

EXAMPLE-BASED PARAMETERIZATION OF
LINEAR BLEND SKINNING FOR SKINNING DECOMPOSITION
(EP-LBS)

By

Kayra M. Hopkins

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computer Science — Doctor of Philosophy

2017

ABSTRACT

EXAMPLE-BASED PARAMETERIZATION OF LINEAR BLEND SKINNING FOR SKINNING DECOMPOSITION (EP-LBS)

By

Kayra M. Hopkins

This thesis presents Example-based Parameterization of Linear Blend Skinning for Skinning Decomposition (EP-LBS), a unified and robust method for using example data to simplify and improve the development and parameterization of high quality 3D models for animation. Animation and three-dimensional (3D) computer graphics have quickly become a popular medium for education, entertainment and scientific simulation. In addition to film, gaming and research applications, recent advancements in augmented reality (AR) and virtual reality (VR) are driving additional demand for 3D content. However, the success of graphics in these arenas depends greatly on the efficiency of model creation and the realism of the animation or 3D image.

A common method for figure animation is skeletal animation using linear blend skinning (LBS). In this method, vertices are deformed based on a weighted sum of displacements due to an embedded skeleton. This research addresses the problem that LBS animation parameter computation, including determining the rig (the skeletal structure), identifying influence bones (which bones influence which vertices), and assigning skinning weights (amounts of influence a bone has on a vertex), is a tedious process that is difficult to get right. Even the most skilled animators must work tirelessly to design an effective character model and often find themselves repeatedly correcting flaws in the parameterization. Significant research, including the use of example-data, has focused on simplifying and automating individual components of the LBS deformation process and increasing the quality of resulting animations.

However, constraints on LBS animation parameters makes automated analytic computation of the values equally as challenging as traditional 3D animation methods. Skinning decomposition is one such method of computing LBS animation LBS parameters from example data. Skinning decomposition challenges include constraint adherence and computationally efficient determination of LBS parameters.

The EP-LBS method presented in this thesis utilizes example data as input to a least-squares non-linear optimization process. Given a model as a set of example poses captured from scan data or manually created, EP-LBS institutes a single optimization equation that allows for simultaneous computation of all animation parameters for the model. An iterative clustering methodology is used to construct an initial parameterization estimate for this model, which is then subjected to non-linear optimization to improve the fitting to the example data. Simultaneous optimization of weights and joint transformations is complicated by a wide range of differing constraints and parameter interdependencies. To address interdependent and conflicting constraints, parameter mapping solutions are presented that map the constraints to an alternative domain more suitable for nonlinear minimization. The presented research is a comprehensive, data-driven solution for automatically determining skeletal structure, influence bones and skinning weights from a set of example data. Results are presented for a range of models that demonstrate the effectiveness of the method.

Copyright by
KAYRA M. HOPKINS
2017

This dissertation is dedicated to my great-grandmother Viola McGee Ellis.

“I am my ancestors’ wildest dreams.”

ACKNOWLEDGMENTS

I could not have achieved all of the many accomplishments associated with this dissertation if it were not for the many vibrant and committed communities to which I belong.

I first thank Spelman College, the Spelman CIS program, Dr. Lawrence, Professor Hardnett, Professor Kearse, Dr. Williams, Professor Hale and Dr. Watkins and all of my Spelman sisters for allowing learn and grow without questions, artificial barriers, assumptions or judgment; and to the four other women in the Fab Five (Nicole, Kamika, Candis and Lynn) - thank you for the phone calls, letting me move in when Michigan winters got the best of me, putting up with my DIY projects, and supporting me always.

This research would not be possible without the financial support of the NASA WISE program, the NASA Harriet Jenkins fellowship program, the GEM program, Intel Corporation and the Alfred P. Sloan Foundation. More than funding, these organizations supported my well-being as a whole; to my Jenkins Fellows - you all inspire me!

My Michigan Family (Sloan, BGSA, GEU) - Thank you for putting up with this California girl with wild ideas and even bigger goals; Dr. Dillon, you are the reason I chose State. Thank you for being a strong and positive influence for me and all women in computer science; Dr. Pierre - you helped my start down this path decades before I would even consider graduate school. Your work supporting students in the sciences has had (and still has) great impact and for that I am especially grateful; Dr. O'Kelly, Dr. Stockman - thank you for the wealth of wisdom, resources and motivation provided throughout the years; Teresa Vandersloot - your encouragement and kind words always came at the right moments. Thank you! Linda Moore your smile always reminded me that there is a such thing as #blackgirlmagic. I needed that! Thank you for your support and encouragement; to my advisor, Dr. Owen and

my committee members, Dr. Chai, Dr. Sherry and Dr. Dysken - thank you for your wisdom and insight;

I am eternally grateful and extremely blessed to have gained many lifelong friends while at Michigan State! I would not have survived without three in particular: Annette - my first year would have been miserable without you as my study partner. Thank you for your encouragement and positivity; Dr. Michelle Mwalimu, my travel partner - I am grateful for our friendship...and our many adventures! Melissa Gomes - you are truly the kind of friend that everyone wishes for. Thanks for always having my back!

To my bravest friend, Norah - I am forever grateful to have you on my team! Thank you for your undying support.

To my SPEC family - Thank you for instilling in me success, heritage, excellence, love, togetherness and opportunities....now, I made it y'all!...and you were the first to tell the world that there was no question if I could! To my sister, Dr. Ashley E. Coleman - Thank you for holding my hand, then and now!

My family has been a source of encouragement and strength throughout this process - To my brothers John, Dar and KJ - Your love and pride for me and my work are sustaining forces. I love you always; to my Baba, Steve Swift - No words can express how lucky I am to have your unwavering support and love...and humor (always at just the right moments); most importantly, I acknowledge and must express my deepest gratitude to my mother, Deborah Ann Moffett-Swift, and grandmother, Mary Etta Jones, for their unconditional love and never-ending support. Because of you two phenomenal women, I truly believe I can succeed at anything I put my mind and effort to.

There are numerous others that I have not called by name who have played an important part in this dissertation. Your love and support is eternally appreciated. Thank you!

TABLE OF CONTENTS

LIST OF TABLES	xi
LIST OF FIGURES	xii
LIST OF ALGORITHMS	xxiv
Chapter 1 Introduction	1
1.1 Overview of Current Method and Challenges	3
1.2 Presented Approach	7
1.2.1 Objectives	7
1.3 Outline of Approach	8
1.4 Contributions of this Thesis	9
1.5 Thesis Structure	10
Chapter 2 Background: Character Models and Linear Blend Skinning . .	13
2.1 Character Model	13
2.1.1 Character Mesh	15
2.1.2 Character Skeleton	17
2.2 Deformation Methods	18
2.2.1 Deformation and Character Animation	20
2.3 Skeletal Animation Subprocesses	20
2.3.1 Rigging	21
2.3.2 Skinning	21
2.3.3 Skinning and Rigging Challenges	21
2.4 Linear Blend Skinning	22
2.4.1 Nomenclature and Notation Conventions	23
2.4.2 Coordinate Systems	24
2.4.3 Linear Blend Skinning Equation	25
2.4.4 Rotation Representations	25
Chapter 3 Related Works	29
3.1 Variations on Linear Blend Skinning	29
3.1.1 Motivations for LBS Variations	30
3.2 Skinning: Computing Weights	32
3.3 Rigging: Computing Skeletal Structure and Bone Rotations	38
3.3.1 Rotation Representations	39
3.3.2 Skeleton Extraction	40
3.3.2.1 Medial Axis Skeleton Extraction Methods	40
3.3.2.2 Clustering and Segmentation Methods for Skeleton Extraction	44
3.3.2.3 Summary of Skeleton Extraction Methods	46

3.3.3	Skeleton Embedding and Rigging Transfer	46
3.3.4	Cages, Exoskeletons and Other Skeletal Abstractions	51
3.4	Example-based Linear Blend Skinning	51
3.4.1	Example-based Shape Deformation	53
3.4.1.1	Interpolation	53
3.4.1.2	Template and Data Fitting Techniques	57
3.4.2	Skinning Decomposition	57
3.4.2.1	This Research and Popular Skinning Decomposition Methods	69
Chapter 4	Example-based Parameterization of Linear Blend Skinning . .	71
4.1	Process Overview	72
4.2	Development of Methods	72
4.2.1	Modified Linear Blend Skinning	74
4.2.2	Parameter Mapping for Constraint Adherence	75
4.2.2.1	Implied Bounds from Competing Constraints	76
4.2.2.2	Weight Mapping	77
4.2.2.3	Rotation Mapping	82
4.2.3	Iterative Motion-based Clustering for Initialization	85
4.2.3.1	Representing High-dimensional Data Sets as Vectors	86
4.2.3.2	Cosine similarity for Vector Comparison	88
4.2.3.3	Converting Translation Vectors for Cosine Similarity Comparison	89
4.2.3.4	Basic Initialization	90
4.2.3.5	NNLS and Iterative Clustering for EP-LBS Initialization	91
4.3	Implementing Example-based Parameterization of Linear Blend Skinning	94
4.3.1	Pose Nomenclature	94
4.3.2	Objective Function: Modified Linear Blend Skinning	95
4.3.3	Initialization: NNLS + Iterative k -means Clustering	96
4.3.4	Optimization & Constraint Adherence: Gradient Descent with Parameter Mapping	98
Chapter 5	Results	102
5.1	Methods of Evaluation	102
5.1.1	Quantitative Analysis	103
5.1.1.1	Model Error	103
5.1.1.2	Execution Time	105
5.1.2	Qualitative Analysis	105
5.1.3	Characteristics of Desired Results	106
5.2	Models	107
5.3	Example-based Parameterization of Linear Blend Skinning Results	108
5.3.1	Model: Cat	112
5.3.2	Model: Lion	119
5.3.3	Model: Snake	126
5.3.4	Model: Horse	131
5.3.5	Model: Woman	139

5.3.6	Model: Elephant	154
5.3.7	Model: Flamingo	162
5.3.8	Model: Camel	170
5.3.9	Model: Dance	178
5.3.10	Model: Jump	186
5.4	Discussion	192
5.4.1	Initialization	194
5.4.2	Optimization and Constraint Adherence	197
5.4.3	Algorithm Complexity	208
5.4.4	Execution Time	209
Chapter 6	Conclusion	211
6.1	Contributions of this Thesis	212
6.2	Applications for this Research	214
6.2.1	Opportunities for Additional Analysis in Related Fields	216
6.3	Future Work	216
APPENDICES	219
	Appendix A Coordinate Systems	220
	Appendix B Weight Mapping	223
	Appendix C Rotation Mapping	225
REFERENCES	226

LIST OF TABLES

Table 2.1:	Symbols and Terminology	24
Table 3.1:	Comparison of Skinning Decomposition approaches. (Abbreviations: NNLS - Non-negative Linear Least Squares, SVD - Singular Value Decomposition, TSVD - Truncated SVD)	68
Table 4.1:	Single-pass Reclustering Sample- Demonstration of improvement in error when a single reclustering pass is used.	93
Table 4.2:	Updated Symbols and Terminology	95
Table 5.1:	Models	109
Table 5.2:	Initialization Results	193
Table 5.3:	Optimization Results	207

LIST OF FIGURES

Figure 1.1:	The process and outcome of binding a skeleton to a character mesh as described by a tutorial on using the COLLADA 3D asset exchange system [109]	3
Figure 1.2:	Motion capture: Liu et al present a method for realtime control of an avatar using motion capture sensors [61].)	7
Figure 1.3:	Example-based Parameterization of LBS for 3D Skinning Decomposition. Given a character mesh and a set of example poses, the examples are divided into training and testing sets. Skinning decomposition is then performed using constraint adherence techniques. This produces the skinning weights and bone transformations needed to generate the new poses. The solution produced is verified using the testing set. Finally, new poses can be created using the skeleton and weighting structure computed during the skinning decomposition process . . .	12
Figure 2.1:	Examples of character representations used in this research: (2.1a) A synthesized mesh of 3003 vertices. (2.1b) A skeletal structure of 22 bones. (2.1c) The skeleton embedded in the character mesh. (2.1d) The final rigged, skinned, and texture-mapped character as seen on-screen.	14
Figure 2.2:	Mesh Topology: Synthesized model with uniform topology (2.2a). Reconstructed model with irregular topology (2.2b).	16
Figure 2.3:	Auto-generated character skeleton estimations [7]	17
Figure 2.4:	Spatial Deformation, before (left image) and after deformation (right image): 2.4a point deformation, 2.4b curve deformation, 2.4c surface deformation, and 2.4d volumetric. (Images from Gain and Bechman [30]).	19
Figure 2.5:	Rigging, skinning and animation. Cheng et al present an automated method for determining the skeletal structure and bone influences needed for animation of a humanoid character [18].)	21
Figure 2.6:	Character mesh in relation to bones with weights indicated on vertices. An elbow model in its base pose (left) and after deformation (center and right). All weights for a vertex sum to one indicating 100% influence	22

Figure 3.1:	Skinning is subject to a variety of artifact. Common forms of deformation artifact occur when an object loses volume when bent, an effect known as “pinch” (a) or when an object is rotated, experiencing the “candy-wrapper effect” (b). (Images courtesy of Merry et al [69].)	30
Figure 3.2:	Blender Painting Weights: Monkey head model and Blender interface for manually painting weights. [9].)	37
Figure 3.3:	Dual-quaternion solution (left) to common linear blend skinning unnatural deformation (right) [29].)	39
Figure 3.4:	Skeleton Extraction Methods demonstrated on Armadillo model : 3.4a medial axis (from the works of Tagliasacchi et al [99]), 3.4b cage control structure (from the works of Ju et al [50]), and 3.4c segmentation (as described by Reniers and Telea [82]).	41
Figure 3.5:	Skeleton extraction by mesh contraction by Au et al [5].	43
Figure 3.6:	Single depth Kinect images are used to segment human meshes (Images courtesy of Shotton et al [94].)	45
Figure 3.7:	Seo et al demonstrate the transfer of an anatomically-based rig from a horse to a camel [89].	46
Figure 3.8:	Rigging Pipeline introduced by Baran and Popovic [7].	48
Figure 3.9:	A kangaroo and various skeletal abstractions used by Capell et al [15].	50
Figure 3.10:	A walk sequence generated from interpolation. Highlighted cells contain example motions. All other poses were generated using the verb/adverb interpolation mechanism introduced by Rose et al[85].	54
Figure 3.11:	Example-based Model Generation System: Seo and Magnenat-Thalmann fit a set of example scans to a template model and use interpolation to generate a new model [88].	56
Figure 3.12:	Optimization pipeline for Learn Skeletons for Shape and Pose solution to skinning decomposition [34]. Example poses are analyzed for shape and pose bone hierarchies. The bone hierarchies are combined to generate a single skeletal structure that defines that model. Additionally, skinning weights are computed for the hierarchies and combined to yield the final LBS weights.	61

Figure 3.13:	Smooth Skinning Decomposition with Rigid Bones (SSDR) incorporates block coordinate descent which alternates between solving for either bones (B) or weights (W), holding the other constant while solving [57].	64
Figure 4.1:	Example-based Parameterization of Linear Blend Skinning for 3D Skeletal Animation, a skinning decomposition pipeline process. . . .	73
Figure 4.2:	Plane containing weights w_1 , w_2 , and w_3 that sum to one with the triangular region of plane bounded to the range $[0, 1]$ that represents valid weights.	80
Figure 4.3:	The 2D projection of weights w_1 , w_2 , and w_3 representing the region of three valid weights as determined by only two weights w_1 and w_2	81
Figure 4.4:	1-ring clusters for a single vertex across multiple poses for a lion model. . . .	86
Figure 4.5:	Sample Single-pass Reclustering for Initialization for Woman Model. First pass clustering (left) and after reclustering (right). Note that the legs of the first clustering pass group the lower leg and the foot together. After splitting bones and another pass at clustering, the left foot and lower left leg as well as the right leg and right foot are identified by four separate bones instead of two.	92
Figure 5.1:	Initialization Results for Cat Model - 5.1a Model. 5.1b Initialization Results: Singleton bone groups.	112
Figure 5.2:	Optimization Results for Cat Model - Progress of EP-LBS minimization of the objective function.	113
Figure 5.3:	Optimization Results for Cat Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 01 [left column] and Pose 02 [right column].	114
Figure 5.4:	Optimization Results for Cat Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 03 [left column] and Pose 04 [right column].	115
Figure 5.5:	Optimization Results for Cat Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 05 [left column] and Pose 06 [right column].	116

