 1.0
35          #and every original value that lays in between minE and maxE
36          #gets scaled accordingly
37          else:
38              scaled = (self.e - self.minE) / self.range
39
40          return scaled
```

Note that not only the rise and fall parameters are applied in this class, but also a re-scaling of the motion control signal takes place (lines 27-38). This feature has been added because in case of the smile, it would look awkward when the character would only smile at 100% if the motion control signal is also at 100%. This is why, as shown below, the smile 'smoother' has a *maxE* value of 0.3, so the smile will be at 100% when the motion control signal is only at 30%. Note that whenever the motion control signal is higher than 30%, the smile will remain at 100% and not scale up to 330%, which would just look silly, as shown in Figure 5.2.

```
#the smoothers that return a smoother and re-scaled vesion of the sound curve
jawSmoother = Smoother(rise = 0.03, fall = 0.01, minE = 0.0, maxE = 0.8)
smileSmoother = Smoother(rise = 0.03, fall = 0.001, minE = 0.0, maxE = 0.3)
chestSmoother = Smoother(rise = 0.2, fall = 0.1, minE = 0.0, maxE = 1.0)
bellySmoother = Smoother(rise = 0.1, fall = 0.05, minE = 0.0, maxE = 1.0)
```

As mentioned in section 3.3.1, the smile smoother has a very low fall value to make sure that the smile of the character will fade away slowly in stead of disappearing in an instant. Also, the belly smoother has lower rise and fall parameters than the one of the chest in order to create a small delay between them to create a slightly more organic and natural look of the breathing. This effect is hardly noticeable, but easily implemented and in some cases might just add a subtle touch to the simulation.

Figure 5.2: Shapes of the face that are exaggerated too much will result in awkward-looking facial expressions.

5.4 Laughing energy

The next piece of code shows what happens to the laughing energy at each time-step, leaving out the energy that is consumed by the active animation which is discussed in section 5.5:

```
1 for each timestep:
2     #decrease laughing energy with a fixed amount.
3     laughingEnergy -= fixedDecreasingFactor
4
5     #cap the laughing energy if it exceeds its maximum
6     laughingEnergy = min(maxLaughingEnergy, laughingEnergy)
7
8     #increase laughing energy with the motion control signal
9     laughingEnergy += motionSignal * dt
```

Line 3 shows that the laughing energy is decreased by a fixed amount every time-step. This is particularly useful in cases of silence, because it can occur that an animation is running, but after it stopped there is still a significant amount of laughing energy left and if the person stopped laughing during this animation, the system would still start a new animation to get rid of the remaining laughing energy. So if the laughing energy gets decreased by a fixed amount every time-step, regardless of any animation that is playing, most of these 'silent laughing' cases will be avoided.

At line 6, we make sure that the laughing energy will not exceed a fixed maximum value. This had to be added to make sure that when the sound is very loud for a longer period of time, the laughing energy would not reach such a high value that not even the most intense animations could get it down again. In the end of course, even without a maximum, the laughing energy would go back to zero, but if the laughing energy is too high, the system will try to get it down using only the most intense animations even while the person may not be laughing that loud any longer.

Line 9 simply adds the current motion control signal value (times the elapsed time), to the current amount of laughing energy.

5.5 Selecting animations

The full body animations that have been used for the simulation each have their own length and energy. As mentioned in chapter 4, this energy value corresponds to the visual intensity of the animation, so for example a 'roll over floor laughing' animation would have a high energy value while mild swaying of the upper body would have a low energy value. To manage these animations and to make sure that they can easily be used in the selection process, the following class was created:

```
class Action:
    #constructor
    def __init__(self, name="", length=0, energy=0):
        self.name = name
        self.length = length
        self.energy = energy
        self.active = False
        self.inactive_tics = 10000
```

Note that this class also keeps track of the time that an animation has been inactive by keeping track of the number of inactive simulation tics. This is information is used in the selection process.

