Université de Montréal

Matrix-based Parameterizations of Skeletal Animated Appearance

par
M. Cihan Özer

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en computer science

février, 2017

RÉSUMÉ

Alors que le rendu réaliste gagne de l'ampleur dans l'industrie, les techniques à la fois photoréalistes et basées sur la physique, complexes en terme de temps de calcul, requièrent souvent une étape de précalcul hors-ligne. Les applications en temps réel, comme les jeux vidéo et la réalité virtuelle, se basent sur des techniques d'approximation et de précalcul pour atteindre des résultats réalistes. L'objectif de ce mémoire est l'investigation de différentes paramétrisations animées pour concevoir une technique d'approximation de rendu réaliste en temps réel.

Notre investigation se concentre sur le rendu d'effets visuels appliqués à des personnages animés par modèle d'armature squelettique. Des paramétrisations combinant des données de mouvement et d'apparence nous permettent l'extraction de paramètres pour le processus en temps réel. Établir une dépendance linéaire entre le mouvement et l'apparence est ainsi au coeur de notre méthode.

Nous nous concentrons sur l'occultation ambiante, où la simulation de l'occultation est causée par des objets à proximité bloquant la lumière environnante, jugée uniforme. L'occultation ambiante est une technique indépendante du point de vue, et elle est désormais essentielle pour le réalisme en temps réel. Nous examinons plusieurs paramétrisations qui traitent l'espace du maillage en fonction de l'information d'animation par squelette et/ou du maillage géométrique.

Nous sommes capables d'approximer la réalité pour l'occultation ambiante avec une faible erreur. Notre technique pourrait également être étendue à d'autres effets visuels tels le rendu de la peau humaine (diffusion sous-surface), les changements de couleur dépendant du point de vue, les déformations musculaires, la fourrure ou encore les vêtements.

Mots clés: infographie, animation squelettique, enveloppe géométrique, déformations, harmoniques de variété, réflectance.

ABSTRACT

While realistic rendering gains more popularity in industry, photorealistic and physically-based techniques often necessitate offline processing due to their computational complexity. Real-time applications, such as video games and virtual reality, rely mostly on approximation and precomputation techniques to achieve realistic results. The objective of this thesis is to investigate different animated parameterizations in order to devise a technique that can approximate realistic rendering results in real time.

Our investigation focuses on rendering visual effects applied to skinned skeleton-based characters. Combined parameterizations of motion and appearance data are used to extract parameters that can be used in a real-time approximation. Trying to establish a linear dependency between motion and appearance is the basis of our method.

We focus on ambient occlusion, a simulation of shadowing caused by objects that block ambient light. Ambient occlusion is a view-independent technique important for realism. We consider different parameterization techniques that treat the mesh space depending on skeletal animation information and/or mesh geometry.

We are able to approximate ground-truth ambient occlusion with low error. Our technique can also be extended to different visual effects, such as rendering human skin (subsurface scattering), changes in color due to the view orientation, deformation of muscles, fur, or clothes.

Keywords: Computer graphics, skeletal animation, skinning, deformations, manifold harmonics, shading.

CONTENTS

RÉSUMÉ . . . . **ii**

ABSTRACT . . . . **iii**

CONTENTS . . . . **iv**

LIST OF TABLES . . . . **vi**

LIST OF FIGURES . . . . **vii**

DEDICATION . . . . **xiii**

ACKNOWLEDGMENTS . . . . **xiv**

CHAPTER 1: INTRODUCTION **1**

CHAPTER 2: PRELIMINARY **3**

2.1 Animation Basics 3

2.2 Skinning 9

2.3 Ambient Occlusion 20

2.4 Ambient Occlusion Approximations 24

2.5 Manifold Harmonics 27

CHAPTER 3: COMBINED PARAMETERIZATIONS **35**

3.1 Joint Angles 37

3.2 Vertex Positions 39

3.3 Joint Positions 46

3.4 Manifold Harmonics 47

CHAPTER 4: RESULTS AND DISCUSSION **58**

CHAPTER 5: CONCLUSION AND FUTURE WORK **66**

BIBLIOGRAPHY **68**

LIST OF TABLES

3.I MSE of striking animation with different numbers of parameters. The same amount of the per-vertex error rates dominates the mean-squared error until less than 50% of the singular values are used. . . 46

4.I MSE of the "striking", "dying", and "walking" animations with joint angle parameterization using a mesh with a poor geometry. The "running" (forward and backward), "dying", and "striking" animations are learnt. 59

4.II MSE of the "striking", "dying", and "walking" animations for clustered vertex position parameterization using a mesh with a poor geometry. "running" (forward and backward), "dying", and "striking" animations are learnt. 61

LIST OF FIGURES

2.1 A scene from "Pinocchio" (1940, ©The Walt Disney Company). Image from Wikipedia "Pinocchio (1940 film)" article (https://en.wikipedia.org/wiki/Pinocchio_(1940_film)). 4

2.2 An animator poses a character for a single frame of an animation sequence. Image from Buzzfeed "20 Crazy Facts About The Making Of "The Nightmare Before Christmas" article by Arielle Calderon (https://www.buzzfeed.com/ariellecalderon/the-making-of-the-nightmare-before-christmas). 5

2.3 A scene from "Toy Story" (1995, ©The Walt Disney Company). Image from "Toy Story" official website (http://toystory.disney.com/). 6

2.4 A skeleton is rigged to a 2D character being animated. Image from Marionette Studio "Skeletal Animation" web page (http://marionettestudio.com/skeletal-animation/). . 7

2.5 Andy Serkis portrays "Gollum" on the performance capture stage. Image from Andy Serkis' official website (http://serkis.com). 8

2.6 Skeleton structure containing joints (blue spheres), bones (yellow tetrahedra) and skin (grey polygons). The root matrix positions the global mesh, and multiplies hierarchically all other matrices of the tree structure. Image from Leonid Sigal, "Human Motion Modeling and Analysis" course slides. 10

2.7 Painting weights on the shoulder. Color range goes from blue (closer to 0), green, to red (closer to 1). Image from Digital Tutors, "Weight Painting in Blender" page (https://www.pluralsight.com/blog/tutorials/weight-painting-in-blender). 11

2.8 A mesh in "T-pose". Image from cgtrader, "Fatman T-Pose 3D model". 13

2.9 Skinning on a 2D cylinder. An unbent knee with skin attached to joints 1 and 2 (left) is skinned using a weighted sum of the skinning transformations (right). $W1$ and $W2$ are world matrices of the joints. 14

2.10 Vertex transformation from local space to world space using matrix W. Vertex v in joint's local space (left). Vertex v in world space (right). 15

2.11 Candy-wrapper artifact of LBS. Reference pose (left), and animated pose (right). 15

2.12 Rest pose (left), candy-wrapper effect (middle), and collapsing elbow (right). Image from "libigl Tutorial" on GitHub. 17

2.13 DQB fails when the rotation centers of the vertices differ considerably. In this case, deformed vertices disintegrate and cracks appear. Image from Kavan et al. [4]. 17

2.14 LMB fails to identify the shortest path. Image from Kavan et al. [4]. 18

2.15 Skin flipping artifact caused by the shortest path when rotating from 179 to 181 degrees. Image from Kavan et al. [4]. 19

2.16 Ray-traced ambient occlusion. Image based on Jorge Pimentel's rendering. 20

2.17 Ray-traced ground truth (left), Kontkanen and Aila [6] (middle), comparison of the images (right). Image from Kontkanen and Aila [6]. 25

2.18 Comparison between Kirk and Arikan [5] (middle) and ray-traced ground truth (left). Image from Kirk and Arikan [5]. 26

2.19 Mesh segmentation application. 3D mesh data is transformed to 2D contour data by preserving the geometric features. Figure based on Zhang et al. [21]) 28

2.20 Opposite angles of the edge ij are used for computing the cotangent Laplacian (Image prepared based on Sorkine [14]). 31

2.21 Because of losing symmetry, all the meshes are deformed except mesh E. Image is taken from Vallet and Levy [18]. 32

2.22 Smoothing process of the Stanford bunny with 3485 (original), 1024, 97, and 6 MH bases, respectively. 33

3.1 Ambient occlusion ground truth (left), and approximation using joint angle parameterization (right). Walking and running animations are learnt, walking animation is approximated. Since the error rate is very small, there is no visual difference between the approximation and ground truth. 38

3.2 Joint angle parameterization suffers flickering for the complex motions that depends multiple animation parameters, i.e., while armpit area suffers flickering, shadows casting by the arms to torso approximated error prone. Ground truth (left), joint angle approximation (middle) and 10 times magnified difference (right). All frames of the striking animation are used to train the model. . . . 39

3.3 Ambient occlusion approximation using clustered vertex position parameterization. Learning and approximating walking animation. Since walking animation does not have many frames (only 30 frames), overfitting does not happen and regularization is not required. 42

3.4 Clustered vertex position parameterization without regularization suffers from wrong approximations since the system gets confused during the parameter extraction process because of working with a large number of animation parameters. 43

3.5 Fixed results using regularization. 44

3.6 Approximation of striking animation using 50% of the singular values (left), and 10% of the singular values (right). The approximation fails for geometries (such as armpit) or areas (such as arms and torso) that break the assumptions that appearance of the object depends linearly on the motion parameters, and that the motion parameters can be handled independently. These assumptions were made by Kontkanen and Aila [6]. 45
3.7 SVD plots for a typical cluster. 46
3.8 Clustered vertex position parameterization also suffers flickering for the complex motions that depends on multiple animation parameters, i.e., while armpit area suffers flickering, shadows casting by the arms to torso approximated error prone. Ground truth (left), joint angle approximation (middle) and 10 $\times magnifieddifference(right).Allframesoft$
3.9 Ambient occlusion approximation based on MH bases parameterization on the bunny. Left, approximation with all MH bases approximation. Right, approximation with only the first (the smallest) base. AO is fully reconstructed with all MH bases, and it is identical for all vertices when only the first basis is used. 48
3.10 Ambient occlusion approximation based on MH bases parameterization. 100%, 91%, 72%, 32%, and 8% of the bases in use respectively. 49
3.11 MH bases approximation using the first basis. Left: $w = 0$ (original). Middle: $w = -0.5$. Right: $w = 0.5$. Only $w = 0$ gets uniform results when only the first basis is used. Gamma correction was applied to these images. 51
3.12 Clusters for the Amazing Spiderman mesh. 0.26 is used as the threshold. "Amazing Spiderman Rigged V2" mesh by "bLender" user on Blend Swap. 54

3.13 Ground truth (left), MH bases parameterization approximation (middle), 100 times magnified comparison (right). Running forward and backward animations are learnt, running forward is approximated. 56

3.14 MH approach does not fail in complex motions. Ground truth (left), MH bases parameterization approximation (middle), and 100 times magnified difference. 57

4.1 Approximation of joint angle parameterization for the "dying" animation with a poor mesh. Ground truth (left), joint angle approximation (middle), and 10$\times$ magnified difference (right). 60

4.2 Approximation of joint angle parameterization for the "striking" animation with a poor mesh. Ground truth (left), joint angle approximation (middle) and 10$\times$ magnified difference (right). . . . 61

4.3 Approximation of unknown (not in the training set), the "walking" animation with joint angle parameterization. Ground truth (left), joint angle approximation (middle), and 10$\times$ magnified difference (right). Regularization parameter $\lambda = 0.001$. 62

4.4 Approximation of clustered vertex position parameterization for the "dying" animation with a poor mesh. Ground truth (left), clustered vertex position parameterization (middle), and 10$\times$ magnified difference (right). 62

4.5 Approximation of clustered vertex position parameterization for the "striking" animation with a poor mesh. Ground truth (left), clustered vertex position parameterization (middle), and 10$\times$ magnified difference (right). 63

4.6 Approximation of unknown, the "walking" animation with clustered vertex position parameterization. Ground truth (left), clustered vertex position approximation (middle), and 10$\times$ magnified difference (right). Regularization parameter $\lambda = 1000$. 64

4.7 Approximation of MH bases parameterization for the "dying" animation with a poor mesh. Ground truth (left), MH bases approximation (middle), and 10× magnified difference (right). 65
4.8 Approximation of MH bases parameterization for the "striking" animation with a poor mesh. Ground truth (left), MH bases approximation (middle), and 10× magnified difference (right). . . . 65
4.9 Approximation of unknown, the "walking" animation with MH bases parameterization. Ground truth (left), MH bases approximation (middle), and 10× magnified difference (right). Regularization parameter $\lambda = -1000$. 65

Dedicated to my mom and the raccoon stealing my mango.

ACKNOWLEDGMENTS

First of all, I would like to thank Professor Derek Nowrouzezahrai and Professor Pierre Poulin for giving me this opportunity and their vital guidance, and great support the last two years. I also would like to thank FRQNT for the Team grant (Modèles couplés pour l'animation et le rendu réalistes multi-échelle) supporting our research.

Next, I would like to thank my family, and my close friends especially for their constant support. You guys made everything easier for me. Also, I feel that I need to open a parenthesis for my lab computer, Cardon, since it never created any problems for me and always worked in full capacity. Last but not least, I would like to thank all the members of the LIGUM for creating a wonderful work environment, and Adrien for helping me to write the Résumé.

Finally, I would like to thank Marco Tarini, Daniele Panozzo, Olga Sorkine, Alec Jacobson and Daniele Panozzo and others, and Keenan Crane, Fernando de Goes, Mathieu Desbrun, Peter Schröder for sharing their source code and data. They helped me a lot for not reinventing the wheel again and again especially for the skinning and manifold harmonics bases parts of my thesis.

Long live rock and roll!

CHAPTER 1

INTRODUCTION

Realistic rendering is an important topic in computer graphics, and it is commonly used, especially in movies and special effects. However, due to the heavy computational cost it requires, realistic rendering techniques are not suitable for real-time applications such as video games and virtual reality. Real-time applications therefore rely on techniques that approximate realistic rendering effects by simplifying calculations or by "learning" from ground-truth values. However, even if approximation techniques can yield good results in general, they may also suffer from some challenging limitations in certain situations.

We investigate an approximation technique that focuses only on common animated objects consisting of a skeleton controlling a skinned character, such as characters in video games, moving clothes, deforming surfaces (e.g., muscles), etc. We assume that there is, or approximate, some correlations with a linear dependency between motion and appearance of objects. We construct a system of equations that "learns" motion and appearance pairs and extracts some approximation parameters. These parameters are then used to approximate realistic rendering of the skinned characters in real time.

To extract approximation parameters, we parameterize the mesh space of our skinned skeletal models based on their joint, spatial, orientation, and manifold harmonic representations. Our main goal is to correlate animation-dependent light transport by investigating these different parameterizations. We construct a linear system based on ground-truth appearance of the mesh (realistic rendering results), and the information linked to the parameterization (joint angles or positions, vertex positions, manifold harmonic bases, etc.). Then we extract the approximation parameters by solving the associated system of linear equations. Real-time approximation is then done by computing an inner product between the approximation parameters and the current animation pose.

We focus on ambient occlusion, a technique that increases realism of the rendered results by giving perceptual clues to depth. It simulates the shadowing caused by objects blocking ambient light and approximates the effects of environmental lighting. Even if ambient occlusion computation is faster than, for instance, the computation of a global illumination, it is still costly for real-time applications and animation sequences.

While our technique is based on parameterizing ambient occlusion, it may also be applicable to other visual effects. For example, our investigation may be expanded to render human skin, changes in color, deforming muscles, fur movement, clothing deformations, etc., by simply "learning" some information, extracting a representation, and then synthesizing new conditions.

This thesis is organized as follows. In Chapter 2, we cover preliminary and previous work related with our approach. We detail our parameterization techniques in Chapter 3. In Chapter 4, we provide and discuss our results. We then conclude our work by discussing avenues for future work, in Chapter 5.

CHAPTER 2

PRELIMINARY

In this chapter, we will introduce some basic concepts and discuss the work related with our research focusing on rendering visual effects applied to skinned characters which contain skeletal and physical animations.

We begin by outlining some basics on animation before introducing important topics related to our research, such as skinning, ambient occlusion, ambient occlusion approximation techniques, and harmonic bases.

2.1 Animation Basics

Animation mimics the motion of objects by displaying a sequence of images, each slightly different than the other, in a rapid succession. This rate changes from product to product: while films in theaters require 24 frames per second (fps), IMAX requires 48 fps, and high-definition videos (such as HD TVs or current high-quality video games) require 60 fps. Variation of frame rates is caused by technical decisions. For example, 24 fps was introduced to synchronize the sound with the film. Cathode ray tube (CRT) used in TVs introduced 30 fps as standard for broadcast production. Due to recent technological development, high frame rates (such as 48 fps and 60 fps) are introduced to increase the quality and realism (such as reducing motion blur and flickering found in films) and align with human vision.

Traditional animations use photographs or drawings on paper. The animator must draw every frame of the movement, then the frames are photographed one-by-one against a painted background onto motion picture film. "Pinocchio" (1940) is a good example of traditional animation (Figure 2.1).

Stop-motion is another animation technique that manipulates an object physically to mimic motions or changes. The object is displaced by small increments between individually photographed frames. Then, these frames are replayed as a fast sequence to

Figure 2.1: A scene from "Pinocchio" (1940, ©The Walt Disney Company). Image from Wikipedia "Pinocchio (1940 film)" article (`https://en.wikipedia.org/wiki/Pinocchio_(1940_film)`).

create the illusion of movement. Puppets, dolls with movable joints, or clay figures are used in stop-motion animations. "The Nightmare Before Christmas" (1993) is a good example of the use of such a technique (Figure 2.2).

Nowadays, computer animation is a widely adopted animation technique. The animated images are generated using a computer, and usually 3D computer graphics is used. On the other hand, 2D computer graphics is still used for stylistic, low bandwidth, and faster real-time renderings. "Toy Story" (1995) is an important example of computer animation (Figure 2.3).

3D animation is digitally modeled and manipulated by an animator using a computer-based system. Polygon meshes are used to represent objects. Often reduced to a mesh of triangles, these are defined by a set of vertex positions connected by edges to define faces. To control its animation, a mesh may be associated to a virtual skeleton structure. This process is called skeletal animation and it is used for character animation (Figure 2.4).

Character animation is a specialized animation focused on bringing animated char-

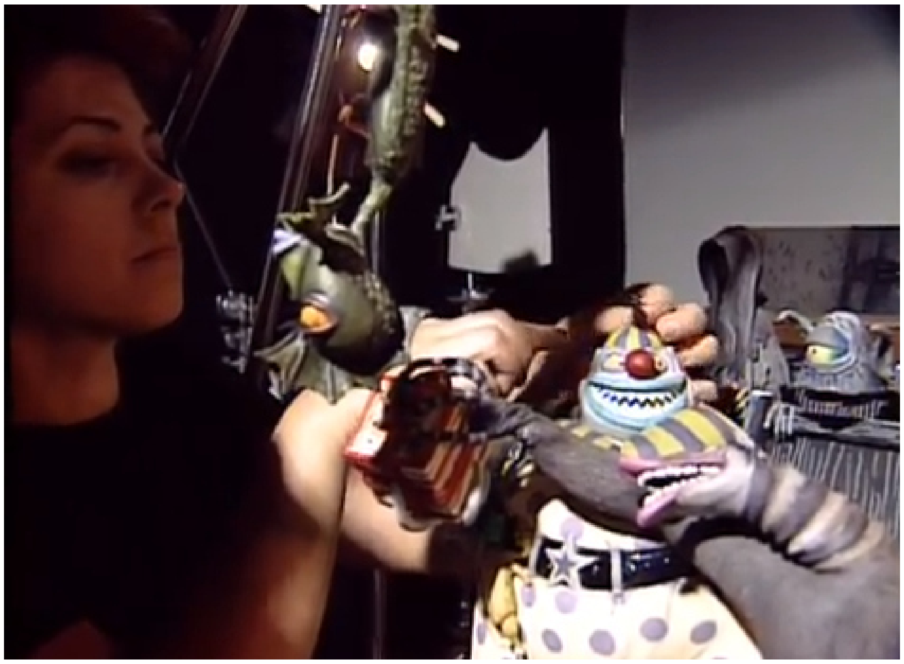


Figure 2.2: An animator poses a character for a single frame of an animation sequence. Image from Buzzfeed "20 Crazy Facts About The Making Of "The Nightmare Before Christmas" article by Arielle Calderon (`https://www.buzzfeed.com/ariellecalderon/the-making-of-the-nightmare-before-christmas`).

acters to life. Typical examples of character animations can be found in video games and animated movies, e.g., in "Sheriff Woody" and "Buzz Lightyear" from "Toy Story" (Figure 2.3). 3D characters thus include a skeleton structure that is posed at each frame of the animation, and the mesh components are manipulated using their associated skeleton "bones".

