

LiU-ITN-TEK-A--12/077--SE

# Improving the visual quality of computer generated crowds for feature film production

Dan Englesson

2012-12-03



**Linköpings universitet**  
TEKNISKA HÖGSKOLAN

LiU-ITN-TEK-A--12/077--SE

# **Improving the visual quality of computer generated crowds for feature film production**

Examensarbete utfört i Medieteknik  
vid Tekniska högskolan vid  
Linköpings universitet

**Dan Englesson**

Handledare Jonas Unger  
Examinator Jonas Unger

Norrköping 2012-12-03

## Upphovsrätt

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida <http://www.ep.liu.se/>

## Copyright

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: <http://www.ep.liu.se/>

## Abstract

In this thesis, several different methods for improving the visual quality of a crowd system for feature film production are presented. The implementations were made for the *Moving Picture Company* for their crowd system software *ALICE*. The methods implemented and presented in this thesis have made improvements on the basic skinning methods by introducing twisting and scaling of the bones with a method called *Stretchable and Twistable Bone Skinning*. Also, complex and detailed deformations on crowd characters such as muscle bulging and wrinkles on clothes, are now possible due to the implementation of *Pose Space Deformation methods* which interpolates stored complex and time-consuming deformations onto a mesh. Several known pose space deformation methods were implemented and compared and resulted in an other method trying to reduce the current pose space deformation method's limitations and is presented as *Three-Joint Pose Space Deformation*. In this thesis, a novel method for *Dynamic Pose Space Deformation* is also presented and a dynamic spring-damper approximation deformation method was also implemented, enabling crowd characters to have dynamic effects, such as jiggling of fat and muscle bouncing, due to joint accelerations.

### **Acknowledgment**

I would like to especially thank Rob Pieké for being such a great mentor and also thank everyone at the Moving Picture Company for their friendliness. Lastly, but most importantly, I would also like to thank my family and friends for their love and support.

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	The Moving Picture Company (MPC)	3
1.2	Motivation	3
1.3	Aim	4
1.4	Structure of the Report	4
<b>2</b>	<b>Background survey</b>	<b>5</b>
2.0.1	Improving basic skinning methods	5
2.0.2	Example-based skinning methods	6
<b>3</b>	<b>Background theory</b>	<b>7</b>
3.1	Skeletal animation	7
3.1.1	The skeleton structure	7
3.1.2	Animating the skeleton	10
3.2	Skinning Methods	11
3.2.1	Linear Blend Skinning	11
3.2.2	Dual Quaternion Skinning	13
<b>4</b>	<b>Implementation</b>	<b>16</b>
4.1	Stretchable and Twistable Bone Skinning	16
4.1.1	Endpoint weights	17
4.1.2	The Stretchable and Twistable Bone skinning method	18
4.1.3	Usage	21
4.2	Pose Space Deformation	21
4.2.1	The Shepard's Method	26
4.2.2	Radial Basis Functions	27
4.2.3	Weighted Pose Space Deformation	29
4.2.4	Pose space deformation with a three-joint local lookup	29
4.2.5	Usage	31
4.3	Dynamic Pose Space Deformation	32
4.3.1	Joint accelerated dynamic deformation	33
4.3.2	Spring-damper function generation	34
4.3.3	Usage	37
<b>5</b>	<b>Results</b>	<b>39</b>
5.1	Stretchable and Twistable Bone Skinning	39
5.2	Pose Space Deformation	40

5.3	Dynamic Pose Space Deformation . . . . .	43
<b>6</b>	<b>Conclusion</b>	<b>45</b>
6.1	Improving the basic skinning methods . . . . .	45
6.2	Pose Space Deformation . . . . .	46
6.3	Dynamic Pose Space Deformation . . . . .	47
6.4	Future work . . . . .	47
6.4.1	Improving the basic skinning methods . . . . .	47
6.4.2	Pose Space Deformation . . . . .	48
6.4.3	Dynamic Pose Space Deformation . . . . .	50

# Chapter 1

## Introduction

### 1.1 The Moving Picture Company (MPC)

The Moving Picture Company, MPC for short, is a post production company that creates visual effects and computer animations for commercials and feature films in the heart of London. MPC is a global company with facilities in London, Vancouver, Los Angeles, New York and Bangalore. They have done numerous visual effects and animations for over 50 feature films such as *Prometheus*, *Pirates of the Caribbean: On Stranger Tides*, *Harry Potter and the Deathly Hallows Part 1 and 2*, *The Chronicles of Narnia: The Voyage of the Dawn Treader* and *Robin Hood* just to name a few. MPC is renowned for their crowd works for feature films, which all have been made possible by using their in-house crowd software *ALICE*. MPC always wants to stay ahead of the game and are therefore always keen on having their softwares such as *ALICE* updated with the latest technology to be able to meet the customers new demands.

### 1.2 Motivation

In previous years, crowds were usually very large and placed in the background, and then live action crowds or fully and complexly rigged characters, so called hero-level characters, were placed close to the camera. The computer generated crowds were placed in the background because their visual quality was not good enough since in order to be able to have a lot of crowd characters on screen, the computations for each character needed to be as fast as possible. It was done by sacrificing visual quality for speed. Since the crowds had to have less visual quality in order to be able to have as many characters as possible on screen, they were usually placed in the background so the loss of visual quality were less visible. However, now days the trend and demand tend to go towards having the crowd characters close to the camera and in some cases the crowd characters would even be placed closer to the camera than the hero-level characters. The crowd characters are, however, often too many to be able to have a complex hero-level rig for each crowd character, but yet so close to the camera so the compromises made for quantity over quality are showing.

By placing the crowd characters closer to the camera, the simple skinning methods built for quantity over quality does not hold anymore. In order to face the new demands of having smaller



crowds with better quality that can be put closer to the camera instead of a large crowd that only works in the background in new feature film productions, MPC's crowd simulation software *ALICE*, needed to be updated.

### 1.3 Aim

The aim of this master thesis was to update MPC's crowd system *ALICE* so that the visual quality of the crowd characters were increased without increasing the computational cost drastically. The computational cost must be strictly less than a hero-rigs computational cost but would ideally not increase much from the current crowd system's computational cost. The specific areas of investigation were to improve the quality for basic skinning methods by removing artifacts caused by basic skinning methods and also to use example-based deformation techniques to apply high-quality details, such as wrinkles on skin and clothes, from a hero-rig directly onto a basic crowds rig without having to have advanced rigs or simulations at run-time, enabling high-detailed deformations at a fraction of the cost compared to if it were to be simulated or have an advanced rig applied at run-time.

### 1.4 Structure of the Report

The report is structured in the following way:

- In chapter 2, a background survey is presented which describes the recent works that have been done on improving the skeletal animation's visual quality by using example-based techniques and also by adding additional information to the basic skinning methods.
- Chapter 3 will present the background theory that is necessary in order to better understand and follow the rest of this report.
- Chapter 4 goes in detail through the algorithms implemented for this master thesis work.
- Chapter 5 will present the results of the speed and the visual quality of the methods implemented.
- Chapter 6 goes through the findings and the conclusions drawn about the implemented methods and thoughts on their limitations and future works are presented as well.

# Chapter 2

## Background survey

Basic skinning methods, Linear Blend Skinning and Dual Quaternion Skinning, particularly Linear Blend Skinning has some artifacts such as the collapsing elbow and candy-wrapper effect, see chapter 3 for more information. In order to reduce these artifacts on hero-level characters extra bones or shape interpolation techniques[1][2] are used. The dual quaternion skinning algorithm however reduces the candy wrapper effect more efficiently than linear blend skinning, but is a bit slower, which have been shown by Kavan et al. [3].

### 2.0.1 Improving basic skinning methods

The dual quaternion skinning method deals with the twisting and rotation deformation much better than the Linear Blend Skinning method, however, it still gives some visible artifacts when rotation and twists occur. Therefore, these two methods need to be improved in order to give better visual quality.

There are several different techniques that uses Linear blend skinning or Dual Quaternion skinning as an underlying method but adds some more functionalities to them. For instance, the work from Jacobson et al. [4] on Stretchable and twistable bone skinning (STBS) removes the candy-wrapper effect almost completely by decomposing the formula for linear blend skinning and adds an extra rotation matrix and scale factor and scales both with a weighted translation per vertex, where the weight depends on where on the bone it is located. It handles the candy wrapper effect much better than the linear blend skinning and also the dual quaternion skinning method.

Scaling of bones are often not used because of unwanted overshoots that appears at for instance a hand if the forearm bone is scaled with linear blend skinning or dual quaternion skinning. However, with stretchable and twistable bone skinning the overshoot is gone, making scaling of bones an interesting possibility and gives more artistic freedom.

Mohr et al. [1] extends the linear blend skinning method by introducing extra joints that are placed and rotated halfway between two joints on bones such as, for instance, on the under arm, giving it a more natural twisting and reducing the candy-wrapper effect.

## 2.0.2 Example-based skinning methods

Many example-based skinning methods exist where the key concept of these methods are to have a set of examples stored in a database and then at run time lookup into the database and interpolate, depending on a specified key, such as a mesh or skeleton, the examples that closest matches that run time mesh/pose. One example-based method is DrivenShape [5] which only uses one mesh as a key into the database and the other mesh as a target mesh, no skeleton is used at all. For instance, the key into the database would be the skin mesh and the mesh to be interpolated would be the cloth mesh.

Methods that use a skeleton as key and a mesh to be interpolated as input are; Pose-Space Deformation(PSD)[2], which have gained popularity in the film industry enabling crowds to have more complex deformations for a slightly longer time and cost compared to linear blend skinning, but still for smaller time and cost than if a full hero rig would have been applied. Other methods are Weighted Pose Space Deformation(WPSD)[6] which is pretty much the same method as normal pose space deformation but is a bit more accurate at a price of being slower than normal pose space deformation, Multi-Weighted Enveloping (MWE) [7] and context-aware skeletal shape deformation[8] are also two other example-based methods. Multi-Weighted Enveloping is very similar to linear blend skinning but differs when it comes to the weights. The Multi-Weighted Enveloping method has for each entry one corresponding weight in the transformation matrix compared to linear blend skinning that only has one weight per transformation matrix. The difficulty for this method is that compared to linear blend skinning, the weights cannot be weighted by hand. Instead a least square fit on a linear equation system must be solved to approximate the weights, where sample-poses are posed by a hero-level rig or hand tweaked to look good. A good thing about this method is that it does not need to have all sample poses loaded at run-time, it only needs to have the weight coefficients loaded per vertex. The context-aware skinning method by Weber et al. [8], has an own underlying skinning method based on log-quaternions which are a vector where the direction is the rotation axis and the magnitude the rotation angle around that axis. The term context refers to the shapes and for the context aware part Weighted Pose Space Deformation is used but instead of using every vertex in the example shapes, anchor points on the mesh are used instead, reducing the amount of computation significantly. Weber et al. was able to reduce the amount of computations in an example from 14,606 anchor points,(100 percent), to 290 anchor points, (2 percent), without losing any significant visible quality but makes it run much faster.

Work has been done on example-based clothing to improve the quality of clothing on characters by Wang et al.[9] and Feng et al.[10]. These methods use pretty much the same methods as with normal pose space deformation but apply the interpolated deformations onto a coarser cloth-mesh that is simulated at run-time to get a dynamically deformed piece of clothing with high detailed wrinkles. An other method presented by Müller et al. [11] is to from a coarse cloth-simulation generate wrinkles procedurally onto a subdivided or tessellated coarse mesh, but that method does not use any example shapes, it only procedurally generate wrinkles.

There are also some work done on trying to approximate dynamic effects such as jiggling fat and bouncing muscles [12], as a spring damper function. The method takes in a set of sample shapes that have been animated in a sequence, in this case motion capture. By calculating the joint acceleration and elongation of the mesh, a spring damper function could be approximated.

# Chapter 3

## Background theory

Vital parts for understanding the process of skeletal animation and skinning methods will be gone through in this section in order to get a good background and understanding of the most used key terms and methods used for skeletal animation.

### 3.1 Skeletal animation

Skeletal animation has been around since the early beginning of computer graphics animation. Almost every computer generated character has bones placed inside them in order to be able to move correctly. It is a very intuitive approach of having bones drive a mesh of vertices since the majority of all living things have bones in order to move properly. Without bones a leg would not work properly, it would just be a useless jelly limb. The bones in the leg gives it its rigidness that will move the leg in a rigid way.

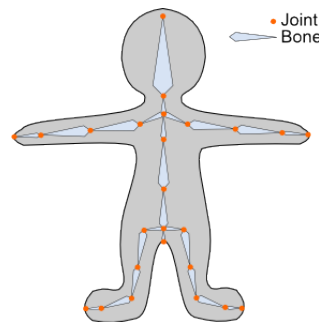
A leg has three distinct rigid parts, upper leg, lower leg and foot. In the early days of computer generated animation the character was usually made up of several rigid parts. For example, the leg would have had a thigh mesh, a calves mesh and a foot mesh that would just be rigidly moved with the corresponding bone. This method is called parenting, where each mesh is parented to the corresponding bone so the mesh will move and rotate in the exact same way as the bone. Later on, new methods for assigning bones to a mesh were invented where there would be no more need for having a separate mesh per bone, this will be described in section 3.2.

#### 3.1.1 The skeleton structure

The mesh is driven by a skeleton. A skeleton is made out of several bones which are ordered in a hierarchical structure where a root bone is the starting bone which is usually connected to the spine. So if the root bone is moved the entire skeleton is moved too, since it is hierarchically structured.

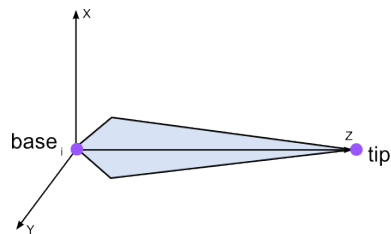
The term for where two or more bones are connected with each other is called a joint. For instance, the left upper arm is connected with the lower arm at the elbow, so the elbow would be

a joint. A skeletal structure of a rigged human is shown in 3.1.



**Figure 3.1:** structure of a skeleton, orange dots are joints and the blue shapes are bones.

While talking about skeletal structures, the terms bone and joint are frequently mixed. The reason for it can be explained by looking at how a bone is expressed in computer graphics. There are several ways of representing a bone in computer graphics. A bone could for instance be just two points or two transformation matrices, also known as frames, positioned at the joints locations. However, if it was to be represented with just two points, rotation along the bone is not captured which is needed in 3D space but in 2D space no twist rotation occur, so representing a bone with just two points in 2D space is a valid representation. The bone is however more likely to be represented as a transformation matrix, that positions the bone's base at the joint position in 3D. The tip of the bones is then expressed as a length in the local z-axis direction and the length often depends on where the next child bone is placed, see figure 3.2. The term child bone refers to a bone that is dependent to a bone called *parent bone* in such a way that it will inherit the transformations of that parent bone. By doing so a hierarchical structure is created. For example, the neck bone is a parent bone to the head bone and the head bone is then called a child bone to the neck bone and will be moved when the neck is moved.

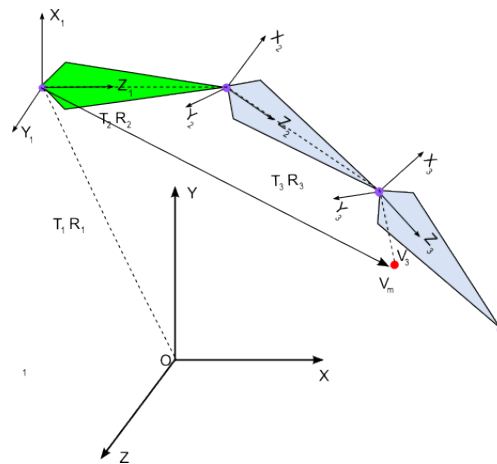


**Figure 3.2:** structure of a bone  $i$  with transformation matrix.

The bones are placed inside the model's mesh in its rest position, also known as default or reference position. A mesh is usually modeled in the pose of Da Vinci's "Vitruvian Man"; a standing pose with arms outstretched, as in figure 3.1. This pose is called the rest pose, or default pose, which is a pose where all the bones have a known position and the mesh has a relative position to the bones which is described in the bone's coordinate system called local space or bone space. The positions of the bones are described in model coordinates which are coordinates described in the root bone's local coordinates space. The root bone are then in turn described in global world coordinates. The reason for describing it in model coordinates is to be able to have the same animation applied to a character independently of rotation. For example, if a guy is walking on the floor, the same animation can be applied to a character walking upside down in

the ceiling since it is animated in model coordinates. It is not possible to reuse the animation in this way if it was animated in world coordinates.