For every animation, one of these actions is created. The following 10 animations were used for our simulation:

```
#create animation actions to be used in the simulation
animationActions = [Action("lean_front1",22,energy=5),
                    Action("lean_front2",42,energy=10),
                    Action("lean_back1",22,energy=5),
                    Action("lean_back2",36,energy=10),
                    Action("clap_hands",46,energy=15),
                    Action("hands_on_belly",73,energy=15),
                    Action("point1",69,energy=20),
                    Action("point2",78,energy=20),
                    Action("slap_leg",80,energy=25),
                    Action("stop_it",79,energy=30),
                    Action("rofl",117,energy=40)]
```

The order of these actions in this list is unimportant because whenever there are multiple actions that meet the constraints, the one with the highest energy value is picked, as discussed in section 4.2. The following piece of code shows the selection process of the actions:

```
1  ############ ANIMATION SELECTION ##########
2  random.shuffle(animationActions)
3  bestAction = None
4  for act in animationActions:
5
6     #activate an animation if it meets the requirements
7     if (laughingEnergy > act.energy and not act.active and
8           act.energy > activeAnimation.energy * overruleThreshold and
9           act.inactive_tics > minInactiveTics):
10
11       #if there is more than one candidate action, pick the one with the highest energy
12       if (bestAction == None or act.energy > bestAction.energy)
13          bestAction = act
14
15    if act.active:
16       #deactivate the action if it has finished
17       if(Blender.getActionFrame(act) == act.length) :
18          act.active = False
19          Blender.stopAction(act)
20       else: #decrease the laughing_energy with the energy of the current animation
21          laughingEnergy -= act.energy * dt
22
23       #set the number of inactive tics to 0 if the action is active...
24       act.inactive_tics = 0
25    else:
26       #increase the number of tics this action has been inactive
27       act.inactive_tics += 1
28
29 if bestAction != None:
30    #deactivate the last action
31    if not lastPlayedAction == None:
32       lastPlayedAction.active = False
33
34    #play the animation with a random speed to introduce some variation
35    Blender.playAction( act.name, blendin=10, speed=random.uniform(0.9, 1.4) )
36    act.active = True
37    lastPlayedAction = act
38    activeAnimation = act
```

At line 2, the array of actions is randomly shuffled. This is to make sure that actions that have the same energy value (like for example *lean_front1* and *lean_back1*) do not always occur in the same order, because if they would only one of them would be selected as the best candidate every time.

Lines 7 to 9 show the constraints that have to be met for an action to be selected, as explained in section 4.2. Note that when the animation is actually started at line 35, the *blendin* and *speed* options are used. The *blendin* option enables us to set the number of frames that has to be used to blend from the animation that was already playing to the animation that was just started. This ensures a smooth transition using linear pose blending when the poses of the animations do not match. The *speed* option lets us play an animation at a given speed. A random value between 0.9 and 1.4 is picked to create some variation in the animations.

Lastly, at line 15 we check wether the animation is active and if it is, the laughing energy is decreased by the energy of the active animation as described in section 4.1. The rest of the code is trivial and needs no further explanation.

Using this implementation, the animations are selected in a way that their intensity matches the intensity of the input sound. Lastly it should be noted that an *idle* animation is playing as soon as no laughter animation is playing, so the character does not seem frozen in cases if silence.

5.6 Results

During the implementation phase, the simulation was run frequently to see how every parameter would effect the end result. Every feature (like breathing or smile intensity) was first isolated in the simulation to fine-tune their parameters. For example, to make sure that the chest and belly work correctly, it is very inconvenient if there are any animations playing that change the pose of the character. Afterwards we checked if all the features also worked well when they were turned on simultaneously and changed their parameters if needed. Video 3 from the playlist shows the results of our simulation using prerecorded and live sound of laughter.

Chapter 6

Conclusion and discussion

In the final chapter of this report, the limitations and future work will be discussed in section 6.1 after which we will finish with the conclusions in section 6.2.