Even though skeletal animation is mainly used to animate humanoid characters (human, animals, etc.), it can animate any kind of deformable objects, such as clothes, faces, trees, etc. However, skeletal animation is usually not a physically-based technique; the realism of the resulting movements is dependent on the talent of the artist, not some underlying physical system.

Keyframing is widely used to generate computer animations. The scene is updated for each key frame of the animation and in-between frames are interpolated to mimic

Figure 2.3: A scene from "Toy Story" (1995, ©The Walt Disney Company). Image from "Toy Story" official website (`http://toystory.disney.com/`).

any intermediate motion. Keyframing can create any kind of motion, however, realism depends on the artist.

Motion capture is a technique to record (and replay) real-world motion. While keyframing can be used to generate any kind of motion, motion capture is used to record the motion of objects, humans, animals, etc. (Figure 2.5). The results of a motion capture session are physically-based and realistic since they come from a "live" performance. Even complex movements, such as secondary motions (such as jiggling of soft tissue), weight and exchange forces can be captured in a physically accurate way. Thus, motion capture can be used to increase the realism of skeletal animation. More details related to rigging and skeletal animation can be found in Section 2.2. Unfortunately, motion capture relies on expensive capture equipment, and results may still require an expensive and cumbersome manual post processing to, e.g., remove capture noise. Motion capture is used both in video games, and movies and keyframing and motion capture are regularly combined.

There are many ways to capture motions [17]: in early prosthetic, a set of armatures are attached over a performer's body, connected to each other using a series of rotational and linear encoders, which are connected to an interface that records all encoders si-

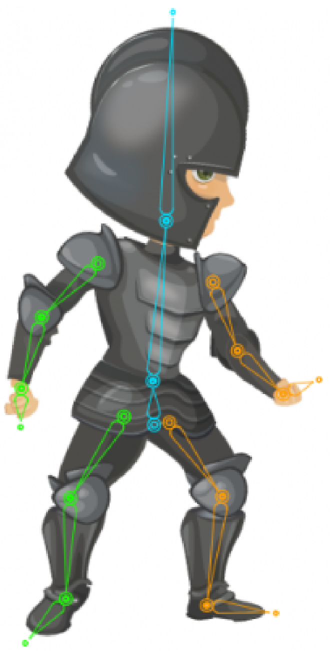


Figure 2.4: A skeleton is rigged to a 2D character being animated. Image from Marionette Studio "Skeletal Animation" web page (`http://marionettestudio.com/skeletal-animation/`).

multaneously in order to prevent distorted or biased data. Finally, performer's motion is analyzed using a set of trigonometric functions. This design has restrictions that are difficult to overcome.

Acoustic motion capture uses a set of audio transmitters strapped to various parts of a performer's body. The transmitters are sequentially triggered to output a clicking sound and its associated receiver measures the time it takes for sound to travel from its transmitter. Even if this method creates accurate results, the sequential nature of the position data created can be an issue since a snap shot of the performer's skeletal position would be preferred instead of a time skewed data (biased data) stream.

Magnetic motion capture uses a centrally located transmitter to capture data. A set of receivers are strapped on a performer's body and they are used to measure their spatial relationship to the transmitter. The resulting data stream contains 3D positions and orientations from each receiver. This data is typically applied to an inverse kinematics system to drive an animated skeleton. The magnetic approach has the same problem as

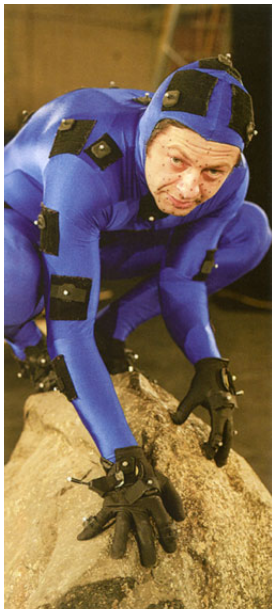


Figure 2.5: Andy Serkis portrays "Gollum" on the performance capture stage. Image from Andy Serkis' official website (`http://serkis.com`).

the audio method. Also, a lack of receivers, and a limited capture range may cause some problems.

Modern motion capture uses optical systems. Here, directionally-reflective balls serve as markers attached to a performer's body. Since this method does not require any cabling, it offers a performer the most freedom of movement, and this is one of its advantages. Video cameras equipped with light sources are connected to a synchronized frame buffer, and acquired data sets are derived by 3D reconstruction from multiple captured images. This method suffers from occlusion of the markers; however, this issue can be reduced by using more cameras. Also, the resolution of the cameras and the quality of the tracking software play an important role in the quality of the data acquired.

We focus our investigation only on skeletal animated characters. We developed a system fed by motion data and corresponding appearance to find connections between motion and appearance. Even though we only use frames generated by keyframing, there is nothing to prevent the use of motion captured data.

2.2 Skinning

Skinning and skeletal animation are the methods used in video games and animation movies to bring 3D characters to life. A renderable skin (mesh) is attached to the character's skeleton which defines how this skin moves and deforms according to the skeletal poses. A skeleton is a rooted tree structure of bones and joints. It is a posable framework used to manipulate skin and other geometric data. A skeleton is posed with transformation matrices at each frame of the animation, the skin is manipulated according to skeleton's new pose, and the character is rendered (Algorithm 1).

Algorithm 1 Basic skinning algorithm

```
1: for each animation frame do
2:     Read the frame information (pose)
3:     Construct the transformation matrices hierarchically depending on the frame information
4:     for each vertex do
5:         Transform the vertex to its deformed location using the transformation matrices
6:     end for
7:     Render the deformed character
8: end for
```

Joints are nodes of the skeleton that allow movements of the skeleton. Each transformation that poses a bone of the skeleton is applied at a joint. A transformation is assumed to be rigid and generally contains a translation plus a rotation. However, some cases (such as crowd simulation) use nonrigid transformations (scale and shear). At each frame of an animation, related transformation matrices are applied to joints and the skeleton is posed in this way. The terms bone and joint are used interchangeably in the literature. We will refer to bones as the edges between joints (Figure 2.6).

Skin is represented as a polygon mesh. Most of the time a triangle mesh is chosen over quad mesh for the representation. The mesh connectivity is assumed constant and only vertex positions change during the skinning process (deformation). The transformations are applied to the vertices to have them at new positions for the current pose (deformed positions). The influence applied to each vertex from a small set of transfor-

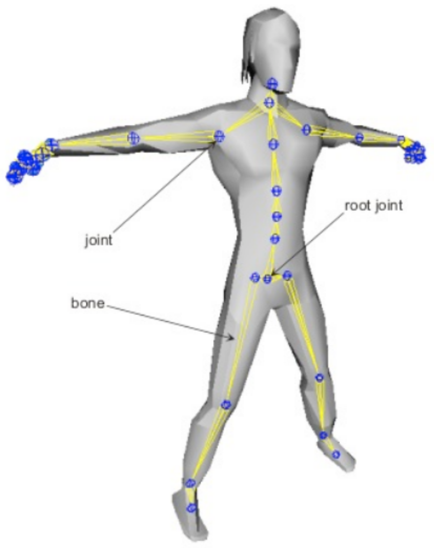


Figure 2.6: Skeleton structure containing joints (blue spheres), bones (yellow tetrahedra) and skin (grey polygons). The root matrix positions the global mesh, and multiplies hierarchically all other matrices of the tree structure. Image from Leonid Sigal, "Human Motion Modeling and Analysis" course slides.

mations is defined by skinning weights (also called vertex weights or bone weights).

Skinning weights bind the vertices to the skeleton. They describe the influence of joints on each vertex. There is usually a fixed upper bound on the number (and weights) of joints influencing a vertex, due to graphics hardware considerations. The weights of a vertex are assumed to be convex (sum of all weights a vertex has is equal to 1); however, there are some other cases that do not assume weights to be convex to let the artists feel free. Even if modelling tools offer algorithms to assign skinning weights automatically, this is not a trivial problem. Most of the time the artist assigns values to the weights by painting them on mesh vertices (Figure 2.7). In this way, the artist can visualize the attachments to discover and fix problems.

Skinning does not necessarily have to be applied to human-like characters as it can be applied to any kind of character containing a skeleton, going from clothes, faces to any deformable geometrical model or parts of. However, Kavan et al. [3] detail the important considerations behind the skinning applied to a character without any skeleton in their SIGGRAPH course.

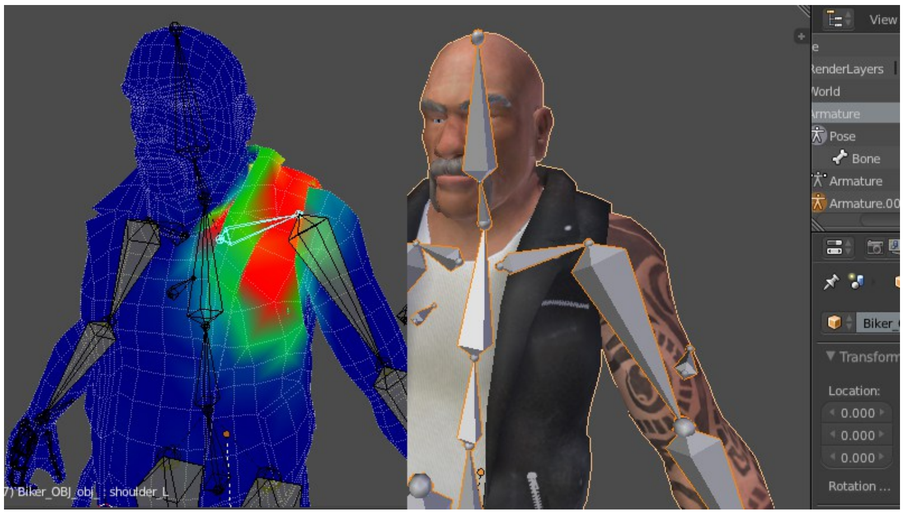


Figure 2.7: Painting weights on the shoulder. Color range goes from blue (closer to 0), green, to red (closer to 1). Image from Digital Tutors, "Weight Painting in Blender" page (`https://www.pluralsight.com/blog/tutorials/weight-painting-in-blender`).

The original idea of skinning, proposed by Magnenat-Thalmann et al. [10], evolved into four different approaches: physically based, volume preserving, example-based, and geometric.

Physically based skinning methods, based on internal structure of the body, create highly realistic deformations. Secondary motion effects (such as jiggling of soft tissue) can be reproduced; however, physically based methods require anatomy knowledge and have higher computational costs. Alternatively, methods capturing real world objects (such as motion capture) can help to decrease computational costs while retaining the same realism; however, they can require expensive hardware set up, and all objects may not be captured easily.

Volume preserving skinning techniques allow the artist to control effects, such as folding or building muscle interactively. A volume correction is applied after each deformation. In this way, unnatural volume changes in deforming joint regions, such as collapsing joints and candy-wrapper effects (explained later) can be prevented [19]. However, volume preserving skinning methods are also expensive when compared to geometric skinning methods. Also, some of volume preserving techniques cannot be used in current standard animation pipelines. One of the important volume preserving methods is lattice-based free form deformation (FFD) [1]. Since FFD provides smooth deformations and preserves skin volume, it is widely used in commercial 3D modelling software [13]. However, FFD also requires additional setup and the deformation is sometimes difficult to predict.

Example-based techniques start from example meshes and use approximations or interpolations to get artifact free realistic results. They consist of direct interpolations between example meshes [9], approximations using principal components of example deformations [7], or fitting linear blending parameters to match provided examples [12]. Realism of the results is limited by the number of input examples, and producing these input examples can be costly.

Geometric skinning techniques are the most common approach for skin deformation. These approaches use a geometric way to bind the skin to the skeleton. Skinning weights and transformation matrices have an important role in these techniques. We mainly focus on geometric skinning techniques since our investigation works with linear blend skinning and dual quaternion skinning [4].

The most popular geometric skinning method is linear blend skinning (LBS). Since it is easy to parallelize on GPU, LBS has become the de-facto standard in video game development. LBS starts from a rest pose shape along with joint transformations and skinning weights. The rest pose shape is actually the skin (mesh) we mentioned earlier in special "T-pose" (Figure 2.8).

LBS computes deformed vertex positions by a weighted sum, which is performed for each vertex at every frame (Figure 2.9). Skinning weights are blended with transformation matrices to transform rest pose vertex positions into their deformed positions (Equation 2.1).

$$\overline{v_i} = \sum_{j=1}^{m} w_{i,j}\, \mathbf{M}_j\, v_i \tag{2.1}$$

where $\overline{v_i}$ is the transformed (deformed) vertex position, v_i is the rest pose position of vertex, $w_{i,j}$ is the weight of the j^{th} joint on the i^{th} vertex, and $\mathbf{M}_j$ is the transformation

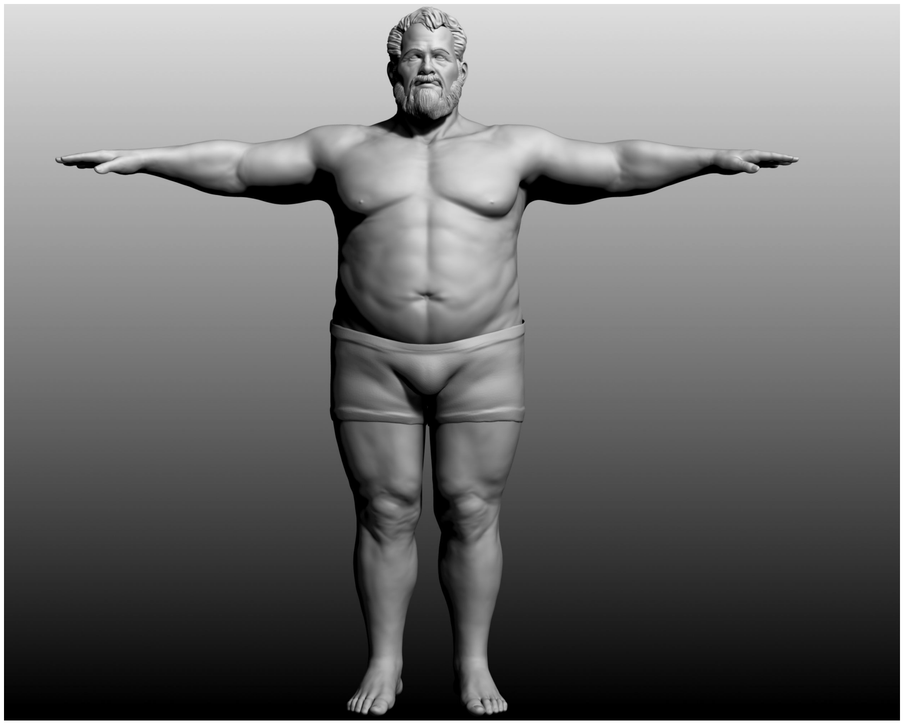


Figure 2.8: A mesh in "T-pose". Image from cgtrader, "Fatman T-Pose 3D model".

matrix of the j^{th} joint. m is the upper bound for skinning weights. It is usually in the [1, 4] range.

The rest pose vertex positions are in $\mathbb{R}^3$ or $\mathbb{R}^4$ (homogeneous space). Bone transformations include transformation matrices that define the animation frames in $\mathbb{R}^4$ (homogeneous space). These matrices are used to pose the skeleton of the character. Skinning weights describe the influence of a bone on a specified vertex. Skinning weights are in $\mathbb{R}$, and usual range is [0, 1].

Equation 2.1 is also used to transform the vertex normal and tangent vectors. Normal and tangent vectors are used for shading operations. For transforming a normal, ideally the inverse transpose of transformation matrix $\mathbf{M}$ should be used; the rest pose's vertex normal should be transformed using the upper 3×3 block of the inverse transposed matrix. However, if $\mathbf{M}$ does not include any nonrigid transformation (such as nonuniform scale and shear), the rest pose normal can be transformed without taking the inverse transpose of $\mathbf{M}$. Still, the resulting normal should be normalized after the

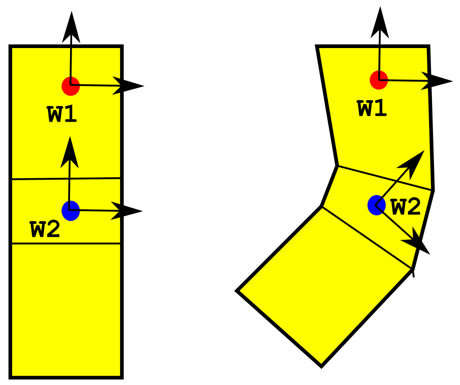


Figure 2.9: Skinning on a 2D cylinder. An unbent knee with skin attached to joints 1 and 2 (left) is skinned using a weighted sum of the skinning transformations (right). $W1$ and $W2$ are world matrices of the joints.

transformation.

Transforming tangent vectors does not require taking the inverse transpose of $\mathbf{M}$; however, normalization is still required after the transformation.

$\mathbf{M}$ is defined as $\mathbf{M} = \mathbf{B}^{-1}\ \mathbf{W}$ where $\mathbf{B}^{-1}$ is the inverse binding matrix that transforms vertex positions from skin local space to joint local space. $\mathbf{W}$ is the world matrix of the joint (Figure 2.10). So, matrix $\mathbf{M}$ provides a transformation for a vertex from skin local space to joint local space, and finally to world space. The matrix $\mathbf{M}$ is updated for each joint at each frame of the animation.

The world matrix for a joint j is defined as $\mathbf{W}_j = \mathbf{L}_j\ \mathbf{W}_{j_{parent}}$, where $\mathbf{L}_j$ is the local matrix of joint j, and $\mathbf{W}_{j_{parent}}$ is its parent's world matrix. For the root joint, the world matrix is equal to its own local matrix. The local matrix of joint j is defined as $\mathbf{L}_j = \mathbf{T}_j\ \mathbf{R}_j\ \mathbf{S}_j$, where $\mathbf{T}_j$ is the translation matrix, $\mathbf{R}_j$ is the rotation matrix, and $\mathbf{S}_j$ is the scale matrix of the joint. These matrices (translation, rotation and scale) are defined for each animation frame in the animation file; $\mathbf{W}_j$ must be recalculated for each joint, at each frame.

The inverse binding matrix $\mathbf{B}_j^{-1}$ of joint j is defined as the inverse of the rest pose world matrix $\mathbf{B}_j^{-1} = \mathbf{W}_{0j}^{-1}$, where $\mathbf{W}_{0j}$ is the rest pose world matrix of joint j. Since the inverse binding matrix will not change between frames, it can be precomputed for each joint of the skeleton.

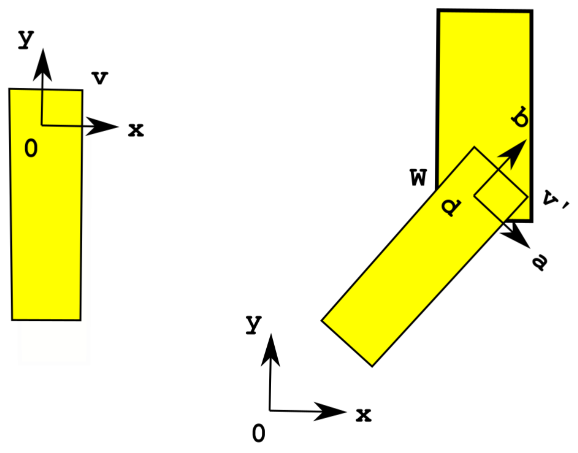


Figure 2.10: Vertex transformation from local space to world space using matrix W. Vertex v in joint's local space (left). Vertex v in world space (right).

You can see LBS algorithm (Algorithm 2) for details of the steps.

The process of blending rigid transformations in LBS can cause some problems. Blending works well if the blending transformation matrices are similar to each other. On the other hand, if the rotation components of the transformation matrices differ, the blended matrix may not necessarily remain a rigid transformation, and the rotation component of the blended matrix will no longer be a proper rotation. This situation causes artifacts known as the candy-wrapper effect and the collapsing elbow.