A thorough understanding of the different coordinate systems is essential, in order to be able to follow this report, so lets go through it thoroughly. In skeletal animation, going back and forth from different coordinate systems are done frequently. The coordinate systems used are in descending order; global coordinates ( also known as world coordinates), model coordinates and lastly local/bone coordinates. The term coordinates are also often referred to as a *space*, for example local coordinates are often referred to as local space as well. The world space is the space which has everything in it, describing the entire scene or shot. The model space is a subspace to the world space where the model is described. The model space describes, in the usual case, a characters position and orientation and is described by the root bone's transformation matrix. In the model space several subspaces exists, one for each bone, called bone space or local space. Now, since the skeletal structure is hierarchical based the child bone of the root bone will be described directly in model space, which is also the root bone's local space. However, the next-coming children will be described in the parent bone's local space. For example, lets look at a bone chain that has three bones as seen in figure 3.3.



**Figure 3.3:** A bone chain representing a vertex position,  $v_3$  (red) in local and global coordinates. The chain has a root bone, seen in green and two child bones with corresponding rotation and translations.

In order to have a vertex follow a specific bone  $i$ , the vertex must be described in bone space to be able to get correct deformations. In figure 3.3,  $v_m$  is the vertex described in model coordinates, and  $v_3$  is the vertex described in bone 3's local space. It should be noted that the model space would be the same as world space if the root bone, shown in green in figure 3.3, would not have been rotated or translated from world space origo.

If a vertex  $v$  is described in bone coordinates as  $v_i$  for a given bone  $i$ , the model coordinates for that vertex can be described as the product of all transformation matrices on the form  $\hat{M}_i$ , where a transformation matrix consists of a transformation vector and a rotation matrix, see equation 3.1,

$$\hat{M}_i = \hat{T}_i \hat{R}_i \quad (3.1)$$

where  $\hat{M}_i$  is a 4x4-transformation Matrix of the  $i$ :th bone,  $\hat{T}_i$  is a translation vector described

in homogeneous coordinates and  $\hat{R}_i$  is a 4x4-rotation matrix. The *hat* expression,  $\hat{\cdot}$ , on the transformation vector, the rotation and transformation matrix denotes that it is described in bone coordinates.

The product of the parent bone's transformation matrices leading up to the child bone are then multiplied with the vertex described in bone coordinates, see equation 3.2. For instance, the vertex  $v_3$  in figure 3.3 needs to have the transformation matrices, in bone coordinates, of the two parent bones and its own transformation matrix in order to describe the vertex that is in bone coordinates, in model coordinates  $v_m$ , see equation 3.2.

It should be noted that the matrices are considered to be read and applied in a right-to-left side order as is done in OpenGL.

$$v_m = \hat{T}_1 \hat{R}_1 \hat{T}_2 \hat{R}_2 \hat{T}_3 \hat{R}_3 v_3 \Leftrightarrow \hat{M}_1 \hat{M}_2 \hat{M}_3 \hat{v}_3 \Leftrightarrow M_3 v_3 \quad (3.2)$$

In order to get the vertex described in bone coordinates, which is what is wanted for animation, the inverse of the matrices must be applied so the vertex that is described in model coordinates is described in bone coordinates instead, see equation 3.3.

$$\hat{v}_3 = \hat{M}_3^{-1} \hat{M}_2^{-1} \hat{M}_1^{-1} v_m \Leftrightarrow M_3^{-1} v_m \quad (3.3)$$

It should be noted that the transformation matrix,  $\hat{M}_i$ , is calculated per bone, not per vertex.

### 3.1.2 Animating the skeleton

Animation of the bones is carried out by multiplying deformed transformation matrices  $\hat{M}'_i$  onto the vertex described in bone coordinates just like equation 3.2. However, as mentioned in section 3.1 the vertex position is described in model coordinates and needs to be transformed with equation 3.3 into bone coordinates. The deformed vertex  $v'_m$  is deformed as in equation 3.4,

$$v'_m = \hat{M}'_1 \hat{M}'_2 \hat{M}'_3 \hat{M}_3^{-1} \hat{M}_2^{-1} \hat{M}_1^{-1} v_m = M'_3 M_3^{-1} v_m \quad (3.4)$$

where  $v'_m$  is the deformed vertex in model coordinates, and the matrices  $\hat{M}'_{1,2,3}$  are the modified transformation matrices which could have been applied a rotation matrix for instance, see equation 3.5.

$$\hat{M}'_i = \hat{M}_{rest(i)} \hat{M}_{anim(i)} \quad (3.5)$$

The modified matrix  $\hat{M}'$  is deformed by multiplying the  $i$ :th bone's rest pose transformation matrix  $\hat{M}_{rest(i)}$  with an animation transformation matrix  $\hat{M}_{anim(i)} = \hat{T}_{anim(i)} \hat{R}_{anim(i)}$ . However, it is more usual to just multiply with a rotation matrix only, since bones are usually never scaled.

The matrices in equation 3.4 are combined into two matrices,  $M'_3 M_3^{-1}$  which in turn for simplicity can be turned into a single matrix  $M_3$  so that only one matrix is stored per bone making it easier to follow and understand, see equation 3.6 where  $i$  denotes the  $i$ :th bone.

$$v'_m = M'_i M_i^{-1} v_m \Leftrightarrow v'_m = M_i v_m \quad (3.6)$$

The right side of equation 3.6, is on the form on which the matrix is often found in literature and it is seldom explained how it is built up, except by Ragnemalm[13] for instance.

By looking at equation 3.6, the vertex belongs to and is moved by one bone only. This method is called *stitching*[13][14], which moves the vertices rigidly with their corresponding bone. It works fairly well for robots that have rigid limbs, but for more lifelike characters, like humans, it will look strange around joints such as elbows, where the skin or in this case vertices would be influenced by more than one bone. Several different *skinning methods* exists to be able to have more than one bone influencing a vertex, which will be gone through in the next section.

## 3.2 Skinning Methods

Skinning is used to assign vertices to a specific bone or bones in such a way that the vertices will follow the bone's or bones' movement. Several skinning methods exists that enable a vertex to be deformed by more than one bone. The difference between stitching and skinning is that only one bone can be stitched to a vertex, whilst the skinning methods can have a vertex being influenced by more than one bone as well.

This section will handle the two most common skinning methods, namely:

- Linear Blend Skinning
- Dual Quaternion Skinning

There exists more advanced skinning methods[4][15], but they usually have either Linear Blend Skinning or Dual Quaternion Skinning as an underlying algorithm as mentioned in chapter 2.

### 3.2.1 Linear Blend Skinning

The Linear Blend Skinning algorithm is the skinning algorithm that is mostly referred to when it comes to skeletal animation due to its simplicity and speed. However, the Linear Blend Skinning algorithm does not have an original paper introducing the subject as mentioned by Lewis et. al [2], which might be the reason for why linear blend skinning exists in a variety of names. Linear Blend Skinning is therefore often referred to as *Skeleton-SubSpace Deformation*, *Vertex Blending* and *Enveloping*. In this report the algorithm will be referred to as Linear Blend Skinning, LBS for short.

The Linear Blend Skinning method closely resembles the stitching method, equation 3.6, but instead of only assigning one bone to a vertex, a weighted summation of all transformation matrices and weights for each bone  $i$  is applied to the vertex, see equation 3.7.

$$v'_m = \sum_{i=1}^N \omega_i M_i v_m \quad (3.7)$$

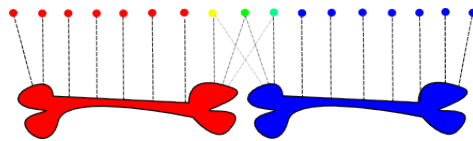


Note that the transformation matrix  $M_i$  is the pre-multiplied matrix as seen in equation 3.4. The variable  $N$  is the number of bones in the skeleton,  $i$  is the current bone index,  $\omega_i$  the weight per vertex that states how much influence bone  $i$  has on that specific vertex and  $v'_m, v_m$  is the vertex in deformed and undeformed state in model view coordinates.

The sum of the weights should always add up to one, see equation 3.8.

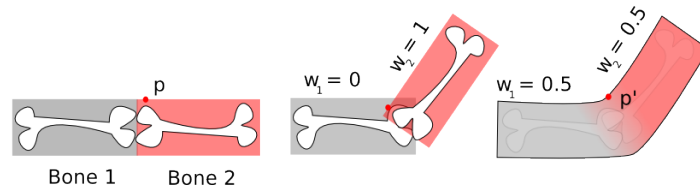
$$\sum_{i=1}^N \omega_i = 1, 0 \leq \omega_i \leq 1 \quad (3.8)$$

The weight  $\omega_i$  describes how much influence a given bone  $i$  has on a particular vertex where a weight of one will make the vertex move rigidly with the bone. Furthermore, A weight of zero for a particular bone means that it will not contribute to the deformation of the vertex, see figures 3.4, 3.5.



**Figure 3.4:** Two bones with assigned weighted vertices visualized by coloring the vertices red and blue depending on which bone it belongs to. Weights that are influenced by more than one bone are visualized as a blended color.

The colored dots in figure 3.4 represents vertices and their color represents which bone they belong to, namely the red or the blue bone. The three vertices in the middle have different colors than red and blue which indicates that they are influenced by both bones.



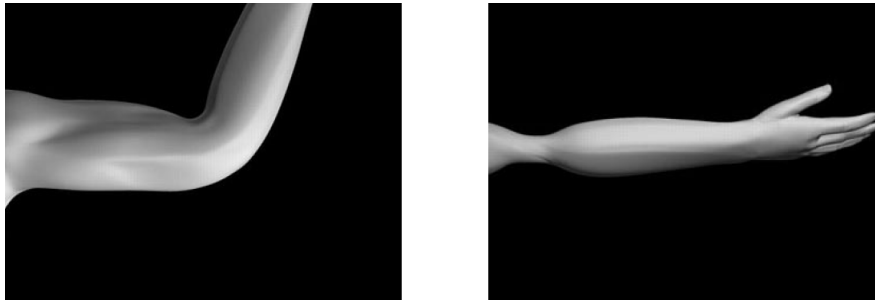
**Figure 3.5:** Visualization of different weights applied to the same vertex. The middle image shows the vertex rigidly moving with bone 2 and the right images shows a blend between the two bones.

Figure 3.5 shows two different weights applied to a vertex  $p$  such that it is influenced by more than one bone in the right most image. In the middle image the vertex is deformed rigidly with bone 2 only since the weight for bone 1 is zero.

### Limitations

The linear blend skinning method has some limitations such as the *collapsing elbow* and the *candy-wrapper* effect, see figure 3.6.

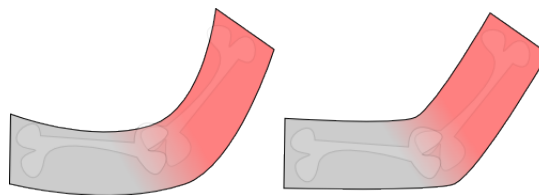
The collapsing elbow effect introduces volume loss around a joint when bending due to simple interpolation and the candy-wrapper effect [2] is due to large twisting rotations of the bone making it pinch at the joint and cause this candy-wrapper artifact. It is the same thing that



**Figure 3.6:** Left: The Collapsing Elbow Effect, Right: the Candy-Wrapper Effect. Images from [2]

happens to a long clown-balloon when making animal figures, where it will pinch due to the large twisting rotations. The vertices further away from the joints will have the same rotation as the bone as seen in the right image in figure 3.6.

A quick fix to the candy-wrapper problem might seem to be to spread out the weights over the entire bone. However, if the weights are spread out over the entire bone, problem will arise when trying to bend the arm. The vertices that should move rigidly when bending will be influenced by the other bone and cause a more undesirable banana shaped bending, see figure 3.7 where the left is the banana shape and the right image is a comparison of how the ideal deformation should look like.



**Figure 3.7:** Left: Banana shape caused by spreading out the weights more evenly to reduce candy-wrapping effect. Right: A comparison to what it should normally look like.

To better address these limitations that Linear Blend Skinning has with the collapsing elbow and candy-wrapper effect, a more advanced skinning method or additional information is needed such like the Dual Quaternion Skinning algorithm.

### 3.2.2 Dual Quaternion Skinning

Dual quaternion skinning by Kavan et al.[3][16] is a very popular method among skeletal animation due to the fact that it is quite fast, however, it is not as fast as linear blend skinning but reduces the candy-wrapper and collapsing elbow issue significantly compared to the linear blend skinning method. There are still some visual artifacts left around the joints when twisted due to the twisting is still concentrated around the joints and not spread out. Kavan et al. reduces the collapsing elbow and candy-wrapper effect by instead of blending rigid bone transformations as matrices, as is done in linear blend skinning, they are instead using dual quaternions.

In order to understand dual quaternions, the term and use of *quaternions* must be understood first. Quaternions was not used much in this thesis, except for that the dual quaternion skinning

method uses it, which was already implemented and therefore, a brief introduction into the dual quaternion skinning method is only presented, letting the reader learn about quaternions elsewhere [17].

## Quaternions

Regular quaternions have been used a lot in computer graphics and computer vision due to its ability to represent rotations in three dimensions in a very compact way. A quaternion consists of a linear combination of the basis elements  $1, i, j, k$  that follows Hamilton's product rule [17], see equation 3.9.

$$i^2 = j^2 = k^2 = ijk = -1 \quad (3.9)$$

A quaternion  $q$  can be written as a sum of a scalar real number  $a_0$  and a vector  $\vec{A}$ , see equation 3.10, where vector  $A = A_x i + A_y j + A_z k$ .

$$q = a_0 + \vec{A} \quad (3.10)$$

## Dual Quaternions

Dual quaternions are on the same form as normal quaternions as in equation 3.10 but instead the elements are dual numbers. Dual numbers are written similar to complex numbers on the form,  $\hat{a} = d_0 + \varepsilon d_\varepsilon$ , where  $d_0$  and  $d_\varepsilon$  are the non-dual and dual part respectively,  $\varepsilon$  is a dual unit that satisfies  $\varepsilon^2 = 0$ . A dual quaternion,  $\hat{q}$ , is a sum of two ordinary quaternions, see equation 3.11.

$$\hat{q} = q_0 + \varepsilon q_\varepsilon \quad (3.11)$$

## The Dual Quaternion Skinning Method

Since the joint transformations are described with transformation matrices, the first task in dual quaternion skinning is then to first convert the joint's transformation matrices into dual quaternions where a dual quaternion can represent both rotation in three dimensions around an arbitrary axis and translation. Dual quaternions can represent a transformation matrix in a much more compact way where it only uses 8 elements instead of 16 elements that is needed for a transformation matrix.

Normal quaternions can only rotate around origo and does not handle translations, which makes them unsuitable for skinning and therefore dual quaternion is needed since they can handle translation and can rotate around an arbitrary axis not centered around origo.

The dual quaternion method is quite similar to linear blend skinning, where instead of multiplying the weights with the the bone's transformation matrices the weights are instead multiplied with the bone's dual quaternions that have been converted from the matrices, see equation 3.12.

$$\hat{b} = \sum_{i=1}^N \omega_i \hat{q}_i \quad (3.12)$$

The term,  $\hat{b}$ , on the left hand side of equation 3.12 is just a weighted summation of all the contributing joint's dual quaternions. The vertex is not applied in this step since the weighted linear combined dual quaternion  $\hat{b}$  needs to undergo some more steps before being able to be applied on the vertex. It has to first be normalized into a dual unit quaternion. As usual, a dual quaternion has a non-dual part  $b_0$  and a dual part  $b_\epsilon$  and is normalized into a dual unit quaternion by normalizing the ordinary quaternions  $b_0$  and  $b_\epsilon$  into unit quaternions, which is done by dividing the two quaternions by the norm of  $b_0$ , see equation 3.13.

$$c_0 = b_0 / \|b_0\|, c_\epsilon = b_\epsilon / \|b_0\| \quad (3.13)$$

These two unit quaternions  $c_0$  and  $c_\epsilon$  can either be converted into a translation vector  $\vec{t}$  and a rotational matrix  $R$  and combined into a single transformation matrix  $M_i$  that is finally multiplied with the vertex  $v$  as is done by Kavan et al. [3], see equation 3.14. The same transformation matrix should be applied to the vertex-normal as well.

$$v' = M_i v \quad (3.14)$$

The two unit quaternions could also be used to directly compute the deformed vertex with cross products as is done by Kavan et al. [16], without converting them into a transformation matrix, see equation 3.15, where the quaternions have been described in scalar and vector parts  $a_x$  and  $\vec{A}_x$  respectively. The vertex normal is calculated as in equation 3.16.

$$v' = v + 2\vec{A}_0 \times (\vec{A}_0 \times v + a_0 v) + 2(a_0 \vec{A}_\epsilon - a_\epsilon \vec{A}_0 + \vec{A}_0 \times \vec{A}_\epsilon) \quad (3.15)$$

$$v'_n = v_n + 2\vec{A}_0 \times (\vec{A}_0 \times v_n + a_0 v_n) \quad (3.16)$$

## Limitations

Even though the dual quaternion skinning method produces much more visibly pleasing results around joints compared to linear blend skinning, it still generates some visible artifacts around the joints. The dual quaternion skinning method is also slower than the linear blend skinning method. However, with today's GPU hardware this method would run very fast on the GPU and would be able to handle more transformations since a quaternion uses less memory compared to a transformation matrix as mentioned earlier.