6.1 Limitations and future work

Our system has not been validated, but this could be done running a user-study that provides the user with different kinds of characters (preferably male and female), a list of prerecorded laughter sounds and the option for the user to laugh in a microphone. Emphasis has to be made on different levels of intensity in the laughter sounds in order for the users to give a fitting rating of the naturalness of the simulated laughter, because it is mostly about the responsiveness to the sound rather than how convincing the animations look. It might also be a good idea to create an extra view that is zoomed in on the face so the user can clearly see what is going on in the face during the simulation.

In our laughter simulation, the breathing and facial animations are synthesized, but the full body animation relies on motion capture recordings or key-framed animations. In our case, we used a set of 10 key-framed animations, varying in intensity, to achieve a sufficient variation of full body animations in the laughter simulation. Furthermore, we randomized the speed of the animations to introduce even more variety. However, this still requires at least 10 animations to be created to prevent clear repetitive patterns in the simulation. Also, the quality of these animations contributes to a significant part of the naturalness of the simulation. In order to improve on this, more procedural ways of animating the body could be considered. For example, the loss of control that sometimes occurs when people can't stop laughing could be simulated by applying *ragdoll physics* [49] on some parts of the body (like the arms or the spine). Also, during this loss of control, people tend to look for objects (wall, table, other people) to support themselves and with *inverse kinematic* constraints on the hands and feet (so they can be controlled directly during the simulation), this effect could interactively be simulated.

In the current state of our system, the energy of the full body animations (representing their intensity) was manually assigned to them. This energy value could also be extracted from the animation itself by defining a function on

the mesh or skeletal structure of the character that measures the amount of movement expressed in energy for any given animation.

As mentioned in section 3.3.1, only *one* shape was used to represent a smiling face. Despite the fact that the facial details are difficult to see when the character is moving his whole body during a bout of laughter, this could be considered a limitation. This is not only because one person does not just have one kind of smile, but also because one particular smile is usually not created in a linear fashion, meaning that the different muscles that are used to obtain a smile do not necessarily move synchronized and symmetric. The facial part of our laughter simulation could be improved by separately driving each deformation that is caused by the contraction of a muscle that creates a smile, possibly using the FACS of Ekman and Friesen, which will allow for variation within a single smile, creating more diversity and naturalness in the face.

The system was build around one male character and not tested with other characters. Although it took quite some time to set up the system for this one character, most of this work could be automated. The full body laughter animations could be stored in a database and be retargeted to the skeletal structure of other humanoid characters. The shapes that were used for the chest and belly could be automatically generated for the mesh of another character using a function that takes into account the lung volume and other physical properties that play a part in the expulsion of the chest and belly during breathing. The shape that was used for the smile could be generated by implementing an extension to the character's face that can handle Ekman and Friesen's FACS. A smile is now simply defined by a code that is the same for each character and tells what muscles are activated to create the smile.

Our laughter simulation relies on the assumption that the input sound is the sound of a person that is already laughing. This means that, for example, when someone starts *crying* in the microphone, the character will just cheerfully laugh along. So in order for our system to be used in games or in serious applications, laughter *detection* must be included. Fortunately, quite some research has been done on laughter detection and for example the results of Truong and Van Leeuwen [16] should work fine in conjunction with our system, resulting in a system that offers interesting possibilities for games and movies involving virtual characters:

- Interactive laughter can be simulated in games where players control an avatar. More and more games are making use of the internet to create interesting multi-player modes. These games, and especially the ones that are played on a PC, increasingly use live voice-streams for the players to communicate. For all of these games were the players control human-like characters, our way of simulating laughter combined with laughter detection could add a whole new dimension to the interaction with the characters.