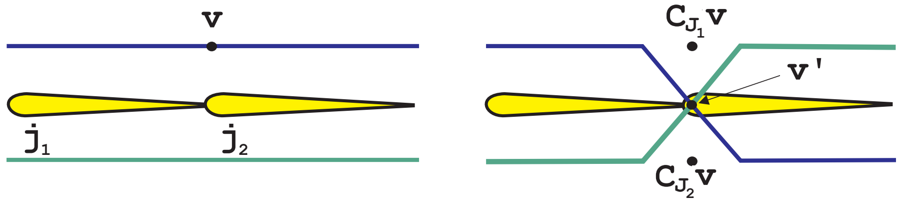


Figure 2.11: Candy-wrapper artifact of LBS. Reference pose (left), and animated pose (right).

The candy-wrapper effect happens if one of the bones is tilted by a large angle, e.g., 180 degrees about its main axis. In this case, vertices with nearly equal weights are transformed to closely located points near this axis. Consider the simple arm rig in

Algorithm 2 Linear blend skinning (LBS) algorithm

```
1: for each joint do
2:     Calculate inverse binding matrices ($\mathbf{B}^{-1}$) as $\mathbf{B}_j^{-1} = \mathbf{W}_{0j}^{-1}$ -Precomputation-
3: end for
4: for each animation frame do
5:     for each joint do
6:         Calculate local matrices ($\mathbf{L}$) as $\mathbf{L}_j = \mathbf{T}_j\ \mathbf{R}_j\ \mathbf{S}_j$
7:         Calculate global matrices ($\mathbf{W}$) as $\mathbf{W}_j = \mathbf{L}_j\ \mathbf{W}_{j_{parent}}$
8:         Calculate transformation matrices ($\mathbf{M}$) by $\mathbf{M} = \mathbf{B}^{-1}\ \mathbf{W}$
9:     end for
10:    for each vertex do
11:        Deform the vertex using Equation 2.1
12:    end for
13: end for
```

Figure 2.11: Vertex v in the figure is influenced equally by the joints j_1 and j_2, and let us assume the arm is animated by twisting the joint j_2 by 180 degrees around the $x-axis$. We can write joint transformation as

$$\mathbf{M}_{j_1} = \begin{pmatrix} I & 0 \\ 0 & 1 \end{pmatrix},\ \mathbf{M}_{j_2} = \begin{pmatrix} R_x(180^\circ) & 0 \\ 0 & 1 \end{pmatrix}$$

where $\mathbf{I}$ is the 3×3 identity matrix and R_x is rotation about the $x-axis$. Averaging $\mathbf{M}_{j_1}\ v$ and $\mathbf{M}_{j_2}\ v$ locates vertex $\bar{v}$ exactly at the position of joint j_2, and the skin collapses to a single point. Another artifact caused by the skin collapse, collapsing elbow, occurs when the angle between the axes of two adjacent bones becomes small. Vertex points on the mesh lying between the two bones move towards the center and thus collapse with each other. Examples of these effects on a 3D mesh are shown in Figure 2.12.

For solving these problems, linear blending of the matrices should be improved with an approach that is coordinate-invariant, always returns a valid rigid transformation, and interpolates two rigid transformations along the shortest path.

Direct quaternion blending (DQB) applies a linear combination of quaternions and generates sufficiently accurate results; however, it fails at complex rotations, such as around the armpit, or results in stretching effects in cloth animation because of switching centers of rotation, leading to gaps (Figure 2.13).

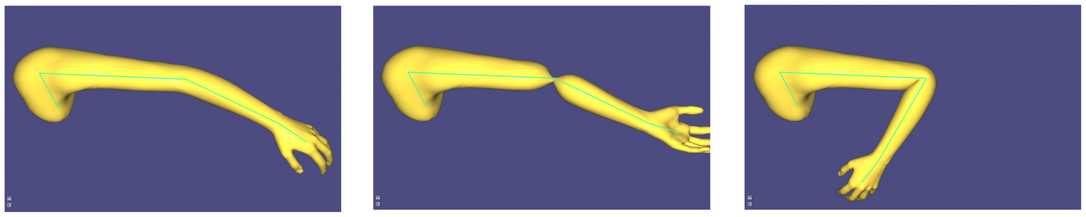


Figure 2.12: Rest pose (left), candy-wrapper effect (middle), and collapsing elbow (right). Image from "libigl Tutorial" on GitHub.

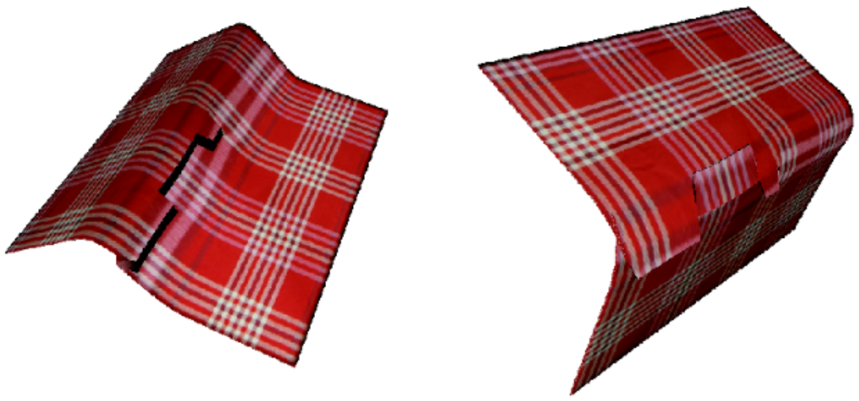


Figure 2.13: DQB fails when the rotation centers of the vertices differ considerably. In this case, deformed vertices disintegrate and cracks appear. Image from Kavan et al. [4].

Spherical blend skinning (SBS) blends rotations and translations independently. Even though SBS generates good results, it is not suitable in real-time applications and real-time version of SBS suffers from the same problems as DQB.

Log-matrix blending (LMB) linearly combines matrix logarithms instead of matrices themselves. In this way, centers of rotation can be handled properly; however, LMB sometimes fails to select the shortest path in interpolation and may generate unnatural skin deformations (Figure 2.14).

Dual quaternion skinning (DQS) solves the problems of LBS, DQB, and LMB. It introduces dual quaternion blending (DLB) that properly blends centers of rotation (Equation 2.2)[4].

$$DLB\left(w;\hat{q}_1 \ldots \hat{q}_n\right) = \frac{w_1\hat{q}_1 + \ldots + w_n\hat{q}_n}{\|w_1\hat{q}_1 + \ldots + w_n\hat{q}_n\|} \tag{2.2}$$

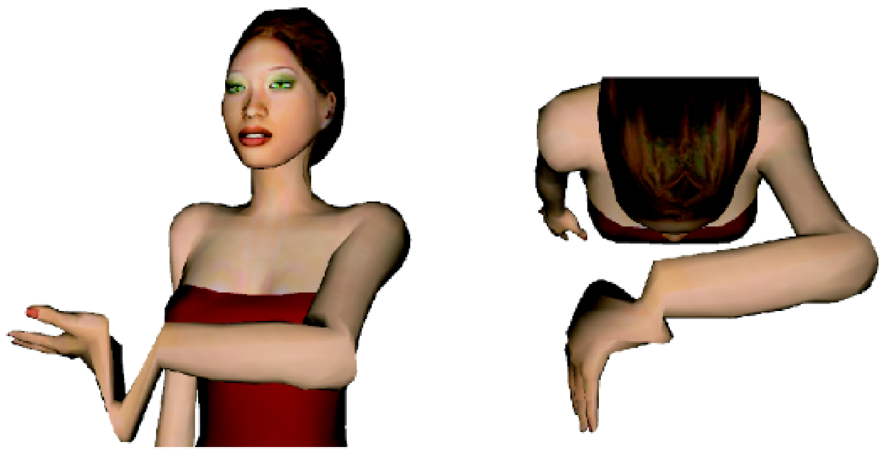


Figure 2.14: LMB fails to identify the shortest path. Image from Kavan et al. [4].

where w is the skinning weight vector and $\hat{q}_1, ..., \hat{q}_n$ are unit dual quaternions. Unit dual quaternions satisfy $||\hat{q}|| = 1$ and they are always invertible. Unit dual quaternions represent a rigid transformation and as ordinary quaternions, they are also associative, distributive, but not commutative.

DLB does not provide a theoretically perfect solution as ScLERP (spherical linear interpolation (SLERP) for dual quaternions); however, results differ little when compared to ScLERP, and DLB thus generates plausible results. Additionally, ScLERP is not suitable for real-time calculation.

DQS requires conversion of all transformation matrices into dual quaternions if the animation data is not already expressed in dual quaternions. This conversion can be done in the CPU, and then dual quaternions are sent to GPU. However, this conversion can create a problem when the appropriate sign for the resulting dual quaternions is chosen. Indeed dual quaternions $\hat{q}_i$ and $-\hat{q}_i$ represent the same rigid transformation, but their interpolation can be different in the interpolated trajectory (shorter and longer path). The sign of the dual quaternion must be handled in the vertex shader since it depends on the joints influencing a vertex. You can see DQS algorithm in Algorithm 3.

DQS cannot handle nonrigid transformations as easily as LBS. Such transformations can be useful in cases, such as crowd simulation to vary a character without modifying the 3D geometric model itself. However, DQS requires higher dimensional geometric

Algorithm 3 Dual quaternion skinning (DQS) algorithm

```
1: if working with transformation matrices then
2:     Convert them into dual quaternions
3: end if
4: Find shortest path for the quaternions
5: Blend them and normalize
6: Perform skinning
```

algebra calculations for handling nonrigid transformations and these calculations have great cost. Thus, if nonrigid transformations are required, a two-step skinning approach is used. Nonrigid parts of the transformation are handled in the first step, and then rigid parts are used to give the mesh its final shape. This also adds extra costs and requires another update in the vertex shader code.

Even though DQS is better at removing some artifacts, it is expensive when compared to LBS. Additionally, DQS also suffers from rotations larger than 180 degrees (Figure 2.15). Because of its inefficiency compared to LBS, DQS could never gain enough interest in video game development, and game developers keep using LBS by limiting the joint's degrees of freedom or adding more joints to the parts that fail.

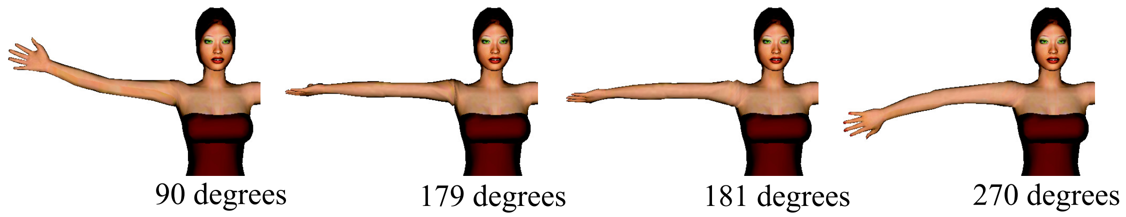


Figure 2.15: Skin flipping artifact caused by the shortest path when rotating from 179 to 181 degrees. Image from Kavan et al. [4].

Even though we only use LBS to generate our results, our approach can work with LBS and DQS since it depends on animation parameters (such as joint angles, vertex positions) instead of skinning process.

2.3 Ambient Occlusion

Ambient occlusion (AO), which is often associated with diffuse indirect illumination, simulates the shadowing caused by surrounding objects blocking the ambient light. This light is assumed omnipresent and omnidirectional, and constant in a 3D scene. It approximates effects due to environmental lighting that are otherwise captured only by global illumination (GI) techniques. Ambient occlusion increases realism and improves the perception of surface immersion in 3D scenes with significantly less computational cost (Figure 2.16).

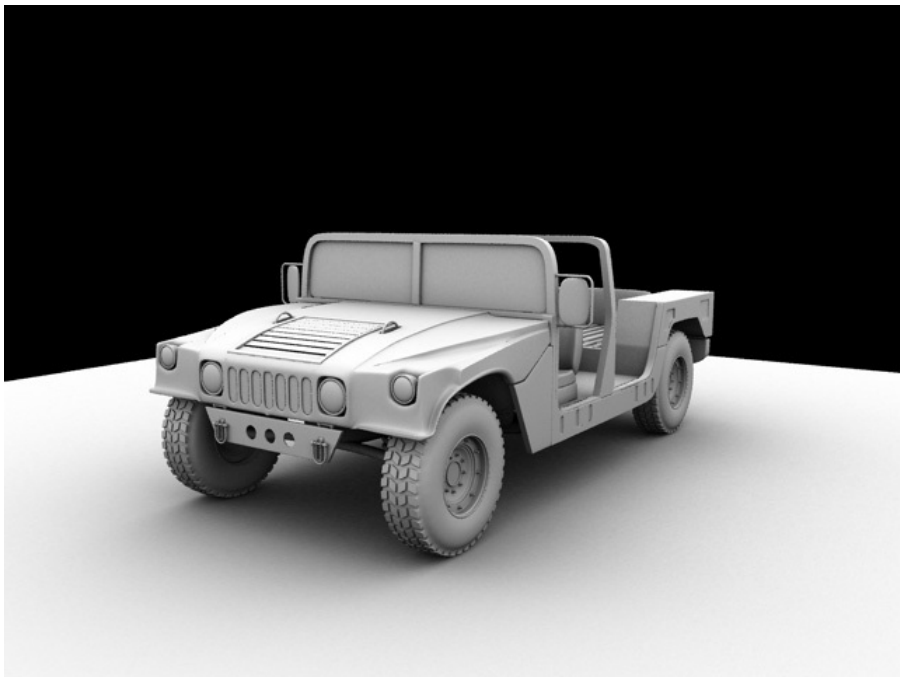


Figure 2.16: Ray-traced ambient occlusion. Image based on Jorge Pimentel's rendering.

While GI techniques compute the illumination in a physically accurate way, ambient occlusion considers only the geometric properties of every point in the scene. Yet, it computes an impression of diffuse indirect illumination in a visually realistic way faster than any other GI technique, such as photon mapping or path tracing. Additionally, since AO considers only the geometry of the scene, changes in direct lighting or average ambient reflected intensity have no effect on ambient occlusion computations.

Some of the ambient occlusion techniques only focus on solving a particular case, such as self-occlusion of an object on itself. Our investigation also only focuses on self-occlusion of deformable objects (skinned characters) over time (animation).

Ambient occlusion is defined as the percentage of occlusion at a point on a surface over the hemisphere above this point (oriented toward the surface normal at this point) (Equation 2.3).

$$AO(x,n) = \frac{1}{\pi} \int_{\Omega} V(x,\omega)\ \max(n \cdot \omega, 0)\ d\omega \tag{2.3}$$

where x is the point at which we compute ambient occlusion, n is the normal at this point, $V(x,\omega)$ is the visibility at x in direction ω (1 if there is no intersection, 0 otherwise).

There are many techniques available for computing ambient occlusion. A detailed investigation can be found in a survey by Méndez-Feliu and Sbert [11]. We use per-vertex ray-traced ambient occlusion computation for ground-truth results in our investigation with sampling with a Monte Carlo method.

A Monte Carlo method estimates numerically the problems too complicated to solve analytically. It allows us to solve multidimensional integrals using probabilistic bases. If we assume that X is a random variable, then the cumulative distribution function (CDF) of X is defined as

$$\text{cdf}(x) = Pr[X \leq x]$$

and the corresponding probability density function (pdf) is

$$\text{pdf}(x) = \frac{d}{dx}\, cdf(x)$$

thus, the probability that x takes a value between a and b:

$$Pr[a \leq X \leq b] = \int_a^b \text{pdf(x)}\ dx$$

The mathematical expectation of a random variable $Y = f(X)$ on a domain Z, and its variance are

$$E[Y] = \int_Z f(Z)\ \mathrm{pdf}(Z)\ dz$$
$$\sigma^2[Y] = E[\ (Y - E[Y]\ )^2] = E[Y^2] - E[Y]^2$$

A Monte Carlo estimator for an integral with arbitrary dimensions is shown mathematically as $E[\overline{F}] = F$, and defined as

$$\overline{F} = \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{\mathrm{pdf}(X_i)} \approx \int_x f(x)\ dx \tag{2.4}$$

where N is the number of samples, $f(x)$ is the function of the integral that will be estimated, such as the ambient occlusion function of Equation 2.3 (Equation 2.5). pdf(x) is used for distributing the sampling points.

$$f(x) = \frac{V(x,\omega)\ \max(n \cdot \omega, 0)}{\pi} \tag{2.5}$$

A major drawback of Monte Carlo integration is the size of the variance of the estimation. If this variance is not reduced, then the results become noisy. Stratification and importance sampling are methods used to decrease the variance. Stratification divides the integration domain into sub-domains and performs Monte Carlo integration on each sub-domain. Importance sampling chooses pdf(x) in order to place the points X_i at locations closer to $f(x)$. Here, we use an importance sampling method, cosine weighted hemisphere, to reduce the variance (Equation 2.6) since cosine weighted hemisphere sampling generates better results (less noise) with fewer samples than other sampling distributions, such as uniform sampling. Algorithm 4 shows how to draw samples (directions) using cosine weighted hemisphere sampling.

$$\mathrm{pdf}(x) = \frac{\max(n \cdot \omega, 0)}{\pi} \tag{2.6}$$

After we plug in $f(x)$ and pdf(x) into a Monte Carlo estimator and make the simplifications, our final ambient occlusion equation reduces to a simple visibility summation

$$AO(x) = \frac{1}{N} \sum_{i=1}^{N} V(x, \omega_i) \tag{2.7}$$

Algorithm 4 Generating random directions using cosine weighted hemisphere sampling

```
1:  for each sample do
2:      $r_1 \leftarrow randomValue()$
3:      $r_2 \leftarrow randomValue()$
4:      $\theta \leftarrow \arccos(\sqrt{r_1})$
5:      $\phi \leftarrow 2\,\pi\,r_2$
6:      $direction.x \leftarrow \sin\theta\ \cos\phi$
7:      $direction.y \leftarrow \sin\theta\ \sin\phi$
8:      $direction.z \leftarrow \cos\theta$
9:      $normalize(direction)$
10: end for
```

Even though ambient occlusion is much faster than global illumination techniques, high-quality AO is not fast enough for real-time rendering. In recent years many investigations focused on approximation techniques to improve or accelerate ambient occlusion calculations. In this and following section, we describe some of these techniques, but the survey by Méndez-Feliu and Sbert [11] can be referred to for more details.

One of the most popular AO approximation techniques is screen-space ambient occlusion (SSAO). SSAO is currently the de-facto standard in video games because of its efficiency. The depth buffer is queried at points derived from samples in a sphere instead of casting rays through the entire scene, and the fraction of occlusion is estimated only within these sampled points. The occlusion factor depends only on the depth difference between samples. Since SSAO is performed in screen space, it becomes independent from scene complexity, while generating pleasant and plausible local results. Additionally, it can be executed completely on the GPU without any CPU usage.

On the other hand, SSAO is not perfect. Because, it lacks spatial location of the entire world geometry, the quality and realism of the results may suffer from missed details absent from the depth buffer. Even if the basic SSAO has improved over time, it still has accuracy problems hard to correct without interfering with depth discontinuities.

Our technique is not a screen-space based technique. It uses the full 3D mesh geometry as it focuses on realism and investigates a different way for realistic ambient occlusion approximation to be delivered in real time. The methods in the following section are more closely related to our technique and investigation.

2.4 Ambient Occlusion Approximations

Even though ambient occlusion is faster to compute than full global illumination solutions, it is still not sufficient for complex animated objects in the sense of both memory and real-time computation complexity. For animated characters, ambient occlusion results should be updated dynamically for every frame of the animation. As long as we do not use an approximation method, the only way to do this in real time is by precomputing all the ambient occlusion information for all the animation frames. When the number of frames and/or animated objects in the scene increases, the memory requirements for precomputed ambient occlusion also increase. This explodes the memory eventually. There is always a realism-performance tradeoff in realistic rendering, and approximation methods are used to minimize this tradeoff to achieve compromised realistic results at real-time frame rates.

Our approach is similar to the ones of Kontkanen and Aila [6] and Kirk and Arikan [5]. We construct our linear system based on the one of Kontkanen and Aila [6]. Details of our technique and experiments will be given in Chapter 3.