Figure 5.6:	Optimization Results for Cat Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 07 [left column] and Pose 08 [right column].	117
Figure 5.7:	Optimization Results for Cat Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 09 [left column] and Pose 10 [right column].	118
Figure 5.8:	Initialization Results for Lion Model - 5.8a Model. 5.8b Initialization Results: Singleton bone groups.	119
Figure 5.9:	Optimization Results for Lion Model - Progress of EP-LBS minimization of the objective function.	120
Figure 5.10:	Optimization Results for Lion Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 01 [left column] and Pose 02 [right column].	121
Figure 5.11:	Optimization Results for Lion Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 03 [left column] and Pose 04 [right column].	122
Figure 5.12:	Optimization Results for Lion Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 05 [left column] and Pose 06 [right column].	123
Figure 5.13:	Optimization Results for Lion Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 07 [left column] and Pose 08 [right column].	124
Figure 5.14:	Optimization Results for Lion Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 09 [left column] and Pose 10 [right column].	125
Figure 5.15:	Initialization Results for Snake Model - 5.15a Model. 5.15b Initialization Results: Singleton bone groups.	126

Figure 5.16:	Optimization Results for Snake Model - Progress of EP-LBS minimization of the objective function.	127
Figure 5.17:	Optimization Results for Snake Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 01 [left column] and Pose 02 [right column].	128
Figure 5.18:	Optimization Results for Snake Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 03 [left column] and Pose 04 [right column].	129
Figure 5.19:	Optimization Results for Snake Model - Expected vertex positions [top:(a)], Computed vertex positions [middle:(b)] and Error Heat Map [bottom:(c)] for Pose 05.	130
Figure 5.20:	Initialization Results for Horse Model - 5.20a Model. 5.20b Initialization Results: Singleton bone groups.	131
Figure 5.21:	Optimization Results for Horse Model - Progress of EP-LBS minimization of the objective function.	132
Figure 5.22:	Optimization Results for Horse Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 01 [left column] and Pose 02 [right column].	133
Figure 5.23:	Optimization Results for Horse Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 03 [left column] and Pose 04 [right column].	134
Figure 5.24:	Optimization Results for Horse Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 05 [left column] and Pose 06 [right column].	135
Figure 5.25:	Optimization Results for Horse Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 07 [left column] and Pose 08 [right column].	136

Figure 5.26:	Optimization Results for Horse Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 09 [left column] and Pose 10 [right column].	137
Figure 5.27:	Optimization Results for Horse Model - Expected vertex positions [top:(a)], Computed vertex positions [middle:(b)] and Error Heat Map [bottom:(c)] for Pose 11.	138
Figure 5.28:	Initialization Results for Woman Model - 5.28a Model. 5.28b Initialization Results: Singleton bone groups.	139
Figure 5.29:	Optimization Results for Woman Model - Progress of EP-LBS minimization of the objective function.	140
Figure 5.30:	Optimization Results for Woman Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 01 [left column] and Pose 02 [right column].	141
Figure 5.31:	Optimization Results for Woman Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 03 [left column] and Pose 04 [right column].	142
Figure 5.32:	Optimization Results for Woman Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 05 [left column] and Pose 06 [right column].	143
Figure 5.33:	Optimization Results for Woman Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 07 [left column] and Pose 08 [right column].	144
Figure 5.34:	Optimization Results for Woman Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 09 [left column] and Pose 10 [right column].	145
Figure 5.35:	Optimization Results for Woman Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 11 [left column] and Pose 12 [right column].	146

Figure 5.36:	Optimization Results for Woman Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 13 [left column] and Pose 14 [right column].	147
Figure 5.37:	Optimization Results for Woman Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 15 [left column] and Pose 16 [right column].	148
Figure 5.38:	Optimization Results for Woman Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 17 [left column] and Pose 18 [right column].	149
Figure 5.39:	Optimization Results for Woman Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 19 [left column] and Pose 20 [right column].	150
Figure 5.40:	Optimization Results for Woman Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 21 [left column] and Pose 22 [right column].	151
Figure 5.41:	Optimization Results for Woman Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 23 [left column] and Pose 24 [right column].	152
Figure 5.42:	Optimization Results for Woman Model - Expected vertex positions [top:(a)], Computed vertex positions [middle:(b)] and Error Heat Map [bottom:(c)] for Pose 15 [left column].	153
Figure 5.43:	Initialization Results for Elephant Model - 5.43a Model. 5.43b Initialization Results: Singleton bone groups.	154
Figure 5.44:	Optimization Results for Elephant Model - Progress of EP-LBS minimization of the objective function.	155
Figure 5.45:	Optimization Results for Elephant Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 01 [left column] and Pose 02 [right column].	156

Figure 5.46:	Optimization Results for Elephant Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 03 [left column] and Pose 04 [right column].	157
Figure 5.47:	Optimization Results for Elephant Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 05 [left column] and Pose 06 [right column].	158
Figure 5.48:	Optimization Results for Elephant Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 07 [left column] and Pose 08 [right column].	159
Figure 5.49:	Optimization Results for Elephant Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 09 [left column] and Pose 10 [right column].	160
Figure 5.50:	Optimization Results for Elephant Model - Expected vertex positions [top:(a)], Computed vertex positions [middle:(b)] and Error Heat Map [bottom:(c)] for Pose 11.	161
Figure 5.51:	Initialization Results for Flamingo Model - 5.51a Model. 5.51b Initialization Results: Singleton bone groups.	162
Figure 5.52:	Optimization Results for Flamingo Model - Progress of EP-LBS minimization of the objective function.	163
Figure 5.53:	Optimization Results for Flamingo Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 01 [left column] and Pose 02 [right column].	164
Figure 5.54:	Optimization Results for Flamingo Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 03 [left column] and Pose 04 [right column].	165
Figure 5.55:	Optimization Results for Flamingo Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 05 [left column] and Pose 06 [right column].	166

Figure 5.56:	Optimization Results for Flamingo Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 07 [left column] and Pose 08 [right column].	167
Figure 5.57:	Optimization Results for Flamingo Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 09 [left column] and Pose 10 [right column].	168
Figure 5.58:	Optimization Results for Flamingo Model - Expected vertex positions [top:(a)], Computed vertex positions [middle:(b)] and Error Heat Map [bottom:(c)] for Pose 11.	169
Figure 5.59:	Initialization Results for Camel Model - 5.59a Model. 5.59b Initialization Results: Singleton bone groups.	170
Figure 5.60:	Optimization Results for Camel Model - Progress of EP-LBS minimization of the objective function.	171
Figure 5.61:	Optimization Results for Camel Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 01 [left column] and Pose 02 [right column].	172
Figure 5.62:	Optimization Results for Camel Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 03 [left column] and Pose 04 [right column].	173
Figure 5.63:	Optimization Results for Camel Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 05 [left column] and Pose 06 [right column].	174
Figure 5.64:	Optimization Results for Camel Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 07 [left column] and Pose 08 [right column].	175
Figure 5.65:	Optimization Results for Camel Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 09 [left column] and Pose 10 [right column].	176

Figure 5.66:	Optimization Results for Camel Model - Expected vertex positions [top:(a)], Computed vertex positions [middle:(b)] and Error Heat Map [bottom:(c)] for Pose 11.	177
Figure 5.67:	Initialization Results for Dance Model - 5.67a Model. 5.67b Initialization Results: Singleton bone groups.	178
Figure 5.68:	Optimization Results for Dance Model - Progress of EP-LBS minimization of the objective function.	179
Figure 5.69:	Optimization Results for Dance Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 01 [left column] and Pose 02 [right column].	180
Figure 5.70:	Optimization Results for Dance Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 03 [left column] and Pose 04 [right column].	181
Figure 5.71:	Optimization Results for Dance Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 05 [left column] and Pose 06 [right column].	182
Figure 5.72:	Optimization Results for Dance Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 07 [left column] and Pose 08 [right column].	183
Figure 5.73:	Optimization Results for Dance Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 09 [left column] and Pose 10 [right column].	184
Figure 5.74:	Optimization Results for Dance Model - Expected vertex positions [top:(a)], Computed vertex positions [middle:(b)] and Error Heat Map [bottom:(c)] for Pose 11.	185
Figure 5.75:	Initialization Results for Jump Model - 5.75a Model. 5.75b Initialization Results: Singleton bone groups.	186
Figure 5.76:	Optimization Results for Jump Model - Progress of EP-LBS minimization of the objective function.	187

Figure 5.77:	Optimization Results for Jump Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 01 [left column] and Pose 02 [right column].	188
Figure 5.78:	Optimization Results for Jump Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 03 [left column] and Pose 04 [right column].	189
Figure 5.79:	Optimization Results for Jump Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 05 [left column] and Pose 06 [right column].	190
Figure 5.80:	Optimization Results for Jump Model - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 07 [left column] and Pose 08 [right column].	191
Figure 5.81:	The computed singleton bone groups for the woman model after initialization (5.81a) and the original bones and skeletal structure (5.81b) used to generate the poses. The initial skeleton contained 20 bones, including three bones for the torso. The initialization process computed 18 bones. Because the example poses did not include articulation of each of the torso bones, the initialization step grouped the three bones together as one bone for the entire torso.	194
Figure 5.82:	Lack of variation of head and neck poses leads to clustering the head and neck regions into a single bone group, causing large errors in that region of the model as indicated by the heat map.	196
Figure 5.83:	Incorrect clustering of bones that move in parallel to one another, such as the clustering of the upper hind legs into one bone group (bone 17), leads to higher model area in those areas, as indicated by the corresponding areas of the error heat map being marked hot. . .	197
Figure 5.84:	Expected Bones for Camel Model - The front right leg of the camel is incorrectly clustered into one large bone group. The heat map for pose 02 gives indication of where bones should be.	198
Figure 5.85:	General Patterns in Optimization Results - Most models begin with an initial objective function value near the solution. In a few iterations the EP-LBS optimization process minimizes the objective function value and converges on a near-zero solution.	200

Figure 5.86: Large undefined groups of vertices, such as the torso regions of biped and quadruped models, may include island patches of vertices within a larger bone group or may not identify additional bones in the region, causing errors in the initial clustering that can not be effectively optimized. 206

LIST OF ALGORITHMS

Algorithm 4.1	Basic LBS Initialization	99
Algorithm 4.2	NNLS and Iterative Clustering EP-LBS Initialization	100
Algorithm 4.3	Gradient Descent with Parameter Mapping	101

Chapter 1

Introduction

Three-dimensional (3D) graphics and animated characters have found their way into hearts and homes by way of video games and popular films and have also forged a place in product development and scientific research. The last 20 years has seen great strides in the development and creation of 3D animated characters, which have become increasingly popular and relevant to numerous fields. In 1996, films made from a combination of computer generated imagery (CGI) and live action accounted for less than 5% of the motion picture market share. By 2009, mixed CGI and live action films accounted for more than 20% of the market share [74]. In 1995, *Toy Story* became the first feature-length computer animated film released in theaters. As the lone movie of its kind it brought in 2.83% of the total market share for ticket sales that year [73, 74]. In the the first half of 2016 alone eight, digital animation films were released, capturing 20.75% of the market share [73].

In 2006, auto manufacturers and suppliers set competition aside and joined forces, pledging an initial \$18 million to advance the development of 3D human body models for use in crash-test simulations. In 2009, the feature film *Avatar*, starring CGI main characters, grossed \$73 million in the first week and has since settled comfortably into the number one position for all time grosses of any film worldwide [1, 112]. With the 2015 and 2016 releases of the Oculus Rift, Samsung Gear VR, Sony Playstation VR and HTC Vive VR hardware, in addition to YouTube 360 Video and countless VR content, the virtual reality market will likely see a significant increase in both use and production, demanding more high-quality

3D characters.

Indeed, in the two decades leading to 2016, three-dimensional graphics has had an increased presence and economic impact in a variety of fields, including entertainment, scientific research, manufacturing, development and simulation. In 2013, the computer graphics industry grossed just over \$25 billion annually. The market is projected to grow to \$32.68 billion by 2019 [66]. From starring in the world’s highest grossing film to their role in simulating difficult situations, the societal advances and positive economic impact resulting from enhanced character animation encourage interest and investment in the study and technological advancement of 3D animation. Additionally, advancements in motion capture, computer vision and position tracking have the potential to significantly reduce research overhead and lower film and video game production costs by cutting animation time, while also increasing character realism. Such improvements ultimately allow larger profit margins for the film and games industries which have been experiencing lower profits in the last decade due to the ever-increasing cost of production. Similarly, 3D imaging and modeling can aid in rapid-prototyping and simulate costly or dangerous environments, reducing overall cost of some scientific research.

Although character animation techniques have been the subject of much research and have been widely successful, they still remain a challenging component in developing compelling animation and 3D content. Automated techniques designed to simplify the animation process often increase the computational complexity of the animation algorithm or produce undesired elements within the animation as a result. The use of recorded or observed motion to drive the animation has the potential to create animation with greater realism in less time and with lower cost. However in practice, the creation of 3D character animation, especially animation derived from observed movement, often requires manual adjustments, significant

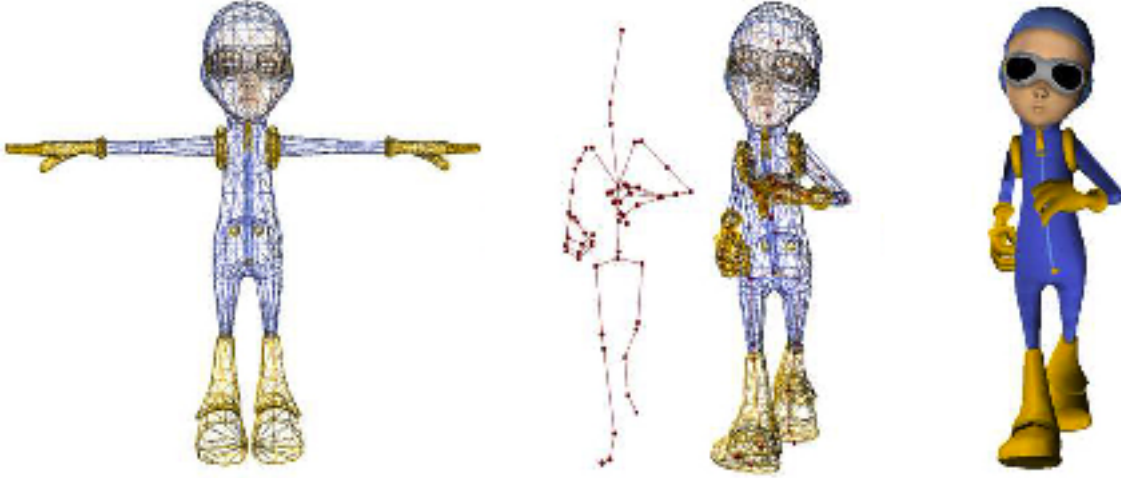


Figure 1.1: The process and outcome of binding a skeleton to a character mesh as described by a tutorial on using the COLLADA 3D asset exchange system [109]

pre- or post-processing or nonintuitive heuristics.

Desirable qualities in 3D animated characters typically must be achieved through the selective trade-off of other coveted features. High-quality characters are created at the expense of time and resources. Larger quantities of characters and animation can be generated, but often at the expense of quality. Significant research has been dedicated to find better solutions to each of these challenges. EP-LBS and the research presented in this thesis aim to reduce the compromises normally required in the 3D animation process and improve the overall process of creating aesthetically-pleasing 3D animated characters generated from observed deformation.

1.1 Overview of Current Method and Challenges

Skeletal animation is the most commonly used method for animating 3D figures. This method assumes an underlying, invisible armature commonly referred to as a *skeleton* that is embedded in a character mesh. The skeleton elicits a weighted deformation of the character

mesh, mimicking the deformation of physical models due to the self-manipulation of skeletal joints. The skeletal structure is associated with the character model and is used to generate the movements of a character.

Skeletal animation facilitates a more efficient animation process by using a simplified skeletal model to determine the movement of the character. The skeletal abstraction greatly simplifies the process of deforming thousands of vertices in a character mesh. Skeletal animation and generating the deformed mesh for a moving character, however, is not a trivial task. Commonly used skeletal animation techniques employ two main processes to associate the skeletal structure with the character model, *rigging* and *skinning*, each of which pose unique challenges.

Rigging is the process of embedding a sufficiently compact skeletal representation of the character into the character mesh. A simple human-like character can have tens of thousands of vertices. Given that character vertices often move in groups, directly computing the independent transformations that define the movement of thousands of vertices, either offline or at 30-75 frames per second can be expensive and redundant. Rigging simplifies the computation of the transformations needed to move a vertex by associating each bone with a collection of vertices that move in a similar fashion. The embedded skeleton is comprised of significantly fewer bones than vertices in the model. By moving one bone, a single transformation can be used to represent the transformation of multiple vertices.