# Chapter 4

## Implementation

This chapter gives an overview of all methods that have been implemented in this master thesis in order to improve the visual quality of the crowd system ALICE at MPC. Moreover, the theory behind the methods will be described and also how they fit into the pipeline.

Firstly, a method called Stretchable and Twistable Bone Skinning, which improves the basic skinning algorithms, is presented. Thereafter, an example-based method called Pose Space Deformation is presented with several different interpolation techniques, which enable characters to have fast and complex skin and cloth deformations at run-time without having to compute anything. Lastly, two dynamic example-based techniques that can create dynamic effects such as jiggling of fat and muscle bouncing due to movement is presented, making it possible for many characters to have dynamic effects applied to them.

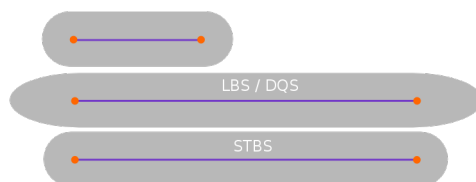
### 4.1 Stretchable and Twistable Bone Skinning

The stretchable and twistable bone skinning method was implemented to improve the basic skinning method currently used at MPC for crowd characters. The usual methods for crowd skinning are linear blend skinning and dual quaternion skinning because both of these methods are very fast. However, they suffer from the collapsing elbow and the candy-wrapper effect, where the latter is the biggest problem because it gives very nasty twisting artifacts around joints, such as when twisting an arm or raising an arm above the shoulder, it causes pinching of the mesh or an unnatural deformed mesh due to rotations. Even though the dual quaternion method deals with the twisting and rotation much better than the linear blend skinning method, it still gives some visible artifacts when rotating and twisting, so an improvement of these two methods is needed.

A new method was recently presented at SIGGRAPH 2011 called Stretchable and Twistable Bone Skinning (STBS) by Jacobson et al. [4], which improves the current skinning methods to be able to scale, rotate and twist the mesh very efficiently with an underlying skeleton with much more realistic results compared to linear blend skinning and dual quaternion skinning.

There are three key improvements when using the stretchable and twistable bone skinning method over the linear blend skinning or dual quaternion skinning method. Firstly, the candy-

wrapper effect, that pinches around joints due to twisting, is gone. Secondly, not only is the candy-wrapper gone, it also for instance, rotates the mesh around an arm more correctly, giving better rotations for problem areas such as armpits. Lastly the bones can be scaled without having any overshoots such as hands getting elongated when scaling the underarm-bone, see figure 4.1.



**Figure 4.1:** Scaling comparison of STBS with LBS and DQS. Note the visible overshoot caused by the LBS and DQS methods.

The Stretchable and Twistable Bone Skinning method is built on top of either linear blend skinning or dual quaternion skinning. The chosen underlying skinning method for this particular implementation for stretchable and twistable bone skinning was the linear blend skinning method because of its simplicity and speed.

In order to fix the candy-wrapper effect when twisting without causing a banana shaped bending effect, as mentioned in section 3.2.1, some additional information needs to be implemented into the linear blend skinning method. The stretchable and twistable bone skinning method fixes this candy-wrapper effect and handles stretching/scaling as well, by decomposing and adding some additional information weighted by an *endpoint weight function*  $e_i(p)$  for each bone  $i$  to the linear blend skinning method, which will be further described below.

### 4.1.1 Endpoint weights

An *endpoint weight function*  $e(p)$ , describes how far from the bone's endpoints, which are the base-joint and the tip-joint, a vertex  $p$  is located.

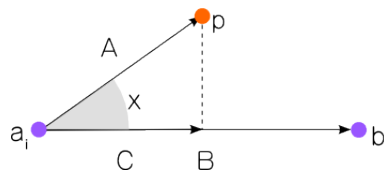
These endpoint weights can be automatically calculated since there exists several different methods for calculating such weights, such as: Simple projection onto the bone, inverse Euclidean distance weighting, automatic weighting by Bone Heat presented by Baran et al.[18] and bounded biharmonic weighting by Jacobson et al.[19]. However, since the endpoint weights have a good geometric and also visual meaning, the weights could also be manually assigned or tweaked, for example, by painting it by hand.

All these methods vary in terms of speed and accuracy, where the two fastest methods are weights generated by simple projection and inverse Euclidean distance weighting. As a result of that, these two methods are computational lightweight and simple, making them very fast, the visual quality is less good compared to more sophisticated methods, because they ignore the shape of the mesh. Since it does not take into account the shape of the mesh, problems occur when, for instance, a neck-bone controls both the head and neck of a dog where the nose is hanging down to the middle of the neck. Then simple projection projects the vertices onto the neck-bone so the endpoint weights for the vertices on the tip of the nose will get the same weights as the vertices placed in the middle of the neck. It results in that, when scaling and rotating the neck-bone, the nose will get longer when scaled and will not rigidly be scaled and

rotated as it should have been. This is because of the fact that the nose-vertices lies beyond the tip-joint and should therefore have had a weight equal to one, but instead have a weight equal to a vertex placed on the neck. However, these cases with over hanging mesh segments are quite rare in a human body and therefore these methods can be considered to be quite suitable for crowd systems since they are very fast.

The other methods, Bone Heat and Bounded Biharmonic weights are shape-aware but are more computational heavy, where for the latter one the mesh needs to be discretized into a volume, which is expensive. The Bone Heat method needs to solve a linear equation system in order to find the weights, but Jacobson et al.[4] states that the quality is more or less worth the cost. They also state that the Bounded Biharmonic weights method would be the best choice if neither ordinary bone weights nor endpoint weights exists. In such a cases, it would be interesting to apply that method since it produces very good weights for both cases. However, the usual case is that the bone weights are already known and no discretized volume of the mesh exists.

The method chosen was the simple projection method that projects the vertex down onto the bone and then the distance to each of the end joints  $a_i, b_i$  are calculated as seen in figure 4.2.



**Figure 4.2:** A point  $p$ , projected onto a bone  $i$  with end joints  $a$  and  $b$ .

With some trigonometry the distance to the projected vertex  $p$  onto the bone  $B$  that goes from joint  $a_i$  to joint  $b_i$ , see figure 4.2, can be described as in equation 4.1. The length of the bone is then factored out to have a point-weight function  $e_{proj}(p)$  between zero and one, as seen in equation 4.2, where  $\vec{A}$ ,  $\vec{B}$  and  $\vec{C}$  are the vectors in figure 4.2.

$$\|\vec{C}\| = \frac{\vec{A} \cdot \vec{B}}{\|\vec{B}\|} \quad (4.1)$$

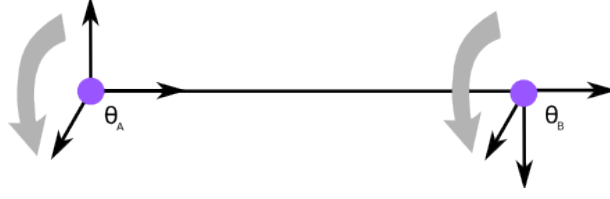
$$e_{proj_i}(p) = \frac{\|\vec{C}\|}{\|\vec{B}\|} \quad (4.2)$$

The projected endpoint weights that are obtained by equation 4.2, are then used to get nice twist rotations around bones and also enables scaling of the mesh in a visually pleasing way. In order to generate nice twisting around the bone, the twisting angles  $\theta$  at the base joint  $A$  and tip joint  $B$ , see figure 4.3, are needed to be taken into account.

In figure 4.3, the dots represents the joints  $A$  and  $B$ , and their twisting around the bone with a certain angle  $\theta$ .

### 4.1.2 The Stretchable and Twistable Bone skinning method

To be able to rotate the vertices around an arbitrary axis, in this case the bone's direction, a rotation matrix  $K$  that handles arbitrary axis-rotations is needed. A rotation matrix can be set up



**Figure 4.3:** Twisting angles  $\theta_A$  and  $\theta_B$ , around the joint.

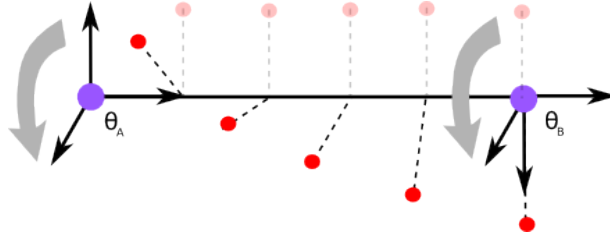
given an normalized rotation vector  $u$  and rotation angle  $\theta$  as seen in equation 4.3.

$$K = \begin{pmatrix} \cos\theta + u_x^2(1 - \cos\theta) & u_x u_y(1 - \cos\theta) - u_z \sin\theta & u_x u_z(1 - \cos\theta) + u_y \sin\theta & 0 \\ u_x u_y(1 - \cos\theta) + u_z \sin\theta & \cos\theta + u_y^2(1 - \cos\theta) & u_z u_y(1 - \cos\theta) - u_x \sin\theta & 0 \\ u_x u_z(1 - \cos\theta) - u_y \sin\theta & u_z u_y(1 - \cos\theta) + u_x \sin\theta & \cos\theta + u_z^2(1 - \cos\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (4.3)$$

However, this matrix setup was already implemented and only a rotation direction vector and a twist angle were needed as input.

The twist angle  $\theta$  is calculated with regards to where on the bone the vertex is located relative to the bone by using the endpoint weight function, and what the two joint angles are, which is done by linearly interpolating the joint angles, see equation 4.4 and the corresponding figure 4.4.

$$\theta = (1 - t)\theta_A + t\theta_B, t = e_{proj_i}(p) \quad (4.4)$$



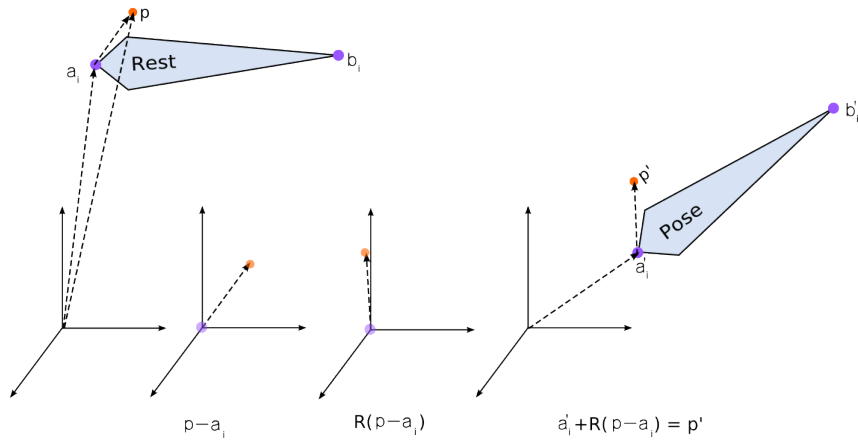
**Figure 4.4:** Linear interpolated twist along a bone, based on the position of the vertex point.

The stretchable and twistable bone skinning method is built on top of the linear blend skinning method by decomposing the transformation matrix  $M$  into a translational and rotational part, see equation 4.5.

$$p' = \sum_{i=1}^N \omega_i M_i p \Leftrightarrow \sum_{i=1}^N \omega_i T_i R_i p \Leftrightarrow \sum_{i=1}^N \omega_i \{a'_i + R_i(-a_i + p)\} \quad (4.5)$$

The decomposition is done by first removing the translation  $a_i$ , describing the vertex  $p$  around origo instead in order to be able to rotate it properly. Then the vertex is properly rotated around origo by using the rotation matrix and then translated back into its deformed position  $a'_i$ , see figure 4.5 for a visual comparison.





**Figure 4.5:** Visual interpretation of the decomposed transformation matrix  $M_i$

When the transformation matrix has been decomposed, the additional information can be added.

When a bone is scaled with linear blend skinning or dual quaternion skinning, overshoots will arise. For example, if the arm bone was scaled the hand would have been scaled as well and would have been elongated due to the fact that it scales every vertex independently of position relative to the bone. With the stretchable and twistable bone skinning method, the scaling occurs where it should occur, for example the arm gets elongated when scaled but the hand stays the same. Scaling is done by adding a stretch vector  $s_i$ , which is a scaled vector in the bone's direction, before the rotation is applied, see equation 4.6 for the stretch vector.

$$s_i = \left( \frac{\|b'_i - a'_i\|}{\|b_i - a_i\|} - 1 \right) (b_i - a_i) \quad (4.6)$$

The scaling factor  $s_i$  in equation 4.6 does not depend on each vertex position which causes the bone to be scaled uniformly resulting in the same overshoot artifacts as found in linear blend skinning. Therefore, the stretch vector  $s_i$  is multiplied with the endpoint weight function  $e_i(p)$  so the scaling is different for each vertex, removing the overshoot artifact. The fully non-uniform scaling implementation is seen in equation 4.7.

$$p' = \sum_{i=1}^N \omega_i \{ a'_i + R_i(e_i(p)s_i + (-a_i + p)) \} \quad (4.7)$$

Since the twisting is a rotation issue, the twisting rotation matrix  $K_i$  is set up by using the rotation matrix that handles rotation around an arbitrary axis as done in equation 4.3. The twist matrix  $K_i$  is created by using the bones direction  $u$  and its rotation angle around the bone  $\theta$ , where the rotation angle  $\theta$  depends on the two joint angles and the projected vertex position onto the bone as seen in equation 4.4. The twist matrix  $K_i$  is then multiplied with the vertex and finally with the normal rotation matrix  $R_i$ .

The final equation for calculating the stretchable and twistable bone skinning method with linear blend skinning as the underlying algorithm is seen in equation 4.8.

$$p' = \sum_{i=1}^N \omega_i \{a'_i + R_i K_i (e_i(p) s_i + (-a_i + p))\} \quad (4.8)$$

It should be noted that the final equation, equation 4.8, only deforms the vertices in a new way, compared to the underlying skinning method, when twisting and scaling occur. At places where twisting and scaling does not occur the stretchable and twistable bone skinning method will work in exactly the same way as the underlying skinning method since the scaling vector  $s_i$  would be zero and the rotation matrix would then be an identity matrix.

### 4.1.3 Usage

The stretchable and twistable bone skinning method was implemented into the skinning geometric operator function called *SkinningGOP(method m)*, in which the linear blend skinning and dual quaternion skinning method already existed. By implementing the stretchable and twistable bone method into the *SkinningGOP*, the method was able to be used in the same way as with the linear blend skinning and dual quaternion skinning methods by just specifying the desired method to be used. No new training, or extra rigging or manual assigning of weights was needed, making it easy to be switched to. The stretchable and twistable bone skinning method was implemented in C++, and then the *SkinningGOP* was used in a Lua-script through Lua-bindings.

## 4.2 Pose Space Deformation

In order to have advanced deformations of skin on characters they will have to be simulated or caused by advanced rigs, which are computational heavy and therefore takes a long time to generate. Because of the fact that it is so computational heavy, only the hero-level characters usually have advanced deformations whilst all crowd characters only have a normal skinning method, such as linear blend skinning, applied to them. So far it has worked to only have normal skinning methods on crowd-level characters, since they usually have been placed in the background of shots and therefore have not had the need to have time-consuming simulations applied to them, or fancy rigs since it would not have been seen anyway. However, nowadays the trend tends to be that crowd-level characters are placed closer and closer to the camera, sometimes closer than the hero-characters, and revealing the artifacts and lack of nice deformations to the audience. Therefore, a new method had to be implemented to address these issues in order to make the crowd characters look good on screen with less of a cost to what it would have taken to have advanced rigs or simulations on every crowd-level character. The method implemented to address these problems is called *Pose Space Deformation*(PSD). A thorough description of the pose space deformation method and different pose space deformation approaches are described in this section.