- Games are not the only real-time applications that involve virtual characters. In the way that our system contributes to a higher level of interactivity to games, it can also work for more serious applications like for example the *TARDIS* [50] project, which aims at *T*raining young *A*dult's *R*egulation of emotions and *D*evelopment of social *I*nteraction *S*kills. It works by simulating and analyzing a job interview situation with the help

of a real-time virtual character that guides the user through the process by reacting to his sound and motions. It is desirable for this virtual character to be as human-like as possible so the user will not get distracted from his tasks. Simulating emotions or non-verbal ways of communicating like laughter will contribute to a more human-like interaction with the virtual character, because for example funny situations might occur during the simulation of a job interview in which it would be awkward if the virtual character does not laugh along.

- Also in movies involving simulated characters, animators might save a lot of time using our system. Instead of having to animate the laughter of a character by hand every time, an animator could just take the laughter sound from the voice actor and simulate the laughter using our system. In movies however, the animations are usually very specific to their current situation. Then again, our system could simulate a basis of the laughter after which the animator would only need to tweak it here and there instead of having to start from the ground up. Furthermore, the simulation does not have to run in real-time, so the algorithm could be optimized by looking ahead in time to give better results.

It might even be possible to simulate other emotions using our system. It has to be noted though, that our laughter simulation relies on very specific characteristics of the sound that are typical to laughter and that happen to match with specific motions of the body and face. So as long as a clear connection between the sound of an emotion and the motion that comes with it can be found, it could be simulated using our system. Example emotions that satisfy this requirement would be screaming of anger and crying.

6.2 Conclusions

The results show that motions typical to laughter, both on the body as well as in the face, can be simulated on a virtual character in real-time by driving motions of the torso caused by the breathing behavior during laughter and facial expressions using a motion control signal extracted from prerecorded or live audio from a laughing person and using this signal to build up laughing energy so an appropriate full body animation can be selected. Although there is still some work to do before the results of this research can be applied in the industry of games or any other application that involves interaction with virtual characters, it is a big step towards creating virtual characters that can laugh in a responsive and natural way, which makes them more fun and more natural to interact with.

Bibliography

[1] League of Legends. http://euw.leagueoflegends.com/, a multiplayer online battle arena video game developed and published by riot games. Accessed: August 2014.

[2] Watch Dogs. http://watchdogs.ubi.com/, an open world action-adventure video game developed by ubisoft montreal. Accessed: August 2014.

[3] Elizabeth Holt. The last laugh: Shared laughter and topic termination. *Journal of Pragmatics*, 42(6):1513–1525, 2010.

[4] Daniel O Dugan. Laughter and tears: best medicine for stress. In *Nursing forum*, volume 24, pages 18–26. Wiley Online Library, 1989.

[5] Willibald Ruch and Paul Ekman. The expressive pattern of laughter. *Emotion, qualia, and consciousness*, pages 426–443, 2001.

[6] Paul Ekman and Wallace V Friesen. *Unmasking the face: A guide to recognizing emotions from facial clues*. Ishk, 2003.

[7] Christopher Creed and Russell Beale. User interactions with an affective nutritional coach. *Interacting with Computers*, 24(5):339–350, 2012.

[8] The Sims. http://www.thesims.com/, a casual life simulation video game series developed by electronic arts. Accessed: August 2014.

[9] Wijnand van Tol and A Egges. Real-time 3d crying simulation. 2009.

[10] Justine Cassell, Timothy Bickmore, Mark Billinghurst, Lee Campbell, Kenny Chang, Hannes Vilhjálmsson, and Hao Yan. Embodiment in conversational interfaces: Rea. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 520–527. ACM, 1999.

[11] Stefan Kopp, Bernhard Jung, Nadine Lessmann, and Ipke Wachsmuth. Max - a multimodal assistant in virtual reality construction. *KI*, 17(4):11, 2003.

[12] Cantoche. http://www.cantoche.com, engage and assist your online users with 3d talking avatars. Accessed: August 2014.

[13] Haptek. http://www.haptek.com, lets you create your own interactive, emoting 3-d characters using simple photographs and your own voice. Accessed: August 2014.