The goal of the rigging process is to determine the positions, orientations, and hierarchical relationships of the bones that make up the skeletal representation. This process is complicated by the fact that these bones are subject to posing and manipulation, effectively dragging associated vertices along with them. Relatively small errors in bone positions or pose orientations can lead to large errors in vertex placement when the model is deformed.

Although rigging simplifies the computation of transformations, rigging is challenging because it is difficult to minimize the number of bones needed to accurately create all of the required movements. When creating the skeletal structure for an animated character, the proper balance between the number of bones and the complexity of the skeletal structure necessary to achieve the possible motions that the skeleton needs to support is essential. A greater number of small bones can vastly increase the number and type of deformations that can be achieved. However, this increased flexibility should be balanced with the increased computational complexity of using more bones and the plausibility of poses that can be created with additional bones. Perfecting this balance is one of the challenges of the animation process.

Skinning associates the vertices of the character mesh with the bones of the skeleton and determines the amount of influence each bone has on a vertex. Skinning takes advantage of the simplified skeletal representation of the character, while still producing the desired deformation. Rather than deforming each of potentially hundreds of thousands of vertices individually, a relatively small number of bones (usually a few dozen) can be transformed, which will in turn deform the associated vertices and produce the desired deformation of the character.

Together the skinning and rigging process determine the bone rotations and bone influence values, referred to collectively as the animation parameters, that can deform a character into new poses. Still, a number of constraints are employed during the skinning process to encourage the production of the desired deformation. The task of finding influence values, or skinning weights, that satisfy the constraints and generate the desired deformation is a considerable challenge in the animation process. Conflicting constraints are perhaps the most significant challenge during the skinning process. Additionally, determining skinning

weights can be counter-intuitive; seemingly valid influence values can result in undesired deformations. As a result, even experienced artists often require multiple iterations of skinning configurations before achieving an acceptable model.

There have been many approaches to improve the quality and efficiency of each of these processes: weight computation (skinning), skeletal structure (rigging), and plausible motion creation (deformation). Many proposed solutions create additional problems and incremental advances in animation and modeling techniques often only partially solve the problem as a result of focusing on only one of the sub-process challenges.

Given a fully skinned and rigged model, motion and animation is created by transforming the invisible bones into new positions, which, in turn, moves the vertices of the character model that is seen on the screen. The process of choosing the new bone configurations is often based on real movement captured using various recording methods including 3D scanning and imaging techniques. Motion capture data has long been used directly to compute bone transformations for new character poses. Imaging techniques such as motion capture, videography, 3D scanning and, more recently, structured light have been used to capture examples for animations and to compute relevant bone transformations. Additionally, such imaging techniques have also been used to automate the computation of bone influence values and aide in other aspects of the skinning and rigging processes.

Although scanning and imaging methods are rich with data that is used to aide in the animation process, much of the data goes unused. Most existing example-based methods using exemplar imagery focus on only one aspect of the animation process, either skinning or rigging. In some cases the example data is used only as an informal guide and is never directly used. However, it is possible to use the data collected using 3D scanning and imaging techniques directly to improve the quality of the resulting deformation and the efficiency of

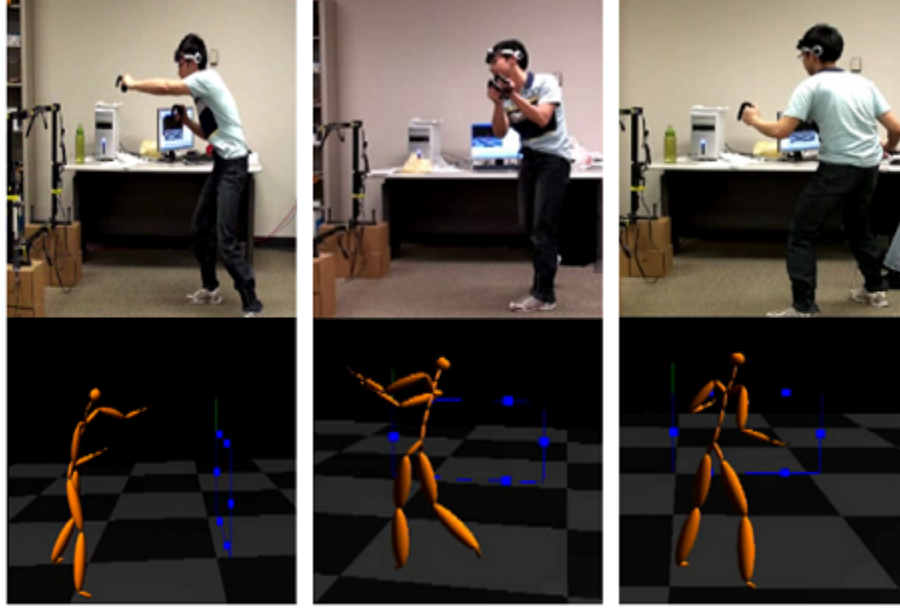


Figure 1.2: Motion capture: Liu et al present a method for realtime control of an avatar using motion capture sensors [61].)

the animation process.

1.2 Presented Approach

This thesis presents Example-based Parameterization of Linear Blend Skinning (EP-LBS), a concise and comprehensive method for direct use of example data to solve rigging and skinning challenges. Using example models with known vertex positions as ground truth, optimization methods are used to simultaneously compute the skeletal structure, weights, and transformations that minimize the difference between the known vertex positions and those determined with the computed animation parameters.

1.2.1 Objectives

The presented EP-LBS method is a simple yet robust solution for skeletal animation based on example data. The desired attributes of a robust solution to the challenges of character

animation include the following:

- ***Automated***: requires little manual adjustment, limited heuristic values, and minimal pre- and post-processing.
- ***Comprehensive***: simultaneously computes both the skinning and rigging parameters.
- ***Constrained***: adheres to standard LBS constraints.
- ***Scalable***: works with high-resolution input models, and large amounts of example data and produces/reproduces high-quality, aesthetically pleasing character animation.

1.3 Outline of Approach

EP-LBS is a comprehensive, data-driven solution to the skinning and rigging processes. The constrained parameterization of LBS animation is refactored and posed as an example-based optimization of a *single objective function*. Composing the rigging and skinning processes as a single objective function addresses the comprehensive quality of the desired solution and formulating LBS parameterization as an optimization problem creates an automated solution requiring minimal processing, manipulation and heuristics. Additionally, an *iterative initialization* process is presented to limit pre-processing and further automate the EP-LBS solution. To address the restrictions on animation parameter values, *parameter mapping* is introduced. Parameter mapping is a method of transforming competing constraints into simple bounding conditions and mapping complex and non-linear constraints to a set of linearly constrained free-variables for increased constraint adherence during example-based skeletal animation. EP-LBS statistical results on a range of example models demonstrate that this method is scalable from modest models of a few hundred vertices, a dozen bones

and limited poses, to high-resolution models of thousands of vertices captured in dozens of poses. The results of this research demonstrate that it is both possible and advantageous to solve for the parameters of the entire animation process in a single unified equation, such that both the skinning and rigging parameters are determined simultaneously. Figure 1.3 provides a visual representation of Example-based Parameterization of LBS for Skinning Decomposition.

1.4 Contributions of this Thesis

The presented EP-LBS method allows an artist to be more efficient in creating animated characters in a data-driven environment. This method has the advantage of being a comprehensive, automated and scalable solution to the challenges and constraints of 3D character animation. This robust EP-LBS solution has implications in many 3D animation fields including scientific modeling, motion pictures, and console gaming.

The primary contributions of this thesis are summarized as follows:

- ***Modified LBS Model***: The standard LBS mathematical model is re-factored to represent rotations using a mixture of quaternions and Euler angle vectors instead of larger nine-element rotation matrices. Quaternions represent the large-scale orienting rotations used to approximate the gross rotation needed to transform the base model to a given example deformation. Euler angle rotations are used for the refined iterative rotation alignment computed during the optimization process.
- ***Parameter Mapping for LBS Constraint Adherence***: Interdependent LBS parameter constraints are mapped to alternate domains better suited to nonlinear

minimization. The benefit of this method is LBS constraint adherence at minimal computational cost.

- ***Unified Least-Squares Non-linear Optimization for LBS*** : Given a set of example poses, the modified LBS model is formulated into a unified least-squares objective function that is minimized to solve for both skinning weights and joint transforms. Parameter mapping and gradient descent are used to minimize the unified objective function. The result is a comprehensive example-based non-linear optimization solution for automated computation of all LBS parameters, simultaneously.
- ***Iterative Motion-based Clustering for Automated LBS Initialization***: Beginning without knowledge of the skeletal structure of a model, motion-based clustering is used to compute initial transforms which are used with non-negative least-squares methods to compute initial weights. This process is executed iteratively, each time adding bones to areas with large error, until the model converges on a basic skeletal structure and vertex-bone associations that yield minimal reconstruction error. The result is an automatic estimation of joints and bone influence for a model requiring no a priori knowledge of the model’s skeletal structure.

1.5 Thesis Structure

This thesis provides the background information that serves as the foundation for EP-LBS, the derivation of EP-LBS processes, a detailed explanation of evaluation methods used to reach the conclusions and results to support the findings. The remainder of this document proceeds as follows: Chapter 2 presents the deformation and skeletal animation background that provides context for this problem, including defining the linear blend skinning algo-

rithm and the mathematical representation of vertex deformation used in this research. Chapter 3 provides a literature review and summary of related research. Topics covered in related works include data-driven LBS techniques, example-based animation methods and constraint-handling procedures. Chapter 4 details the development of a modified LBS model. Theories of related works and the modified LBS model are combined to present a new, comprehensive example-based skeletal animation process, EP-LBS. Additionally, the main contributions of the EP-LBS research, which include iterative motion-based LBS initialization, parameter mapping and the unified non-linear squares optimization solution for skinning decomposition, are detailed. Chapter 5 presents evaluation methods and results and Chapter 6 concludes with the details of the findings of this research and suggestions for related study and future research.

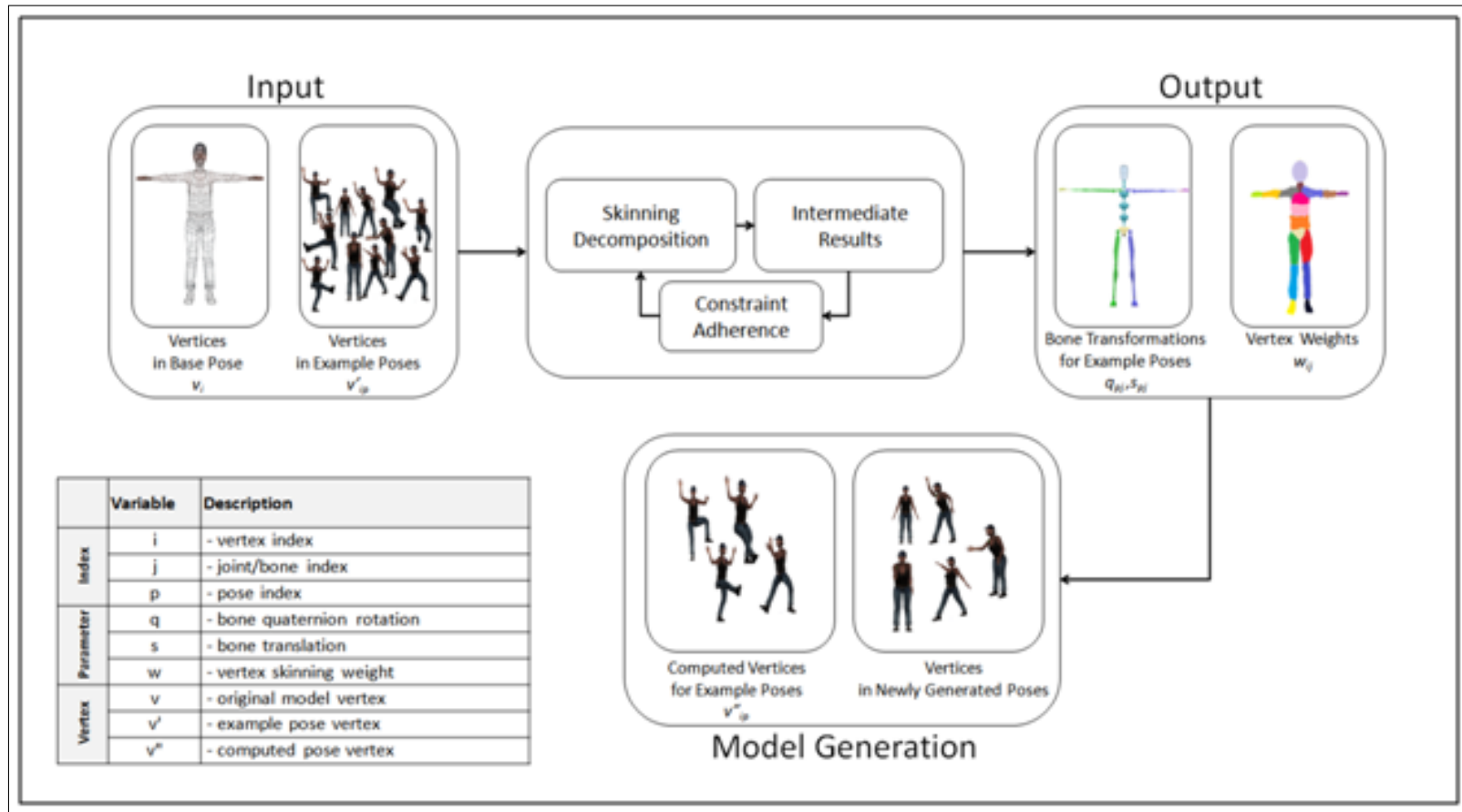


Figure 1.3: Example-based Parameterization of LBS for 3D Skinning Decomposition. Given a character mesh and a set of example poses, the examples are divided into training and testing sets. Skinning decomposition is then performed using constraint adherence techniques. This produces the skinning weights and bone transformations needed to generate the new poses. The solution produced is verified using the testing set. Finally, new poses can be created using the skeleton and weighting structure computed during the skinning decomposition process

Chapter 2

Background: Character Models and Linear Blend Skinning

Animation is the process of presenting a series of character deformations sequentially on-screen to create the illusion of motion. Therefore accurate deformation is an integral component of high-quality animation. Character deformation is often accomplished using skeletal animation, which can be modeled mathematically with Linear Blend Skinning (LBS) [71]. This chapter defines the components of a character model and introduces the mathematical implementation of the Linear Blend Skinning method for skeletal animation of a character model.

2.1 Character Model

In the simplest terms, a 3D character is composed of a mesh of interconnected vertices and an embedded skeleton (as shown in Figure 2.1). The *skeleton* and *mesh* are symbolic abstractions of the on-screen character. These symbolic constructs also have a mathematical representation that is used to implement the Linear Blend Skinning algorithm for deformation. Both the mathematical and metaphorical representations are analyzed here to provide a broad introduction to skeletal animation.