The pose space deformation method was pioneered by Lewis et al.[2] in 2000, and was popular at that time but never really got picked up in the visual effects industry until recently. Pose space deformation has become an interesting research area again in the computer graphics industry and is currently seen as a very hot topic due to todays demands of being able to have many high-detailed characters on screen.

Pose space deformation is a lookup and interpolation technique that enables complex skin deformations on, for example, crowd characters that would not have been possible to have been applied to before, since it has been too time-consuming to set up complex rigs or have heavy skin deformation simulations simulated for each crowd character. This technique takes advantage of interpolation and instead interpolates from a database with nicely deformed meshes, which might have been sculpted by hand, simulated with an advanced simulation or have had a complex rig applied to it, onto a run-time mesh. Therefore, at run-time, there is no need to have a complex simulation or advanced rig for each character, which gives a significant speedup and enables a larger amount of characters to be able to have better skin deformations than ever before.

The pose space deformation method is also useful for fixing errors on a mesh, such as intersections or loss of volume. The problem areas can be hand-sculpted and tweaked in such a way that it looks good for that particular pose and then gets stored in the database so that when a similar pose at run-time appears it will get that hand sculpted fix, enabling artistic shape deformation on characters as well.

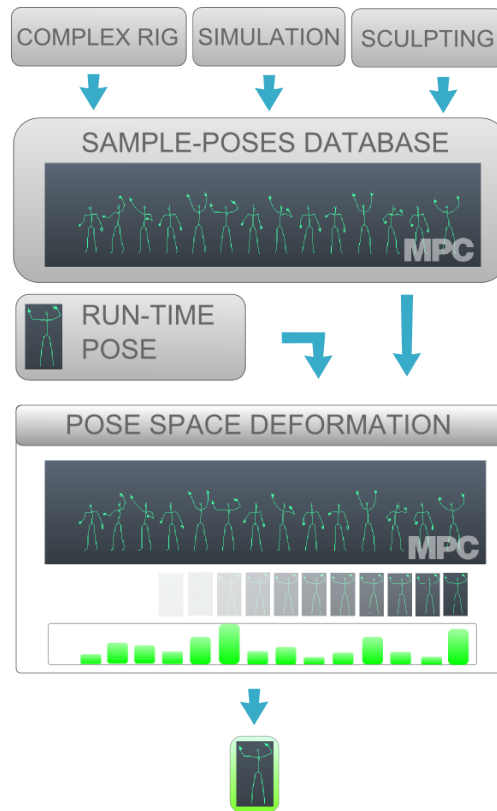
The pose space deformation method consists of two steps. The first step is an off-line processing step where a database is created and populated with simulated, complexly rigged and/or hand sculpted poses where for each pose a skeleton and its deformed mesh is stored. The second step is done at run-time where an arbitrary pose is sent into the database and gets compared to all skeleton poses in the database and takes out the poses that closest resembles the arbitrary pose. By cheaply interpolating the closest poses' meshes from the database, weighted by how close they are to the arbitrary skeletal run-time pose, a very close resemblance of a hero-level character quality mesh can be achieved for a fraction of what it would have taken to simulate it with a complex rig or an advanced skin deformation simulation. See figure 4.6 for a graphical overview of the pose space deformation pipeline, where the inputs to the pose space deformation algorithm is a complex rig, an advanced simulation and/or a hand sculpt correction that will make up the sample-pose database. The run-time pose is an arbitrary posed skeleton that only has a simple skinning method applied to it that deforms a skinned mesh. Onto this run-time pose, nicer deformations which are interpolated from the sample-pose database are applied, making it look much better.

Pose space deformation can simply be described as a scattered data interpolation problem described in the local coordinates space of the skeleton, which given a set of nicely deformed example-poses containing a skeleton and corresponding deformed mesh, can generate a high-detailed interpolated deformed mesh for an arbitrary posed skeleton that only have a simple and fast skinning method applied to it.

Several different pose space deformation techniques exists where different scattered data interpolation techniques are used. Below follows the most commonly used pose space deformation methods and a novel method that tries to improve the most common methods for pose space deformation is also presented.

### **Database creation**

The first step in a pose space deformation method is, as mentioned, to generate some data in the form of a set of sample-poses which are extracted from an advanced simulated, complexly rigged or hand-sculpted sequence and is therefore needed to be described before the pose



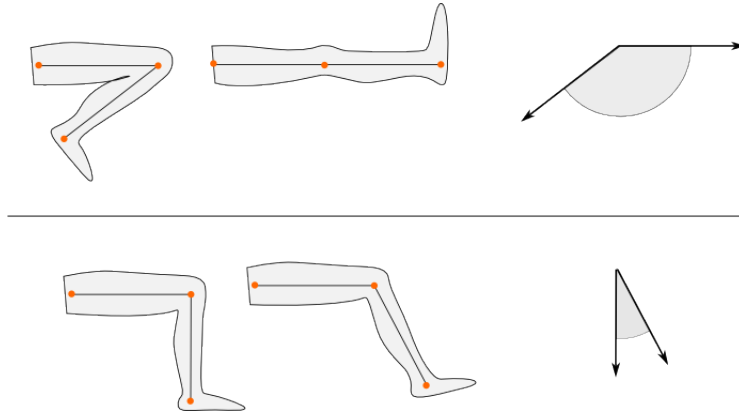
**Figure 4.6:** A scheme on how pose space deformation works in general.

space deformation methods are gone through, in order to get a good understanding of the algorithms.

Two methods for selecting the poses for the database were implemented. One is a very simple method that splits an animated sequence into  $N$  parts, where  $N$  is the number of sample poses wanted in the database. By dividing the number of frames in the sequence with the number of poses  $N$  a fixed jump  $M$  is achieved, where for each  $M$ :th frame a pose is stored. This method is very fast but has no knowledge of how the poses look like, which can result in sample-poses that are not that good for interpolation.

The best sample poses to interpolate with are the extreme poses, since interpolation between two extreme poses can interpolate more poses in between, compared to two non-extreme poses which cannot in any way generate good deformations at an extreme pose since it has lost that information. See figure 4.7 where the top images are extreme-poses and the bottom images are non-extreme-poses. So the best way is to store extreme poses instead, which can interpolate many more poses as demonstrated with the angles to the right in figure 4.7. The angles are much larger for the top extreme-poses in comparison to the non-extreme-poses.

In order to find the extreme poses, a greedy method can be used that first stores the rest-pose and then loops through the entire sequence to find the pose that is furthest away from the rest-pose sample. Once the sequence has been looped through, the pose which was furthest away is stored in the database along with the rest-pose. The sequence is then looped through again in



**Figure 4.7:** Extreme poses (Top images) versus non-extreme poses (Bottom images).

order to find the pose that is furthest away from both the rest pose and the pose that was newly added and store it in the database. The procedure continues by comparing all sample poses that was currently stored in the database to the current frame in the sequence until  $N$  desired sample-poses have been selected. As mentioned, it is a very greedy method and it takes some time to calculate the database, where the time depends mostly on the length of the sequence and the amount of sample-poses needed.

The measurement that described the differences between two poses  $p_a$  and sample-pose  $p_i$  was done by summing the euclidean square distances between each joint  $j$  and in order to be able to capture similar poses but with different twists around the bones, the local twist-rotations per joint  $\theta$  around the bone were taken into account as well and scaled by a factor  $k$  in order to be able to store different head rotations, see equation 4.9.

$$d(p_a, p_i) = \sum_j^{n_{joints}} (p_{a,j} - p_{i,j})^2 + k(\theta_{a,j} - \theta_{i,j})^2 \quad (4.9)$$

Each sample-pose found by the methods described above contains; the skeletal position and its corresponding mesh, a delta-vector between every vertex  $v_{i,j}$  between an undeformed mesh, which is in rest-pose position, and the same vertex from a sample-pose  $k$ . The delta-vectors are obtained by first transforming the sample-pose mesh from its posed position into its rest-pose position, which is the undeformed position it had before any skinning was applied to it, and extract the delta-vectors in rest-pose position. This is done in order to get a rotational invariant local-space for comparing the locations of the vertices and also make the delta vector only depend on the deformation produced from sculpting, a complex rig and/or a simulation and not by the skinning itself. The rotational invariant local-space is obtained by applying the *inverse* of the bone's transformation matrix,  $(\omega_j M_j)$  that transformed the vertices into their posed positions with for example linear blend skinning or dual quaternion skinning, see equation 4.10.

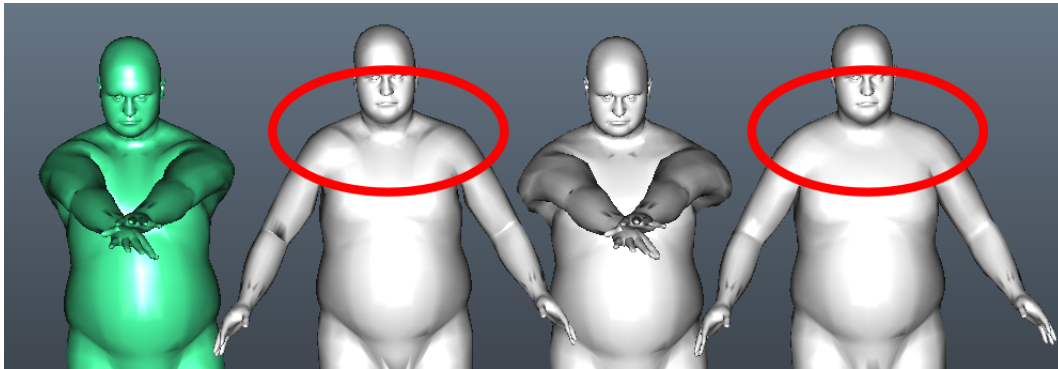
$$\hat{d}_{i,k} = \sum_{j=1}^{n_{joint}} (\omega_{i,j} M_j)^{-1} v_{i,j} - v_{i,0} \quad (4.10)$$

In equation 4.10,  $\hat{d}_{i,k}$  is the delta-vector for sample-pose  $k$  for a vertex  $i$  and  $\omega_{i,j}$  is the bone-weight

that says how much the joint  $j$  influences the  $i$ :th vertex  $v_{i,j}$ .  $M_j$  is the transformation matrix which transforms the vertex from its rest position to its posed state, and lastly  $v_{i,0}$  is the  $i$ :th vertex in the untransformed rest-pose mesh.

By applying the inverse of the underlying skinning method, such as linear blend skinning, the mesh is put into the rest position but still has deformed vertices where the more advanced rigging or simulation methods, such as muscle bulging and skin twisting, had effect, see figure 4.8. By applying the inverse of the skinning method used, both the deformed and the undeformed vertex are then described in the same space and also deformations caused by rotations and translations from the simple skinning method itself are removed. This inverse skinning step is used on all pose space deformation methods presented in this thesis.

It should be noted as well that the same underlying simple skinning method used for creating the high-detail sample poses must be used when skinning the interpolated mesh again. This constrains these methods a bit since the same underlying skinning method have to be used. There have been some publications on removing this step, or replacing it to remove the need of being dependent on the same underlying skinning method. One publication for instance, removes this step by estimating the delta vectors directly by using a Powell optimization approach [20].



**Figure 4.8:** From left to right: Advanced deformed mesh with a simple underlying skinning method, the advanced skinning method put into the rest position by applying the inverse of the underlying skinning method, simple skinning, rest pose. Note the difference between the second and fourth character around the clavicles inside the red circles.

Figure 4.8 shows from left to right, a complexly rigged pose, the complex rigged pose transformed back into the rest position, as described in equation 4.10, and the two right-most images are the same poses but instead normally skinned and rigged. Note how the shoulders differs between the characters that are in rest-position which are the deformations generated from for instance, as in this case, a complex rig. Since all rotations caused by the skinning of the mesh are removed when transformed back into the rest-position, it is possible to get a rotational-invariant local delta-vector/displacement-vector,  $\hat{d}_{i,k}$ , for each vertex that says how much the vertex needs to be pushed in a certain direction in order to reproduce a complexly deformed version, for example the pose furthest to the left in figure 4.8.

These delta-vectors are then stored in a database or directly used for calculating weights for the pose space deformation methods described below that uses radial basis functions and only stores those weight values instead.

### 4.2.1 The Shepard's Method

The simple and very popular scattered data interpolation method called the Shepard's Method [2][21], interpolates a delta-vector  $\hat{d}_{i,k}$ , for a vertex  $i$  in sample-pose  $x_k$ , by calculating the weighted sum of the surrounding data-points  $d$  and is normalized by the sum of the weights, see left part of equation 4.11.

$$\hat{d}_{i,k} = \sum_{k=1}^{n_{poses}} \frac{\omega_k}{\sum_{k=1}^{n_{poses}} \omega_k} d_{i,k}, \quad \omega_k = \|x - x_k\|^{-p} \quad (4.11)$$

The data-point  $d_{i,k}$  is, in this case, a delta vector between the deformed vertex,  $v'_{i,k}$  and the vertex  $v_i$  that has not been deformed at all,  $d_{i,k} = v'_{i,k} - v_i$ . The deformed vertex  $v'_{i,k}$  has been deformed into the sample pose position  $k$  by a skinning method such as linear blend skinning which also has a complex skeletal rig or physical simulation method applied to it as well, making the skin deformation look really good. If the deformed vertex would be directly used to calculate a delta vector, if the vertices are described in model coordinates, then the delta vector might be incorrect due to rotations and translations caused by the underlying skinning method used as described in section 4.2.

Note that the deformed vertex is deformed per pose whilst the undeformed vertex in rest pose has the same value for all poses. The weight  $\omega_{i,k}$  is the inverse distance between the arbitrary pose  $x$  and sample pose  $x_k$  which is scaled by a scaling factor  $p$  that should be larger than one otherwise it is discontinuous at the data points, see equation 4.11. The distance calculations when calculation  $\omega_{i,k}$  is done by summing all the joint distances between the joints  $j$ , as seen in equation 4.12.

$$\|x - x_k\| = \sum_j^{n_{joints}} \|(x_j - x_{k,j})\| \quad (4.12)$$

Because of the simplicity of the Shepard's method, it has some drawbacks. For instance, far from the data-points it will converge to the average of the data points, since the weights will be approximately the same. This is due to the fact that when the data points are far from each other, the weights will be very large, making it not depend much on how the run-time arbitrary pose looks like. This results in that both the weight at the numerator and denominator will be approximately the same, which in turn results in a convergence to the average of the data points, see equation 4.13.

$$\hat{d}_{i,k} = \sum_{k=1}^{n_{poses}} \frac{x}{\sum_{k=1}^{n_{poses}} x} d_{i,k}, \quad x \rightarrow \infty \Rightarrow \sum_{k=1}^{n_{poses}} \frac{d_{i,k}}{\sum_{k=1}^{n_{poses}} 1} \Rightarrow \frac{\sum_{k=1}^{n_{poses}} d_k}{N} \quad (4.13)$$

As mentioned, the choice of power  $p$  constant matters;

- For  $0 < p \leq 1$ , discontinuity arises at the interpolated data-points showing as sharp creases.
- For  $p > 1$ , the derivate at the interpolated data-points is zero resulting in a smooth but uneven interpolation without any sharp creases but instead flat spots around the data-points occur.

These issues of converging to the average far from sample poses and have flat spots around the data points, makes it not the most suitable scattering interpolation algorithm for pose space deformation.

## 4.2.2 Radial Basis Functions

The radial basis function method is the most commonly used scattering data interpolation method[21] and it is also well suited for Pose-Space Deformation as presented by Lewis et al. [2].

Radial Basis Functions, RBF for short, are commonly used to approximate functions which are approximated by linear combinations of  $N$  non-linear radial basis functions  $\phi()$ , see equation 4.14.

$$\hat{d}_{i,k} = \sum_{k=1}^N \omega_{i,k} \phi(\|x - x_k\|) \quad (4.14)$$

In this case,  $N$  is the number of poses. The distance between two poses is obtained by summing the joints for each pose, as is done in equation 4.12, whilst using the square distance instead. The radial basis functions are weighted for each sample-pose  $x_k$  by a weight  $\omega_{i,k}$ .