Kontkanen and Aila [6] extract some static per-vertex coefficients by solving a linear system. These coefficients depend on animation parameters of reference poses and corresponding per-vertex ambient occlusion values. Their approach parameterizes ambient occlusion as a linear combination of the animation parameters. Their system is fed with the reference poses containing joint angle values along with the precomputed per-vertex ambient occlusion values. They establish a linear mapping from an arbitrary pose to approximated per-vertex ambient occlusion values. This mapping corresponds to the dot product between the arbitrary pose vector (vector of joint angles) and the coefficients extracted by solving the system. Details will be covered in Chapter 3.

Kontkanen and Aila [6] assume that ambient occlusion depends linearly on the animation parameters (joint angles). Their second assumption is that ambient occlusion is a sum of occlusions resulting from individual parameters. These assumptions cause a loss of some higher order effects and make the approximation fail for some complicated motions dependent on many animation parameters.

Comparison between Kontkanen and Aila [6] and ray-traced ground truth can be seen in Figure 2.17.

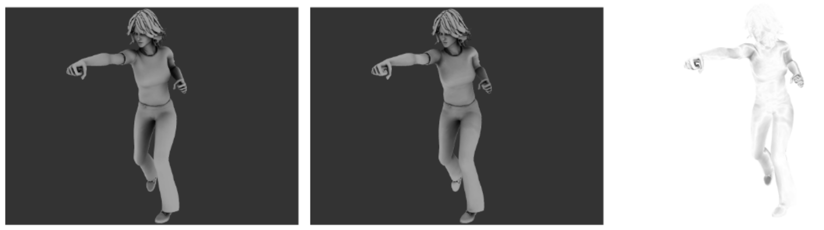


Figure 2.17: Ray-traced ground truth (left), Kontkanen and Aila [6] (middle), comparison of the images (right). Image from Kontkanen and Aila [6].

Kirk and Arikan [5] also construct their approach based on the one of Kontkanen and Aila [6]; however, they use a different parameterization and compress their data to increase memory efficiency.

Kirk and Arikan [5] work with a localized multilinear model while Kontkanen and Aila [6] work with the entire space of character poses. Additionally, Kirk and Arikan [5] use joint positions supported with additional handle position information, which defines a coordinate frame at each bone, as animation parameters. In this way, their method can capture some effects which cannot be captured with using only joint angles, and this increases their accuracy. A comparison between Kirk and Arikan [5] and ray-traced ground truth can be seen in Figure 2.18.

Kirk and Arikan [5] cluster the reference poses according to their motion and construct a linear system for each cluster based on the per-vertex ray-traced ambient occlusion values, as well as joint and handle positions. These systems are solved to get some coefficients for every vertex in the pose cluster. Then, vertices are clustered and another linear system is constructed to get coefficients for vertex clusters.

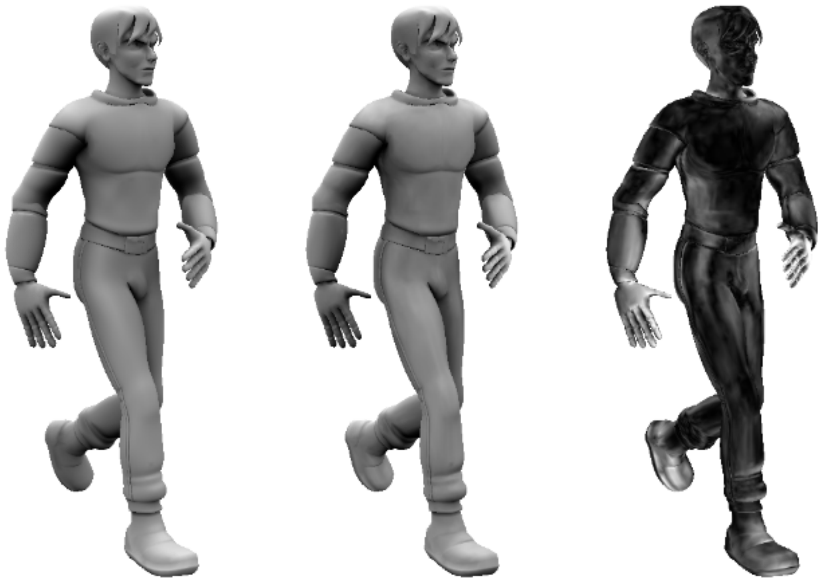


Figure 2.18: Comparison between Kirk and Arikan [5] (middle) and ray-traced ground truth (left). Image from Kirk and Arikan [5].

The approach of Kirk and Arikan [5] cannot approximate the per-vertex ambient occlusion results directly using a single dot product as do Kontkanen and Aila [6]. Kirk and Arikan [5] use additional functions for transition, which is caused by clustering, between several subspaces and a blending parameter. The blending parameter is user-defined, mesh-dependent, and must be found by experiment.

As any kind of machine learning technique, both approaches depend on large reference pose collections that cover many different motions. Learning many different motion examples also increases the generalization of these approaches to approximate arbitrary animations.

We build our approach based on Kontkanen and Aila [6] and perform different parameterizations from joint angles to joint positions (without handles), vertex positions and manifold harmonics. Details of our technique and experiments will be given in Chapter 3.

2.5 Manifold Harmonics

Spectral methods solve problems by examining or manipulating the eigenvalues, eigenvectors, eigenspace projections, or a combination of these quantities. These methods can be applied to solve geometry processing problems such as mesh compression, correspondence, parameterization, segmentation, smoothing, symmetry detection, surface reconstruction, remeshing, etc. The main threads for the development of spectral methods are spectral graph theory, signal processing related to Fourier analysis, and works on kernel principal component analysis and spectral clustering.

The focus in spectral graph theory derives relationships between the eigenvalues of the Laplacian or adjacency matrices of a graph and its various fundamental properties, such as diameter and connectivity. Spectral graph theory provides a basis for the Laplace operators used in the computer graphics and geometry processing community, i.e., Laplace-Beltrami operator.

The Fourier transform, a classical tool for signal processing, decomposes a function of time (a signal) into a sum of sinusoidal bases (Fourier function bases). Thus, the Fourier transform can be used to implement low-pass or more general convolution filters. This idea can also be extended to arbitrary manifolds by considering the Laplace operator. The Laplacian provides insights in the structure and morphology of the shape. This idea was applied to geometry processing by Taubin [16] for the first time using the Tutte Laplacian. In time, different Laplacians were used for geometry processing. The paper by Zhang [20] can be referred to for the details of Laplacians used in geometry processing.

Most of the spectral methods have a basic framework in common, which can be considered as three main steps (Figure 2.19). Firstly, a matrix representing a discrete linear operator based on the structure of the input mesh is constructed. Each entry of this matrix represents the relation between the vertices (faces or other primitives) of the mesh. Secondly, eigenvalues and eigenvectors of the mesh are computed (eigen decomposition). Finally, resulting structures from the decomposition are used in a problem-specific manner to obtain a solution.

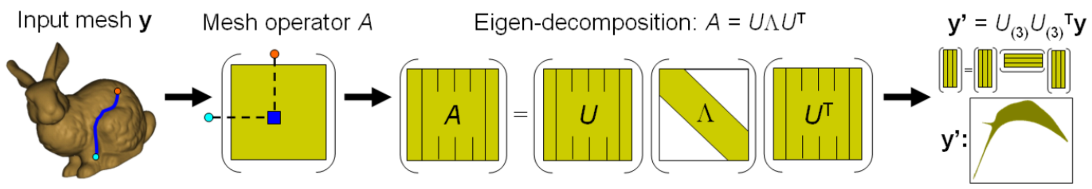

Input mesh **y**
Mesh operator A
Eigen-decomposition: $A = U\Lambda U^T$
$\mathbf{y}' = U_{(3)}U_{(3)}^T\mathbf{y}$
A
$=$
U
Λ
U^T
$=$
$\mathbf{y}'$:

Figure 2.19: Mesh segmentation application. 3D mesh data is transformed to 2D contour data by preserving the geometric features. Figure based on Zhang et al. [21])

Combinatorial and geometric Laplacians are commonly used for spectral mesh processing because their eigenvectors have similar properties to the classical Fourier basis functions. Fourier transform of such a signal can be derived by an eigenspace projection of the signal along the eigenvectors of a mesh Laplacian by representing mesh geometry using a discrete signal defined over the manifold mesh surface. The eigenvectors of the mesh Laplacians present "harmonic behavior", and they are considered as the vibration modes or the harmonics of the mesh surface with their corresponding eigenvalues as the associated frequencies.

Combinatorial Laplacians are constructed based on the connectivity of the graph that is the 1-skeleton of the mesh and they do not explicitly encode geometric information. Geometric Laplacians discretize the Laplace-Beltrami operator from Riemannian geometry, and explicitly encode geometric information of the mesh. Geometric Laplacians require a manifold triangle mesh.

Even though they have a distinct heritage both Laplacians have a single mathematical definition, and several fundamental properties. Laplacian operators are linear operators acting on functions, which are specified by their values at the vertices, defined on a mesh. So, if a mesh has n vertices, then the functions on the mesh are represented by vectors with n components, and the Laplacian is expressed by an $n \times n$ matrix. Laplacian operator locally takes the difference between the function value at a vertex and a weighted average of its values at the first-order or immediate neighbor vertices. This form is given by [21]

$$(L\mathbf{f})_i = b_i^{-1} \sum_{j \in N(i)} w_{ij}\,(f_i - f_j) \tag{2.8}$$

where L is the Laplacian, $\mathbf{f}$ is a vector with n components, b_i^{-1} is a positive number, and w_{ij} are the symmetric edge weights.

Eigenvector orthogonality and positive semi-definiteness are important properties for Laplacian operators. An operator can be written as the product of a diagonal and a symmetric matrix as

$$L = B^{-1}\,S$$

where B^{-1} is a diagonal matrix whose diagonal entries are b_i^{-1}, S is a symmetric matrix whose diagonal entries are given by $S_{ii} = \sum_{j \in N(i)} w_{ij}$ and whose off diagonal entries are w_{ij} [21]. Even though L may not be symmetric, it is similar to a symmetric matrix $O = B^{-\frac{1}{2}}\,S\,B^{-\frac{1}{2}}$. So, L and O have the same real eigenvalues, and if v is an eigenvector of O with eigenvalue λ, then $u = B^{-\frac{1}{2}}v$ is an eigenvector of L with the same eigenvalue [21]. Since O is symmetric, its eigenvectors are mutually orthogonal. However, this may not be true for L in general, and an additional scalar product (Equation 2.9) might be required to make eigenvectors of L orthogonal

$$\langle \mathbf{f}, \mathbf{g} \rangle_B = \mathbf{f}^T\,B\,\mathbf{g} \tag{2.9}$$

The eigenvectors of L become orthogonal with respect to that product as

$$\left\langle u_i, u_j \right\rangle_B = u_i^T\,B\,u_j = v_i^T\,vj = \delta_{ij}$$

Another desirable property, being positive semi-definite, is also not guaranteed for L. However, L can be made positive semi-definite with respect to the proper inner product based on Equation 2.9 (weights w_{ij} are assumed non negative)

$$\langle \mathbf{f}, L\,\mathbf{f} \rangle_B = \mathbf{f}^T\,S\,\mathbf{f} = \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}(f_i - f_j)^2 \geq 0$$

A geometric mesh Laplacian is assembled using a discrete approximation to the Laplace-Beltrami operator on a smooth surface. On a C^∞ surface without boundary (ℓ), the Laplacian is a self-adjoint positive semi-definite operator; $\Delta_\ell : C^\infty(\ell) \to C^\infty(\ell)$.

An important property of Δ_ℓ is defined as [21]

$$\int_\ell f\ \Delta_\ell\ g\ da = \int_\ell \nabla\ f \cdot \nabla\ g\ da$$

On a surface with boundary, if von Neumann boundary conditions are established, the Laplacian remains self-adjoint. Choosing $g = f$ yields and establishes the positive semi-definite property [21]

$$\int_\ell f\ \Delta_\ell\ f\ da = \int_\ell \|\nabla f\|^2\, da$$

For a given triangulated mesh, the Laplace operator is constructed similar to Δ_ℓ. For this task, an n dimensional vector space of functions on the mesh are used. These functions represent piecewise linear (continuous) functions on the mesh, and they are defined by their values at the vertices. The Laplacian for the mesh is defined as

$$[C\ f]_i = \sum_{j \in N(i)} \frac{1}{2}(\cot\alpha_{ij} + \cot\beta_{ij})(f_i - f_j) \tag{2.10}$$

where the angles α_{ij} and β_{ij} are subtended by the edge (i, j). With respect to Equation 2.8, C is obtained by setting $b_i = 1$ for all i and $w_{ij} = \frac{1}{2}(\cot\alpha_{ij} + \cot\beta_{ij})$ for all i and j. Because of imposing von Neumann boundary conditions, the $\cot\beta_{ij}$ term vanishes if (i, j) is a boundary edge [21]. The geometric Laplacians based on cotangent formula (Equation 2.10) are currently the most popular discrete approximations to the Laplace-Beltrami operator used for geometry processing.

We use the geometric Laplacian defined by Vallet and Levy [18] for parameterization of ambient occlusion over an animated mesh. Vallet and Levy [18] present a method to convert the geometry of a mesh into frequency space. They use a special Laplace operator based on manifold harmonics, which is a generalization of spherical harmonics for arbitrary manifolds. Fourier-like function bases can be defined using eigenvectors of a discrete Laplacian. Discretization is done by finite element methods (FEM) and discrete exterior calculus (DEC). However, an orthogonal Laplace operator taking geometry into account is also needed to get correct results. Vallet and Levy [18] handle both of these issues (orthogonal and geometry-aware Laplacian). An algorithm for fast eigenvector calculation of this Laplacian is also given and a band-by-band spectrum computation algorithm is proposed.

Our approach only uses the orthogonal, geometry-aware Laplace operator to explore ambient occlusion parameterization. Details of our approach will be introduced in Chapter 3.

Eigenvectors of a Laplace operator are used to calculate manifold harmonics bases (MHB). Cotangent Laplacian (L), which is a Laplace operator used only for triangulated meshes, is used for setting up the Laplacian by Vallet and Levy [18]. It is represented as an $N \times N$ matrix, where N is the number of vertices. The cotangent Laplacian is calculated over a one-ring neighborhood (Figure 2.20).

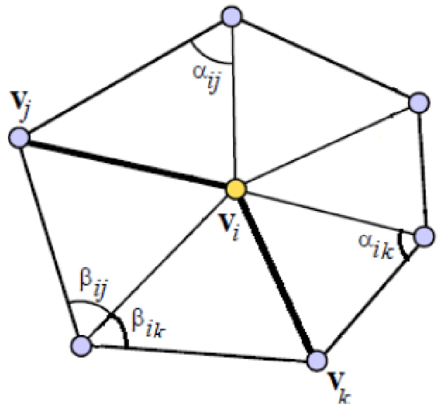


Figure 2.20: Opposite angles of the edge ij are used for computing the cotangent Laplacian (Image prepared based on Sorkine [14]).

The cotangent Laplacian takes geometry of the mesh into account; however, it is not orthogonal. By using discrete exterior calculus (DEC), we can make the cotangent Laplacian symmetric, and in this way, it becomes orthogonal. Symmetry for the Laplacian is quite important. If the Laplacian (Δ) loses its symmetry ($\Delta_{ij} \neq \Delta_{ji}$), the eigenfunction basis is no longer orthonormal, and this does not allow for the correct mesh construction when we transform the mesh from frequency space to geometry space, and the mesh becomes deformed (Figure 2.21).

Hodge star $\star_0$ is used to recover the symmetry. $\star_0$ is a diagonal matrix where its

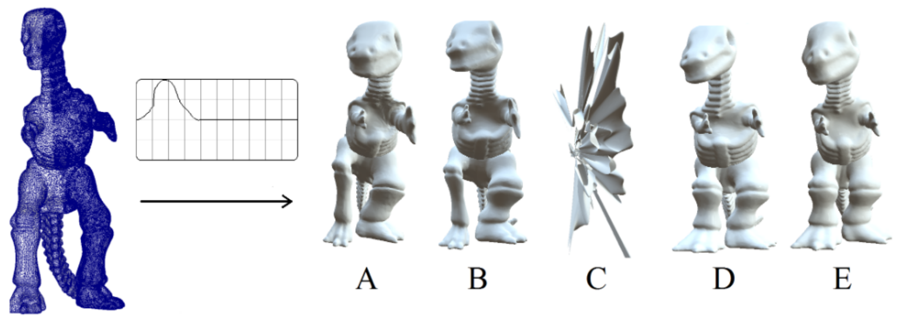


Figure 2.21: Because of losing symmetry, all the meshes are deformed except mesh E. Image is taken from Vallet and Levy [18].

diagonal elements correspond to the dual area associated with the vertex. Crane et al. [2] define this dual area as the total area of the one-ring neighborhood of a vertex divided by 3.

After the Hodge star $\star_0$ is calculated for the manifold, the Laplacian can be made symmetric (and its eigenvectors orthonormal) using Equation 2.11. Alternatively, each coefficient of this symmetric Laplace operator ($\bar{\Delta}$) can be computed as in Equation 2.12

$$\bar{\Delta} = \star_0^{-\frac{1}{2}} L \star_0^{-\frac{1}{2}} \tag{2.11}$$

where $\star_0^{-\frac{1}{2}}$ is the inverse square root of the Hodge star, and L is the cotangent Laplacian

$$\bar{\Delta}_{ij} = -\frac{\cot\beta_{ij} + \cot\beta_{ij}^{'}}{\sqrt{\left|v_i^*\right|\left|v_j^*\right|}} \tag{2.12}$$

where $\cot\beta$ and $\cot\beta^{'}$ are the cotangent values of the angles across edge ij, and v^* is the dual area of the given vertex.

One of the important steps in this implementation is handling numerical precision issues. They may make eigenvectors of $\bar{\Delta}$ not orthonormal even if the Laplacian is symmetric. Numerical precision issues can be solved by using $\bar{\Delta} = \left(\bar{\Delta} + \bar{\Delta}^T\right)/2$ operation.

After a positive semi-definite discrete Laplacian $\bar{\Delta}$ is assembled, its eigenvectors are computed to get the manifold harmonics bases. These bases are used for manifold harmonic transformations, which are a generalization of Fourier transform for transforming geometries (meshes) between frequency space and geometry space. When we transform the mesh into frequency space, we can filter it or smooth it out, and then transform it back to geometry space. These transformations are based on eigenvectors of $\bar{\Delta}$.

All eigenvector v^k and eigenvalue λ_k pairs of the Laplacian on a manifold satisfy $-\Delta v^k = \lambda_k v^k$. This minus sign is required for the eigenvalues to be positive [8]. This is also the reason why $\bar{\Delta}$ is multiplied by -1 in Algorithm 5.

Algorithm 5 Setting up the Laplacian and getting MH bases

```
1: Calculate the cotangent Laplacian ($L$) of the mesh
2: Calculate the dual area ($\star_0$) of the vertices
3: Assemble the positive semi-definite discrete Laplacian ($\bar{\Delta}$) using Equation 2.11
4: Multiply $\bar{\Delta}$ by $-1$ to get positive eigenvalues
5: Compute the eigenvalues and eigenvectors of $\bar{\Delta}$
6: Sort the eigenvalues and eigenvectors (ascending order)
7: Map the MH bases into canonical bases using $H = \star_0^{-\frac{1}{2}} \bar{H}$
```

Another important point is to order the eigenvalues and eigenvectors. The eigenvectors are used either sorted by increasing or by decreasing eigenvalues. Vallet and Levy [18] use ascending order. Thus, lower eigenvectors contain a general idea of the shape whereas higher eigenvectors contain details of the shape. For example, a mesh smoothing process is performed by excluding these higher bases (Figure 2.22).

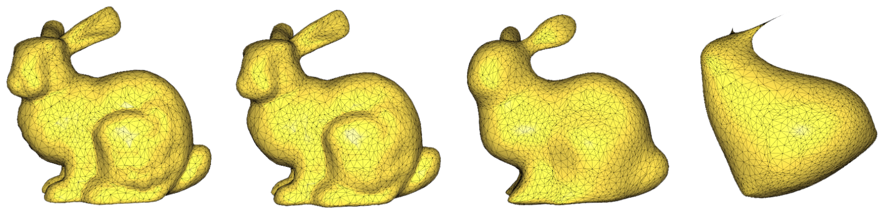


Figure 2.22: Smoothing process of the Stanford bunny with 3485 (original), 1024, 97, and 6 MH bases, respectively.