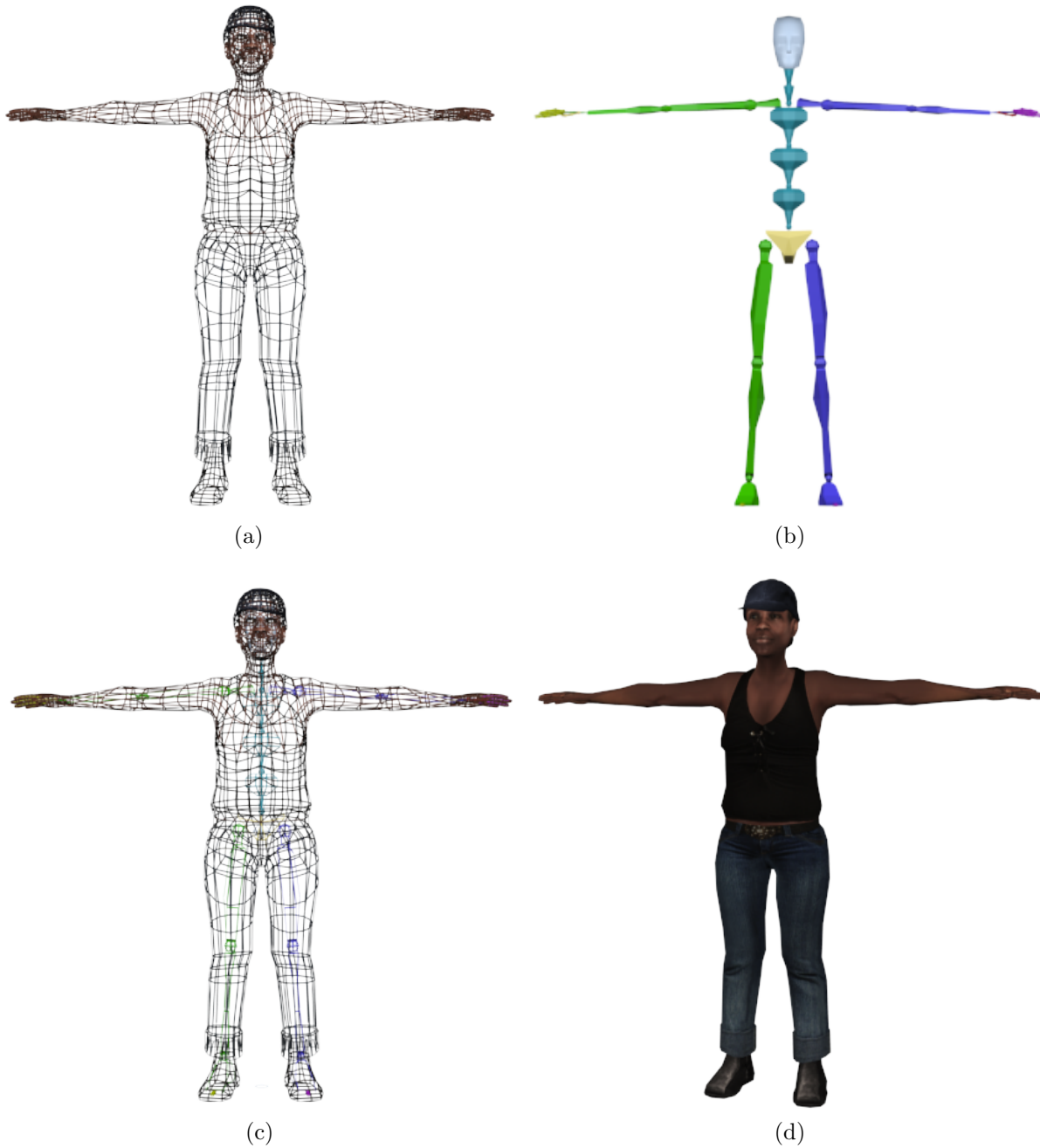


Figure 2.1: Examples of character representations used in this research: (2.1a) A synthesized mesh of 3003 vertices. (2.1b) A skeletal structure of 22 bones. (2.1c) The skeleton embedded in the character mesh. (2.1d) The final rigged, skinned, and texture-mapped character as seen on-screen.

2.1.1 Character Mesh

A character or model is represented on-screen as a collection of points in space referred to as vertices and a defined geometry incident on these vertices, typically triangles or larger polygons. The structure of vertices and polygons is referred to as the character mesh. Any polygon may be used for the geometry, however triangles are typically preferred because graphics hardware is typically optimized for fast and efficient rendering of triangles. The spatial relationships between the vertices and polygons of the character mesh form the topology of the model. Topology refers to a variety of geometric properties of the character mesh, such as vertex connectivity, uniformity, resolution, edgeloops and polygonal subdivisions, that remain unchanged by deformations and continuous transformation such as bending, twisting and stretching.

There are two basic methods for creating the vertices and geometry of a character mesh: *synthesis*, in which vertices and polygons are fully generated, or synthesized, by an artist or with the help of some software package, and *reconstruction*, where vertices and polygons are reconstructed from a tangible model via scanning or imaging techniques. The greatest difference between synthesized and scanned meshes is the resulting topology of the model. The creation of the character mesh often requires compromise between quality and simplicity. Synthesized models often have uniform topology resulting in a simplified model compared to the irregular topology and higher resolution often found in scans generated with model reconstruction.

The resolution of a mesh, which refers to the number of vertices per unit area, strongly influences the on-screen appearance of a character and the types of deformation that can be modeled for that character. Models with a higher resolution are able to display greater detail,

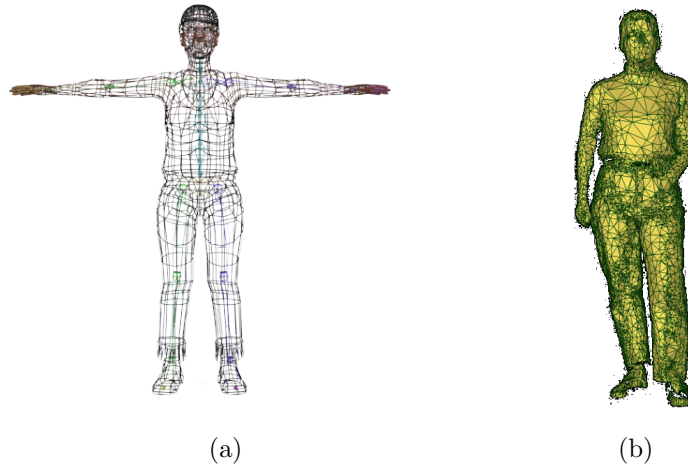


Figure 2.2: **Mesh Topology:** Synthesized model with uniform topology (2.2a). Reconstructed model with irregular topology (2.2b).

and appear more vivid on screen. Reconstructed models often generate higher resolution character meshes, although the increased number of vertices may be misplaced or otherwise malformed during the reconstruction process.

Characters of 5,000-7,000 vertices are standard for desktop gaming applications. Character models on mobile devices typically have much fewer vertices (50-70% fewer vertices) as a result of hardware limitations common in these devices. Character models used in film demand much higher resolution, 20,000-50,000 vertices more or more. Additionally, high resolution models may be created initially, with deformation performed on some reduced-resolution version. Increased resolution, however, comes at increased computational expense. Although skeletal animation techniques use a skeleton abstraction to decrease the complexity of computing vertex displacements, the number of vertices in a model remains the determining factor for animation algorithm complexity.

2.1.2 Character Skeleton

In skeletal animation, the movement of the vertices of a character mesh is inferred from the explicitly defined motion of an underlying skeleton. The skeleton is defined by a set of joints and their hierarchical relationships. Typically, a few dozen joints form a sufficient abstraction of the character. Effectively, the embedded skeletal structure reduces the deformation task from the individual determination of thousands of vertex movements to the computation of a few dozen degrees of freedom that dictate the movement of many vertices at once.

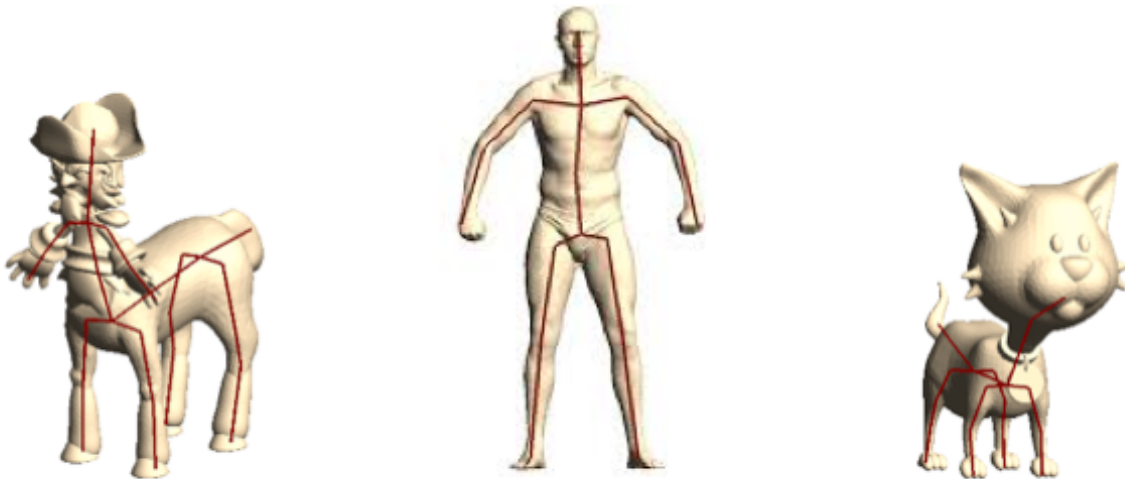


Figure 2.3: Auto-generated character skeleton estimations [7]

Skeletal animation also provides an abstraction of the common biological system, wherein muscles predominately modify the position of bones, which then deform the surface of the system due to what is effectively a spring system consisting of flesh that attaches and fills the space between the skin and the bones. The distribution of flesh under the skin implies that points on the surface are often influenced by multiple physical bones. The LBS model provides an approximation of that effect. However, skeletal systems and linear blended

skinning are by no means limited to the modeling of biological systems based on bones and skinning.

Skeleton joints are mathematically represented by transformations, which are the combination of translation and rotation that define a bone’s pose. It is common that bones will be defined in association with each joint as a tool for visualizing the articulation of the skeleton. The bones do not directly impact the deformation of the model. However, since bones are always associated with joints, the terms bone and joint are often used interchangeably. Animators and animation systems typically refer to “bones”, since this is more physically relevant than the technically correct “joint”. This research uses both terms interchangeably.

2.2 Deformation Methods

Character deformation and skeletal animation techniques are derived from basic deformation methods in 3D computer graphics. For the purpose of this thesis deformation techniques are categorized as either *spatial* or *physically-based* methods. This classification system is based on a combination of the classifications found in the works of Gibson et al [31], Gain et al [30], and Chaudhry et al [16].

Physically-based deformation manipulates objects based on physical science principles, such as mass, force, compressibility and inertia. This deformation technique is an extremely powerful tool for generating realistic deformation. The ability to generate realistic deformation frequently implies the use of physically-based techniques for highly-detailed and precision-sensitive applications, such as fabric deformation and human body modeling for medical applications. Physically-based deformations require a great deal of a priori knowledge and are more complex than other deformation techniques, making them not well-suited

for real-time systems due to the intensive computational requirements. To take advantage of the deformation benefits and compensate for computational complexity, physically-based techniques are often used in combination with other modeling techniques.

Geometric deformation techniques focus on the change in the shape of the geometry of an object without consideration of the physical nature of the object or its deformation. Geometric methods may induce deformation of a mesh by direct manipulation of the mesh or may use indirect association methods. *Spatial deformation* is a form of indirect geometry manipulation that changes the shape of an object indirectly by manipulating a control structure associated with the object to be deformed. The control structure may define any geometry including a volume, patch, curve, or simply a point in space. The association between the control representation and the deformable object allows changes in the control structure to drive the deformation of the object. Spatial deformation has the benefit of abstracting away some of the complexity of the physics of movement. Spatial deformation can be categorized as volumetric, point-based, curve-based or surface-based spatial deformation, as demonstrated in Figure 2.4.

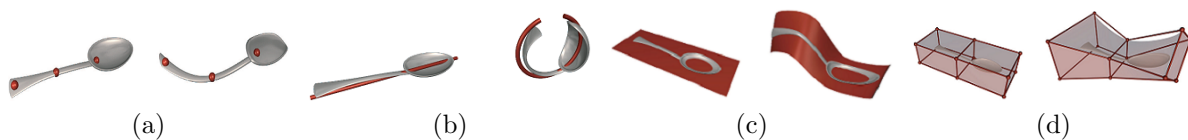


Figure 2.4: Spatial Deformation, before (left image) and after deformation (right image): 2.4a point deformation, 2.4b curve deformation, 2.4c surface deformation, and 2.4d volumetric. (Images from Gain and Bechman [30]).

2.2.1 Deformation and Character Animation

Although animation often recreates physical movement of the character, spatial deformation abstractions are more common than physically-based methods for skeletal animation due to their simplicity and reduced computational expense. Standard skeletal methods for character animation fall into a number of sub-categorizations within the set of spatial deformation abstractions.

Linear Blend Skinning, the skeletal animation technique that is the subject of this research, employs a combination of volumetric and curve-based deformation to simplify the skeletal animation process. The skeletal structure embedded in a character is an example of curve-based deformation. Distance-based methods for computing weights demonstrates volumetric deformation. These skeletal abstractions reduce the computational expense of determining thousands of individual transformations required to reposition vertices to a more moderate computational task of calculating a few dozen bone transformations that are then methodically applied to groups of vertices that move in similar manners.

Understanding the characteristics of each type of deformation on which skeletal animation techniques are built affords opportunities to leverage the strength of each method to advance the LBS process and counteract the negative trade-offs inherent to each.

2.3 Skeletal Animation Subprocesses

A skeletal structure must be embedded into the character mesh and its bones associated with the vertices. These processes are referred to as rigging and skinning.

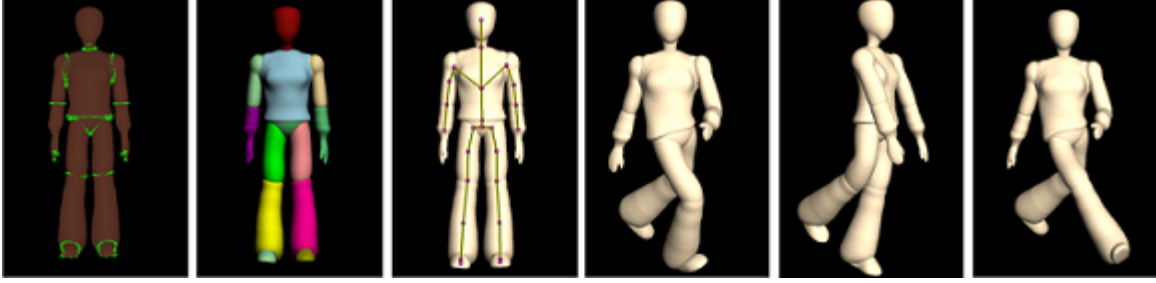


Figure 2.5: Rigging, skinning and animation. Cheng et al present an automated method for determining the skeletal structure and bone influences needed for animation of a humanoid character [18].)

2.3.1 Rigging

Rigging is defined as the determination of the shape, topography, and relative placement of the embedded skeletal structure, such that transformations can lead to natural-appearing deformations.

2.3.2 Skinning

Once a character is rigged, the animation process continues with the attachment of the skeleton to the mesh, or skinning. *Skinning* is the association of the vertices of the character mesh with the joints of the skeletal structure. Each vertex is assumed to be influenced by a subset of joints in the skeletal model. Skinning defines which joints will influence the movement of each vertex and the degree of influence of each joint. The degree of influence for joints are referred to as *skinning weights*.

2.3.3 Skinning and Rigging Challenges

Skinning and rigging can be tedious tasks requiring a great amount of skill and experience to do well and are tasks typically performed by a skilled animator. The placement of the skeleton is refined by hand with success determined mostly by the skill and experience of the

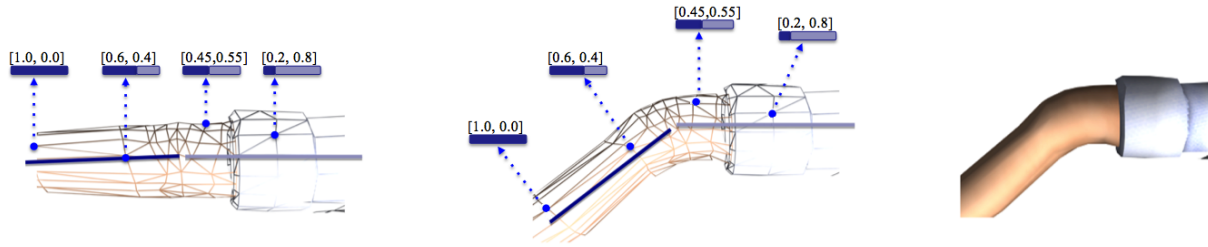


Figure 2.6: Character mesh in relation to bones with weights indicated on vertices. An elbow model in its base pose (left) and after deformation (center and right). All weights for a vertex sum to one indicating 100% influence

animator. Skilled animators learn through experience where to place and orient the joints to enable the desired movement of the character. Once rigged, a character can undergo the skinning process.

In most instances the skinning and rigging processes are highly iterative, requiring continuous adjustment of both bone positions and skinning weights while manually manipulating the model. The parameters are all highly interdependent; adjusting a bone often introduces new flaws that require changes in skinning weights to fix. It is also difficult for an animator to determine all potential poses and often a model will have to be reworked when an unexpected pose exposes weaknesses in the design.

2.4 Linear Blend Skinning

The most common mathematical framework used to implement the skinning and rigging processes is Linear Blend Skinning (LBS). LBS is a popular skeletal animation technique, also referred to as single weight enveloping, smooth skinning and skeletal subspace deformation. LBS forms the basis for many skinning algorithms, is often the point of focus for

improvements in skeletal animation, and serves as the foundation for this research. EP-LBS constructs a scalable system for computing LBS-compatible skinning and rigging parameters from example data.