Radial basis functions are functions which values depends only on the distances from an arbitrary data point  $x$ , in this case a pose, to a sample data point  $x_k$  which is referred to as a center. The term radial comes from the fact that all radial basis functions depend only on the distance from a center  $x_k$ , making it only depend on a radius  $r$ . Below is a list of different radial basis functions [21]:

- Gauss:  $\phi(r) = e^{-\frac{r^2}{2\sigma^2}}$
- Thin plate spline:  $\phi(r) = r^2 \log(r)$
- Hardy multiquadratic:  $\phi(r) = \sqrt{r^2 + c^2}, c > 0$

The radial basis function used in this implementation was a Gaussian function, which is the most commonly used kernel since it so well behaved[2]. The user can also manipulate and affect the variance variable  $\sigma$  in the Gauss kernel, see equation 4.15, which in this case controls how many poses that should be affecting the current run-time pose for interpolation.

$$\phi(\|x - x_k\|) = e^{-\frac{(\|x - x_k\|)^2}{2\sigma^2}} \quad (4.15)$$

A small value of  $\sigma$  results in a very narrow falloff which causes only the closest sample-poses to be interpolated. If a large value of  $\sigma$  is applied every sample-pose, or many sample-poses, in the database will be taken into account in the interpolation resulting in a more averaged deformed pose but it handles more arbitrary poses better. However, the value of  $\sigma$  does not directly correspond to the amount of poses wanted to be interpolated, instead it gives a vague meaning of how many sample-poses that should be interpolated.

In order to interpolate the delta-vectors for an arbitrary pose at run-time, the weights in equation 4.14 were needed to be solved, which can be set up as a least-square fit problem and was solved in the following way;



First a NxN matrix  $\Phi$  was created where the  $(i, j)^{th}$  element is the radial basis function  $\phi$  between sample-pose  $x_i$  and sample-pose  $x_j$  in the database with the euclidean distance between the poses as input. The Euclidean distance between two poses,  $x_a$  and  $x_b$ , was obtained by summing together the distances for each joint  $j$ , see equation 4.16.

$$\|x_a - x_b\| = \sqrt{\sum_{j=1}^{n_{joints}} (x_{a,j} - x_{b,j})^2} \quad (4.16)$$

Since the actual distances  $\hat{d}_{i,k}$ , that were captured in the same way as in equation 4.13, and all sample-poses  $x_k$  are known, the unknown weights  $\omega_i$  could be estimated and later be used for deforming an arbitrary pose at run-time. This is done by finding the pseudo-inverse of the  $\Phi$ -matrix, which is an approximation of the inverse matrix, and is found by firstly describing the summation equation 4.14 in vector and matrix notations instead, see equation 4.17 and equation 4.18. The linear equation system is a linear least-square fit problem, which can efficiently be solved by using *Singular Value Decomposition* (SVD), which approximates the pseudo-inverse of the  $\Phi$ -matrix. A *Neural network*, could also be used to approximate the weights by training it with the known data, the  $\Phi$ -matrix and distance weight vector  $d$ , but the least-square fitting with singular value decomposition was the approach chosen in this thesis.

$$\begin{bmatrix} \phi_{1,1} & \phi_{1,2} & \phi_{1,3} & \cdots \\ \phi_{2,1} & \phi_{2,2} & \cdots & \\ \phi_{3,1} & \cdots & & \\ \vdots & & & \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \vdots \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \end{bmatrix} \quad (4.17)$$

By applying Singular value decomposition on the  $\phi$ -matrix in equation 4.17, a pseudo-inverse of the  $\phi$ -matrix could be obtained and used to generate the unknown weights as is done in equation 4.18, where  $\Phi^{*-1}$  is the psuedo-inverse matrix,  $d$  is the distance vector and  $\omega$  is the weight vector.

$$d = \Phi\omega \Rightarrow \omega = \Phi^{*-1}d \quad (4.18)$$

When the weights have been solved by using singular value decomposition, the interpolated delta vector  $\hat{d}_{i,k}$  can be obtained by using equation 4.14. Equation 4.14 is calculated at run-time, where  $\omega_i$  is the solved weight, the euclidean distance passed into the radial basis function  $\phi$  is the distance from the arbitrary run-time pose  $x$  and the k:th sample pose  $x_k$  in the database. The  $\Phi$ -matrix does not change for every vertex, so only one  $\Phi$ -matrix per frame needs to be calculated and used for every vertex which makes this method really fast.

The newly interpolated delta-vector  $\hat{d}_i$  is added to an undeformed base-mesh's vertex at rest-pose position before the skinning is done, in order to prepare it for the deformed position. After every delta-vector has been applied to its corresponding undeformed vertex, the same skinning method used for putting the sample-poses into rest-position, as is described in section 4.2, is applied to the mesh transforming it into the current pose with the newly deformed vertices, see equation 4.19,

$$v_i^t = SKINNING(v_i^r + \hat{d}_i) \quad (4.19)$$

where  $v_i^t$  is the newly transformed vertex position generated by applying a skinning method such as linear blend skinning to the undeformed vertex at rest-position,  $v_i^r$ , which has been deformed by the newly calculated delta-vector  $\hat{d}_i$ .

### 4.2.3 Weighted Pose Space Deformation

The weighted Pose space deformation method described by Kurihara [6], goes through exactly the same procedure as the normal pose space deformation technique with Radial Basis Functions. The only difference is that the  $\Phi$ -matrix is computed for each vertex instead, whilst for the normal pose space deformation method, only one  $\Phi$ -matrix was calculated and used for all vertices at run-time. The distance calculation as seen in equation 4.16 is now calculated for each vertex instead and weighted with its corresponding bone-skinning weight  $w_i^b$ , which results in better deformation with less sample-poses. This is due to the fact that it describes the position of the vertex more correctly by weighting each vertex with the bone-weight  $w_i^b$ , see equation 4.20.

$$\|x_a - x_b\| = \sqrt{\sum_{j=1}^{n_{joints}} \omega_{i,j}^b (x_{a,j} - x_{b,j})^2} \quad (4.20)$$

By taking into account the bone-weight  $w_i^b$ , better deformations are achieved since every vertex is weighted differently resulting in that less sample-poses are needed, compared to normal pose space deformation, to get good results. However, since the  $\Phi$ -matrix needs to be calculated for each vertex, this method is noticeably slower than normal pose space deformation, but it does not need to have as many sample-poses in order to produce good results. The weighted pose space deformation method also handles run-time poses that are far from the sample poses much better than the normal pose space deformation method does, due to the fact that it takes into account the bone-weights as well.

### 4.2.4 Pose space deformation with a three-joint local lookup

The pose space deformation methods presented above tries to find a best fit for an entire pose, which makes them not optimal to be used if a small amount of poses are used, or if the arbitrary pose at run-time does not look anything like the sample-poses stored in the database. For instance, if a database only had two poses, for example a guy with both arms and legs spread out like a starfish and a second pose as a casual standing guy with his arms hanging along the side of the body, and the pose to be interpolated at run-time is a standing guy waving his right arm. When looking at the entire pose as input, none of the poses in the database would fit the waiving guy's pose very well, which would result in a poorly interpolated pose at run-time. However, if the input is only a small part of the body, a better approximation of the arbitrary body can be achieved since it will only try to fit a local part of the body with the poses in the database. By using a more local lookup into the database by only looking at a section of the body such as an arm or leg, the right waving arm could then be fitted nicely with the starfish pose and the rest of the body could be fitted nicely with the second pose, which results in a nicely interpolated run-time pose with very few samples.

The method is a three-joint lookup method that for each joint, only takes into account its parent, the current joint and child/children joint(s) when finding the best match in the sample-pose database instead of taking into account the entire pose.

The previous methods also used the distances between the poses on a per joint basis in global space, which made them sensitive to how the poses were posed and especially how they were rotated in global space. In order to interpolate smoothly all fine detail on a very arbitrary guy, for example a guy lying on the floor and only having standing sample-poses in the database, a rotational invariant local-space is needed. Several different local-space methods have been implemented and are described below:

1. By using local space positions of the joints generated by having one of the three joints as origo and comparing positions in this local space. It was performed by applying the inverse of the transformation matrix of the current joint in global space to the parent and child joint which translated and rotated every joint back into a rotational invariant local joint-space with the current joint as origo. It gave good results for arbitrary rotated poses but was a bit slow, due to the fact of having to calculate and apply the inverse matrix of the current joint.
2. By taking out the vectors from the current joint to the parent and child joint and performing various operations such as taking out the area and length between the positions of the parent and child joint. The method is therefore not depending on any position and rotation but solely on angles and areas, which does not depend on how the pose is placed in world space. However, different poses can also produce the same angles between joints and the same area which makes it unstable.
3. By only taking into account the current joint's local Euler-rotations as done by Wang et al. [9], taken out from the local Pose-matrix the method is positional as well as rotational invariant. The method performs well, however, a larger amount of poses are needed to be able to interpolate the correct deformations since it is very sensitive to all the rotations. For example, if a bone is slightly twisted around the bone but otherwise has the same angles it might still not choose that pose because of the slight twist, even-though it might be the best fit in terms of visual quality.
4. By using the Euler-rotations as done in 3. but instead weighing the different Euler-angles differently between zero and one, where the rotations around the bone has the largest weight and the other two Euler-angles shares the rest of the weights so the total adds up to one, a good result can be achieved. This method tends to produce very good results for arbitrary poses and does not need a large amount of sample-poses stored in the database in order to produce visually pleasing results.

The three-joint pose space deformation method was almost as fast as the normal pose space deformation, but the visual quality for arbitrary poses was much better.

The preferred methods to be used for the three-joint pose space deformation method is to use method one, three or four above to get the best results where method one has proven to be the best just before method four. The other methods were removed. However, method three was able to be obtained with method four, having all weights equal to one.

By making the lookup in a rotational and translational invariant local space, such a bone space, as in method one above, a good match in the database was easily found without having to have many sample-poses. By describing the parent and child joints in the current joint's local bone space, a match in the sample-pose database could easily be found, since it does not depend

on how the bones were placed in global space. For example, if an arm in the arbitrary pose is bent along side the body as it was carrying a cup of tea, a good match can be found on all sample-poses in the data-base that have a bent arm in any position, such as a guy typing on a keyboard, flexing his muscles and so forth. By having many similar results, but not an exact match to interpolate with, compared to having only one similar and several poor matches, as it would be with normal pose space deformations, the three joint local lookup method can better handle arbitrary poses and reduce flickering that occurs when few or poor matches are only found in the sample-database.

### **A localized joint deformation area of the mesh**

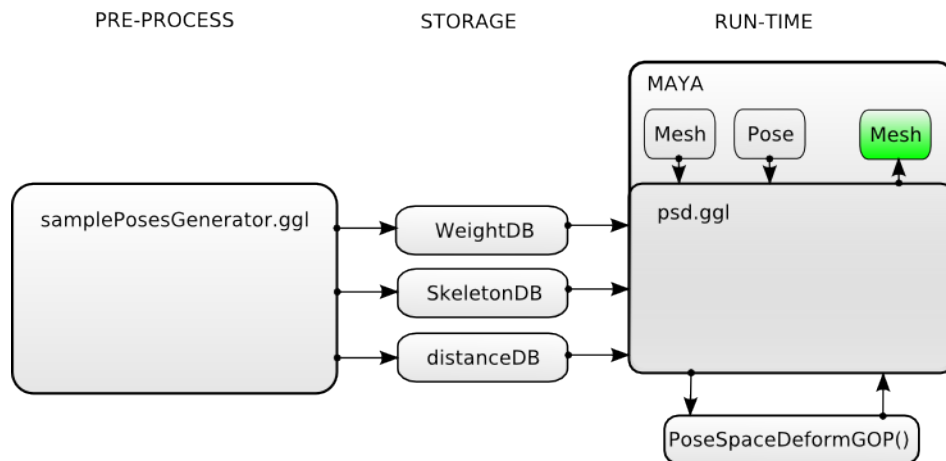
Instead of applying the local deformations onto the entire mesh the three joint method only applies the local deformations onto a local area. If the whole mesh was taken into account, unnecessary computations would then be made since only a small portion of the entire mesh would be affected by a single joint, resulting in a much slower algorithm.

For each joint in the three-joint algorithm a local area of the mesh is calculated by storing its parent's vertices and own vertices as well. By doing so a larger area close to the joint is affected compared to if only the vertices affected by that joint would be taken into consideration. For example, if the elbow joint is taken into consideration, the lower arm vertices will already have been assigned to the elbow joint but vertices that are close but are making up the upper arm will not be accounted for unless the parent joint's vertices are also taken into account. However, this results in that several vertices will be affected and deformed more than once. For instance, a vertex placed on the lower arm of a character will have deformations applied to it twice, once when calculating the deformations produced by the elbow-joint and once for the deformations produced by the wrist-joint. It results in deformations that are often larger than expected, and in order to fix the problem each vertex deformation vector is divide by the amount of times it has been deformed by a joint, which results in an averaged deformation vector. A better method is to also take into account the vertex position relative to the joints so that if the vertex is close to a joint, say the elbow-joint, it will almost only get the deformation produced by that joint and if a vertex is exactly in between two joints it will get the average deformation from the two. It results in a more accurate deformation vector since the deformations around joints are mostly driven by the closest joint and not so much by the neighboring joints.

### **4.2.5 Usage**

As mentioned earlier, the pose space deformation method is done in two steps. The first step is to gather the set of sample-data needed and store it in a database. This was done through a giggle-script which is a lua-script but has connections and set-ups with MPC's software stack. The giggle-script that generates the sample-pose database is called *samplePosesGenerator.ggl* and takes in the number of samples wanted from a specified animated sequence, see figure 4.9. The database consists of three different databases where the needed information is stored.

The second step in pose space deformation is to do the actual pose space deformation method. This method is done in C++ code in the method *PoseSpaceDeformationGOP()* and is called in the *psd.ggl* script through lua-bindings and is run at run-time inside MAYA. The run-time script takes in the run-time pose and mesh and interpolates using the sample-pose database a new mesh seen in green in figure 4.9.



**Figure 4.9:** Overview of the Pose Space Deformation method implementation

Here follows a simple step by step usage of pose space deformation that enables fast and complex deformations at run-time:

1. Create a set of high-detailed animated sequences generated from a complex rig, simulation or by hand-sculpting.
2. Choose from the high-detailed generated sequences a set of different poses, preferably extreme poses, and store them in a sample-pose database.
3. Apply pose space deformation, that will use the sample-pose database, on an arbitrary run-time pose in order to get a high-detailed interpolated run-time character.

### 4.3 Dynamic Pose Space Deformation

Normal pose space deformation can only handle static deformations such as muscle bulging caused by positioning the bones in a certain way. It cannot handle dynamic effects such as muscles bouncing and jiggling of fat due to acceleration of the body. Dynamic effects on a character's body are very important in making a computer generated character look real when moving. Without dynamic effects a character would look unnaturally stiff and rigid.

Dynamic effects are, however, time consuming to produce and therefore more or less only used on hero-level characters because of the time it takes to rig up or to simulate the effects. The dynamic effects on hero-level characters are often created by using an advanced rig with muscles attached to it that are simulated using for example a Finite Element Method [15]. Another method to simulate jiggling of fat, which is much cheaper than a finite element simulation, is to use a mass-spring system[12], which drives each vertex on the mesh with a spring where the spring coefficients are usually painted manually on the mesh.

In this section two different approaches that were implemented in this thesis are presented, where the first one is more or less a normal pose space deformation method but uses the joint accelerations as a key input to a sample-pose database instead of joint positions. The second method approximates a spring-damper function, creating a mass-spring system which estimates

the spring coefficients based on the sample-poses in the database, as is done by Park and Hodgins [12].

### 4.3.1 Joint accelerated dynamic deformation

Park and Hodgins [12] showed that dynamic effects could be efficiently approximated in pose space by using the joint accelerations as input, but they used it as input to approximate a spring-damper function. Since normal pose space deformation is working in pose space, however, only for one frame, it could be extended to be used to approximate dynamic effects as well by taking into account the previous frames' skeleton positions. Normal pose space deformation uses the positions of the joints as input to find the closest match, which does not depend on the previous frames, but in order to find the dynamic deformations which is history-dependent the key to the dynamic pose space deformation must be history-dependent. The joint acceleration, which is history-dependent, is used as key to find the closest dynamic matches in a database.