[14] Anton Nijholt. Embodied agents: A new impetus to humor research. 2002.

[15] ILHAIRE. http://www.ilhaire.eu, incorporating laughter into human avatar interactions: Research and experiments. Accessed: August 2014.

[16] Khiet P Truong and David A Van Leeuwen. Automatic discrimination between laughter and speech. *Speech Communication*, 49(2):144–158, 2007.

[17] Mary Tai Knox and Nikki Mirghafori. Automatic laughter detection using neural networks. In *INTERSPEECH*, pages 2973–2976, 2007.

[18] Stavros Petridis and Maja Pantic. Is this joke really funny? judging the mirth by audiovisual laughter analysis. In *Multimedia and Expo, 2009. ICME 2009. IEEE International Conference on*, pages 1444–1447. IEEE, 2009.

[19] Harry J Griffin, Min SH Aung, Bernardino Romera-Paredes, Ciaran McLoughlin, Gary McKeown, William Curran, and Nadia Bianchi-Berthouze. Laughter type recognition from whole body motion. In *Affective Computing and Intelligent Interaction (ACII), 2013 Humaine Association Conference on*, pages 349–355. IEEE, 2013.

[20] Eva Lasarcyk and Jürgen Trouvain. Imitating conversational laughter with an articulatory speech synthesizer. 2008.

[21] Shiva Sundaram and Shrikanth Narayanan. Automatic acoustic synthesis of human-like laughtera). *The Journal of the Acoustical Society of America*, 121(1):527–535, 2007.

[22] Jürgen Trouvain. Segmenting phonetic units in laughter. In *Proceedings of the 15th International Congress of Phonetic Sciences. Barcelona: Universitat Autònoma de Barcelona*, pages 2793–2796, 2003.

[23] Darren Cosker and James Edge. Laughing, crying, sneezing and yawning: Automatic voice driven animation of non-speech articulations. *Proceedings of Computer Animation and Social Agents, CASA*, 2009.

[24] Jérôme Urbain, Elisabetta Bevacqua, Thierry Dutoit, Alexis Moinet, Radoslaw Niewiadomski, Catherine Pelachaud, Benjamin Picart, Joëlle Tilmanne, and Johannes Wagner. Avlaughtercycle: An audiovisual laughing machine. In *Proceedings of the 5th International Summer Workshop on Multimodal Interfaces*, pages 79–87. Citeseer, 2009.

[25] Paul C DiLorenzo, Victor B Zordan, and Benjamin L Sanders. Laughing out loud: control for modeling anatomically inspired laughter using audio. *ACM Transactions on Graphics (TOG)*, 27(5):125, 2008.

[26] Yao-Yang Tsai, Wen-Chieh Lin, Kuangyou B Cheng, Jehee Lee, and Tong-Yee Lee. Real-time physics-based 3d biped character animation using an inverted pendulum model. *Visualization and Computer Graphics, IEEE Transactions on*, 16(2):325–337, 2010.

[27] Petros Faloutsos, Michiel Van de Panne, and Demetri Terzopoulos. Composable controllers for physics-based character animation. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 251–260. ACM, 2001.

[28] Larry Gritz and James K Hahn. Genetic programming evolution of controllers for 3-d character animation. *Genetic Programming*, 97, 1997.

[29] Fábio Roberto Miranda, João Eduardo Kögler Jr, Emílio Del Moral Hernandez, and Márcio Lobo Netto. An artificial life approach for the animation of cognitive characters. *Computers & Graphics*, 25(6):955–964, 2001.

[30] Keith Grochow, Steven L Martin, Aaron Hertzmann, and Zoran Popović. Style-based inverse kinematics. In *ACM Transactions on Graphics (TOG)*, volume 23, pages 522–531. ACM, 2004.

[31] Sybren A Stüvel and A Egges. Stride space: Humanoid walking animation interpolation using 3d delaunay databases. 2010.