2.4.1 Nomenclature and Notation Conventions

To inform the discussion of a mathematical representation of LBS, the following terms, variables and notation conventions are defined. First, formatting conventions are used to readily identify variable types. Scalar values are formatted in italics, as in w . Vector values are lowercase and bold, as in \mathbf{x} . Capital bold letters are reserved for matrix values, for instance \mathbf{Y} . Indexing variables are listed as subscripts, such as the z in Y_z .

Each skeleton joint is mathematically represented by a bone transformation, which, without explicit regard to its presentation, is indicated with a \mathbf{T} . Transformations are a combination of rotation and translation. Rotations can be presented in either a matrix or vector form. Rotations are identified by \mathbf{R} in matrix form and by \mathbf{q} or \mathbf{e} for quaternion and Euler angle vector representations, respectively. In this thesis, translations are presented in vector-form and are indicated by \mathbf{s} . Skinning weights, identified by the variable w , represent scalar values. Lower case variables i and j refer to the indices for vertices and joints respectively. Bone transformations, including rotation and translation components, are indexed by joint, as in \mathbf{T}_j .

Given a model for which many poses have been observed, one pose is designated as the *base pose*. All other poses are assumed to be deformations of the base pose. Vertices in the base pose, or original position prior to deformation, are represented by the variable \mathbf{v} . A vertex after deformation is denoted as \mathbf{v}' . Vertex positions computed using the methods presented in this research are indicated by \mathbf{v}'' . Lowercase subscript variable i indicates the

Object	Variable	Subscript
Base Vertex	\mathbf{v}	i
Computed Vertex	\mathbf{v}'	i
Rotation Matrix	\mathbf{R}	j
Quaternion Rotation	\mathbf{q}	j
Euler Angle Rotation	\mathbf{e}	j
Translation	\mathbf{s}	j
Transformation	\mathbf{T}	j
Weight	w	i, j

Table 2.1: Symbols and Terminology

index for a vertex. This research refers to the i^{th} vertex in its original position and prior to deformation as \mathbf{v}_i . Similarly, \mathbf{v}'_i is the i^{th} vertex in some observed transformed position and \mathbf{v}''_i is the i^{th} vertex in its transformed position as determined by EP-LBS optimization calculations. Weights are indexed by both vertex and joint. The term w_{ij} refers to the influence weight of joint j on vertex i .

Terms and variables used throughout this document are summarized in Table 2.1.

2.4.2 Coordinate Systems

The geometry of a character is defined using a model coordinate system, or model space. In addition to the model coordinate system, each bone in the skeletal structure has its own coordinate system. For every bone there is typically a matrix that transforms the bone in the bone coordinate system, or bone space, to model space. For hierarchical models, bones are transformed from the bone space to the parent bone space. This research adopts the global model space representation followed by local parent spaces. For further explanation of the coordinate systems and their usage, please refer to Appendix A.

2.4.3 Linear Blend Skinning Equation

For a deformed skeletal structure, the LBS algorithm defines a vertex position after deformation to be a linear combination of the weighted joint transformations applied to the original vertex position. A deformed vertex can be modeled by the following equation:

$$\mathbf{v}'_i = \sum_j w_{i,j} \mathbf{T}_j \mathbf{v}_i \quad (2.1a)$$

Subject to constraints:

$$0 \leq w_{ij} \quad (2.1b)$$

$$\sum_j w_{ij} = 1 \quad (2.1c)$$

The transformed vertex position, \mathbf{v}'_i , is the summation of the the weighted transformations of each bone $w_{i,j} \mathbf{T}_j$ applied to the original vertex position \mathbf{v}_i , as expressed in Equation (2.1a), where i is the vertex index, j is the bone index, and \mathbf{T} is a transformation in matrix form. The weights are a decimal representation of the percentage of total influence a particular bone has on a vertex. This lead to the non-negative (Equation (2.1b)) and summation (Equation (2.1c)) constraints that work together to confine weight values to positive decimal percentages that sum to one, representing 100% total influence for all bones.

2.4.4 Rotation Representations

Transformations are composed of both rotations and translations. Rotations can be represented as a matrix, Euler angle rotations or quaternions. Most commonly a 4x4 affine

transformation matrix is used to express the transformation due to rotation, translation, and scaling of the joint from a base pose position in world coordinates. A transformation matrix can be decomposed into a rotation matrix \mathbf{R}_j and a translation vector \mathbf{s}_j , yielding:

$$\mathbf{T}_j \mathbf{v}_i = \mathbf{R}_j \mathbf{v}_i + \mathbf{s}_j \quad (2.2)$$

where

$$\mathbf{R}_j = \begin{bmatrix} r_{11_j} & r_{12_j} & r_{13_j} & 0 \\ r_{21_j} & r_{22_j} & r_{23_j} & 0 \\ r_{31_j} & r_{32_j} & r_{33_j} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.3)$$

Rotation matrices are most often presented as a 4x4 affine matrix. The nine values of the rotation matrix are highly correlated, which makes computing the individual elements of a rotation matrix challenging from an optimization perspective. However, current computer hardware has been optimized to perform matrix computations quickly, resulting in the matrix form of rotations being the de facto standard.

Quaternions provide a concise alternative to matrix rotations, requiring fewer numeric values to represent a single rotation. Quaternion rotations are represented by four values, where rotation matrices are comprised of nine elements. Additionally, the individual elements of a unit quaternion are highly correlated. This correlation implies the need for additional constraints when quaternions are used for LBS rotations. However, the correlation can be exploited to simplify the process of determining the individual elements of a quaternion vector during EP-LBS optimization. The equivalent equation for quaternion

based transformations is:

$$\mathbf{T}_j \mathbf{v}_i = \dot{\mathbf{q}}_j \mathbf{v}_i \dot{\mathbf{q}}_j^* + \mathbf{s}_j \quad (2.4)$$

where

$$\mathbf{q} = [q_x, q_y, q_z] \quad (2.5)$$

$$\dot{\mathbf{q}} = [q_w, \mathbf{q}] \quad (2.6)$$

$$\dot{\mathbf{q}}^* = [q_w, -\mathbf{q}] \quad (2.7)$$

Quaternions, $\dot{\mathbf{q}}$, are represented by four values: a single scalar component, q_w , and a three element vector component, \mathbf{q} . Unit quaternion rotation is defined by the pre-multiplication of a point by the quaternion, $\dot{\mathbf{q}}$, and the post-multiplication by its conjugate, $\dot{\mathbf{q}}^*$. A unit quaternion conjugate may also be represented by $\dot{\mathbf{q}}_j^{-1}$.

Using a quaternion representation of rotations, an alternate formulation of the linear blend skin equation (presented previously as Equation (2.1a)) is introduced.

$$\mathbf{v}'_i = \sum_j w_{i,j} \left(\dot{\mathbf{q}}_j \mathbf{v}_i \dot{\mathbf{q}}_j^{-1} + \mathbf{s}_j \right) \quad (2.8)$$

When quaternions are used to represent rotations, an additional constraint is needed to guarantee unit quaternions for the skinning decomposition. The following requirement is added for the scalar quaternion components to the LBS constraints.

$$q_w^2 + q_x^2 + q_y^2 + q_z^2 = 1 \quad (2.9)$$

Of course, this constraint is considerably simpler than the constraints on a rotation matrix. For a concise, yet thorough, explanation of quaternion vector and matrix conversions readers are referred to Horn [42]. Readers are additionally referred to Henderson [38], Hogar [40], Watt and Watt [108] and Horn [42] for a detailed derivation of the quaternion matrix from the pre- and post-multiplied quaternion vectors.

The quaternion-based LBS representation in Equation (2.8) takes advantage of the benefits of quaternion representations of rotations without increasing the complexity of the overall deformation calculation. Additionally, the pre- and post-multiplication of a vertex by a quaternion and its conjugate can be written in matrix form as shown in Equation (2.10). The quaternion matrix, \mathbf{Q} , takes advantage of the highly correlated, reduced numeric representation of rotations as well as the computational efficiencies of matrix calculations.

$$\mathbf{v}'_i = \sum_j w_{i,j} (\mathbf{Q}_j \mathbf{v}_i + \mathbf{s}_j) \quad (2.10)$$

Equation (2.10) is the fundamental LBS model used for the EP-LBS research presented here and will be referenced as the Quaternion LBS Equation for the remainder of this thesis.

Chapter 3

Related Works

In addition to general research on improving the effectiveness and efficiency of the LBS algorithms, the formative concepts on which this research is built include constraint adherence, constrained optimization and example-based skinning decomposition. This chapter discusses research relevant to this thesis spanning each of these topics and places EP-LBS research in the context of the these fundamental works.

3.1 Variations on Linear Blend Skinning

As the prevailing method for 3D skeletal animation, LBS is known to have problematic characteristics that limit its application. These characteristics often result in visible artifacts, or undesired deformations, some of which are known by fanciful names such as “pinch” and “candy-wrapper”. Challenges for LBS implementations also include efficiently constraining the transformations applied to joints as well as performance and implementation issues that impact computation and memory requirements. To address these challenges, alternate methods for representing rotations, computing skinning weights and determining skeletal structure have been explored and are discussed here.



Figure 3.1: Skinning is subject to a variety of artifact. Common forms of deformation artifact occur when an object loses volume when bent, an effect known as “pinch” (a) or when an object is rotated, experiencing the “candy-wrapper effect” (b). (Images courtesy of Merry et al [69].)

3.1.1 Motivations for LBS Variations

For successful application of LBS, weights and transforms must satisfy certain properties and the model must produce the desired deformations at reasonable computational expense. As such, artifacts, or undesired deformations, constraint adherence and time and memory performance are common motivating factors for research and modification of Linear Blend Skinning.

Artifacts: Skinning methods for deforming 3D objects can result in the desired overall deformation but can also lead to unexpected or undesired deformations on a smaller scale, known as artifacts. Two common types of artifact are pinch and the candy-wrapper effect.

Pinch describes the artifact that occurs when an volume is bent significantly. This artifact is a result of the collapse of the object volume due to the bend. Pinch is problematic because many artists prefer to model figures in a base pose with the arms and legs straight, the so-called “da Vinci pose” or the “T-pose”. Since the pose is at one limit of elbow deformation, the other extreme of elbow deformation results in the greatest amount of pinch and the average elbow deformation always exhibits some visible pinch.

The *candy-wrapper effect* describes the loss of object volume when a rotation is applied to a portion of the object. The visual effect of this artifact resembles a twisted candy wrapper.

The candy-wrapper effect is particularly problematic on joints that rotate around an axis through the joint, such as a wrist joint.

LBS defines deformations by compounding transformations. Mathematically this takes the form of blending, or interpolating, matrices. However, significantly dissimilar transformations can result in degenerate blended transformations that cause the geometry to collapse [72]. Enhancements to LBS often aim to minimize some or all of these deformation artifacts.

Constraint Adherence: Many LBS algorithms assume rigid affine transformations for all deformations. Affine transformations include translation, rotation, scaling and shear. Affinity ensures the linearity of points is preserved; points that are collinear prior to affine transformation remain collinear after transformation.

Rigid transformations, while not required for all skeletal animation algorithms, ensure the distance between vertices remains the same after deformation in addition to remaining collinear. Rigid transformations are a subset of affine transformations. Transformations that have the property of rigidity are translation and rotation.

Bounds and constraints on weight and transformation values can be used to encourage rigid and affine transformations. However, computation of weight values that adhere to these conditions is not a trivial task. Significant research has been dedicated to addressing the constraints of the LBS model.

Performance: The use of animation in real-time environments for entertainment motivates the need for efficient skeletal animation. Methods to reduce running time or memory consumption form major areas of research focus for 3D animation.

Similar to this work, learning algorithms have been used to predict deformation gradients for the computation of LBS bone rotations in real-time environments [104, 13, 102, 92]. Hardware-accelerated LBS GPU implementations have been explored for increased compu-

tational efficiency [83, 28, 17]. The number of bone influences is frequently limited to four bones per vertex. This allows efficient use of hardware (one bone per byte and four bytes per 32-bit word) and avoids linear dependencies or under-determined systems. Additionally, compression methods have been employed to improve storage and processing efficiency of high-resolution models [90].

The research presented in this thesis is an offline process intended to simplify the computation of animation parameters needed for the creation of animated figures. While this work is not a real-time system, its goal is to facilitate simplified generation of animation parameters that may be used in real-time systems. The computed parameterization remains compatible with both basic and enhanced variations of LBS algorithms.

3.2 Skinning: Computing Weights

Skinning weights form the majority of the parameters in the LBS model. As such, numerous methods have been employed to generate weights and determine bone influence. Perhaps the most intuitive technique for determining bone influence is to associate each vertex with the nearest bone. This concept can be expanded to reflect the influence of multiple bones on the movement of a single vertex by assigning each bone a percentage of the total influence, or weight, based on the distance between the vertex and the bone. This forms a basic distance-based weighting scheme. Other analogous models and the introduction of additional control parameters are two other common approaches to computing skinning weights.

Constraint Adherence for LBS Skinning Weights: Standard LBS models constrain weight values to be greater than or equal to zero. In addition to the non-negative condition, all weights for a vertex must sum to a value of one. The standard LBS model and

weight constraints from Chapter 2 are listed here for convenience.

$$\mathbf{v}'_i = \sum_j w_{i,j} \mathbf{T}_j \mathbf{v}_i \quad (3.1a)$$

Subject to constraints:

$$0 \leq w_{ij} \quad (3.1b)$$

$$\sum_j w_{ij} = 1 \quad (3.1c)$$

A common approach to ensuring the sum-to-one constraint (Equation 3.1c) for rigid affine deformations is to solve for all weights save one such as in the work of and Mohr and Gleicher [72], Mohr et al [71], and Merry et al [69]. The value of the final weight, w_J , is computed as:

$$w_J = 1 - \sum_{j=1}^{J-1} w_j \quad (3.2)$$

This save-one, deductive reasoning method for computing weights ensures that all weights for a vertex sum to a value of one. However, care must be taken to ensure that the final weight is also non-negative, adhering to Equation (3.1b). Negative weights can afford greater freedom and variety in the types of deformation produced but may also introduce undesired and unexpected artifacts [71].

A common approach to guarantee adherence to the summation constraint for a set of non-negative weights is to normalize each weight for a vertex by dividing its value by the sum of all weights for the vertex. Normalization does not address the bounding condition for

weights, but can be used in combination with other bounded techniques to ensure weights that are both non-negative and sum to one.

Many solutions for computing skinning weights solve a system of linear equations given by the standard LBS model and a set of example data. This is a useful method when the transformations are known and there is sufficient example data available.

In addition to the non-negative and sum-to-one constraints, Mohr et al introduce additional variables and constraints to handle underdetermined cases [71]. A new variable y_j is introduced for each influence bone computed. y_j is computed as the minimization of $\sum_{j=1}^J y_j$, with the added constraints that $-y \leq w - w_p \leq y$, where w_p is the sum of the previous weights. The solution for this weighting scheme is achieved using a linear programming solver.

Hasler et al also introduce an additional constraint to the weighting process [34]. They introduced the concept of minimizing the L1-norm of the weight vector as the weight optimization is being simultaneously computed. The L1-norm in this case is the sum of the absolute values of the weights, so this additional constraint effectively minimizes the values of the weights. That minimization conflicts with the sum-to-one affine constraint, so the authors choose to re-normalize the results after optimization to ensure affine transformation. Since the minimization has been computed prior to normalizations, the normalized weight vector no longer guarantees an optimum solution. The additional constraint is added to improve the results of NNLS solutions, which often are not sparse, resulting in less arbitrary influence of bones.

The use of additional constraints can limit the search space for various solvers, but the cost of its implementation must be balanced with the overall program efficiency.

Distance-based Methods for Computing Weights: Simple distance based metrics

associating a bone or set of bones with each vertex have been used and improved upon for weight computation such as in the work of Kry et al [56], Bloomenthal [10], Lu et al [63], Madras et al [64], and Cheng et al [18]. Additionally, many 3D modeling and animation software packages, such as Blender, Maya and 3D Studio Max, use some distance-based metrics as the basis of proprietary, built-in “weight painting” features [9, 68, 67].

Kry et al do not specifically address bounds or constraints on the weights but suggest that weights at the ends of a bone’s influence should be forced to zero to ensure continuity and various optimization techniques can be used to compute weights that result in better quality deformations [56]. Bloomenthal suggests weight computation based on the medial axis of a three-dimensional model, taking into account thickness of the model [10]. Bloomenthal’s method, however, does not consider the constraints required to maintain affine transformations. Lu et al assume vertices are influenced by at most two bones and does not consider the possibility of negative weights (as a consequence of the problem structure weights are assumed nonnegative and will sum to one) [63].