The acceleration of the joints were generated by taking into account the previous, current and the next positions of the joints, see equation 4.21,

$$\begin{aligned} a_j(t) &= \frac{\Delta v_j}{\Delta t}, v_j(t) = \frac{q_j(t) - q_j(t-1)}{\Delta t} \Rightarrow \\ \Delta v_j &= \frac{v_j(t+1) - v_j(t)}{\Delta t} \Leftrightarrow \frac{q_j(t+1) - 2q_j(t) + q_j(t-1)}{\Delta t} \Rightarrow \\ a_j(t) &= \frac{\Delta v_j}{\Delta t} = \frac{q_j(t+1) - 2q_j(t) + q_j(t-1)}{\Delta t^2} \end{aligned} \quad (4.21)$$

where  $q_j(t)$  is the  $i$ :th joints position at frame  $t$  and  $\Delta t$  is the time interval between two consecutive frames.

Since the method uses the cached skeleton poses and interpolates a new deformed mesh for each frame independently from the last frame, it will in some sense not be entirely history dependent other than that it takes into account the previous frames in order to calculate the acceleration. However, a good thing though is that it does not have to be run in a specific order in order to collect any data from the previous frames such as the last frames delta-vectors. It makes it very flexible and if the variation in acceleration is smoothly varying then it will work well, but for not so smooth variation in acceleration some popping in the mesh might occur. This is due to the fact that it does only interpolate the delta vectors depending on the current frame acceleration. For example, if it in the previous frame had a large impulse, the fat should have been elongated somewhat but if it for the next frames does not have any accelerations, then it will not have any deformations interpolated to the mesh, which results in some flickering. The small flickering can be reduced by taking into account more samples in the database for the interpolation.

### Dynamic Database Creation

As done with normal pose space deformation, a sample database is created in a pre-processing stage where the key input is the acceleration and the acceleration-key points to the delta-vectors which are extracted by subtracting the deformed mesh with the undeformed mesh in rest-pose space. the deformed mesh is put into rest-pose space by applying the inverse of the transformation matrix to the vertices, as was done with normal pose space deformation, see equation 4.10.

The positions of the joints are extracted in the global world coordinate space and the accelerations are calculated in world coordinate space as well because of the fact that if the positions would have been described locally for each frame and for each joint, accelerations due to for example falling would not have been captured since the positions for each joint would have been the same for all frames in model space.

The acceleration vector described in global space is described in rest-pose coordinate space by multiplying the acceleration vector described in world space with the inverse of the rotational matrix of the *globalPose* transformation-Matrix, which is the transformation matrix that transform the skeleton from rest-pose to its world position. It is done in order to get a rotational invariant acceleration, which means that the deformations will not depend on how the pose is positioned and rotated in global space. Only the inverse of the rotational matrix part of the *globalPose* matrix is applied since the acceleration is a vector, not a point in space and therefore only needs to be rotated into a different space. The reason for choosing to describe it in rest-pose space and not in joint-space is that the delta-vectors are described in rest-pose space and therefore the acceleration needs to be described in the same space.

### Run-time

When the database is created, the second stage is to generate, for each frame at run-time, a key to look up into the database with and the key is the run-time acceleration. It is calculated for each joint for each frame as done in 4.21 and it is also described in joint coordinate space by multiplying the acceleration with the inverse of the rotational part of the global transformation matrix to describe the acceleration in rest-pose coordinate space.

The calculated acceleration described in joint space is used as input into the sample database and the same methods, as in normal pose space deformation with Radial Basis Functions, are used in order to interpolate a new dynamically deformed mesh from the samples in the database, where the input keys are the run-time joint accelerations instead.

Because of the fact that the vertices are only depending on the deformations of their closest joints when generating the acceleration needed, the vertices were classified to be influenced only by their closest joint and the joint's parent in order to nicely capture deformations around the overlapping regions as well. Not only does it capture deformation around overlapping regions nicely, it also speeds up the entire process significantly since for each joint, only a fraction of the mesh's vertices are needed in the calculation, instead of the entire mesh.

### 4.3.2 Spring-damper function generation

Dynamic deformations by using the pose space have, as mentioned, been efficiently done by Park and Hodgins [12], where the dynamic deformations are driven by the joint accelerations in Pose Space. Their approach was to separate the static deformations and the dynamic deformations, such as jiggling fat and muscle movement, and for the latter estimate a spring-damper function for each motion-capture marker. A spring-damper function is a good approximation since it is history dependent and can model the bouncing of the muscles or the jiggling of the fat as a spring which decays over time, a damper, if no more forces are applied to it.

This method generates for each vertex a spring-damper function by approximating the spring and damper coefficients by using captured data from for example a finite element muscle simulation

or, as in Park's case, motion captured data.

By approximating a spring-damper function for each vertex based on accurate simulation data that might have taken hours to produce, almost the same quality can be achieved but instead at near real-time frame rates. Here follows the method used to estimate a spring-damper function for each vertex based on simulated data.

### Spring-damper function generation

A spring is described by Hooks law as in equation 4.22

$$F_s = -kx \quad (4.22)$$

where  $F_s$  is the spring force stored by the spring,  $k$  is the spring constant and  $x$  is the displaced distance from the equilibrium position. The damping force is described in equation 4.23.

$$F_d = -cv = -c \frac{dx}{dt} \quad (4.23)$$

Where  $F_d$  is the damping force,  $c$  is the damping constant and  $v$  is the velocity of the spring.

The total force acting on the spring is the sum of the spring force  $F_s$ , the damping force  $F_d$  and the external force  $F_e$  which gives the following second order differential equation 4.24

$$F_{tot} = F_s + F_d + F_e, F_{tot} = ma \Leftrightarrow m \frac{d^2x}{dt^2} \Rightarrow m \frac{d^2x}{dt^2} + c \frac{dx}{dt} + kx = F_e \quad (4.24)$$

The external force  $F_e$  is in this case only the joint acceleration from the skeletal motion, where  $F_e$  is described as a weighted linear combination of the six acceleration components. Three dimensions for the linear accelerations  $a$  and three dimension for the angular accelerations  $\alpha$ , as seen in equation 4.25,

$$F_e = w_1a_x + w_2a_y + w_3a_z + w_4\alpha_x + w_5\alpha_y + w_6\alpha_z \quad (4.25)$$

where  $w_{1..6}$  are their corresponding weights.

The mass is not known but can be baked into the other constants by dividing each side in equation 4.24 with the mass, resulting in a second order differential equation described in per unit mass instead, which is here denoted as  $*$ , see equation 4.26

$$\frac{d^2x}{dt^2} + c^* \frac{dx}{dt} + k^*x = F_e^* \quad (4.26)$$



## Estimating the Spring-damper coefficients

The known inputs to equation 4.26 are only the joint accelerations captured from the skeletal motion, which means that there are eight unknown constants, namely  $c^*, k^*, w_1^*, w_2^*, w_3^*, w_4^*, w_5^*, w_6^*$ , in the second-order differential equation above that needs to be estimated for each vertex. Principal Component Analysis (PCA) can be applied to project it into a lower dimensional space to speed it up and also easier find a solution, but it has not been implemented due to the fact that this is a pre-processing stage and is therefore not that focused on speed.

Equation 4.26 can be linearized by describing the acceleration  $\frac{d^2x}{dt^2}$  and velocity  $\frac{dx}{dt}$  with their numerical derivatives see equations 4.27 and 4.28 below,

$$\frac{dx}{dt} = \frac{x(t) - x(t-1)}{\Delta t} \quad (4.27)$$

$$\frac{d^2x}{dt^2} = \frac{x(t+1) - 2x(t) + x(t-1)}{\Delta t^2} \quad (4.28)$$

which results in a linear equation with the eight unknown parameters, for each frame.

By putting together for each vertex all equations, a linear equation system is achieved, where the inverse of matrix  $A$  needs to be estimated in order to estimate the eight unknown parameters, see equation 4.29.

$$Ax = bx = A^{-1}b \quad (4.29)$$

In equation 4.29,  $A$  is a  $N \times 8$ -matrix containing all known parameters such as joint positions and frame rate, where  $N$  is the number of sample-frames. The vector containing all the eight unknown parameters is  $x$  which is the vector that needs to be solved by finding the inverse of matrix  $A$  and multiply it with the  $b$  vector, which contains known parameters of joint positions and frame rate. The matrix and vector structures for equation 4.29 is described in equation 4.30.

$$\begin{bmatrix} a_x & a_y & a_z & \alpha_x & \alpha_y & \alpha_z & -(x_i^t - x_i^{t-1})/\Delta t & -x_i^t \\ & & & & & \cdot & & \\ & & & & & \cdot & & \\ & & & & & \cdot & & \\ & & & & & \cdot & & \\ & & & & & \cdot & & \\ & & & & & \cdot & & \\ a_x & a_y & a_z & \alpha_x & \alpha_y & \alpha_z & -(x_N^t - x_N^{t-1})/\Delta t & -x_N^t \end{bmatrix} \begin{bmatrix} w_1^* \\ w_2^* \\ w_3^* \\ w_4^* \\ w_5^* \\ w_6^* \\ c^* \\ k^* \end{bmatrix} = \begin{bmatrix} (x_i^{t+1} + x_i^{t-1} - 2x_i^t)/\Delta t^2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ (x_N^{t+1} + x_N^{t-1} - 2x_N^t)/\Delta t^2 \end{bmatrix} \quad (4.30)$$

This linear equation system in equation 4.30, was efficiently solved by finding the pseudo-inverse of matrix  $A$  by using Singular Value Decomposition(SVD).

The Singular Value Decomposition was achieved by using the open-source linear algebra library called *Eigen*, where the chosen method used was the *JacobiSVD()* with *FullPivHouseholderQRPreconditioner* as preconditioner because of its accuracy. A preconditioner was necessary since the matrix was rectangular.

Once the eight unknown parameters were solved for each vertex, they were stored to be used at runtime for each vertex.

### 4.3.3 Usage

The pipeline of the dynamic pose space deformation method is pretty much the same as was done for the normal pose space deformation method. The thing added was that when generating the sample-pose database, a C++ function was written for the spring-damper approximation of the weight-coefficients that needed to solve a linear equation system. An extra database was created called *Dynamic-database* consisting of two different databases within, one for each dynamic method. Otherwise, the structure of the implementations is the same where only new features were hooked on to the existing structure as seen in figure 4.10.

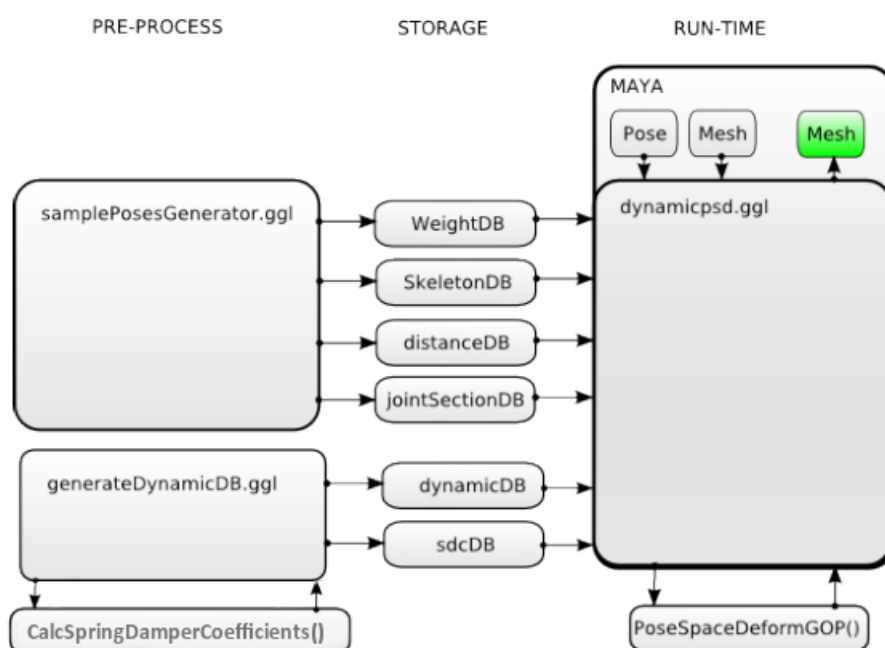


Figure 4.10: Overview of the Dynamic Pose Space Deformation method implementation

Here follows a simple step by step usage of the dynamic pose space deformation method that enables fast and complex dynamic deformations at run-time:

1. Create a set of high-detailed animated sequences which has deformations due to movement, such as a character jumping up and down with a complex muscle simulation, applied to it.
2. Take out a set of different poses based on the acceleration and store in a dynamic sample-pose database.
3. Apply dynamic pose space deformation, that will use the dynamic sample-pose database, on an arbitrary run-time pose where the acceleration at the current frame for the character's limbs are captured and used to interpolate a dynamically deformed mesh from the database at run-time.

And here follows a simple step by step usage of the spring damper approximation method that enables fast and complex dynamic deformations at run-time:

1. Create a set of high-detailed animated sequences which has deformations due to movement, such as a character jumping up and down with a complex muscle simulation, applied to it.
2. Approximate based on one these sequences for each vertex a spring-damper function which has eight parameters and store in a database.
3. use these eight approximated spring damper values per vertex on the run-time mesh and calculate at run-time the external forces produced by the joint accelerations.

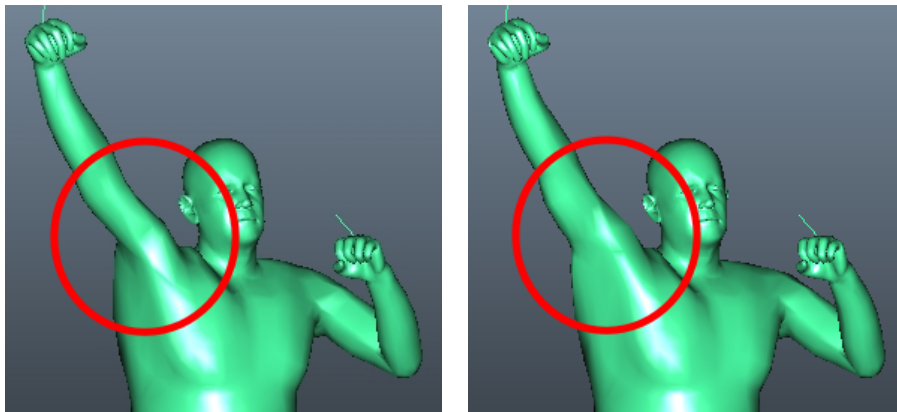
# Chapter 5

## Results

All results produced by the Stretchable and twistable bone skinning method, the pose space deformation methods and also the dynamic pose space deformation methods implemented for this thesis are shown below.

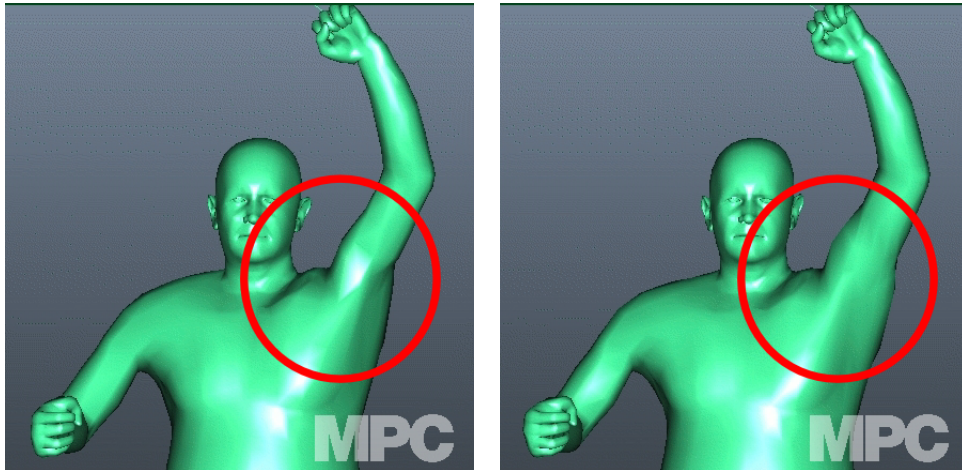
### 5.1 Stretchable and Twistable Bone Skinning

Figure 5.1 and 5.2 shows that the stretchable and twistable bone skinning methods improves the visual quality around rotated and twisted joints, such as around the armpits. The crowd characters can now have their arms raised with much more correctly rotated armpits and shoulders.



**Figure 5.1:** a) Normal linear blend skinning applied. b) Stretchable and twistable bone skinning applied. Note the difference around the shoulder for the raised arm in the red circle.

There are three key improvements when using the stretchable and twistable bone skinning method over the linear blend skinning or dual quaternion skinning method. Firstly, the candy-wrapper effect, that pinches around joints due to twisting, is gone when rotating an arm.