Manifold harmonic transform (MHT) is used to transform the mesh into frequency space

$$\tilde{x} = x^T \star_0 H \tag{2.13}$$

where x is the vector of the x coordinate of the vertices of the mesh, $\tilde{x}$ is the frequency space coefficients for those x coordinates, and H is the manifold harmonics bases of the mesh. The same idea is also applied to the y and z coordinates of the mesh vertices if necessary. During our investigation, we used the per-vertex ray-traced ambient occlusion results instead of the positions (coordinate values), so we obtained the ambient occlusion coefficients in frequency space.

The inverse manifold harmonic transform (MHT^{-1}) is used to transform the mesh in frequency space back into geometry-space (Equation 2.14). For this operation, the coefficients in frequency space (such as $\tilde{x}_k$) are multiplied with the corresponding manifold harmonic basis vector (H_k), and the results are summed to get the results in geometry space, e.g., the x coordinates of the vertices of the mesh. Algorithm 6 can be followed for the geometry processing using manifold harmonics bases.

$$x = \sum_{k=1}^{m} \tilde{x}_k \, H_k \tag{2.14}$$

where x are the x coordinates of the vertices of the mesh transformed back into geometry-space, m is the number of MH bases in use, $\tilde{x}_k$ is the k^{th} frequency-space coefficient, and H_k is the basis vector of the k^{th} MHB.

Algorithm 6 Geometry processing using MH bases

```
1: Assemble the positive semi-definite discrete Laplacian (Equation 2.11)
2: Compute the eigenvectors of the Laplacian to get MH bases
3: Map the bases into canonical bases (Step 7 in Algorithm 5)
4: Transform the mesh into frequency space using MHT (Equation 2.13)
5: Perform the operation (smoothing, filtering, etc.)
6: Transform the mesh back into geometry space using MHT$^{-1}$ (Equation 2.14)
```

We apply the idea presented by Vallet and Levy [18] with the per-vertex, ray-traced ambient occlusion values to parameterize with manifold harmonics bases. We also experiment with a smoothing process by decreasing the number of frequency-space coefficients. Details can be found in Chapter 3.

CHAPTER 3

COMBINED PARAMETERIZATIONS

The main goal of our investigation is the study of different parameterizations (such as joint, spatial, orientation, etc.) to reduce animation-dependent light transportation. We would like to come up with a system to approximate realistic real-time rendering. As an important visual localized effect, we start our investigation with ambient occlusion, which will simplify our first steps. Precomputed per-vertex ray-traced ambient occlusion results are parameterized according to skeletal joint angles, joint positions, surface vertex positions, and manifold harmonics. In this chapter, we describe our parameterization systems, how we express them in matricial forms, and discuss their advantages and disadvantages.

We start by assuming that there is a linear dependency between the motion and the appearance of surfaces associated to bones, and we construct a linear system (Equation 3.1) to express this relation based on Kontkanen and Aila [6]. Our system is fed with a selected motion (reference poses) and the corresponding appearance data. These motion-appearance pairs are learnt by our system and parameters are extracted. These parameters are then used to approximate the appearance of arbitrary motions in real time, thanks to the following expression

$$\mathbf{A} = \mathbf{J}\mathbf{T} \tag{3.1}$$

where $\mathbf{A}$ is a $P \times N$ matrix containing the appearance information of P reference poses, for a surface mesh of V vertices, $\mathbf{J}$ is a $P \times M$ matrix containing motion information (joint angles, vertex positions, manifold harmonic (MH) bases, etc.) of reference poses, and $\mathbf{T}$ is an $M \times N$ matrix containing the extracted parameters. Here, M refers to the number of motion information components, e.g., vertex positions in 3D such that $P \times M = P \times 3N$.

During our investigation, we consider that $\mathbf{A}$ is always filled with the precomputed per-vertex ray-traced ambient occlusion results. We also refer to them as our ground-truth results. **J** is constructed with different kinds of coefficients depending on our parameterization systems. **T** is extracted by solving the linear system.

Since Equation 3.1 is an overdetermined system of linear equations, there is no exact solution for **T** and an optimal solution can be computed using the pseudoinverse $\mathbf{J}^{+}$ of **J** (Equation 3.2). The pseudoinverse is defined and unique for all real or complex matrices.

$$\mathbf{J}^{+}\mathbf{A} = \mathbf{T} \tag{3.2}$$

Each column of **T** includes the per-vertex approximation parameters (except for the MH parameterization) and defines the visual effects due to all the motion information (values in **J**) on a particular vertex. Thus, these parameters and the current motion information can be used to approximate in real time, the per-vertex ambient occlusion for arbitrary poses. This is done by establishing a linear mapping using the dot product between the current motion information and the approximation parameters, such that

$$a_i = j^T t_i \tag{3.3}$$

where a_i is the ambient occlusion value at vertex i, j is the motion information of the current pose, e.g., vertex positions, and t_i is the approximation parameters associated with the i^{th} vertex (i^{th} column of **T**).

The general steps of the parameterization process follow Algorithm 7.

Algorithm 7 General steps of parameterization

```
1: Feed A with the precomputed appearance data
2: Feed J with the corresponding motion data
3: Take the pseudoinverse of J
4: Multiply the pseudoinverse of J with A to get the approximation parameters (Equation 3.2)
5: Use the related parameters in T for the approximation in real time (Equation 3.3)
```

3.1 Joint Angles

We first parameterize the ambient occlusion using joint angles of the mesh skeleton. This is also what Kontkanen and Aila [6] did. Joint parameterization helped us to construct the basis of our system.

Matrix **A** is filled with the per-vertex ambient occlusion of each reference pose *P*. Ambient occlusion is precomputed using ray tracing for each pose of the animated mesh. These AO values are also used as ground-truth values in our investigation.

$$\mathbf{A} = \begin{bmatrix} AO(v_1, p_1) & \dots & AO(v_N, p_1) \\ \vdots & & \vdots \\ AO(v_1, p_{\hat{p}}) & \dots & AO(v_N, p_{\hat{p}}) \end{bmatrix}$$

Matrix **J** is filled with joint angles defining by rotation *R* of each joint for the related pose *P*. Most of the time this information is available in the animation file; however, it is not directly used in the skinning process. Joint angles are used to generate the rotation component of the transformation matrices. Thus, joint angle parameterization requires extra information, which is not needed for skinning, both for precomputation and real-time approximation. Additionally, joint angle parameterization cannot be plugged in the current skinning pipeline right away, as some modifications are required in the shader code.

$$\mathbf{J} = \begin{bmatrix} JA(R_1, p_1) & \dots & JA(R_M, p_1) & \lambda \\ \vdots & & \vdots & \vdots \\ JA(R_1, p_{\hat{p}}) & \dots & JA(R_M, p_{\hat{p}}) & \lambda \end{bmatrix}$$

Joint angle parameterization also requires each row of **J** to be extended with a constant value λ, which is the same for each reference pose in order to capture ambient occlusion independently of any actual animation parameters. Our experiments on this extra constant value showed that its effect can be crucial to reduce errors between ground truth and approximated results.

T is extracted using Equation 3.2. Each row of **T** includes the effects of the related pose information (joint angles) to all vertices. Thus, columns of **T** include the approximation parameters of each vertex ($\tilde{AO}$) and they are used for the per-vertex ambient occlusion approximation (Figure 3.1). Equation 3.3 is used for this task. Current pose vector j in the Equation 3.3 is filled with the joint angles of the current animation pose.

$$\mathbf{T} = \begin{bmatrix} \tilde{AO}(v_1,t_1) & \dots & \tilde{AO}(v_N,t_1) \\ \vdots & & \vdots \\ \tilde{AO}(v_1,t_M) & \dots & \tilde{AO}(v_N,t_M) \end{bmatrix}$$

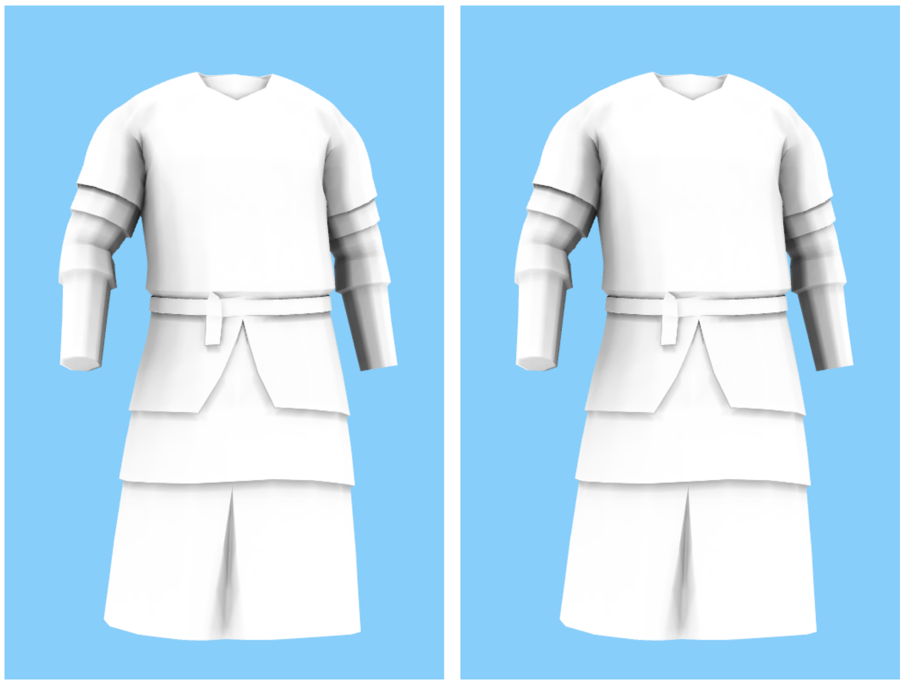


Figure 3.1: Ambient occlusion ground truth (left), and approximation using joint angle parameterization (right). Walking and running animations are learnt, walking animation is approximated. Since the error rate is very small, there is no visual difference between the approximation and ground truth.

Memory requirements for joint angle parameterization is rather low, e.g., 3.76 MB is required for 15410 mesh vertices. However, because of the reasons mentioned in

Section 2.4, some "higher" effects in motions depending on many parameters cannot be captured, such as the shadow casted on the mesh body (face, torso, legs, etc.) by the hand. Therefore joint angle parameterization fails for some more complex animation sequences (Figure 3.2).

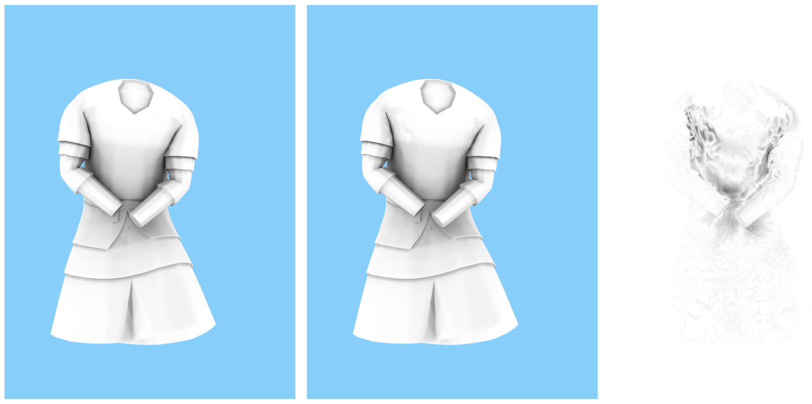


Figure 3.2: Joint angle parameterization suffers flickering for the complex motions that depends multiple animation parameters, i.e., while armpit area suffers flickering, shadows casting by the arms to torso approximated error prone. Ground truth (left), joint angle approximation (middle) and 10 times magnified difference (right). All frames of the striking animation are used to train the model.

3.2 Vertex Positions

In the parameterization with vertex positions, we keep matrix **A** the same; however, matrix **J** is updated with vertex positions of the mesh in the reference pose:

$$\mathbf{J} = \begin{bmatrix} JA(P_{v_1}, p_1) & \dots & JA(P_{v_M}, p_1) & \lambda \\ \vdots & & \vdots & \vdots \\ JA(P_{v_1}, p_{\hat{p}}) & \dots & JA(P_{v_M}, p_{\hat{p}}) & \lambda \end{bmatrix}$$

T is extracted in the same fashion (using Equation 3.2), and rows of **T** include effects of vertex positions of the reference poses to all vertices. Real-time approximation of vertex position parameterization is also done in the same fashion as joint angle parameterization using Equation 3.3. However, this time vector j includes the vertex position of the current pose coming from the skinning of the mesh with the current pose. Thus, theoretically, vertex position parameterization can be plugged in current skinning pipelines, and the per-vertex ambient occlusion can be approximated during skinning.

Since positions of all the mesh vertices are used, memory requirements of vertex position parameterization are higher than joint angle parameterization. For finely tessellated meshes or multiple meshes in a scene, this can explode memory requirements and it is not suitable for real-time rendering. To address this issue, we introduce clustering, which decreases memory requirements, and allows us to experiment with local mesh space instead of global space.

We cluster mesh vertices based on their skinning weights. Here, the joint with the largest impact (weight) on a vertex is chosen as its cluster, and all the vertices affected the most by the same joint are clustered together. After creating the clusters, an individual linear system is constructed for each such cluster. Consequently, one matrix **A** per cluster is filled with reference pose ambient occlusion of the vertices in the clusters.

$$\mathbf{A}_c = \begin{bmatrix} AO_c(v_1, p_1) & \dots & AO_c(v_N, p_1) \\ \vdots & & \vdots \\ AO_c(v_1, p_{\hat{p}}) & \dots & AO_c(v_N, p_{\hat{p}}) \end{bmatrix}$$

Similarly, $\mathbf{J}_c$ is filled with the vertex positions of the reference poses in the cluster.

$$\mathbf{J}_c = \begin{bmatrix} JA_c(P_1, p_1) & \dots & JA_c(P_M, p_1) & \lambda \\ \vdots & & \vdots & \vdots \\ JA_c(P_1, p_{\hat{p}}) & \dots & JA_c(P_M, p_{\hat{p}}) & \lambda \end{bmatrix}$$

Thus, we get one matrix **T** for each cluster, and ambient occlusion is approximated using Equation 3.3 for each cluster using the same j, the current pose vector, with each cluster's approximation parameters (Figure 3.3). Algorithm 8 can be followed for the clustering steps.

Algorithm 8 Steps of clustered vertex position parameterization

```
1: Cluster each vertex according to the joint with the largest weight on this vertex
2: Generate $\mathbf{A}_c$ with the precomputed appearance data of related vertices for each cluster
3: Generate $\mathbf{J}_c$ with the reference pose vertex positions of related vertices for each cluster
4: Solve the linear system (Equation 3.2) to get the approximation parameters ($\mathbf{T}$ matrix) of each cluster
5: Approximate the ambient occlusion in real time using Equation 3.3 for each cluster
```

Since there are many more vertex positions than joint angles, vertex position parameterization causes the system to face overfitting. In this case, the system gets confused and begins to make wrong decisions. For example, the meshes in our experiments can generate about 48000 parameters in total. For complex motions defined with many animation frames, approximation results thus become inaccurate (Figure 3.4).

Regularization reduces the overfitting problem. Regularization simplifies the hypothesis made (such as the linear dependency between the animation and the approximation parameters for ambient occlusion) by penalizing some of the parameters (approximation parameters in this case) using the regularization parameter λ. Since we multiply the extracted parameters with the regularization parameter λ, the effects of some parameters are reduced or not considered at all. In this way, the system stops getting confused since only the effects of the important parameters are considered.

One of the linear regularization methods is based on a gradient descent algorithm (Equation 3.4). However, this approach depends on parameters such as a learning rate α and a regularization parameter λ. Moreover, a gradient descent algorithm works iteratively to only reach a local minimum. If the learning rate α is too large, a local minimum can be missed. On the contrary, if the learning rate is too small, it takes more iterations to reach a local minimum. Additionally, the initial guess (starting solution) plays a crucial role in a gradient descent algorithm.

$$\theta_j = \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i) x_j^i + \frac{\lambda}{m} \theta_j \right] \tag{3.4}$$

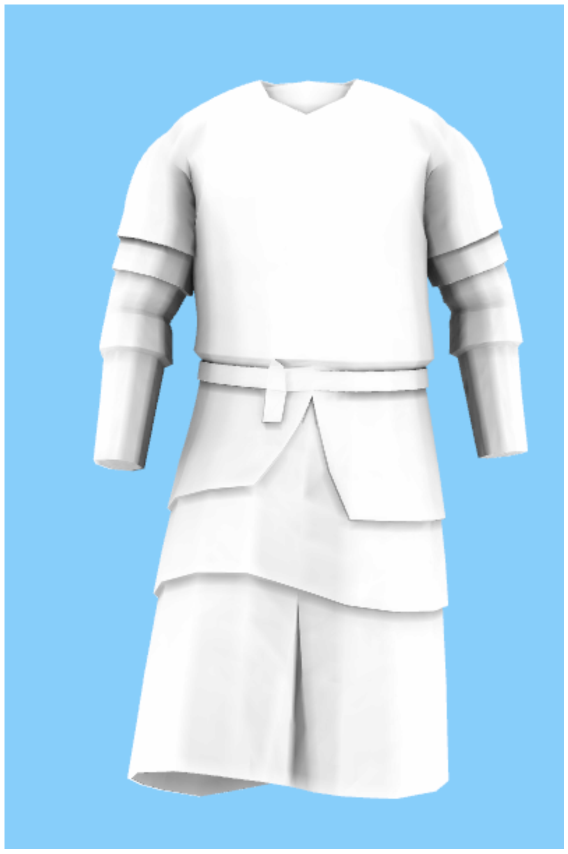


Figure 3.3: Ambient occlusion approximation using clustered vertex position parameterization. Learning and approximating walking animation. Since walking animation does not have many frames (only 30 frames), overfitting does not happen and regularization is not required.

where θ_j is the j^{th} parameter extracted, α is the learning rate, m is the number of samples in the training data, $h_\theta(x^i)$ is the approximated value of the i^{th} sample, y^i is the ground-truth value of the i^{th} sample, x^i_j is the value of feature j for the i^{th} sample (animation parameters in our case), and λ is the regularization parameter.

Using regularization with the normal equation (Equation 3.5), we can eliminate the learning rate α and the importance of the initial guess.

$$\mathbf{\Theta} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{M})^{-1}\mathbf{X}^T\mathbf{Y} \tag{3.5}$$

where $\mathbf{\Theta}$ contains all the approximation parameters, $\mathbf{X}$ contains all the variables effecting

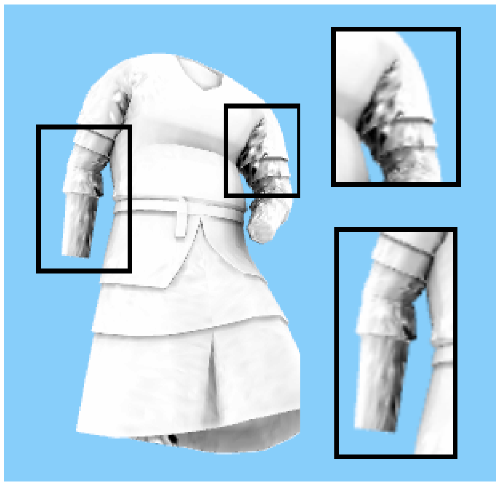


Figure 3.4: Clustered vertex position parameterization without regularization suffers from wrong approximations since the system gets confused during the parameter extraction process because of working with a large number of animation parameters.

the approximation, λ is the regularization parameter, $\mathbf{Y}$ contains the ground-truth values, and $\mathbf{M}$ is the identity matrix with a zero in the upper left diagonal entry.

Additionally, the normal equation solves for an optimal value directly. Thus, it eliminates iterations and results are directly optimal solutions. Even though the normal equation can be costly to evaluate for some systems since it requires expensive matrix operations, such as transpose, multiplication, and inverse, this is not our case. Our approach is more suitable to work with the normal equation, and getting the most optimal results depending on less parameters is more important for us to increase the realism of AO approximations.