Madaras uses geodesic distances to determine weights. The distances are computed using an algorithm with $O(n^3)$ time complexity, so computation on most meshes used in practice is impractical. Therefore, in most applications the mesh is downsampled prior to running the distance algorithm. Cheng et al use Euclidean distance to automate the commonly used “painting weights” metaphor, in which weights are assigned to the model via a graphical interface that mimics painting the model with colors defined by weight values, blending colors to indicate the influence of multiple bones.

Distance-based methods are easy to understand and, therefore, are a natural medium for assigning weights. However, distance-based methods are limited in the types of deformation they can produce and, as a result, often yield deformation artifacts. Additionally, separate

measures must be taken to ensure constraint adherence. The methods for computing skinning weights presented in this thesis handle constraints while computing weight values.

Analogous Weight Models: In an effort to minimize folding artifacts during extreme bends of the armature, Baran and Popovic use the analogy of heat transfer to compute bone weights [7]. Bone influence is determined based on a heat transfer equation in a method called *bone heat*. In this method the concept of heat equilibrium is applied to the character mesh in order to ensure that the width of the transition between two bones is roughly proportional to the distance from the joint to the surface, thereby minimizing the folding artifact, or pinch. The bone heat technique specifies that the amount of influence a bone has over a vertex (effectively the skinning weight) is equal to the amount of heat the vertex receives from that bone. Baran and Popovic’s heat equilibrium analogy employs distance-based metrics in combination with the save-one technique to compute affine weights.

While bone heat reduces the folding artifact of common vertex weighting techniques, it still does not provide the most optimal vertex weighting scheme. The master’s thesis research that precedes this dissertation shows that the use of example data can produce improve results in comparison with bone heat [41]. Bone heat has been adopted by Blender as the basis for its built-in weighting algorithm and is used in various automated animation research, such as in the work of Pan et al [78].

Analogous methods provide useful abstractions to compute weights. The computational expense of the method and its ability to accurately model the desired set of values are the challenging trade-offs considered when using such models. This research uses abstractions (not tied to any analogous model) for computing weights and rotations, considering the trade-offs between computational expense and the ability to produce values adherent to all constraints.

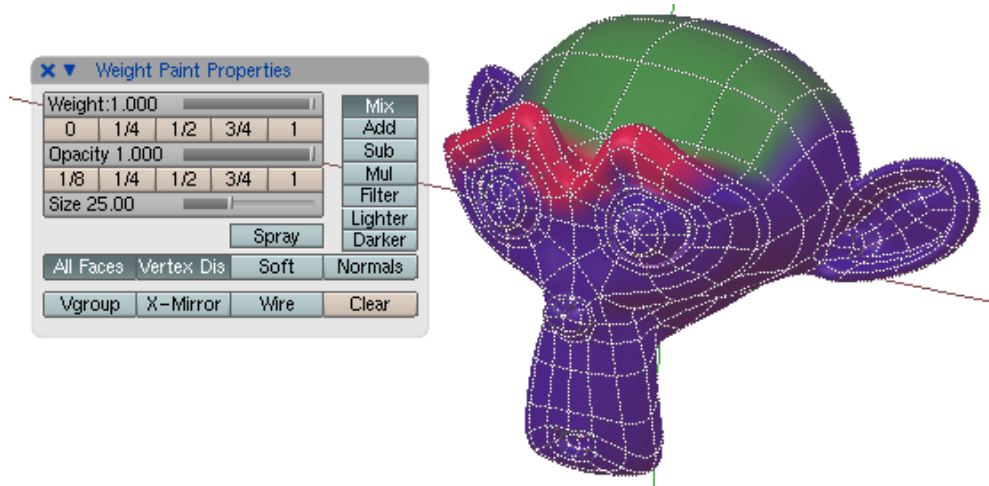


Figure 3.2: Blender Painting Weights: Monkey head model and Blender interface for manually painting weights. [9].)

Using Additional Parameters with Skinning Weights: Skinning methods often strive to achieve more refined control over character movement. A common solution is to increase the number of weights or controls used. By using more control parameters, an artist can create more refined movement. Instead of associating a single weight with each bone transform, Wang and Phillips incorporate an additional set of weights, or coefficients, for each element of the bone transformation matrix, a process referred to as *multi-weight enveloping* [105]. This results in a total of 12 weights for every bone that is influencing a given vertex. In addition to the added bone transformation matrix coefficients, the rows and columns of the matrix are scaled (multiplied) by constant scalars to manipulate the influence of a particular pose or a particular bone. This allows for more flexibility in armature movement. Wang and Phillips utilize example poses and a least-squares approach to compute weights, much like this research. Although this technique utilizes inverse matrix approaches to solve for the weights, the process does not lead to intuitive weight values and still requires considerable ‘hand tweaking’ and comes at increased computational expense. Still, this work was pivotal in exposing weights as the focus for increased control over deformation results.

Recently, Jacobson et al have proposed the use of additional weights computed from cubic spline basis functions in addition to standard weights to address common artifacts [46]. This technique employs the same theory as in the work of Wang and Phillips: more weights provide greater control over deformation [105]. However, all of the augmented weights approaches result in significantly increased computational requirements. The additional weights used by Jacobson et al require random sampling of the mesh because the Delaunay triangulation of high resolution meshes is computationally expensive. Jacobson et al acknowledge the increased computation cost but note that it is faster than nonlinear systems [46].

The trade-off between performance and control of deformation is one that must be carefully considered. For standard LBS systems, computation and optimization of weights is done once for a model prior to animation as an offline process. Any added parameters that impact the computation of deformed vertices during the animation process will be costly every time a model is rendered. This research also introduces extra parameters. However, the new parameters are used in place of standard LBS parameters only during optimization, thereby not incurring overhead during the runtime use of the model.

3.3 Rigging: Computing Skeletal Structure and Bone Rotations

Determining the shape, topography, and relative placement of the embedded skeletal structure such that transformations can lead to natural-appearing deformations is an integral component of the Linear Blend Skinning system.

This section focuses on the automatic determination of the skeletal structure based on a single mesh. A significant limitation to automated rigging from a single rest pose mesh is



Figure 3.3: Dual-quaternion solution (left) to common linear blend skinning unnatural deformation (right) [29].)

that the articulation and range of motion for the model is unknown at the time of rigging. Therefore, the rigging task is based solely on the static shape and topological characteristics and what a priori knowledge of specific topological characteristics infers about the placement of joints necessary to achieve a desired range of motion. When additional poses are available it is possible to compute a more robust skeletal structure from the available data. The use of example data to compute skeletal structures is discussed as a component of example-based LBS in Section 3.4.

3.3.1 Rotation Representations

Standard LBS implementation uses matrices to represent rotations. However, quaternions have been found to be computationally more efficient and offer deformations with fewer rotation-related artifacts. Frey and Herzog use quaternions to reduce the algorithm overhead of effects such as motion blur by using a rotation representation that requires fewer values than standard matrices [29]. Kavan et al use dual-quaternions to reduce skinning artifact

[52, 53]. Hejl introduces an additional vertex attribute for quaternion spherical skinning [36]. Unfortunately, it has been noted that this method does not work well for vertices assigned more than two skinning weights.

Similar to the works discussed in this section, the research presented in this thesis takes advantage of processing improvements achieved from vectors compared to matrices. However, this research employs both quaternion and Euler angle vectors. Quaternions are used to represent the large-scale orientation of the bones, while Euler angles model fine adjustments to the bone placement that are iteratively refined during optimization.

3.3.2 Skeleton Extraction

Skeleton extraction leverages the topological characteristics of a model to automatically infer its skeletal structure. Common approaches have been classified in a variety of manners such as geometric and volumetric [78], surface flow and segmentation [47] or surface and curve methods [21]. This thesis broadly classifies skeletal extraction methods as either medial axis methods or segmentation methods. Single pose, automated methods for skeleton extraction from one static mesh are discussed here. Multi-pose, example-based skeleton extraction methods are described in Section 3.4.

3.3.2.1 Medial Axis Skeleton Extraction Methods

The concept of a medial axis for shapes was initially introduced by Blum as a mathematical description of shape for biological applications [12]. Blum described the excitation of points on a plane generating waves that flow uniformly in all directions but cannot flow through one another. The points of intersection of the waves, or loci of wave discontinuities, form a reduced representation of shape that is referred to as the medial axis. This medial axis

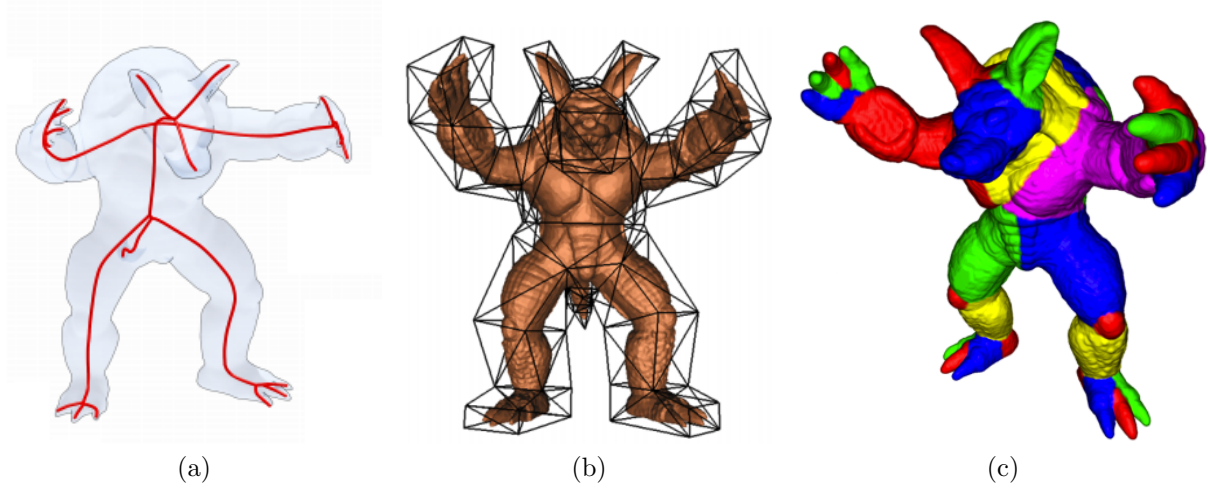


Figure 3.4: Skeleton Extraction Methods demonstrated on Armadillo model : 3.4a medial axis (from the works of Tagliasacchi et al [99]), 3.4b cage control structure (from the works of Ju et al [50]), and 3.4c segmentation (as described by Reniers and Telea [82]).

concept has since been more succinctly described as the set of points enclosed in a mesh having multiple closest points on the surface of the mesh [11].

For the purpose of computing a character skeleton, popular means for computing the medial axis include mesh contraction, or thinning, field-based methods, Reeb graph methods and Voronoi diagrams. Each is treated briefly here. For further reading on skeletal extraction methods, readers are referred to the works of Cornea and Min [21] and Siddiqi and Pizer [95].

Voronoi diagrams partition a model into regions such that each region contains a generator point or seed and all other points in the region are closer to that seed than any other seed. These seeds can then be connected into a skeleton-like graph structure and used to define a medial axis skeletal abstraction.

Ogniewicz et al use hierarchical clustering of the Voronoi diagram of a shape to generate a hierarchical skeleton [75, 76]. Yoshizawa et al approximate the medial axis for a shape using a two-sided Voronoi-based approximation to extract a skeletal mesh [115]. Teichmann and

Teller compute and simplify a medial axis from a 3D Voronoi diagram to generate a skeletal structure [100]. Dey and Sun define a medial geodesic function based on an approximate medial axis computed from a subset of the Voronoi diagram for the model to extract a skeletal structure for 3D models [22].

Medial axis models generated from Voronoi diagrams form an intuitive method for extracting a skeletal structure from a single static pose of a model. However, 3D model Voronoi diagrams typically result in medial surfaces that must be further pruned or otherwise refined to reduce the dimensionality of the diagram and generate an appropriate LBS skeleton representation. As a result, Voronoi medial axis methods for skeletonization require additional computational expense beyond Voronoi diagram generation.

Reeb graphs directly compute a 1D structure, diminishing the need for post-processing to reduce the dimensionality of the resulting skeletal structure. Reeb graphs are computed by a user-defined function that encodes the topology of the model. The critical point of the function define nodes of the computed skeletal structure. Reeb graph implementations can be computationally expensive, many with a running time of $O(n^2)$. Additionally, Reeb graphs are not ideal because the user is required to define the boundary conditions for the function.

Hilaga et al define multiresolutional Reeb graphs to define the skeletal structure of a model which serves as a search key for collections of 3D shape data [39]. Tierny et al define a unified method to construct and simplify a Reeb graph skeletal representat of a model [101]. Patanè et al compute Reeb graphs based on iso-contours at saddle points of a continuous function which can be used to define skeletal structures with lower memory overhead [79].

Doraiswamy et al introduce a more optimal Reeb graph construction algorithm [23]. Aujay et al use a $O(n \log n)$ method to compute the Reeb graph. The authors compute the

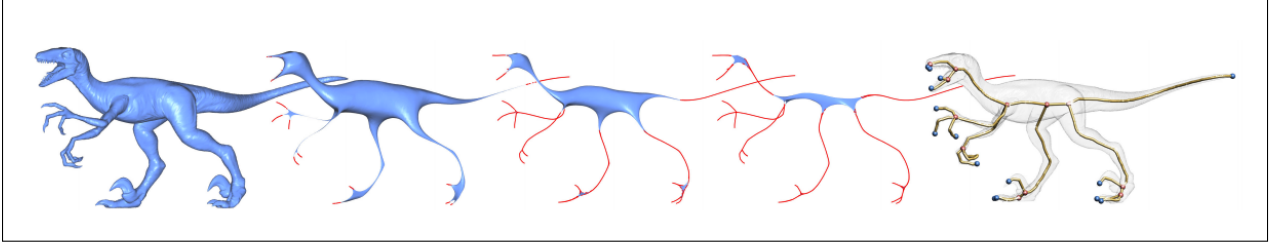


Figure 3.5: Skeleton extraction by mesh contraction by Au et al [5].

skeleton on anatomically-based models by computing a Reeb graph of a harmonic function operating over the surface of a model [6]. The approach presented by Aujay et al aims to improve the skeletonization process by generating realistic skeletal structures that reside fully inside a model, including the endpoints of bone, making this method best suited for models for which an anatomical skeletal structure is available.

Various *field-based* approaches to the extraction of curve skeletons have been explored, including electrostatic fields [32], repulsive force fields [62], potential fields [20, 19], tensor fields [114] and, most commonly, Euclidean distance fields, as demonstrated in the works of Malandain et al [65], Shilane et al [93], Hassouna et al [35] and Huang et al [43]. Basic distance fields minimize the shortest distance from embedded skeletal points to some surface boundary. More general potential fields, define a skeletal structure based on the sum of potentials generated by a model’s surface or boundary, defining the potential of an interior skeletal point. These generalized methods do not always create ideal curve skeletons from which the final skeleton can be produced. General field methods take into account larger surface areas to compute the skeletal structure, yielding more ideal curves at the expense of increased computation.

Laplacian smoothing is often used as a method of *mesh contraction* for generating medial axis curve skeletons for models, such as in the works of Au et al [5], Wang et al [106] and Tagliasacchi et al [99]. Additional smoothing methods compute a skeleton from point

clouds as in the works of Cao et al [14] and Huang et al [43]. Forstmann et al [27], Yang et al [113], Forstmann et al [28], Jacobson et al [48] and Ozetirelli et al [77] present various methods for converting curve skeletons generated by mesh contraction into LBS skeletal structures. Ultimately, like other medial axis methods, mesh contraction methods for curve extraction must employ an additional process to compute a rigid skeleton for use with LBS.

The various medial axis representations of shape form a good initial approximation for a skeleton because they generate a simplified embedded representation of a characters shape that closely mimics the desired skeletal structure. However, the medial axis for a character is not sufficient to create a LBS skeletal structure in 3D because 3D medial axes are typically 2D surfaces rather than one-dimensional splines or curves [47]. The desired curve skeleton can be extracted from a medial surface. Additionally, curve skeletons defined by post-processing a medial axis may not be direct correlations to the rigid skeletal structures needed for LBS deformations. Therefore, medial axis methods are used in conjunction with other post-processing to generate the LBS skeletal structure for a model. Still, medial axis methods are intuitive, foundational skeletal extraction methods that remain relevant to the study of computing of skeletal structures for Linear Blend Skinning.