**Figure 5.2:** a) Normal linear blend skinning applied. b) Stretchable and twistable bone skinning applied. Note the difference around the shoulder for the raised arm in the red circle.

Secondly, not only is the candy-wrapper gone, it also rotates, for instance, an arm more correctly as seen in figures 5.1 and 5.2, giving better rotations for problem areas such as armpits, and lastly the bones can be scaled without having any overshoots such as hands getting elongated when scaling an forearm. The three key improvements are listed below.

- The candy-wrapper effect is gone
- Twisting is captured correctly and rotates naturally.
- Scaling bones is now possible without getting weird scaling issues on the mesh.

Unfortunately, no results of scaling the bones were captured since the current system didn't support scaling of the bones. The stretching part of equation 4.8 implemented was therefore commented out since it will not be used for now and would only add extra computational cost without doing anything.

The performance of the stretchable and twistable bone skinning method compared to linear blend skinning, which is the underlying skinning method for the stretchable and twistable bone skinning implementation, is shown in table 5.1.

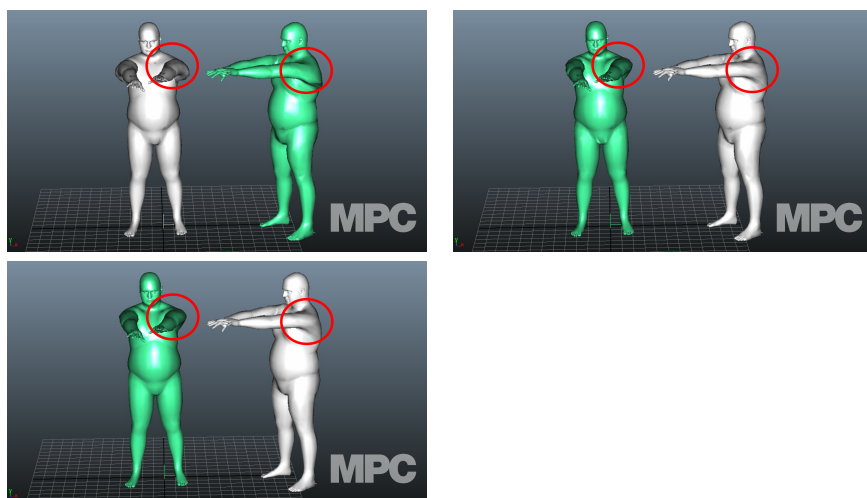
**Table 5.1:** Performance of the Stretchable and Twistable Bone Skinning method compared to linear blend skinning, which is the underlying skinning method for the STBS implementation.

Skinning method	Frames per second (FPS)
Linear Blend Skinning	24-25 FPS
Stretchable and Twistable Bone Skinning	20-22 FPS

## 5.2 Pose Space Deformation

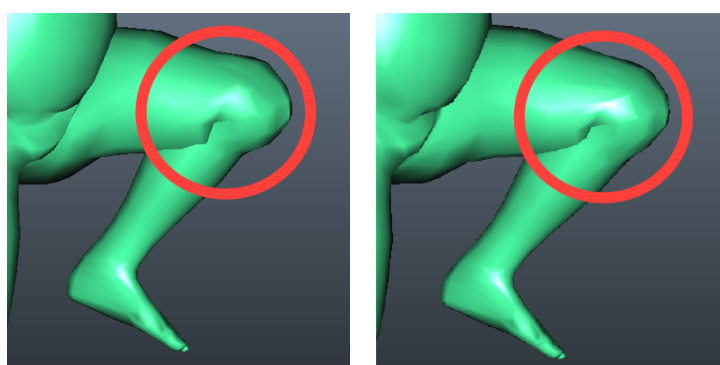
The figures shown in figure 5.3 are from top to bottom; normal dual quaternion skinning, pose space deformation applied and the bottom image is a reference mesh made with an advanced

rigging method for comparison. Note especially around the shoulder area where muscle bulging is added and skinning errors are reduced.



**Figure 5.3:** a) Normal Dual Quaternion Skinning applied. b) Pose space Deformation applied. c) Aimed simulated deformed mesh. Note the deformations around the armpits and clavicles.

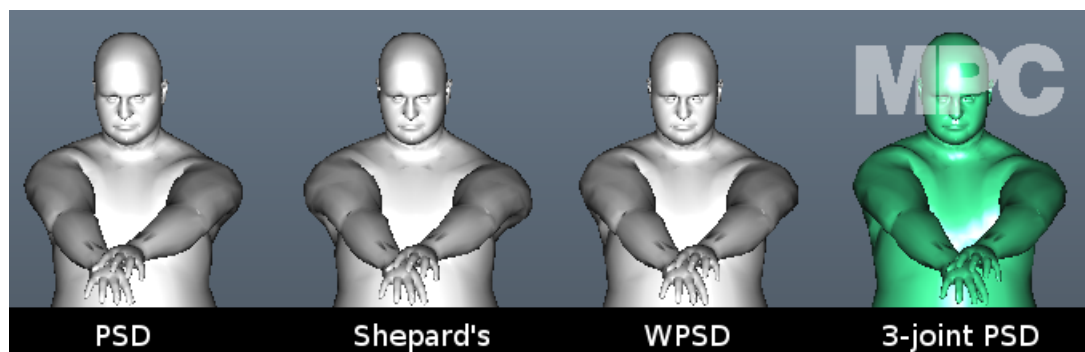
Pose space deformation also handle manual tweaking of the mesh as well so that for example a knee will deform nicely. It could have been the case for figure 5.4, however this was done with an advanced rig, but it could just as well have been manually hand-sculpted, making pose space deformation a powerful correction tool. In figure 5.4 note especially around the knee inside the red circle how the pose space deformation method deforms the knee into a much more pleasing result.



**Figure 5.4:** a) Normal Dual Quaternion Skinning applied. b) Pose space Deformation applied. Note the deformation difference around the knee in the red circle.

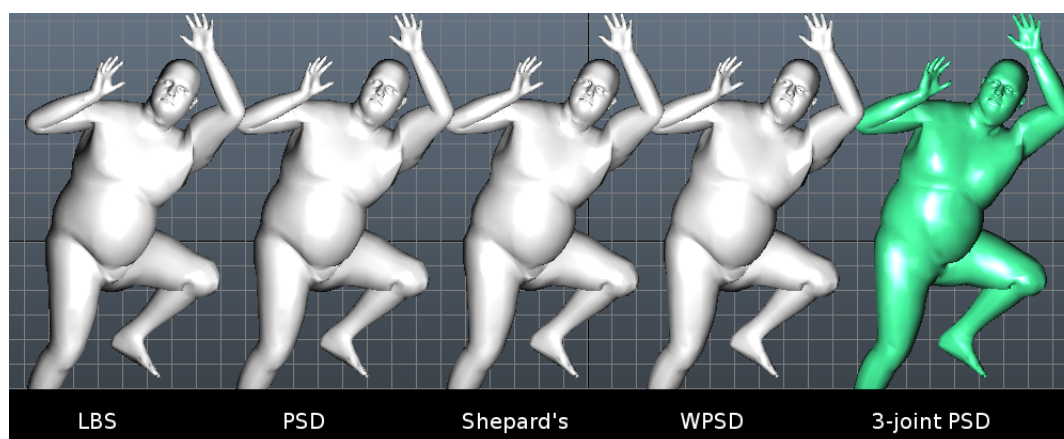
Several different pose space deformation methods were implemented, and figure 5.5 shows a comparison between the normal pose space deformation, the shepard's method, the weighted pose space deformation and also the three-joint bone-space pose space deformation method. It should be noted that this pose closely resembles a pose in the sample-pose database. Figure 5.3 c) would give a good hint on what the sample-pose would have looked like. As a result

all methods produce good results with muscle bulging around the chest and shoulder area, where the shepard's method is the worst of them, and the others are pretty much the same quality.



**Figure 5.5:** Pose space deformation comparison between from left to right: Normal Pose Space Deformation, Shepard's method, Weighted pose space deformation, 3-joint pose space deformation.

Figure 5.5 showed the difference between the different pose space deformation methods with a run-time pose chosen deliberately to be similar to a pose in the sample-pose database to show how it would look like. Furthermore, figure 5.6 shows how the different methods cope with trying to interpolate deformations onto a very arbitrary pose that is chosen to be very far from the sample-poses in the database. All sample-poses in the database are only standing poses and are somewhat left and right symmetric between the arms and legs, whilst the run-time pose is of a guy crawling on the floor with non symmetric movement between the left and right arm and leg. The result of the different pose space deformation methods are shown in figure 5.6 where the methods are from left to right; Linear blend skinning, normal pose space deformation, Shepard's method, weighted pose space deformation and 3-joint pose space deformation.



**Figure 5.6:** Comparison between several different pose space deformation methods and even linear blend skinning at a very arbitrary pose. The methods from left to right: Linear blend skinning, normal pose space deformation, Shepard's method, weighted pose space deformation, 3-joint pose space deformation.

The result of this comparison showed that all methods except for the 3-joint local look-up pose space deformation method did not do so well. By comparing the linear blend skinning method

with the normal pose space deformation, Shepard's and weighted pose space deformation method, they all pretty much look the same as linear blend skinning, concluding that good sample-poses were hard to find for interpolation. The 3-joint pose space deformation method, which does a local lookup and do not have to worry about symmetry or anything of that sort, works much better in finding and interpolating good mesh deformations. Note especially around the knees and shoulder in figure 5.4. It should also be noted that the Shepard's method is somewhat the second best method in this test if compared to the other methods in figure 5.6. This is due to the fact that it will, if far from any sample-poses, converge to the average of all sample-poses in the database, which is not desired.

Table 5.2 shows the speed of which all the different pose space deformation algorithms runs at with a sample-database of 120 samples. It should be noted that the frames per seconds drops as the number of samples increases. Furthermore, the time for the Shepard's method is very slow because of the brute force implementation of it. Not much attention was given to the Shepard's method since it had the worst visual quality and the theory behind it also revealed that it would perform worst of the bunch.

**Table 5.2:** Performance of the different Pose Space Deformation techniques implemented.

Pose Space Deformation method (120 samples)	Speed for performing the function
PSD	65ms (14-16 fps)
Shepard's method	18300ms (0.05 fps)
WPSD	1020ms (1 fps)
Local PSD - bone space	260ms (4 fps)
Local PSD - angles	260ms (4 fps)

### 5.3 Dynamic Pose Space Deformation

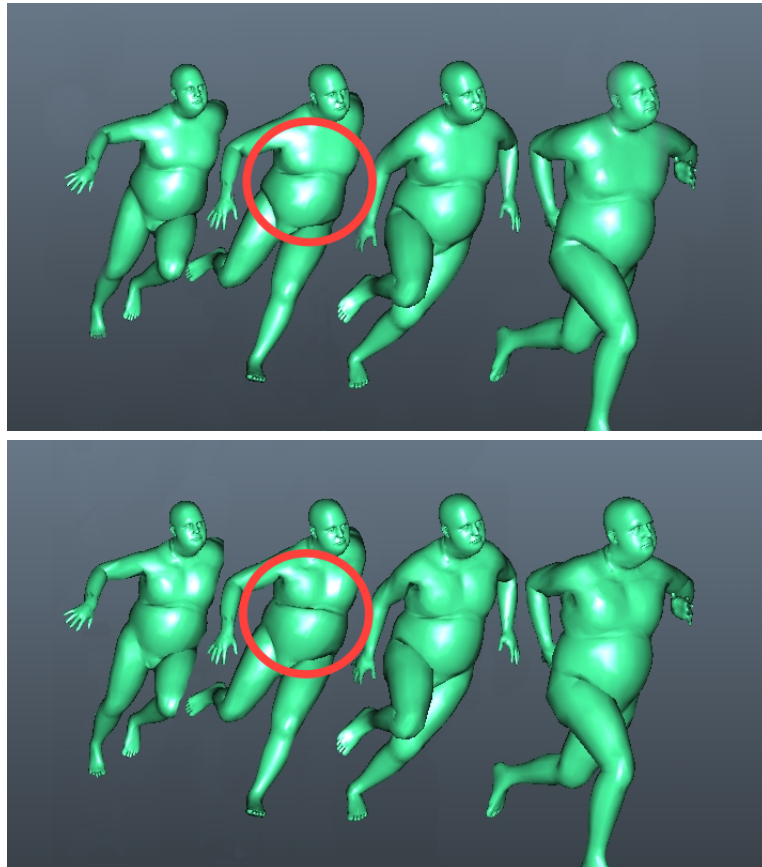
Dynamic effects are very important in order to make a character look real when moving. Without dynamic effects the character's movements would look stiff. Here follows some results made with dynamic pose space deformations.

Figure 5.7 shows first a basic skinning deformation sequence of a running guy, the second image below shows the dynamic pose space deformation method using the joint accelerations method described in section 4.3.1.

The speed of the dynamic pose space deformation is comparable to the speed of local pose space deformation methods but it is a bit slower. It runs at around 1-3 frames per second, when having a sample-pose database consisting of between 20-30 sample-poses.

Results of the approximated spring-damper method was never produced since it had trouble estimating all the unknown coefficients. More varied simulations were probably needed, but more on that in the next section.





**Figure 5.7:** a) Normal Dual Quaternion Skinning applied. b) Joint accelerated Dynamic Pose space Deformation applied. Note the deformation difference around the belly in the red circle.

# Chapter 6

## Conclusion

In this chapter, conclusions are drawn about all implemented methods. Moreover, limitations and future works are also gone through.

### 6.1 Improving the basic skinning methods

It was easy for MPC to switch to the Twistable bone skinning method since the pipeline for it was the same as for the basic skinning methods, linear blend skinning and dual quaternion skinning, used at MPC. Furthermore, the method did not require any new inputs nor any new training of the staff as well, which made it very easy for the users to start using it.

The method will work and look exactly the same as linear blend skinning for cases where no twisting or scaling is applied. The stretchable and twistable bone skinning method also runs at only 2-4 frames per seconds slower than the basic skinning methods, but generates much better visual quality compared to the normal linear blend skinning method.

When no rotation or scaling is applied the method will work and look exactly the same as linear blend skinning. There are not much of a performance loss, where only 2-4 frames per seconds are lost but generates much better visual quality compared to the normal linear blend skinning method.

The conclusion is that the stretchable and twistable bone skinning method was a good choice to implement in order to improve the basic skinning methods, since it improves the visual quality of the crowd characters without being very computational heavy, making the method run almost as fast as the underlying skinning method.

As of now, the stretchable and twistable bone skinning method is currently used in production within MPC's crowd system software *ALICE*, producing better visual quality than in the past for cases with scaled and twisted bones.

## 6.2 Pose Space Deformation

Pose Space deformation is a powerful interpolation tool that can be used to correct errors such as the collapsing-elbow effect and, moreover, makes it possible to have complex skin deformations applied to the mesh at a fraction of the time it would have taken if it would have been driven by a complex rig or simulation. The pose space deformation method also enables the user to manually correct errors or tweak the look of a deformation by hand, making it possible to have art-directed deformations.

It can also significantly enhance the visual quality of a character since it can interpolate already pre-calculated muscle deformations at near real-time onto the run-time mesh, which might otherwise have taken a very long time to simulate. The method also makes it possible for characters, such as crowds, to actually get complex deformations applied to them. Something which could not have been done in the past since it would have been too time consuming to simulate.

A thorough implementation of several pose space deformation methods currently used in the industry have been implemented and also some improvements to the current methods have been proposed. The improvements made it possible to deform an arbitrary pose independently of how it was rotated in world space which also resulted in that far less sample poses were needed.

Since the improved method does the lookup and matching of the sample-poses in the database on a per 3-joint basis, a smaller amount of sample-poses are needed compared to the normal pose space deformation method. The amount of sample-poses needed was as low as 40-50 poses when using the 3-joint pose space deformation method without reducing the visual quality so much compared to the normal pose space deformation method which needed at least 80-120 samples. Preferably, however, more sample-poses are necessary to make the methods robust for production. The quality of the pose space deformations methods mostly depended on the poses chosen to be stored in the database and how close the run-time pose resembled those stored poses. To be on the safe side it was better to store additional sample-poses in the database, even-though the speed was somewhat reduced.

The method to be preferred of the two database-creation methods was the first method in terms of speed and a good variation in the database, which is true if, and only if, the animation sequence is varied and is not a sequence with very similar or many looping animations. However, if the second method would have been faster and never gotten stuck in a local-minimum it would have been the method to prefer since the extreme poses are the best poses to be stored in the database.