We update Equation 3.5 with our data (the precomputed ray-traced ambient occlusion and clustered vertex positions) and use Equation 3.6 to get the approximation parameters $\mathbf{T}$. This operation is performed for each cluster.

$$\mathbf{T} = (\mathbf{J}^T\mathbf{J} + \lambda\mathbf{M})^{-1}\mathbf{J}^T\mathbf{A} \tag{3.6}$$

where $\mathbf{T}$ is the approximation parameters, $\mathbf{J}$ is the vertex positions of reference poses, λ is the regularization parameter, $\mathbf{M}$ is the identity matrix with a zero in the upper left diagonal entry, and $\mathbf{A}$ is the per-vertex ambient occlusion of reference poses.

We also performed some experiments on regularization parameter λ. A set of values in the range $[10, -10]$ is assigned to λ, and the approximation parameters are recomputed, then both visual results and error rate of the approximation are considered. For the mesh we used in Figure 3.5, $\lambda = 10^{-5}$ generated the best results.

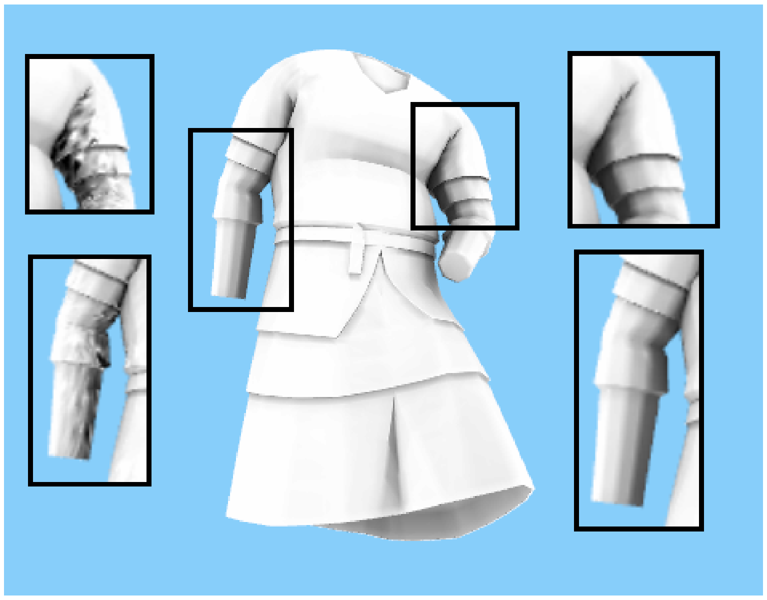


Figure 3.5: Fixed results using regularization.

We also performed a singular value decomposition (SVD) experiment to reduce the number of approximation parameters. A square matrix $\mathbf{E}$ can be decomposed into its eigenvalues and eigenvectors as $\mathbf{E}v = \lambda v$ where v contains the eigenvectors of $\mathbf{E}$ and λ the eigenvalues of $\mathbf{E}$. An eigenvector is a vector whose direction does not change by the transformation. Even if the system may be stretched, the vectors point the same direction. Each eigenvector has a corresponding eigenvalue that gives the scaling factor. In other words, an eigenvector is the direction, and its eigenvalue tells the variance in that direction (how the data spreads out on the line). So, the eigenvector with the highest eigenvalue is therefore the principal component.

Using these characteristics we can reduce the number of operations needed. For example, we can predict the final results without making all shading calculations, we can decide to sample more or less some parts of the scene, or we can remove some unnecessary parameters to reduce computational or storage costs.

While square matrices ($N \times N$) have eigenvalues, rectangular matrices ($M \times N$) have singular values. Singular values can be used for the same purpose by applying singular value decomposition (SVD). We applied SVD to our $\mathbf{T}_{P\times 3V}$ matrix (where P is the number of the reference poses and V is the number of vertices in the cluster), and experimented reducing the singular values. Our visual and mathematical results showed that $\mathbf{T}$ can be reduced by more than half while keeping the same error rate and visual quality (Figure 3.6).

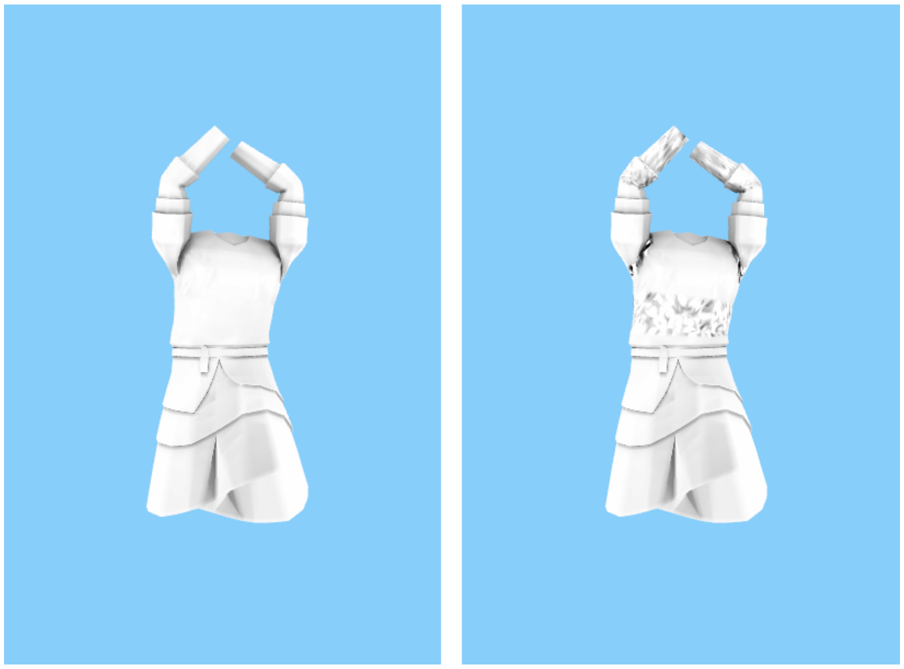


Figure 3.6: Approximation of striking animation using 50% of the singular values (left), and 10% of the singular values (right). The approximation fails for geometries (such as armpit) or areas (such as arms and torso) that break the assumptions that appearance of the object depends linearly on the motion parameters, and that the motion parameters can be handled independently. These assumptions were made by Kontkanen and Aila [6].

Table 3.I and Figure 3.7 can be referred for the error rates and plots of the singular values.

Vertex position parameterization also requires the constant value λ to capture AO independently of any actual animation parameters, and fails at complex motions as joint

	Max Error	Min Error	Mean of MSE (all frames)	Variance of MSE (all frames)
10%	0.0177	0.0161	0.0168	1.9054×10^{-7}
50%	5.6585×10^{-4}	4.3844×10^{-5}	1.7870×10^{-4}	1.4322×10^{-8}
70%	5.6585×10^{-4}	4.3844×10^{-5}	1.7870×10^{-4}	1.4322×10^{-8}
90%	5.6585×10^{-4}	4.3844×10^{-5}	1.7870×10^{-4}	1.4322×10^{-8}
100%	5.6585×10^{-4}	4.3844×10^{-5}	1.7870×10^{-4}	1.4322×10^{-8}

Table 3.I: MSE of striking animation with different numbers of parameters. The same amount of the per-vertex error rates dominates the mean-squared error until less than 50% of the singular values are used.

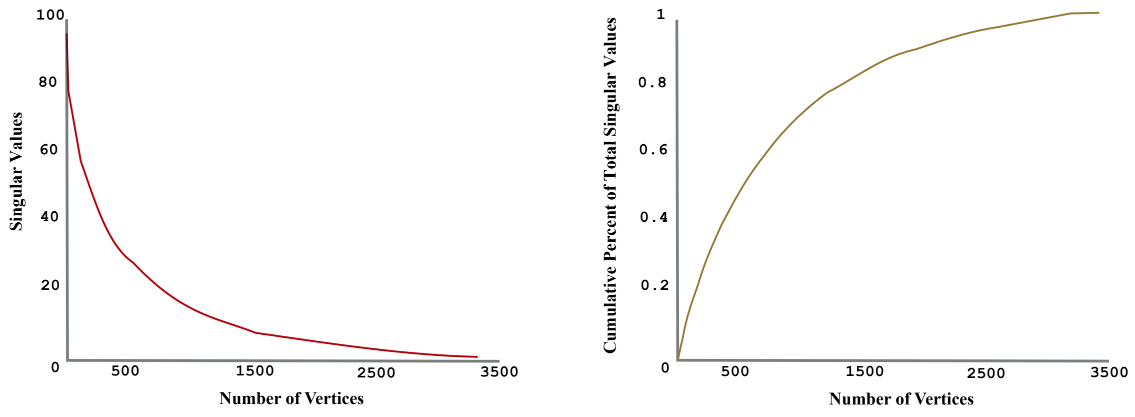
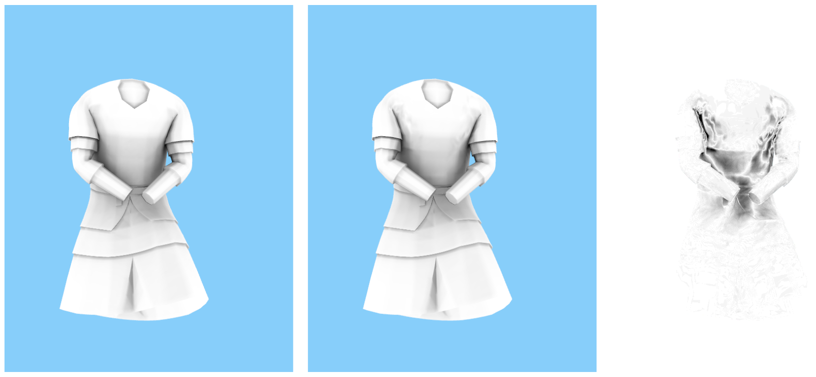


Figure 3.7: SVD plots for a typical cluster.

angle parameterization (Figure 3.8). Memory requirements of clustered vertex position parameterization are higher than joint angle parameterization, e.g., for the same mesh with 15410 vertices, clustered vertex parameterization requires 338 MB. On the other hand, vertex position parameterization does not require any extra information that is not used in the skinning process, and can be plugged in any current skinning pipeline.

3.3 Joint Positions

We also investigate parameterization with joint positions similar to Kirk and Arikan [5], but without any kind of clustering and only considering joint positions. The same pre-computed per-vertex ray-traced ambient occlusion matrix **A** is kept, and **J** is filled with the joint positions of the reference poses.



$$\mathbf{J} = \begin{bmatrix} JA(P_{j_1}, p_1) & \dots & JA(P_{j_M}, p_1) & \lambda \\ \vdots & & \vdots & \vdots \\ JA(P_{j_1}, p_{\hat{p}}) & \dots & JA(P_{j_M}, p_{\hat{p}}) & \lambda \end{bmatrix}$$

Equation 3.2 is used to solve the linear system, and matrix **T**, containing effects of joint positions, is extracted. Equation 3.3 is used for real-time approximation of ambient occlusion with the j vector containing joint positions of the current pose.

Similar to joint angle parameterization, joint position parameterization has low memory requirements; however, it requires extra information not needed by the skinning process both for extracting **T** and for real-time approximation. Additionally, the constant value λ is required during the approximation of **T**, and joint position parameterization fails at complex motions as do joint angle and vertex position parameterizations.

3.4 Manifold Harmonics

We begin our investigation on parameterization with manifold harmonics using a static mesh. We construct the bases of the mesh, and get the spectral coefficients of ambient occlusion using Equation 3.7 based on Equation 2.13.

$$\tilde{a} = a^T \star_0 H \tag{3.7}$$

where a is the vector of precomputed per-vertex ray-traced ambient occlusion (ground truth), $\tilde{a}$ is the frequency-space coefficients of ambient occlusion, $\star_0$ is the Hodge star operator (dual area of vertices), and H is the manifold harmonics bases of the mesh. For details, please refer to Section 2.5.

Then, we use Equation 3.8 based on Equation 2.14 and reconstruct ambient occlusion using the frequency-space ambient occlusion coefficients and the manifold harmonics bases of the mesh as

$$a = \sum_{k=1}^{m} \tilde{a}_k H_k \tag{3.8}$$

where a is the reconstructed per-vertex ambient occlusion, m is the number of MH bases in use, $\tilde{a}_k$ is the k^{th} frequency-space ambient occlusion coefficient, and H_k is the basis vector of the k^{th} MH basis.

Because of MH properties, when all the MH bases are used, the exact same results as the ground truth can be obtained if the Laplacian is set properly. Similarly, when only one basis (MH basis with the smallest eigenvalue) is used, the result is an identical value for all vertices. We applied these two observations to our ambient occlusion reconstruction to verify our results (Figure 3.9).

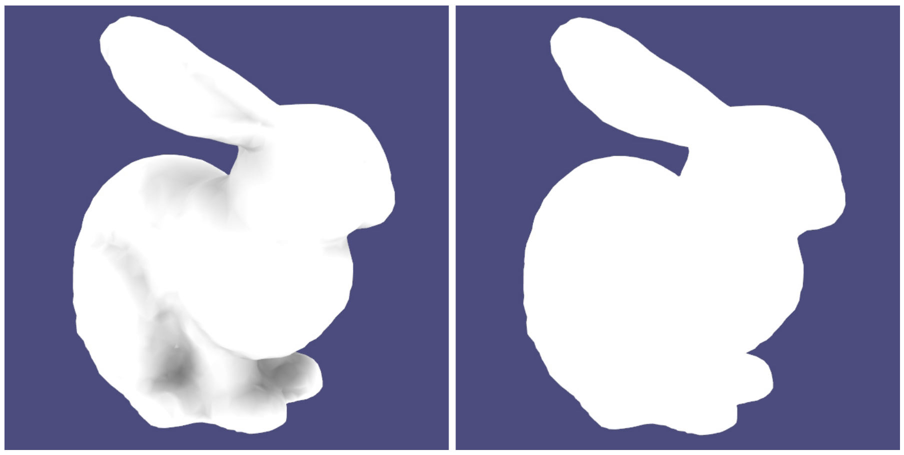


Figure 3.9: Ambient occlusion approximation based on MH bases parameterization on the bunny. Left, approximation with all MH bases approximation. Right, approximation with only the first (the smallest) base. AO is fully reconstructed with all MH bases, and it is identical for all vertices when only the first basis is used.

We also experimented on the visual effects resulting from the number of bases in use. Our experiments show that when we decrease the number of bases by more than 30%, we start to lose details (Figure 3.10). On the other hand, the visual appearance of ambient occlusion remains even if we decrease by 85% of the bases.

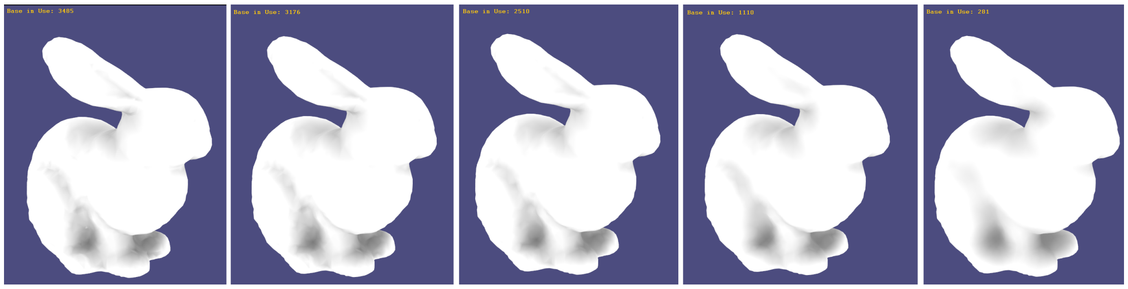


Figure 3.10: Ambient occlusion approximation based on MH bases parameterization. 100%, 91%, 72%, 32%, and 8% of the bases in use respectively.

Algorithm 9 details the process to compute the frequency-space ambient occlusion coefficients, and the reconstruction using the fewer bases.

Algorithm 9 Ambient occlusion approximation using MH parameterization for a static mesh

```
1: Assemble the positive semi-definite discrete Laplacian (Equation 2.11)
2: Get MH bases of the Laplacian
3: Compute the frequency-space ambient occlusion coefficients (Equation 3.7)
4: repeat
5:     Multiply and sum the related frequency-space coefficients and bases (Equation 3.8)
6: until $k$ reaches a user-specified upper bound ($k \in [1,m]$, where $m$ is the number of MH bases in use)
```

The Laplace operator defined by Vallet and Levy [18] is a 3D Laplacian that is not convenient to capture ambient occlusion over time (animation). For this task, the current Laplacian $\bar{\Delta}$ must be extended to 5D, which includes ambient occlusion (4D) over time (5D).

Ambient occlusion can be associated with the curvature of the manifold. The local bending of the surface is measured by the curvature, i.e., how fast a curve is changing direction at a given point. High changes in curvature translate to high frequencies. In the ambient occlusion case, we differentiate between positive and negative curvatures since a positive (convex) curvature gives brighter shading (e.g., bump) and a negative curvature (concave) gives dark shading (e.g., dip).

This idea is applied to our $\bar{\Delta}$ to extend it to 4D. Since $\bar{\Delta}$ is positive semi-definite, $\bar{\Delta} + \mathbf{D}$ is also positive semi-definite, where $\mathbf{D}$ is a diagonal matrix with positive entries. The emphasis of the bases can be pushed towards concave regions by clamping the curvature, c_i in the $[0, +\infty]$ range for each vertex i, and using these values for the diagonal entries of $\bar{\Delta}$. This operation is performed using Equation 3.9.

$$\tilde{\Delta} = (\bar{\Delta} + w\mathbf{D}) \tag{3.9}$$

where $\tilde{\Delta}$ is the 5D, extended Laplacian, $\bar{\Delta}$ is our original Laplacian defined in [18], w is the scalar weight mentioned above, $\mathbf{D}$ is a diagonal matrix with positive curvature entries.

This kind of curvature is not the real curvature but it behaves in the same way. Additionally, for extending $\bar{\Delta}$ to time (animation), instead of using a scalar weight for the basis coefficients, each weight becomes a function of time, and this function can be approximated with a continuous Fourier transform.

We start our investigation by considering only ambient occlusion (4D case), and we use scalar weights $W = \{-10, -1, -0.5, 0, 0.5, 1, 10\}$. Setting $w = 0$ generates the same results with $\bar{\Delta}$ as expected. However, all other $\tilde{\Delta}$ Laplacians, computed with scalar weights other than 0, fail to generate correct results including the case that only the first basis of $\tilde{\Delta}$ in use (Figure 3.11).

After the curvature-based approach failed to obtain generalized MH bases, we decided to investigate the linear system used in other parameterization techniques. We started our investigation by focusing on Equation 3.8. We can write the system in Equation 3.8 as

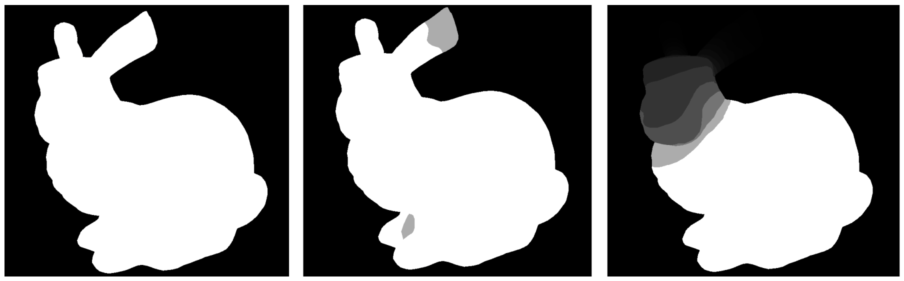


Figure 3.11: MH bases approximation using the first basis. Left: $w = 0$ (original). Middle: $w = -0.5$. Right: $w = 0.5$. Only $w = 0$ gets uniform results when only the first basis is used. Gamma correction was applied to these images.

$$\begin{aligned}
a_1 &= \tilde{a}_1 H_{11} + \tilde{a}_2 H_{21} + \tilde{a}_3 H_{31} + \ldots + \tilde{a}_v H_{v1} \\
a_2 &= \tilde{a}_1 H_{12} + \tilde{a}_2 H_{22} + \tilde{a}_3 H_{32} + \ldots + \tilde{a}_v H_{v2} \\
a_3 &= \tilde{a}_1 H_{13} + \tilde{a}_2 H_{23} + \tilde{a}_3 H_{33} + \ldots + \tilde{a}_v H_{v3} \\
&\vdots \\
a_v &= \tilde{a}_1 H_{1v} + \tilde{a}_2 H_{2v} + \tilde{a}_3 H_{3v} + \ldots + \tilde{a}_v H_{vv}
\end{aligned}$$

where a_i is the ambient occlusion of vertex i, the $\tilde{a}$ vectors are the frequency-space coefficients, and the H_{ij} vectors are the coefficients related manifold harmonics bases. This operation can also be written using matrix multiplication as in Equation 3.10.

$$\begin{bmatrix} a_1 & a_2 & \ldots & a_v \end{bmatrix} = \begin{bmatrix} \tilde{a}_1 & \tilde{a}_2 & \ldots & \tilde{a}_v \end{bmatrix} \begin{bmatrix} H_{11} & \ldots & H_{1v} \\ \vdots & \ddots & \vdots \\ H_{v1} & \ldots & H_{vv} \end{bmatrix} \tag{3.10}$$

Equation 3.10 is for a single pose; it can be extended to all poses by adding all the frequency-space coefficients of the poses along with MH bases. One important step is to fill the unrelated part of the MH bases matrix with zeroes. Thus, Equation 3.10 becomes a vector-matrix multiplication as

$$A_{1\times PV} = \tilde{A}_{1\times PV}\ \hat{\mathbf{H}}_{PV\times PV} \tag{3.11}$$

where A is the per-vertex ambient occlusion vector for P poses, $\tilde{A}$ is the frequency-space coefficient vector of P poses, $\hat{\mathbf{H}}$ is the matrix filled with the MH bases of the poses and zeroes.