3.3.2.2 Clustering and Segmentation Methods for Skeleton Extraction

Clustering and segmentation methods for skeleton extraction divide a character or model into segments based on properties of its mesh geometry. The method used to cluster geometry and find "cuts" or boundaries for the segments distinguishes one segmentation method from another. Katz and Tal determine segment boundaries based on extreme concavities, or creases, and fuzzy clustering of the mesh [51]. Lien et al use a combination of mesh openings centroids and convex hull principal axis centroids to define mesh segments and extract the



Figure 3.6: Single depth Kinect images are used to segment human meshes (Images courtesy of Shotton et al [94].)

skeleton [60]. Huang et al use modal analysis to analyze surface deformation energy to segment the mesh [44]. Cheng et al use principal component analysis (PCA) and clustering to segment the character mesh [18]. Similar to the work presented in this research Bharaj et al analyze connected components to segment the mesh [8]. Bharaj et al establish the base components as two triangles sharing an edge. Such small components tend to lead to oversampling. Bharaj et al exploit the oversampling to segment the mesh and generate a graph that forms the basis of the skeletal structure, but must also employ iterative edge-collapse to reduce the dense graph produced into a reasonable skeletal structure. Shotton et al use a Microsoft Kinect depth image to segment human character meshes [94].

Most single-pose segmentation methods rely on convex triangular meshes, but provide little guarantee that the skeleton generated will exist entirely within the character mesh. While, a skeleton that is fully encapsulated by the character mesh is a common characteristic, it is not an explicit requirement by default.

Segmentation works well when shape mimics the skeletal structure of the character mesh. However, clustering and segmentation methods are susceptible to the challenges of any skele-

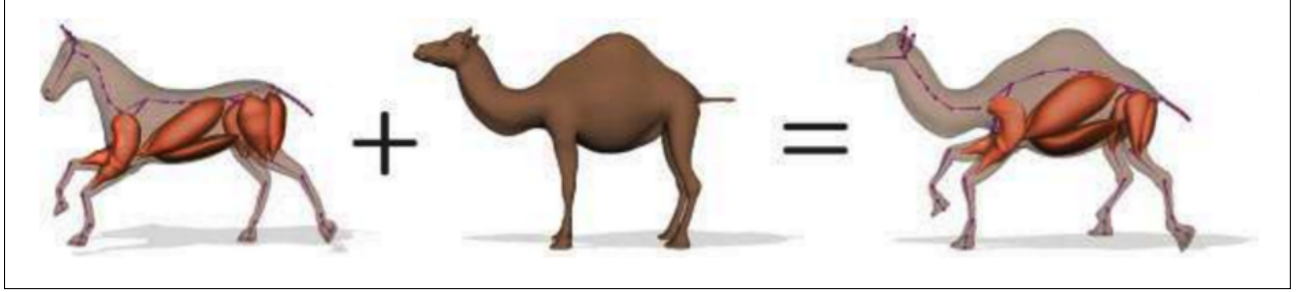


Figure 3.7: Seo et al demonstrate the transfer of an anatomically-based rig from a horse to a camel [89].

tonization methods based on a single pose. If the pose from which the skeleton is being extracted significantly obscures the range of possible deformations, skeleton extraction methods will fail to generate an appropriate skeleton without additional information.

3.3.2.3 Summary of Skeleton Extraction Methods

Skeleton extraction methods attempt to automatically compute a skeletal structure from single static character model. Automatic, single-pose skeleton extraction methods, however, are severely restricted by the limited information that can be derived from a model in a single pose. Utilizing additional information beyond a character mesh in a single pose, making some assumptions about the skeletal structure, or employing additional heuristics can improve extraction results. This thesis uses the additional data gained from a collection of example poses to further inform the clustering and segmentation-based skeletal extraction process.

3.3.3 Skeleton Embedding and Rigging Transfer

Skeleton embedding takes advantage of a priori assumptions made about the general shape of a new model to embed and adjust an existing skeletal structure for a similarly shaped model in the new model. Skeleton embedding based on a single pose of a character mesh alleviates

many of the challenges of insufficient information faced by skeleton extraction methods. The challenges are addressed by relying on the additional information provided by an example or template skeleton.

The template skeleton must often be modified to best fit the new model. Jacobson summarizes the most common desired properties of an embedded rig [47]. Ideal embedding approaches strive to:

- Avoid degenerate bone properties (ex: short, zero-length and overlapping bones)
- Maintain structure and topology (ex: symmetry, angles, orientations and proportions)
- Place bones logically ("feet" bones at below others, extremities near surface)

Because the template skeletal structure is not an exact match for the model in which it will be embedded, fitting or parameterization of the template skeleton is necessary. Frequently, the fitting, parameterization and embedding problem is structured as an optimization problem and the embedding goals are expressed mathematically as cost functions or constraints. The skeletal structure (such as hierarchy and topology) are predefined by the template skeleton and the goal is then to compute the appropriate adjusted joint positions and bone lengths for the model, ideally adhering to the aforementioned properties of an embedded rig.

Moccozet et al define template meshes and skeletons and characteristic points on the mesh to parameterize a human body mesh that can be used to reconstruct human body shapes [70].

Miller et al take advantage of a database of partial source rigs and a target mesh to semi-automatically determine the animation parameters needed to animate the mesh. The

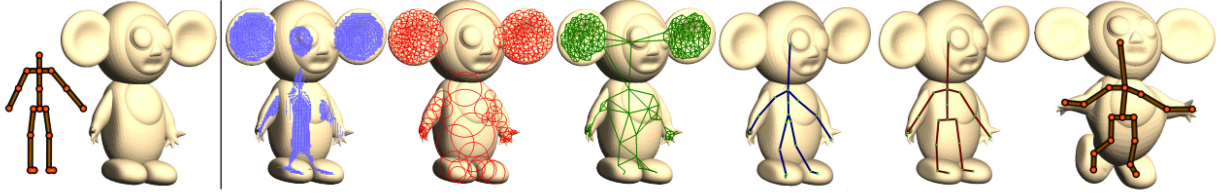


Figure 3.8: Rigging Pipeline introduced by Baran and Popovic [7].

animation parameters are computed by fitting various known parts from the database to parts on the mesh and transferring the known animation parameters to the new model. This method relies on the user tagging parts of the new model so that they can be matched with parts in the database. Joint positions and bone lengths of all the source parts are adjusted to fit the new model and scored for comparisons. Parts with the best score are used to generate the new skeleton. There is also an option for user-override of part choices.

Rigging transfer refers to a special cases of skeleton embedding in which the class of skeleton does not match the class of character to which the skeleton is being applied or the shape and class of character in which the skeleton is being embedded is not known in advance. Poirier and Paquette use skeletons rigs, both in part and in their entirety, to transfer animation and “remix” animations by combining parts of multiple skeletons [80]. Their rig retargeting technique generates a topology graph from the mesh and compares it to the known skeletal graph. The skeleton is then adjusted to roughly match the mesh graph, by comparing local arc shapes in the graphs.

Given a skeletal embedding problem, the inclination is often to structure the embedding process as an optimization process with the goal of finding the ideal joint positions. However, as Jacobson points out, the theoretical goal of an ideal joint embedding often entails a complex translation into mathematical form (goals that can be expressed as a mathematical equation) and must address competing and interdependent goals [47].

Baran and Popovic combine skeleton extraction and embedding to introduce a rigging pipeline that successively applies a variety of skeletonization techniques to compute and refine the placement of skeletal structure into a character [7]. First a skeleton is extracted from a static mesh using successive extraction methods. The computed skeleton is then embedded into the model and the embedding is refined by minimizing a penalty function that models skeleton embedding.

The successive extraction methods include computing an approximate medial surface, deriving a graph from the medial surface and simplifying the graph. The approximated medial surface is computed from a sampling of points on the medial surface, computed using distance-based methods. To reduce the dimensionality of the medial surface and produce a graph structure that can be used to generate a skeleton, a sphere packing method is introduced. Sphere packing is used to identify graph vertices that represent a set of potential joint positions. The centroids of the spheres are strategically connected to create a graph. A reduced skeleton is then produced by systematically merging edges of the graph. Baran and Popovic introduce a penalty function that aids in refining how the computed skeleton is embedded into the mesh.

In some cases, skeleton fitting or embedding can offer a more robust alternative to single-pose skeleton extraction. For instance, common classes of skeletons such as humanoids, bipeds, quadrupeds, human hands and insect-like models may have an abundance of template models to be used as the basis for embedding. However, if the new model is significantly different from the available template models, the skeleton embedding process may not significantly benefit from the data provided by the example. Even given a well-chosen template skeleton, the exact placement of joints within the mesh is still critical to ensure quality deformations.



Figure 3.9: A kangaroo and various skeletal abstractions used by Capell et al [15].

3.3.4 Cages, Exoskeletons and Other Skeletal Abstractions

In addition to embedded skeletons, the concepts of cages, exoskeletons and other skeletal abstractions may be used to define bone groups for LBS models. Capell et al define a volumetric control lattice that can be used to control the embedded skeleton of a character which in turn deforms the rigged character. This method is used for interactive deformation and uses a finite element method [15].

Skeletal abstractions attempt to define the skeletal structure without explicitly defining a graph like hierarchical structure of embedded nodes and edges to represent bones and joints. The most common skeletal abstraction is the clustering of vertices that move together to define a bone without explicitly identifying the bone or joint skeletal structure. Much like this work, van Kaick et al use an abstraction to generate clusters that define the skeleton [103]. van Kaick analyzes the convexity of the mesh to generate clusters. In contrast, this research analyzes transforms of connected components. This research relies on a form of clustering segmentation to identify an initial skeletal abstraction that is then used to initialize LBS parameters.

Exoskeletons, cages and other skeletal abstractions may be used alone or may be used as an intermediary to compute standard LBS skeletal structures. These abstractions allow greater flexibility when defining and deforming a character skeleton. However, not all skeletal abstractions can be used to compute standard LBS skeletons or animation parameters.

3.4 Example-based Linear Blend Skinning

The long-standing tradition of indirect use of example data as the archetype for character deformation suggests that data-driven and example-based skinning and rigging techniques can

be a powerful resource for computing skeletal animation parameters. In practice, animators may visually compare a deformation with scanned models or recorded motions. During the production of the movie *Jurassic Park*, animators working on the scene presenting a running flock of Gallimimus filmed themselves running in the parking lot pretending to be dinosaurs [97]. The film captured the accidental trip and fall of one of the animators, an element that actually appears in the film. After observing an example pose or motion the animator can modify the model’s animation parameters until it is capable of undergoing comparable deformation and movement. While beneficial to the animation process, this type of indirect example-based approach inefficiently utilizes the information contained in the examples and does not take full advantage of the rich data set available in example scans.

Direct example-based techniques, also known as data-driven techniques, use observed example data to make educated inferences on the smooth skinning equation, inform the computation of unknown animation parameters and generate new data. Data-driven techniques have the benefit of both simplifying the animation process through automation and improving the quality of generated deformations by directly using the example data.

Automated LBS methods, such as those described in sections 3.2 and 3.3, use a single pose as example input for determining the skinning and rigging parameters for a model. This provides a starting point from which to build the animation. However, when additional example poses are available, more powerful techniques can be used to compute animation parameters. Example-based methods define the use of multiple poses for automated computation of skinning and rigging parameters.

Example-based LBS methods fall into two broad sub-categories: example-based shape deformation and example-based skinning decomposition. Shape deformation includes data fitting and interpolation methods for computing intermediate poses and fitting new data to

template models. Skinning decomposition is the general purpose pipeline for solving LBS optimization functions to compute animation parameters that can be used to both recreate the example data and also generate new poses, whether intermediate to example poses or entirely different poses. Each method has unique benefits and challenges.

3.4.1 Example-based Shape Deformation

Example-based shape deformation emphasizes the creation of reasonable new poses from a collection of example data. Less focus is placed on the computation of specific animation parameters, in favor of an emphasis on reasonable shape and pose creation.

3.4.1.1 Interpolation

Interpolation of character motion is a fundamental and pervasive technique in animation and has been extended to interpolation of various other animation components. Interpolation of character movement is the computation and completion or insertion of intermediary poses or frames between known poses or frames, known as key frames. Interpolation can be applied directly to parameters such as rotations or to end effector positions. Inverse kinematics is the common form of pose interpolation used to compute the bone positions of intermediary poses. Other forms of interpolation may interpolate poses, shapes, transformations, or other numeric parameter values. In fact, the entire smooth skinning process can be viewed as an interpolation or blending of transformations.

Pose Space Deformation (PSD), introduced by Lewis et al, was developed as a solution to common smooth skinning artifacts caused by the linear blending of transformations [59]. PSD is a form of displacement interpolation in which the space of possible poses is defined by a set of example poses. Pose parameters for new poses that fall within the defined

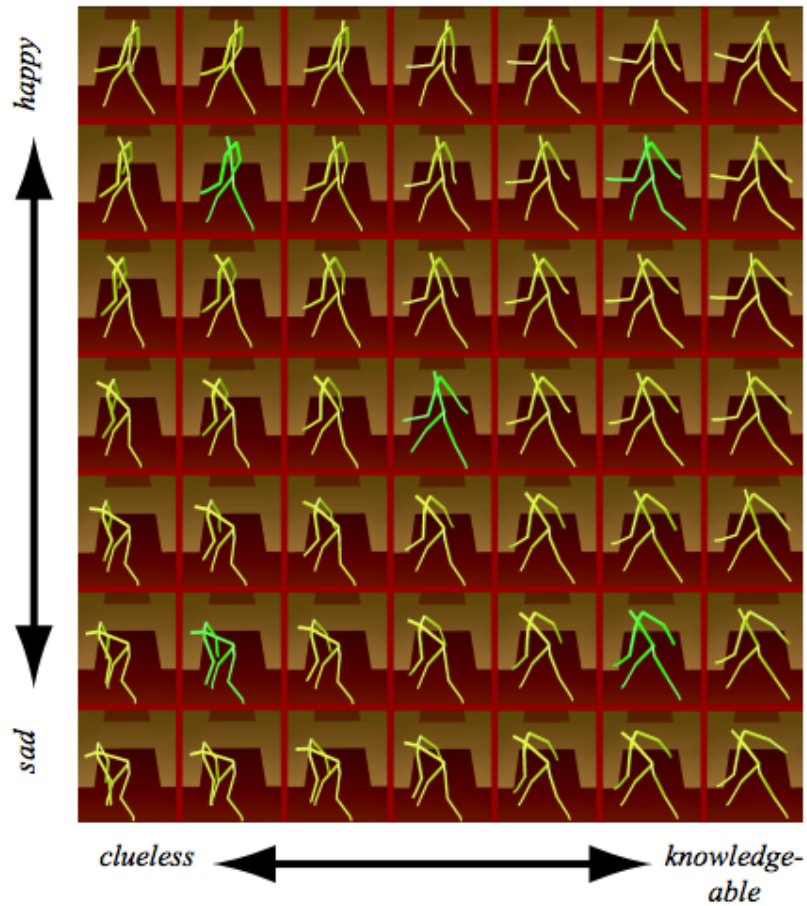


Figure 3.10: A walk sequence generated from interpolation. Highlighted cells contain example motions. All other poses were generated using the verb/adverb interpolation mechanism introduced by Rose et al[85].

space can be interpolated from stored example poses. Radial basis functions are used to compute interpolation weights. This method focuses primarily on the rigging process and the generation of skeletal parameters for new poses. A problem with any PSD-based method is that all example poses must be available during rendering, whereas for conventional LBS only a base pose is necessary. Since graphical models can be very large, this can result in a large memory cost. In contrast, the research presented in this thesis is a comprehensive solution for both the skinning and rigging processes and retains the basic LBS approach of deformations relative to a base pose.

While other research focuses on either blending between shapes or blending transforms, Sloan et al use a combination of both to interpolate between poses [96]. Input scan data is utilized to compute weights for each vertex in the model as a function of a weighted radial basis function and a linear polynomial. Sloan et al use a linear system per example pose which is more efficient than per degree of freedom linear systems such as those in Rose et al [85] and Lewis et al [59].

Instead of per-vertex or pose interpolation, Allen et al interpolate the shape of each body part individually and combine the results to form interpolated poses [2]. To determine the weights for blending the composed body parts, Allen et al use k -nearest neighbors interpolation to compute the body part interpolation based on pose and cosine functions. The research presented by Allen et al provides a robust solution, however some of the computed values can be counterintuitive due to the weighted compositions of different body parts.

EigenSkin is a model developed by Kry et al that interpolates eigendisplacement coordinates using radial basis interpolation [56]. Eigendisplacements are used to reduce redundancy between deformations. Kry et al present a solution that is more expensive computationally, but provides improved results and decreased artifacts in human hand deformations.