[32] Katherine Pullen and Christoph Bregler. Motion capture assisted animation: Texturing and synthesis. *ACM Transactions on Graphics (TOG)*, 21 (3):501–508, 2002.

[33] Thanh Giang, Robert Mooney, Christopher Peters, and Carol O'Sullivan. Real-time character animation techniques. 2000.

[34] Paul Ekman and Wallace V Friesen. Facial action coding system: A technique for the measurement of facial movement. palo alto, 1978.

[35] Demetri Terzopoulos and Keith Waters. Analysis and synthesis of facial image sequences using physical and anatomical models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 15(6):569–579, 1993.

[36] Eftychios Sifakis, Andrew Selle, Avram Robinson-Mosher, and Ronald Fedkiw. Simulating speech with a physics-based facial muscle model. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 261–270. Eurographics Association, 2006.

[37] Demetri Terzopoulos and Keith Waters. Physically-based facial modelling, analysis, and animation. *The journal of visualization and computer animation*, 1(2):73–80, 1990.

[38] Wikipedia Loudness. http://en.wikipedia.org/wiki/Loudness, wikipedia, the free encyclopedia. Accessed: August 2014.

[39] Wikipedia Amplitude. http://en.wikipedia.org/wiki/Amplitude, wikipedia, the free encyclopedia. Accessed: August 2014.

[40] Wikipedia Root mean square. http://en.wikipedia.org/wiki/Root_mean_square, wikipedia, the free encyclopedia. Accessed: August 2014.

[41] Mario Filippelli, Riccardo Pellegrino, Iacopo Iandelli, Gianni Misuri, Joseph R Rodarte, Roberto Duranti, Vito Brusasco, and Giorgio Scano. Respiratory dynamics during laughter. *Journal of Applied Physiology*, 90 (4):1441–1446, 2001.

[42] Wikipedia Extrapolation. http://en.wikipedia.org/wiki/Extrapolation, wikipedia, the free encyclopedia. Accessed: August 2014.

[43] Kazuo Tsubota, Seiichiro Hata, Yukio Okusawa, Fuminobu Egami, Tomohiro Ohtsuki, and Katsu Nakamori. Quantitative videographic analysis of blinking in normal subjects and patients with dry eye. *Archives of ophthalmology*, 114(6):715–720, 1996.

[44] Wikipedia Linear programming. http://en.wikipedia.org/wiki/Linear_programming, wikipedia, the free encyclopedia. Accessed: August 2014.

[45] Wikipedia Recurrence relation. http://en.wikipedia.org/wiki/Recurrence_relation, wikipedia, the free encyclopedia. Accessed: August 2014.

[46] Blender. http://www.blender.org, a free and open source 3d animation suite. Accessed: August 2014.

[47] PortAudio. http://www.portaudio.com/, a free, cross-platform, open-source, audio i/o library. Accessed: August 2014.

[48] SoundAnalyse. https://code.google.com/p/pygalaxy/wiki/SoundAnalyse, a module that provides functions to analyse sound chunks to detect loudness and pitch. Accessed: August 2014.

[49] Wikipedia Ragdoll physics. http://en.wikipedia.org/wiki/Ragdoll_physics, wikipedia, the free encyclopedia. Accessed: August 2014.

[50] Keith Anderson, Elisabeth Andre, Tobias Baur, Sara Bernardini, Matthieu Chollet, Evangelia Chryssafidou, Ionut Damian, Arjan Egges, Patrick Gebhard, Hazael Jones, Magalie Ochs, Catherine Pelachaud, Kaska Porayska-Pomsta, Paola Rizzo, and Nicolas Sabouret. *The TARDIS framework: intelligent virtual agents for social coaching in job interviews*. Lecture Notes in Computer Science. Springer, 2013. ISBN 9783319031606. doi: 10.1007/978-3-319-03161-3_35.