The pose space deformation methods can only handle static deformations such as flexing muscles, skin deformations and corrections, they do not handle dynamic deformations such as fat and muscle bouncing due to a force impulse created when for example running or jumping. They cannot handle dynamic effects since the values in the methods are not history dependent which is necessary for dynamic effects.

## 6.3 Dynamic Pose Space Deformation

Previously, only a few hero-level characters were able to have complex dynamic simulations applied, such as fat or muscle deformations, due to the large computational costs. With dynamic pose space deformation though, the heavy simulations only need to be simulated once in a pre-processing stage and only lookups at run-time are required, which makes the dynamic pose space deformation much faster and very well suited for crowd characters. It makes it possible to have more characters get complex deformations applied, which greatly improves the visual quality of the crowds.

When accelerations are varied smoothly over time, such as when running, then the difference in terms of visual quality compared to a computer-heavy simulated muscle deformation sequence will be very small but it will run at a much greater speed than the advanced simulation. When the accelerations are not smoothly varying, say an impulse is created only when a guy is hitting the floor from falling, the impulse will be instant and this method will not work that well. This is because of the fact that since dynamic effects are history dependent, they will depend on what happened in the previous frames where this method only takes into account three frames at a time in order to calculate the acceleration at that particular frame. By doing so the deformation effect of the impulse would be gone after three frames and the bouncing of fat would stop instantly. However, it is not that noticeable since muscles and fat do not tend to oscillate much when an impulse is applied, making this a valid choice.

Currently, the method to be preferred is the joint accelerated method which is based on the normal pose space deformation interpolation method. The spring-damper approximation method is currently not working very well, and probably needs to have a new and more varied sample sequence in order to efficiently estimate the eight unknown parameters from the linear equation system.

The spring-damper approximation method is, however, not a pose space deformation method in the sense that it will not interpolate between any known samples. It will estimate from a set of sample-poses coefficients for a mass-spring damper system that is run at run-time. This method could have been applied directly if the coefficients were manually painted on to the mesh, but it is time consuming to paint weights for hundreds of characters.

## 6.4 Future work

The methods implemented for improving the visual quality of the crowd system *ALICE* have still some limitations that could be further worked on and improved. Here follows some limitations and future work for the stretchable and twistable bone skinning method, the pose space deformation methods and the dynamic pose space deformation methods.

### 6.4.1 Improving the basic skinning methods

Since the chosen method was the projection method, which was the fastest method of them all for determining the point-weight for a given vertex, issues arise for vertices which are located further away from the end joints of the bone. A projected weight could be for instance, 1.5 when the maximum weight is one and in order to correct this the point-weights that are less than zero

or larger than one are clamped so every weight is between zero and one. This, however, causes some artifacts as can be seen in figure 6.1, which is due to the fact that the vertices that have a weight one, are rigidly following the bone's rotation. These weights should instead have a weight that would be less than one in such a way that they will not rigidly be rotated with the bone. A falloff weight function could be used, or something similar, in order to remove this problem in such a way that far from the bone the point-weights will decrease. This problem could also be removed by using better point-weight approximation methods such as the Bone Heat Weights method or Bounded Biharmonic Weights method which are far more accurate but are much more computational heavy.

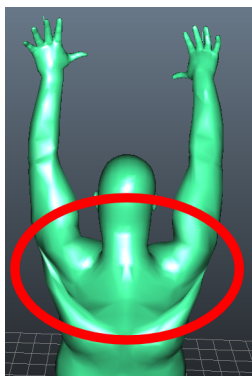


Figure 6.1: Weight issues for vertices that have a point-weight larger than one.

## 6.4.2 Pose Space Deformation

Out of all the Pose-Space deformation methods implemented, only two out of five works well for arbitrary poses, both of which are based on the proposed local 3-joint look-up method. The other ones have a lot of trouble finding a good pose for an arbitrary pose, with few samples in the database, and usually fail miserably.

Since pose space deformation is an interpolation method which relies only on the pose at the current frame and not on how it was deformed in the previous frames, some flickering might occur at low values of  $\sigma$ . This is because when a low value is chosen it tries to find one or a few sample-poses that are close to the run-time pose which might from frame to frame vary heavily and therefore give some flickering because it chooses different poses every time. If instead a higher value is used, more samples are taken into account where among these samples, a good chance is that the same sample poses from the previous frame are also being taken into account in the current frame making the flickering go away.

The test cases have only been tested with one piece of geometry. When two pieces of geometry are used, two databases needs to be stored, one for each piece of geometry, for example one for the human mesh, and one for the shirt. The pose space deformation method could be applied to both meshes separately if the two meshes have gone through the same simulation. If not, the shirt would have to know how the skin has been deformed in order to follow the skin nicely and not intersect with it. In that case, one approach would be to implement an output function that outputs only the delta-vectors from the pose space deformation method with the skin so those delta-vectors can be used as inputs to the second pose space deformation method that is applied

on the shirt.

The cloth mesh needs to have some knowledge of which joint or joints it is most affected by, for instance if it could be skinned in the same way as is done with a skinned mesh, which have been assigned bone weights for each vertex, the cloth mesh could have pose space deformation assigned to it as well. Since the cloth mesh is usually never assigned to a specific bone and is instead just being deformed by the movement of the skin, no skinning data is assigned to the mesh. In order for the pose space deformation method to work, it needs to transform the meshes into the rest-position space which is done by applying the inverse of the global transformation matrix on each vertex. However, as mentioned, cloth meshes usually do not have any bone-weights assigned to them resulting in that a transformation matrix does not exist. A future implementation could be to, for each cloth vertex, find the closest vertex on the skinned mesh and use that vertex's bone-weights and transformation matrix in order to transform the cloth into the rest position. It could be done as a preprocessing step, but if the cloth should be able to slide along the skin, different vertices will be the closest ones depending on each frame, so by finding the closest skin-vertices on a per frame basis could improve the quality but it will most certainly decrease the run-time speed by a lot. The simplest solution would be to have the cloth skinned to the skeleton and in that way get the bone-weights and transformation matrices needed in order to perform pose space deformation, which would work perfectly for tight-fitted cloth.

Another limitation is that the pose space deformation, as of now, only can deform the same mesh that has been stored and generated for the sample-database, because it is vertex-based driven. If the delta vectors generated from the sample-data for each vertex are not exactly matching the same vertices in the run-time mesh, the pose space deformation methods will fail because the delta-vectors will not be contributing to the correct vertex any more. It increases the amount of databases needed and also the amount of work needed to generate pose space deformations, since for each mesh a database needs to be stored.

A future improvement could be to implement an interpolation method that can transfer deformations from one similar mesh to another mesh. For example, if two characters are very similar in size and muscle-type, only one database could be used for both instead since the vertices will be closely matching each-other. It could be done by simulating a database for one of the characters and then for the other character use the same database but for each vertex in the database find the closest vertex/vertices on the character's mesh and interpolate the deformation values stored in the database onto them by using barycentric coordinates, as was done by Wang et al.[9].

The two database creation methods that chooses the different sample-poses to be stored in the database was far from the best techniques to be used, but worked quite well. The simple method is fast but does not know about the poses it stores. It only stores every  $N$ :th pose in an animated sequence regardless of how the pose looks. So a varied sequence is necessary in that case, for example a range-of-motion sequence is needed to get a good variation of sample-poses in the database. For the second method a greedy method was implemented in order to find extreme poses for an arbitrary sequence. However, since it was a greedy method it took some time to generate the samples needed, where it got slower for each sample pose added to the database. In the case of longer sequences with at least one very arbitrary pose, a local-minimum was often found and this generated a lot of the same, or similar, sample-poses which were not desirable, so the other method is the one to prefer at the moment. However if no local-minimum would be found and if it would be much faster to compute, this method would have been the preferred method since it extracts extreme poses, which are the best poses to interpolate with.

Normal pose space deformation only takes into account the current pose and interpolates a deformed mesh for that pose. It does not handle any form of dynamic deformation such as bouncing fat or muscles due to movement of the skeleton and external forces. A dynamic pose space deformation method have been implemented for that case which is discussed in section 6.4.3.

Another future improvement is to get rid of the fact that the skinning method that is to be used at run-time, such as linear blend skinning or dual quaternion skinning, must be the same as the underlying skinning method used when simulating the data for the sample-pose database. A Powell optimization method[20] could be used instead to remove the need of having to have the same underlying skinning method as the one used for creating the sample-poses in the sample-pose database.

### 6.4.3 Dynamic Pose Space Deformation

The joint acceleration method have some flickering if a small amount of dynamic variance  $\sigma$  is applied to it, otherwise it works quite well. The spring-damper approximation method is not working very well, however the implementation for it is all there. The reason for it not behaving nicely is probably mostly because of the fact that the training data is not varied enough to be able to accurately generate spring-damper and force parameters when solving the linear equation system mentioned above. Park and Hodgins had a sample sequence of 600 frames whilst this method has only been tested with 200 samples where most of the samples were very similar. The spring and damper constants looks like they are both well approximated but the force weights tend to be too small, or badly approximated, so the total external force gets either too small to have an impact on the spring-damper function so it looks stiff, or it gets too high and breaks the mesh. A new longer sample sequence with more varying skeletal motion would therefore be interesting to test in order to see if it could improve the estimated values and actually generate good results with this method.

Since the spring-damper approximation method works with the deformed delta-vectors produced by the spring-damper function, which are described as the delta-vectors between the deformed mesh in rest-pose space and the undeformed mesh, problems can occur if normal pose space deformation is applied as well. It is because of the fact that the delta-vectors will also take into account the delta-vectors generated by the normal pose space deformation method, which makes the delta-vectors different, often longer, than what they should be and causes the spring-damper function to behave in unexpected ways. To fix this, the delta-vectors should be derived from the deformed mesh with normal pose space deformation applied to it in rest pose space and the deformed mesh with dynamics applied in rest-pose space instead of doing as is normally done, to take out the delta-vectors between the deformed mesh in rest-pose and the undeformed mesh.

A test has been made for a very arbitrary pose compared to the poses in the database in order to test how well the dynamic effects perform on arbitrary poses. It looks as if it works but since the test has only been performed on one arbitrary sequence of a guy crawling on the floor and only having standing sample-poses in the database, other cases might not work. More tests have to be made to make sure that it works in several cases.

The dynamic joint acceleration method, which is a variant of the normal pose space deformation method, calculates for each frame its accelerations and uses only that acceleration as lookup into a sample-pose database as done with normal pose space deformation. By only deforming the

mesh on a per frame basis even though the acceleration relies on what happened in the previous frames it can cause some flickering if the acceleration is not smooth enough. For instance, if a guy is running the method works well since the acceleration varies slowly. If the acceleration varies very rapidly, say only one impulse occurs at one frame, the forthcoming frames should have some bouncing and dynamic deformations of the skin and fat due to that impulse even-though there might not be any acceleration/impulse at the current frame. This method will not be able to capture those forthcoming bouncing effects since it interpolates a new mesh for each frame, depending only on what the current frame's acceleration is. However, muscles and fat tend to become stationary very quickly after an impulse which makes this method still a valid method, especially for smoothly varying accelerations.

A hybrid method of both the joint acceleration method and the spring-damper approximation method could be a future implementation where the interpolated delta-vectors generated from the joint acceleration method could be fed into the spring-damper system approximation method, where the external force has been removed to only simulate a spring-damper function that decays over time to its equilibrium. It will then get nicely deformed delta-vectors produced from simulated data by the joint acceleration method, and then decay over time in a history-dependent manner with estimated spring-damper coefficients generated from real simulation data with the spring-damper approximation method.

As with normal pose space deformation, the dynamic pose space deformation only works at the moment with the same mesh that was used to create the database and therefore each mesh needs a dynamic database. The same approach as described in section 6.4.2, for removing the need of having one database per mesh, could be used for the dynamic pose space deformation method as well in order to reduce the number of dynamic databases needed.



# Bibliography

- [1] A. Mohr and M. Gleicher, "Building efficient, accurate character skins from examples," in *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, (New York, NY, USA), pp. 562–568, ACM, 2003.
- [2] J. P. Lewis, M. Cordner, and N. Fong, "Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, (New York, NY, USA), pp. 165–172, ACM Press/Addison-Wesley Publishing Co., 2000.
- [3] L. Kavan, S. Collins, J. Žára, and C. O'Sullivan, "Skinning with dual quaternions," in *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, I3D '07, (New York, NY, USA), pp. 39–46, ACM, 2007.
- [4] A. Jacobson and O. Sorkine, "Stretchable and twistable bones for skeletal shape deformation," in *Proceedings of the 2011 SIGGRAPH Asia Conference*, SA '11, (New York, NY, USA), pp. 165:1–165:8, ACM, 2011.
- [5] T.-Y. Kim and E. Vendrovsky, "Drivenshape: a data-driven approach for shape deformation," in *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '08, (Aire-la-Ville, Switzerland, Switzerland), pp. 49–55, Eurographics Association, 2008.
- [6] T. Kurihara and N. Miyata, "Modeling deformable human hands from medical images," in *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '04, (Aire-la-Ville, Switzerland, Switzerland), pp. 355–363, Eurographics Association, 2004.
- [7] X. C. Wang and C. Phillips, "Multi-weight enveloping: least-squares approximation techniques for skin animation," in *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '02, (New York, NY, USA), pp. 129–138, ACM, 2002.
- [8] O. Weber, O. Sorkine, Y. Lipman, and C. Gotsman, "Context-aware skeletal shape deformation," *Computer Graphics Forum (Proceedings of Eurographics)*, vol. 26, no. 3, 2007.
- [9] H. Wang, F. Hecht, R. Ramamoorthi, and J. O'Brien, "Example-based wrinkle synthesis for clothing animation," *ACM Trans. Graph.*, vol. 29, pp. 107:1–107:8, July 2010.
- [10] W.-W. Feng, Y. Yu, and B.-U. Kim, "A deformation transformer for real-time cloth animation," *ACM Trans. Graph.*, vol. 29, pp. 108:1–108:9, July 2010.
- [11] M. Müller and N. Chentanez, "Wrinkle meshes," in *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, (Aire-la-Ville, Switzerland, Switzerland), pp. 85–92, Eurographics Association, 2010.

- [12] S. I. Park and J. K. Hodgins, "Data-driven modeling of skin and muscle deformation," in *ACM SIGGRAPH 2008 papers*, SIGGRAPH '08, (New York, NY, USA), pp. 96:1–96:6, ACM, 2008.
- [13] I. Ragnemalm, *Polygons Feel No Pain So How Can We Make Them Scream?*, vol. 2, ch. 11 Body Animation, skinning, pp. 145–156. Ingemar Ragnemalm, 2011.
- [14] M. DeLoura, *Game Programming Gems 1*, ch. 4.15 Filling the Gaps - Advanced Animation Using Stitching and Skinning, pp. 476–483. Charles River Media, 2000.
- [15] D. T. Chen and D. Zeltzer, "Pump it up: computer animation of a biomechanically based model of muscle using the finite element method," in *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '92, (New York, NY, USA), pp. 89–98, ACM, 1992.
- [16] L. Kavan, S. Collins, J. Žára, and C. O'Sullivan, "Geometric skinning with approximate dual quaternion blending," *ACM Trans. Graph.*, vol. 27, pp. 105:1–105:23, Nov. 2008.
- [17] E. Dam, M. Koch, and M. Lillholm, "Quaternions, interpolation and animation," tech. rep., 1998.
- [18] I. Baran and J. Popović, "Automatic rigging and animation of 3d characters," in *ACM SIGGRAPH 2007 papers*, SIGGRAPH '07, (New York, NY, USA), ACM, 2007.
- [19] A. Jacobson, I. Baran, J. Popović, and O. Sorkine, "Bounded biharmonic weights for real-time deformation," in *ACM SIGGRAPH 2011 papers*, SIGGRAPH '11, (New York, NY, USA), pp. 78:1–78:8, ACM, 2011.
- [20] Xian, Soon, Feng, Lewis, and Fong, "A powell optimization approach for example-based skinning in a production animation environment," *In Computer Animation and Social Agents*, pp. 141–150, 2006.
- [21] J. P. Lewis, F. Pighin, and K. Anjyo, "Scattered data interpolation and approximation for computer graphics," in *ACM SIGGRAPH ASIA 2010 Courses*, SA '10, (New York, NY, USA), pp. 2:1–2:73, ACM, 2010.