Equation 3.11 establishes our basis to generate the extended MH bases animation (time) and ambient occlusion. As you can see, Equation 3.11 is similar to Equation 3.1 and the per-vertex ambient occlusion depends linearly on the frequency-space coefficients and the MH bases. Thus, we can rewrite Equation 3.11 and get

$$\mathbf{A} = \tilde{\mathbf{A}}\tilde{H} \tag{3.12}$$

where $\mathbf{A}$ includes the per-vertex ambient occlusion of P reference poses, $\tilde{\mathbf{A}}$ the frequency-space coefficients of P reference poses, and $\tilde{H}$ is the generic MH bases.

$\mathbf{A}$ is filled by the precomputed per-vertex ray-traced ambient occlusion of each reference pose P;

$$\mathbf{A} = \begin{bmatrix} AO(v_1, p_1) & \dots & AO(v_N, p_1) \\ \vdots & & \vdots \\ AO(v_1, p_{\hat{p}}) & \dots & AO(v_N, p_{\hat{p}}) \end{bmatrix}$$

Similarly, $\tilde{\mathbf{A}}$ is filled with the ambient occlusion frequency-space coefficients of reference poses P. Since the frequency-space coefficients are gathered by multiplying the ground-truth ambient occlusion with dual area and manifold harmonics bases of the mesh (Equation 3.7), they include both geometry and ambient occlusion information.

$$\tilde{\mathbf{A}} = \begin{bmatrix} JA(\tilde{ao}_1, p_1) & \dots & JA(\tilde{ao}_M, p_1) \\ \vdots & & \vdots \\ JA(\tilde{ao}_1, p_{\hat{p}}) & \dots & JA(\tilde{ao}_M, p_{\hat{p}}) \end{bmatrix}$$

Finally, $\tilde{\mathbf{H}}$ is a $V \times V$ matrix that includes animation and ambient occlusion related information since we get it by solving the linear system. $\tilde{\mathbf{H}}$ is generic and used to approximate the per-vertex ambient occlusion in real time using Equation 3.13

$$a = \tilde{a}\tilde{\mathbf{H}} \tag{3.13}$$

where a is the per-vertex ambient occlusion of the current pose, $\tilde{a}$ is the frequency-space coefficients of the current pose, and $\tilde{\mathbf{H}}$ is the generic, 5D MH bases.

Using the bases for the whole mesh is not handy for real-time operations since $\tilde{\mathbf{H}}$ has large memory requirements. Thus, we again introduce clustering to decrease memory requirements. For not losing any information from the animated mesh, the vertices are clustered depending on as many joints as possible. However, this creates another problem. Considering all possible joints having an effect on a vertex leads to many clusters with few vertices. Small clusters tend to generate error prone results in the MH bases parameterization. Thus, we introduce a threshold to consider only joints having influence above this threshold on the vertex. In this way, the multiple joints having a large influence on vertices are considered, but the joints having a small impact are eliminated while keeping the quality of the results. Our experiments with the threshold showed that values in 0.2 level are good for the meshes with low tessellation or bad geometry, and values in 0.1 for well generated meshes (Figure 3.12).

After clustering, each cluster has its own linear system and generic MH bases $\tilde{\mathbf{H}}$. Additionally, the per-vertex ambient occlusion is calculated using each cluster's generic basis with the current pose's frequency space coefficients.

One important point at this step is setting the related data for each cluster. Clustering according to multiple skinning weights generates geometries with discontinuities and some important geometric information can be lost for calculation of the cotangent Laplacian (L) or the dual area (Hodge Star $\star_0$). This leads to incorrect MH bases, and correct rendering results cannot be obtained. The easiest way to solve this issue and get the correct MH bases is calculating the cotangent Laplacian and the dual area for the whole mesh and gathering the related parts for the clusters from these matrices. These

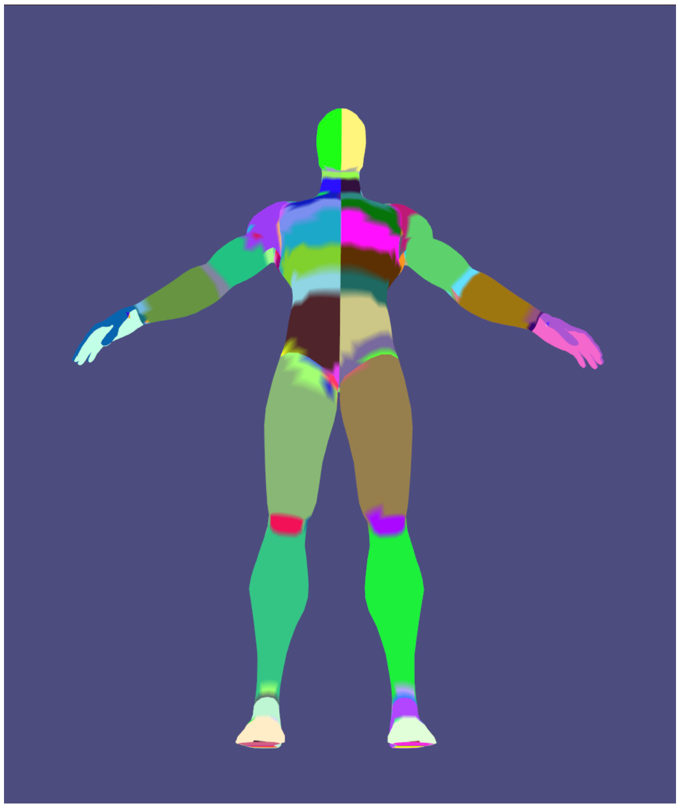


Figure 3.12: Clusters for the Amazing Spiderman mesh. 0.26 is used as the threshold. "Amazing Spiderman Rigged V2" mesh by "bLender" user on Blend Swap.

steps are performed for each the reference pose.

Additionally, even if we set correct values for the positive semi-definite Laplacian, we may face some non-positive coefficients when we get MH bases for some of the clusters. This situation leads to mesh faces with sudden changes in the results when the number of bases in use is decreased because the clusters include some subclusters sharing the same MH basis as the first basis in their own subcluster but different than the first basis of the main cluster. These clusters can be detected and separated from their parent cluster after MH bases of the parent cluster is computed. To do so, we introduce a two-step algorithm to get the correct generic MH bases $\bar{\Delta}$. In the first pass, we divide the vertices into clusters according to the joints having skinning weights larger than the threshold. Then, if necessary, we divide these clusters into the subclusters according to their $\tilde{H}$ values (Algorithm 10).

Algorithm 10 Two-step algorithm to obtain the clustered generic MH bases $\bar{\Delta}$

```
 1: Cluster the vertices according to the threshold.
 2: for each reference pose do
 3:     Pose the mesh with the current reference pose
 4:     Calculate $L$ and $\star_0$ for the whole mesh
 5:     for each cluster do
 6:         Set the related information and calculate the generic bases $\tilde{H}$
 7:         if the first basis is not uniform (not sharing the same value for the first basis) in the cluster then
 8:             Gather the vertices with the similar first basis and create subclusters
 9:             Add the subclusters into the cluster list
10:             Perform Step 6 for the main cluster
11:         end if
12:     end for
13: end for
```

These subclusters are considered only during the precomputation of the frequency-space coefficients $\tilde{a}$. The linear system (Equation 3.12) is constructed by considering the main clusters, including subclusters. After the correct frequency-space coefficients are calculated for each subcluster, these coefficients are put into correct positions in the frequency-space matrix of the main clusters $\tilde{\mathbf{A}}$, and the linear system is solved to get generic MH bases $\tilde{\mathbf{H}}$ (Algorithm 11).

In Algorithm 11, the first three steps are performed as a precomputation. The frequency-space coefficients of the current pose $\tilde{a}$ is required to perform Step 6. Thus, the frequency-space coefficients are needed to calculate for the pose before Step 6 is performed if this information is not available beforehand.

Algorithm 11 Parameterization with MH bases

```
1: Perform the two-step clustering (Algorithm 10)
2: Fill $\mathbf{A}$ and $\tilde{\mathbf{A}}$ with the correct values
3: Solve the linear system (Equation 3.12) for each cluster to get $\tilde{\mathbf{H}}$
4: for each pose do
5:     for each cluster do
6:         Compute ambient occlusion using Equation 3.13
7:     end for
8: end for
```

Since manifold harmonics parameterization focuses on pure geometry of the mesh and extends it with the ground-truth ambient occlusion, it is the most accurate approach to approximate ambient occlusion (Figure 3.13). Ambient occlusion can be approximated with a 10^{-30} error rate, and this approach does not fail at complex movements (Figure 3.14). Also the constant value λ to capture AO independently of any actual animation parameters is eliminated. Even for clusters with few vertices, this constant parameter does not have a crucial role as it does with joint angles, joint positions, or vertex positions parameterizations. Memory requirements of MH parameterization are lower than clustered vertex position parameterization; however, it is larger than the joint angles or joint position parameterizations, e.g., 144 MB is required for a 15410 vertex mesh.

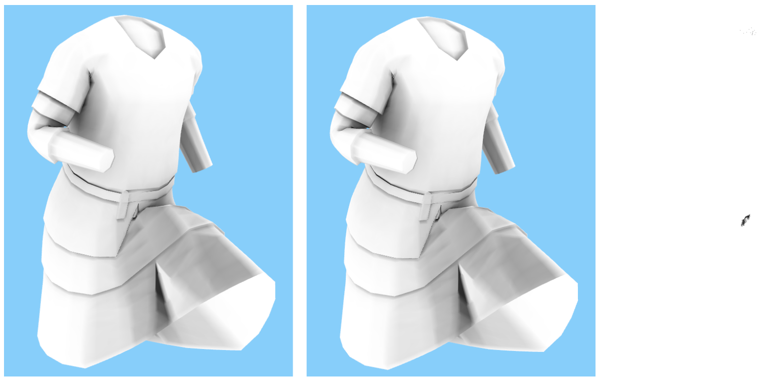


Figure 3.13: Ground truth (left), MH bases parameterization approximation (middle), 100 times magnified comparison (right). Running forward and backward animations are learnt, running forward is approximated.

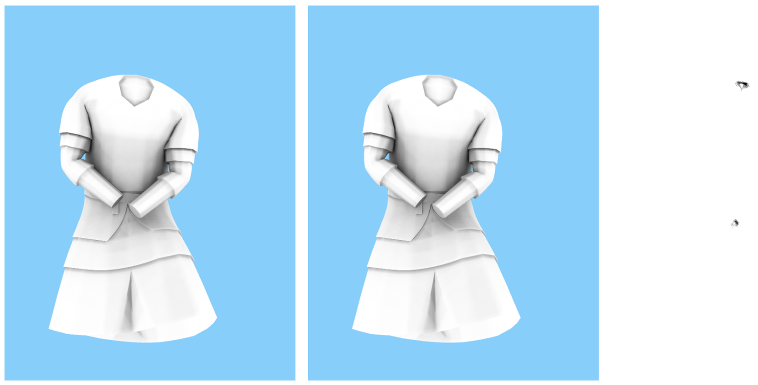


Figure 3.14: MH approach does not fail in complex motions. Ground truth (left), MH bases parameterization approximation (middle), and 100 times magnified difference.

CHAPTER 4

RESULTS AND DISCUSSION

We tested our parameterization techniques using five different skeletal animations on five different skinned characters. The animation data and some of the meshes come from Tarini et al. [15]. We use the precomputed per-vertex ray-traced ambient occlusion as ground truth. We shoot 10000 rays during the precomputation, and we choose max distance (ray length) as 25% the diagonal of bounding volume of the mesh. We also use the precomputed ray-traced ambient occlusion to train the linear systems of equations.

The linear system constructed based on Kontkanen and Aila [6] works well with the motions learnt. Approximation results are close to the ground truth both mathematically and visually as long as the assumptions of Kontkanen and Aila [6] are valid (appearance of the object linearly depends on the motion parameters, and the motion parameters can be handled independently). However, generalizations of this system (approximation parameters) to any kind of random motion requires learning a large number of motion examples with many different aspects as any other machine learning techniques. Otherwise, the generalization fails at poses never treated before.

Correct approximation results also depend on the mesh geometry. If the mesh is not well generated (broken parts, geometrically wrong tessellation, etc.), and/or the animations produce incorrect skinning results (such as self-intersections), then the approximation parameters cannot be extracted correctly since the ground-truth values (ray-traced ambient occlusion) would not be correct, and approximation results are then prone to errors. The more the system learns these error-prone reference poses, the more it gets confused. Therefore, the errors in the approximation results (both visual and error rate) increase and even the approximation of the motions learnt gets affected. These kinds of problems are common in real-world applications, such as video games since low-polygon meshes with problematic geometry and few animation frames that may lead to error-prone skinning results, are used to decrease memory requirements.

We experiment and verify this situation by using a mesh with a poor geometry and animations that include self-intersections that actually come from a video game [15]. During the experiment, we only increase the tessellation of the mesh, and "running" (forward and backward), "dying", and "striking" animations are learnt. Animations of "striking", "dying" (complex motion), and "walking" (unknown motion) are approximated.

Kontkanen and Aila [6] tackle this problem by generating their own "valid" reference pose set, and our results supported their method and their findings. Approximation results of our approach can be increased either by eliminating invalid motions (such as self-intersections) or fixing the mesh geometry.

Our parameterization with joint angles is the same as the parameterization from Kontkanen and Aila [6]. As long as their assumptions are satisfied, joint angle parameterization can approximate ambient occlusion with a precision of 10^{-13} (Figure 3.1). However, geometry parts such as the armpits or situations such as the shadow casted by the hand on the torso are problematic cases for this approach. Hence, the error rate increases to a precision of 10^{-4}, and flickering appears in the approximated results. The most complex animation we worked on is the "striking" animation since it breaks many of the assumptions of Kontkanen and Aila [6], such as two arms of the humanoid geometry that cast shadows on the torso. For this animation, joint angle parameterization generates visible artifacts even if all the poses of the animation are learnt and approximated (Figure 3.2).

Parameterization with joint angles is also affected by the mesh geometry and invalid poses (Figures 4.1 and 4.2). A result of the generalization of the joint angle approximation can also be seen in Figure 4.3. Error rates are listed in Table 4.I.

Animation sequence	Max error	Min error	Mean of MSE (all frames)	Variance of MSE (all frames)
Dying	4.0975×10^{-3}	5.4389×10^{-4}	1.6232×10^{-3}	5.1312×10^{-7}
Striking	4.0846×10^{-3}	2.0865×10^{-4}	9.1364×10^{-4}	4.214×10^{-7}
Walking	3.3542×10^{-3}	5.9681×10^{-4}	1.4838×10^{-3}	4.2904×10^{-7}

Table 4.I: MSE of the "striking", "dying", and "walking" animations with joint angle parameterization using a mesh with a poor geometry. The "running" (forward and backward), "dying", and "striking" animations are learnt.

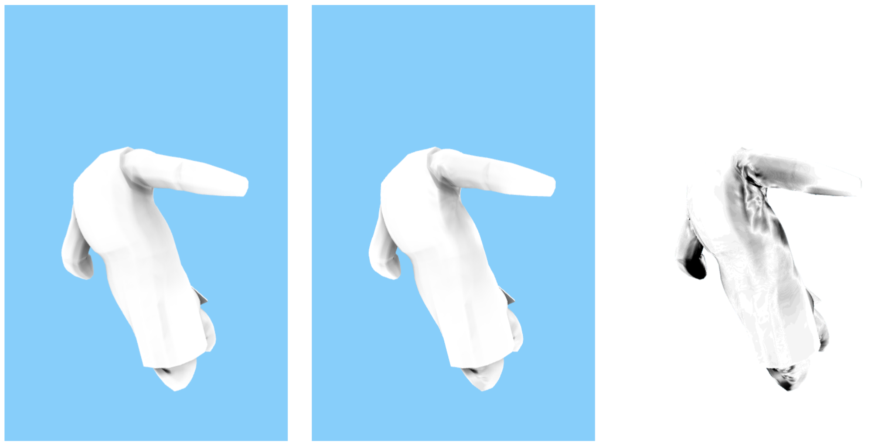


Figure 4.1: Approximation of joint angle parameterization for the "dying" animation with a poor mesh. Ground truth (left), joint angle approximation (middle), and 10× magnified difference (right).

Even though memory requirements of joint angle parameterization is low, investigating and/or applying data compression methods can make joint angle parameterization more useful for highly tessellated meshes or scenes consisting multiple meshes.

The parameterization with clustered vertex positions produces visually similar results as the joint angle parameterization. However, it tends to fail more than the joint angle parameterization when the assumptions are not held (Figures 4.4 and 4.5). Vertices breaking the assumptions generate higher error rates and this also increases variance in mean squared error (MSE) of the clustered vertex position parameterization.

Clustered vertex position parameterization can be integrated into the current skinning pipelines. In this way, ambient occlusion can be approximated during the skinning process by multiplying the skinned vertex position with the approximation parameters. However, the biggest obstacle to its integration is the high memory requirements of this approach. Even though SVD experiments showed that more than half of the parameters can be eliminated, this may not be enough for highly tessellated meshes or complex scenes with multiple meshes. Therefore, the further elimination of parameters along a data compression technique can be investigated as future work.

The generalization of clustered vertex position parameterization follows in a similar fashion with joint angle parameterization. If the system does not cover the "valid"

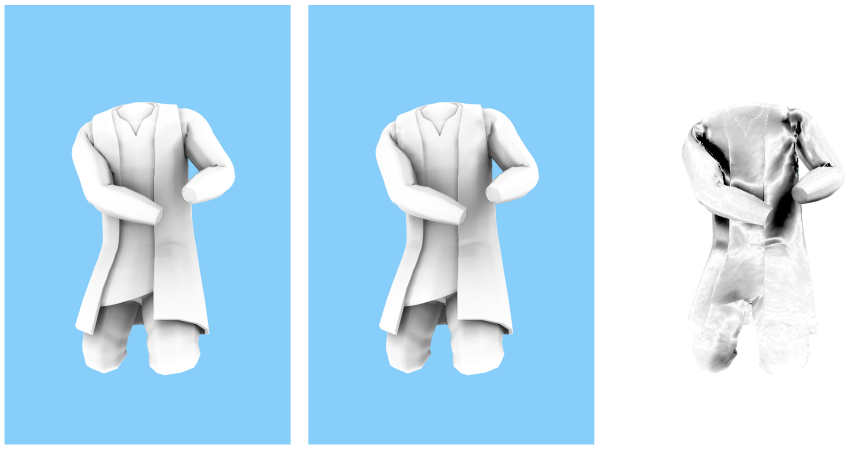


Figure 4.2: Approximation of joint angle parameterization for the "striking" animation with a poor mesh. Ground truth (left), joint angle approximation (middle) and $10\times$ magnified difference (right).

reference poses capturing most of the possible motion then, clustered vertex position parameterization tends to approximate error prone results for the motions not seen (Figure 4.6). Error rates can be seen in Table 4.II.

Animation sequence	Max error	Min error	Mean of MSE (all frames)	Variance of MSE (all frames)
Dying	7.4977×10^{-3}	1.4174×10^{-3}	3.1699×10^{-3}	2.1491×10^{-6}
Striking	4.684×10^{-3}	3.4522×10^{-4}	1.7549×10^{-3}	1.4655×10^{-6}
Walking	3.8607×10^{-2}	2.4583×10^{-2}	0.0304	2.1944×10^{-5}

Table 4.II: MSE of the "striking", "dying", and "walking" animations for clustered vertex position parameterization using a mesh with a poor geometry. "running" (forward and backward), "dying", and "striking" animations are learnt.

We only considered visual errors of joint position parameterization for a complex motion (the "striking" animation), and did not perform any detailed investigation since it could not beat either parameterization with joint angles nor parameterization with vertex points.

The parameterization with manifold harmonics bases produces the best results for the approximation of the motions learnt (Figure 3.13). The ground truth can be approximated with an error in the 10^{-30} range, and only clusters with few vertices (such as five

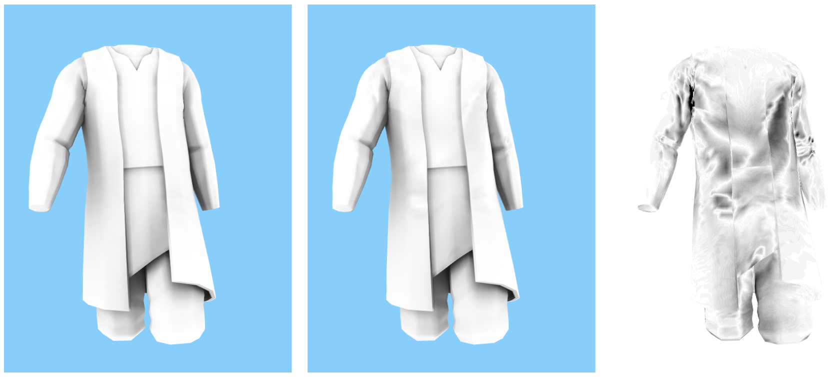


Figure 4.3: Approximation of unknown (not in the training set), the "walking" animation with joint angle parameterization. Ground truth (left), joint angle approximation (middle), and $10\times$ magnified difference (right). Regularization parameter $\lambda = 0.001$.

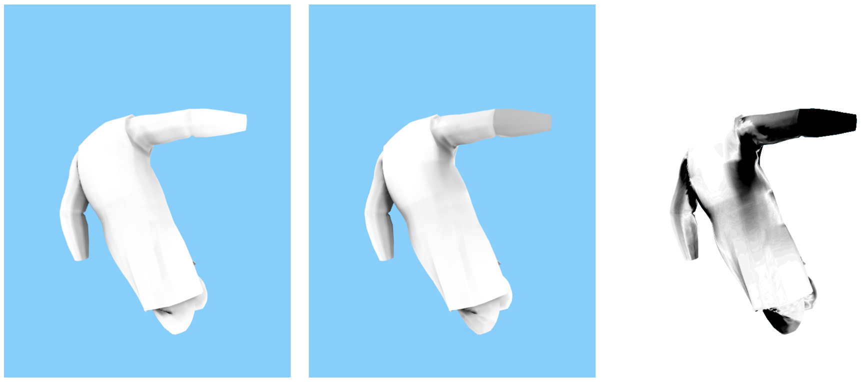


Figure 4.4: Approximation of clustered vertex position parameterization for the "dying" animation with a poor mesh. Ground truth (left), clustered vertex position parameterization (middle), and $10\times$ magnified difference (right).

vertices) produce approximation with a precision of 10^{-5} even for a motion sequence that does not satisfy the assumptions (Figure 3.14). The MH bases parameterization extracts the approximation parameters based on the ground-truth values and the mesh's own geometry. Thus, MH bases approximation may be affected more by poor mesh geometry. As a result, relatively small clusters (e.g., clusters with less than 200 vertices) may cause some visible flickering (because of the precision of 10^{-5}) even though other clusters are approximated with an error in the 10^{-30} range without any problems (Figures 4.7 and 4.8).

On the other hand, MH bases approximation cannot be generalized as other parame-

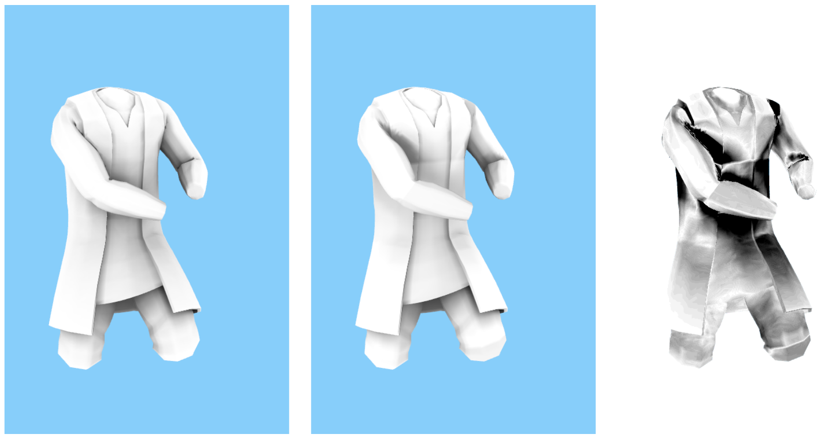


Figure 4.5: Approximation of clustered vertex position parameterization for the "striking" animation with a poor mesh. Ground truth (left), clustered vertex position parameterization (middle), and 10× magnified difference (right).

terization systems by simply learning more reference poses. MH bases parameterization approximates error prone results regardless of increasing the number of the reference poses covering many aspects (Figure 4.9). Increasing the number of the reference poses may help to improve results in some clusters; however, it does not help to improve the precision in the mesh's itself.

The MH bases parameterization requires less memory than clustered vertex parameterization. However, depending on the desired level of quality or the detail of the mesh, some clusters can use fewer parameters to approximate ambient occlusion. In this way, memory requirements of MH bases parameterization can be decreased more, and a data compression technique could be investigated to reduce the memory requirements further.

The MH bases parameterization also requires frequency-space coefficients (animation parameters in parameterization sense) of the current pose for ambient occlusion approximation. These coefficients are not related to the skinning process either, and computing them in real time for arbitrary poses may not be possible. Thus, related frequency-space coefficients should be available before the approximation process. This issue may be solved by also learning the frequency-space coefficients of the reference poses and extracting some approximated parameters for them. Thus, another machine learning technique can be applied to extract generalized frequency-space coefficients

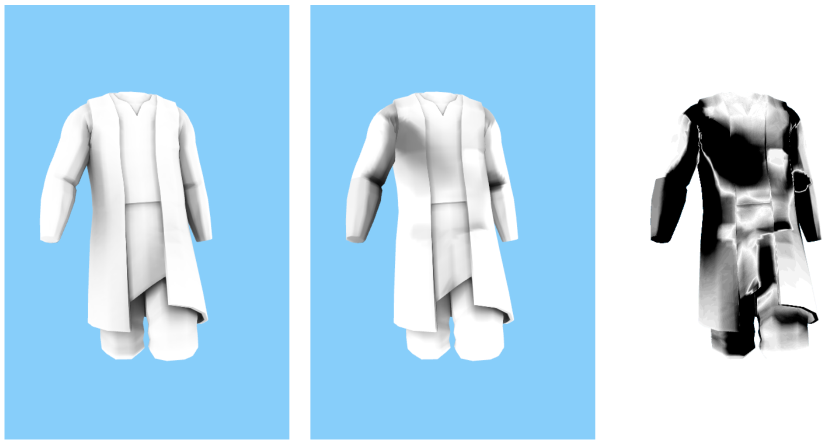


Figure 4.6: Approximation of unknown, the "walking" animation with clustered vertex position parameterization. Ground truth (left), clustered vertex position approximation (middle), and $10\times$ magnified difference (right). Regularization parameter $\lambda = 1000$.

which will be used with the generalized MH bases. However, the effect of using generalized frequency-space coefficients along the generalized MH bases on the quality of the rendered images is unknown.

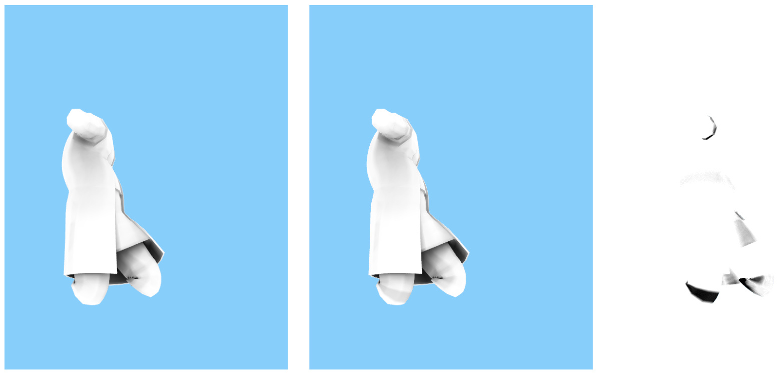


Figure 4.7: Approximation of MH bases parameterization for the "dying" animation with a poor mesh. Ground truth (left), MH bases approximation (middle), and 10× magnified difference (right).

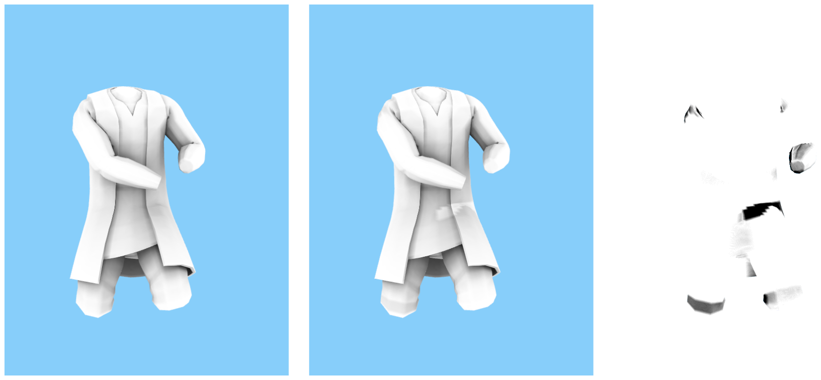


Figure 4.8: Approximation of MH bases parameterization for the "striking" animation with a poor mesh. Ground truth (left), MH bases approximation (middle), and 10× magnified difference (right).

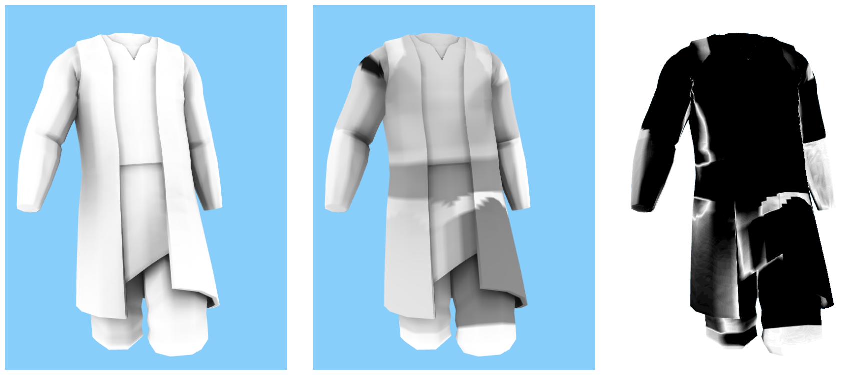


Figure 4.9: Approximation of unknown, the "walking" animation with MH bases parameterization. Ground truth (left), MH bases approximation (middle), and 10× magnified difference (right). Regularization parameter $\lambda = -1000$.

CHAPTER 5

CONCLUSION AND FUTURE WORK

We have presented a technique that approximates appearance of skeletal animated skinned characters by "learning" motion-appearance data using parameterization techniques of the mesh space based on joint angles and positions, vertex positions, or manifold harmonics bases. We tested our technique on ambient occlusion. Our approximation and parameterization techniques can approximate the ground-truth values with small error rates, and the quality of the results can be extended by learning all "valid" possible motions. This also helps to extend our technique to arbitrary motions.

The linear system we construct for learning motion-appearance pairs and the approximation parameters extracted can experience difficulties to approximate complex geometries and geometric configurations, such as an armpit and shadowing that depend on many animation parameters, the shadows cast by a hand of the character on his torso, etc. Additionally, geometry of the mesh may cause some problems in the approximation results, e.g., mesh discontinuities and self-intersections are causes for failed results.

The parameterization with manifold harmonics bases approximates ground truth directly and does not fail as much as other parameterization techniques to approximate complex geometries or motions because mesh geometry and ground-truth appearance data are parameterized together. However, geometrical problems in the mesh and self-intersections can still cause problems. The parameterization with vertex positions tends to fail more frequently at complex motions and the data requirement is higher than other parameterization techniques; however, it can be plugged directly into current skinning pipelines and the appearance of the meshes can be approximated during the skinning process.

Even though the parameterization with joint angles fails less than the parameterization with vertex positions for complex motions and geometries, results are not as good as the parameterization with manifold harmonics.

Improvements in the learning algorithm should be investigated to minimize the error-prone approximation results, especially with complex motions or geometries. In this way, the training of the system should be performed without learning every possible motion to improve rendering results. Indeed, we need to have a better understanding about what a given geometric configuration brings to the system of equations to analyze its impact, and maybe better selection of the proper poses with impact on appearance.

Ambient occlusion results prove promising to extend our technique to other appearance phenomena. Since ambient occlusion is view-independent, diffuse shading, changes in colors, subsurface scattering, etc., can be the next steps of the investigation. Additionally, there is a prior work about approximating global illumination based on precomputed radiance transfer (PRT) and bases of the Laplace operator. Based on the similarities these approaches share with our MH bases parameterization, an investigation should be performed about global illumination approximation. Also, other animated meshes without a standard skeleton like deforming under gravity, motion or muscle actions, or using a skeleton differently like for facial animations could generalize our approach.

A study about directions could also be important for extending our technique. In this way, our approach can be extended to caustics (incident light direction) or specular shading (light and view directions), transparency and refraction (view direction). This notion of directions also brings other possible applications to surface orientations to model and animate surface features such as realistic fur, hair, etc.

Approximation of physically based animations is another interesting avenue. A linear system on physically based animated vertex positions and joint angles can be constructed, and by extracting parameters, realistic vertex positions can be approximated. This idea can be applied to deformation of muscles, fur, clothes, etc.

With all those applications, maybe even combined memory requirements of our approach would quickly become a major hurdle, especially for scenes consisting of multiple meshes, such as in video games. Data compression of the approximation parameters is critical to the acceptance of our approach in the real-time scenarios where it could apply. Transferring the learned approximations to procedural models could be an another route to alleviate the large memory requirements.

BIBLIOGRAPHY

[1] Cheng-Hao Chen, I-Chen Lin, Ming-Han Tsai, and Pin-Hua Lu. Lattice-based skinning and deformation for real-time skeleton-driven animation. In *Proceedings of the 2011 12th International Conference on Computer-Aided Design and Computer Graphics*, CADGRAPHICS '11, pages 306–312. IEEE Computer Society, 2011. ISBN 978-0-7695-4497-7. doi: 10.1109/CAD/Graphics.2011.41. URL `http://dx.doi.org/10.1109/CAD/Graphics.2011.41`.

[2] Keenan Crane, Fernando de Goes, Mathieu Desbrun, and Peter Schroder. Digital geometry processing with discrete exterior calculus. In *SIGGRAPH courses*. ACM, 2013.

[3] Alec Jacobson, Zhigang Deng, Ladislav Kavan, and JP Lewis. Skinning: Real-time shape deformation. In *SIGGRAPH courses*. ACM, 2014.

[4] Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. Geometric skinning with approximate dual quaternion blending. *ACM Trans. Graph.*, 27(4):105, 2008.

[5] Adam G. Kirk and Okan Arikan. Real-time ambient occlusion for dynamic character skins. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, I3D '07, pages 47–52. ACM, 2007. ISBN 978-1-59593-628-8. doi: 10.1145/1230100.1230108. URL `http://doi.acm.org/10.1145/1230100.1230108`.

[6] Janne Kontkanen and Timo Aila. Ambient Occlusion for Animated Characters. In Tomas Akenine-Moeller and Wolfgang Heidrich, editors, *Symposium on Rendering*. The Eurographics Association, 2006. ISBN 3-905673-35-5. doi: 10.2312/EGWR/EGSR06/343-348.

[7] Paul G. Kry, Doug L. James, and Dinesh K. Pai. Eigenskin: Real time large deformation character skinning in hardware. In *Proceedings of the 2002 ACM*

SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '02, pages 153–159. ACM, 2002. ISBN 1-58113-573-4. doi: 10.1145/545261.545286. URL `http://doi.acm.org/10.1145/545261.545286`.

[8] Bruno Lévy and Hao (Richard) Zhang. Spectral mesh processing. In *SIGGRAPH courses*, pages 8:1–8:312. ACM, 2010. ISBN 978-1-4503-0395-8. doi: 10.1145/1837101.1837109. URL `http://doi.acm.org/10.1145/1837101.1837109`.

[9] JP Lewis, Matt Cordner, and Nickson 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, pages 165–172. ACM Press/Addison-Wesley Publishing Co., 2000. ISBN 1-58113-208-5. doi: 10.1145/344779.344862. URL `http://dx.doi.org/10.1145/344779.344862`.

[10] Nadia Magnenat-Thalmann, Richard Laperrière, and Daniel Thalmann. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings of Graphics Interface '88*, pages 26–33. Canadian Information Processing Society, 1988. URL `http://dl.acm.org/citation.cfm?id=102313.102317`.

[11] Alex Mendez-Feliu and Mateu Sbert. From obscurances to ambient occlusion: A survey. *Vis. Comput.*, 25(2):181–196, January 2009. ISSN 0178-2789. doi: 10.1007/s00371-008-0213-4. URL `http://dx.doi.org/10.1007/s00371-008-0213-4`.

[12] Alex Mohr and Michael Gleicher. Building efficient, accurate character skins from examples. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, pages 562–568. ACM, 2003. ISBN 1-58113-709-5. doi: 10.1145/1201775.882308. URL `http://doi.acm.org/10.1145/1201775.882308`.

[13] Nadine Abu Rumman and Marco Fratarcangeli. *State of the art in skinning tech-*

niques for articulated deformable characters, pages 200–212. SciTePress, 2016. ISBN 9789897581755.

[14] Olga Sorkine. Laplacian Mesh Processing. In Yiorgos Chrysanthou and Marcus Magnor, editors, *Eurographics 2005 - State of the Art Reports*. The Eurographics Association, 2005. doi: 10.2312/egst.20051044.

[15] Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. Accurate and efficient lighting for skinned models. *Computer Graphics Forum*, 33(2):421–428, 2014. ISSN 1467-8659. doi: 10.1111/cgf.12330. URL `http://dx.doi.org/10.1111/cgf.12330`.

[16] Gabriel Taubin. A signal processing approach to fair surface design. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, pages 351–358. ACM, 1995. ISBN 0-89791-701-4. doi: 10.1145/218380.218473. URL `http://doi.acm.org/10.1145/218380.218473`.

[17] Wes Trager. A practical approach to motion capture: Acclaim's optical motion capture system. In *SIGGRAPH course*, 1994.

[18] Bruno Vallet and Bruno Levy. Spectral Geometry Processing with Manifold Harmonics. *Computer Graphics Forum*, 2008. ISSN 1467-8659. doi: 10.1111/j.1467-8659.2008.01122.x.

[19] Wolfram von Funck, Holger Theisel, and Hans-Peter Seidel. Volume-preserving mesh skinning. In *VMV*, 2008.

[20] Hao Zhang. Discrete combinatorial Laplacian operators for digital geometry processing. In *SIAM Conference on Geometric Design, 2004*, pages 575–592. Press, 2004.

[21] Hao Zhang, Oliver van Kaick, and Ramsay Dyer. Spectral mesh processing. *Computer Graphics Forum*, 29(6):1865–1894, 2010.