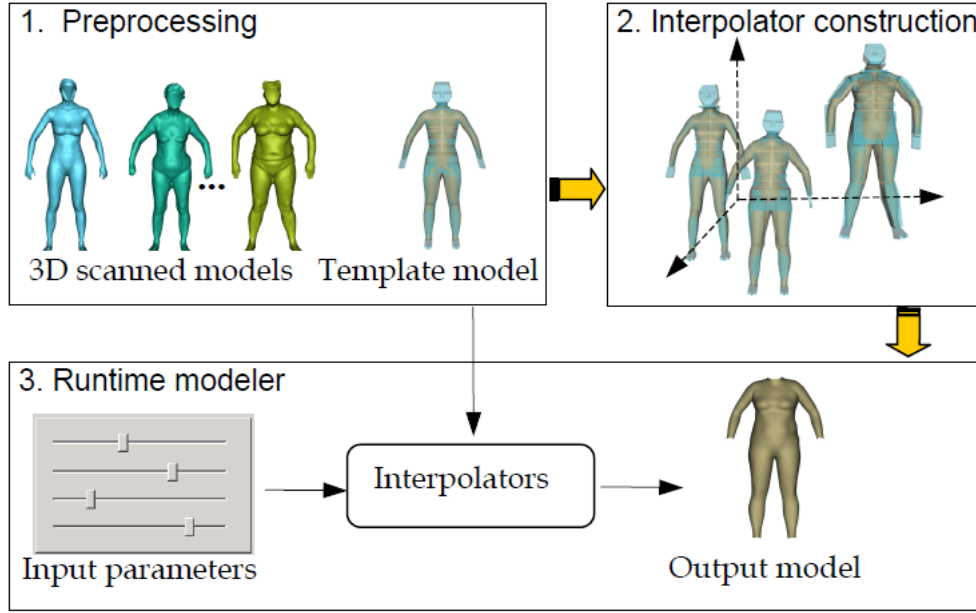


Figure 3.11: Example-based Model Generation System: Seo and Magnenat-Thalmann fit a set of example scans to a template model and use interpolation to generate a new model [88].

A combination of various interpolation approaches is used in the work of Seo and Magnenat-Thalmann [88]. To automate the modeling of diverse human bodies, they use a layered interpolation approach, where each component of the body geometry is modeled with an interpolator. Gaussian radial basis functions are used to create the linear and residual components of the deformations. Joint transformations are interpolated in a manner similar to EigenSkin and eigendisplacements are used to create displacement interpolators to express vertex movement. Seo and Magnenat-Thalmann’s system is more complex than other frameworks and works well in creating the realistic human body shape diversity that the researchers desired. However, there are other, less complex methods for computing comparable shape diversity.

Mohr and Gleicher also use interpolation and additional joints to solve problems with artifact [72]. Additional joint positions are computed as a halfway spherical linear interpolation

of the rotation of the joint and its rotation in the example pose.

3.4.1.2 Template and Data Fitting Techniques

Template fitting is an alternate example-based shape deformation technique that uses a collection or database of known data that has been processed and categorized to match and adjust known template data to new data. Template models work well when there is large amount of relevant input data on which to train the processing algorithm and when the structure of models is well-understood, such as human body shapes.

Allen et al parameterize range scan data to produce high resolution template meshes that can be used to generate a wide-range of realistic human body shapes [3]. They demonstrate an application that allows the generation of new "keyframe" models and linear interpolation of keyframe models to generate new models with widely-varied realistic human appearance.

Template models are typically an estimation of the actual data. The success of such models for LBS is greatly dependent on the similarity of the new data to the previously observed data in the database. The success of template and data fitting techniques also depends significantly on the ability of the data processing methods used for training to successfully parameterize the template models. Thus if the new pose or data is significantly different from all of the template data or the template model parameterizes the wrong features, the fitting process may have a difficult time or be unable to find the appropriate parameters to match the new data.

3.4.2 Skinning Decomposition

Fitting and interpolation are intuitive means of computing deformations. However fitting is inexact and interpolation is unable to compute LBS animation parameters, specifically

skinning weights. Although some fitting techniques attempt to account for differences in the template models and the new models being parameterized, the new models are generally forced to fit the shapes of an observed model.

Interpolation computes intermediate transformations to smoothly transition from one pose to another. This, however, is not the same as blending transformations as is required of Linear Blend Skinning. LBS computes a weighted combination of transformations to generate a deformation. Although weighted transforms may be used to interpolate between observed poses and generate new intermediate poses, there is no conversion from the new interpolated transforms to the corresponding weighted transforms for LBS [52]. Therefore, interpolation can be used to generate intermediate poses but cannot be used to compute the LBS skinning weights for those intermediate poses.

Historically, automated example-based methods for computing animation parameters have focused either on the computation of skinning weights or bone rotations. The complete process of computing the set of animation parameters, both weights and transforms, that recreate a set of example LBS deformations and can be used to generate new poses forms a workflow that is known by a variety of names. This research uses the term skinning decomposition as coined by Kavan et al to refer to the inverse problem of computing LBS animation LBS parameters from example data [54].

A few cornerstone works have defined a full skinning decomposition pipeline. James and Twigg coined their skinning decomposition approach *Skinning Mesh Animation* (SMA) [49]. Unique to their work at the time, James and Twigg incorporate flexible bones as an extension of the LBS model, in addition to the standard rigid bones, to achieve greater accuracy of resulting reconstructed data. Kavan et al consider the summation constraint for skinning weights to be a soft optimization constraint in their work titled *Fast and Efficient Skinning*

of *Animated Meshes* (FSD) [54]. In *Learning Skeleton for Shape and Pose* (LSSP), Hasler et al combine skeletal abstractions for defining character shape with standard LBS hierarchical skeletons that define pose, to reduce the reconstruction error of computing animation parameters from examples[34]. Le and Deng, in their work *Smooth Skinning Decomposition with Rigid Bones* (SSDR), use a layered approach to optimization, first optimizing a set of values that adhere to a subset of the constraints, before running the optimization additional times to compute the optimized values with respect to the remaining constraints and then compute the final optimized value, adherent to all constraints [57].

Regardless of name or title, all techniques for multi-pose example-based skinning decomposition have a similar problem structure and workflow. This research adopts the example-based skinning decomposition problem structure and workflow described by Le and Deng [58]. Typically, the problem structure is determined by a least squares objective function, subject to minimization. For skinning decomposition this objective function is defined by a specific LBS formulation used to compute character poses from animation parameters, any associated constraints on the animation parameters, such as summation and nonnegative constraints for skinning weights and some error expression to evaluate the reconstruction of the example poses. The problem structure serves as the foundation of any skinning decomposition solution and is critical to the implementation of the three major subprocesses of example-based skinning decomposition.

Given a problem structure, formulated as a set of parameter constraints and a parameterized least-squares reconstruction error objective function, Le and Deng describe the workflow, or solution pipeline, for example-based skinning decomposition as consisting of initialization and optimization. This research identifies a third subprocess of constraint adherence. Constraint adherence is often a component of optimization. However, since there are a variety

of approaches to constraint adherence, some of which are stand-alone processes, this research considers constraint adherence separately. In all, this research identifies three major components of any skinning decomposition problem:

- ***Initialization*** - Initial computation of animation parameters that reduce the search space and provide a reasonable starting point for the optimization process.
- ***Optimization*** - Optimization approach to the minimization of the reconstruction error function that defines the skinning decomposition problem.
- ***Constraint Adherence*** - Measures taken to honor standard LBS constraints.

Additionally, it is relevant to define the characteristics of standard inputs and outputs for skinning decomposition problems. Skinning decomposition input is typically a series of character meshes captured in a variety of poses. The minimal required pose information for skinning decomposition is vertex connectivity and the vertex positions for the deformed example pose. It is not necessary to define a based pose, from which all other example poses are computed. Any of the example poses could be used. The standard output is the set of weights and transformations (often separated in their rotation and translation components) that best recreate the example poses given as input.

For each component of the skinning decomposition problem there is room for flexibility and creativity when making implementation decisions. The specific implementations are the key components to varied skinning decomposition approaches. The remainder of this chapter discusses popular implementation choices for each component, briefly discusses unique implementations, and compares similarities and differences between this research, standard approaches and the approaches of the four cornerstone skinning decomposition papers mentioned above.

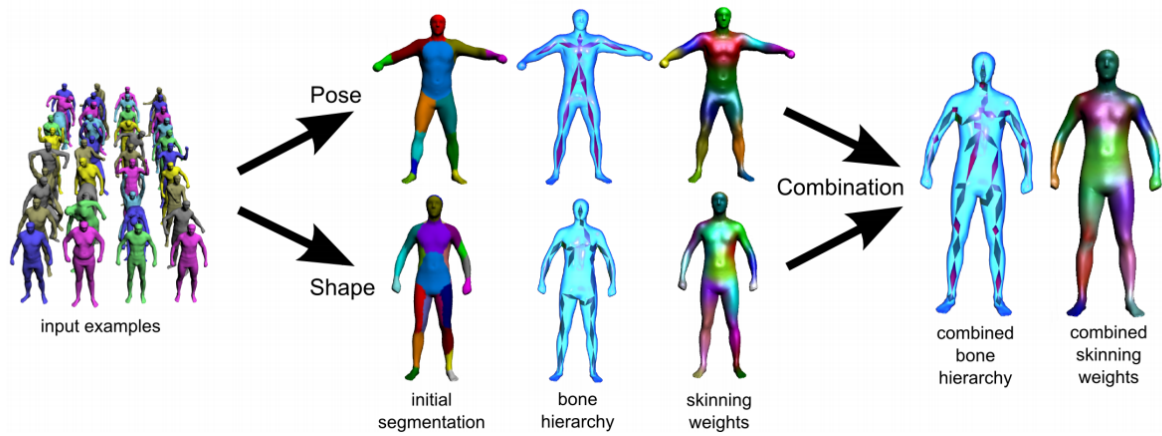


Figure 3.12: Optimization pipeline for Learn Skeletons for Shape and Pose solution to skinning decomposition [34]. Example poses are analyzed for shape and pose bone hierarchies. The bone hierarchies are combined to generate a single skeletal structure that defines that model. Additionally, skinning weights are computed for the hierarchies and combined to yield the final LBS weights.

Problem Structure - The problem structure is the most consistent component of skinning decomposition. In each case skinning decomposition is posed as a constrained optimization problem of computing the underlying animation parameters that best reconstruct the example poses using a basic LBS deformation model. The objective function is often defined by a least squares representation of the vertex reconstruction error. The goal of the objective function is to find the animation parameters that minimizes the difference between each computed vertex position and the corresponding vertex positions given by the initial example poses.

The objective function may be normalized by the number of vertices, poses or dimensions, or some combination of the three to get a approximation or reconstruction error value for the model. The method of normalization is a unique characteristic of problem structure that varies between skinning decomposition implementations and must be accounted for when comparing techniques. The SMA approach defines the approximation error as a percentage of

average distortion by normalizing by average reconstruction error [49]. Similar to this work, FSD defines the model reconstruction error on a normalized per element basis. Computed error values are normalized by the total number of individual elements of the animation parameter matrix, in this case the number of spatial dimensions (three) times the number of vertices times the number of poses [54]. SSDR uses the same normalization metrics [57]. Both implementations scale the normalized error metric by 1000 for convenience. LSSP uses a straightforward residual RMSE that is normalized by the bounding volume diagonal [34]. Regardless of technique, normalization aids in comprehension and comparison of data.

Initialization - Initialization of the parameters used in the optimization function has significant implications on optimization algorithm efficiency. The basic approach is to use some form of clustering of either vertex or triangle mesh geometry to compute a rough set of bone groups, often referred to as proxy bones. These approximations may be used to further refine the initialization using some iterative approach, such as in the works of Kavan et al (FSD)[54] and Le and Deng (SSDR) [57]. Alternately, SMA computes first pass proxy bones that are used directly in the skinning decomposition process [49].

It is important to note that initialization often includes any necessary preprocessing of the data. Common preprocessing activities that occur prior to initialization are dimension reduction, mesh triangulation and inter-pose vertex correspondence for non-uniform input meshes. FSD performs dimension reduction by computing a set of representative vertices that can be used to represent a larger collection of vertices whose positions can be computed from linear combinations of the representative vertex trajectories [54]. Only the representative vertices are used for optimization thereby reducing the dimensionality of the skinning decomposition problem.

Optimization - Typically, standard constrained optimization techniques are used for

minimization of the objective function. The constrained minimization sub-topic of optimization is expansive. While discussing a variety of minimization approaches, Press et al suggest that, although there is no reasonable means to find the perfect optimization algorithm, informed decisions of which approach to choose can be made by considering the characteristics of the problem, such as problem structure, constraints, dimensionality, and the ability to compute the derivative of the objective function [81].

Considerations and trade-offs to evaluate include the scale of memory and storage requirements, the ability to compute the derivative of the objective function and the ability to compute solutions to subproblems of the overall optimization task, amongst other qualities. This research chooses to distinguish between direct and indirect optimization algorithms for skinning decomposition. Specifically, this research covers the gradient and block coordinate descent indirect methods as well as three popular matrix decomposition approaches (non-negative least squares, singular value decomposition and Cholesky decomposition) that are commonly used to compute skinning decomposition solutions directly.

Gradient descent is a fundamental optimization algorithm that serves as the basis for other more complex optimization algorithms. Basic gradient descent provides a means for finding a local minimum of a function by repeatedly stepping in the direction of the negative gradient until the gradient converges to zero (or near zero).

$$\mathbf{x}' = \mathbf{x} - \gamma \nabla f(\mathbf{x}) \tag{3.3}$$

Gradient descent methods assert that, given a function, $f(\mathbf{x})$, that is defined at a point, \mathbf{x} , a local minimum for the function can be computed by repeatedly taking a step, γ , in the direction of the negative gradient, $-\nabla f(\mathbf{x})$, until the difference in the function value at the

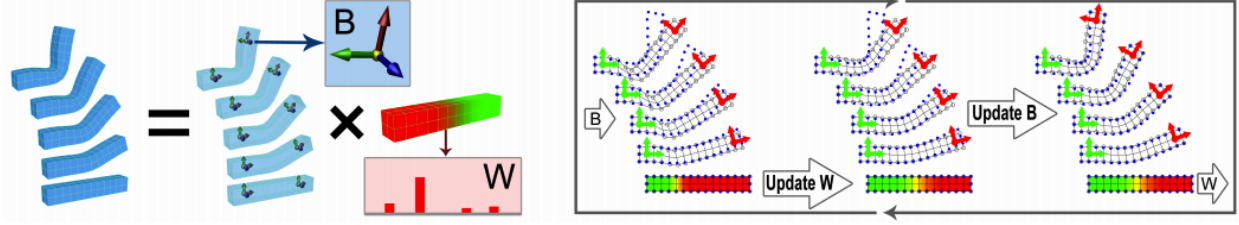


Figure 3.13: Smooth Skinning Decomposition with Rigid Bones (SSDR) incorporates block coordinate descent which alternates between solving for either bones (B) or weights (W), holding the other constant while solving [57].

current point and the function value at the previous point is negligible.

For gradient descent, the derivative of the objective function may be determined either directly in the form of partial derivatives taken with respect to each parameter element or by finite difference estimation [110].

$$\nabla f(\mathbf{x}) = \frac{f(\mathbf{x} + \mathbf{h}) - f(\mathbf{x})}{h} \quad (3.4)$$

One variation on gradient descent involves adjusting the step size after each iteration to take advantage of successful steps and allow the algorithm to converge more quickly. Another common gradient descent variation uses the conjugate gradient method. For conjugate gradient descent instead of taking infinitely many small steps in the same general direction to reach the minimum, you take a calculated few steps such that each step is orthogonal to the last and the length of each step is maximized so that the total number of steps taken is greatly reduced [91]. The choice of step size and direction forms the basis of many variations of gradient descent algorithms.

One of the major challenges with example-based skinning decomposition is the interdependent nature of the LBS animation parameters and their competing constraints. SSDR uses block coordinate descent to address this problem [57]. Block coordinate descent is an

iterative, alternating minimization combination of least squares methods that does not inherently require gradient calculations. Block coordinate descent iteratively computes optimal weights, holding transforms constant and then computes optimal transforms, blocking the weights and holding them constant. This process is repeated until the result converges on a solution. SSDR uses the Active Set Method (ASM) with bounded constraints to optimize the skinning weights and Singular Value Decomposition (SVD) to compute optimal rotations during each iteration. ASM and SVD are chosen based on the properties of each subproblem. ASM is a common method for computing solutions to systems of equations and parametric programming problems in which the solution to the problem is a parameterization of another problem with a known (or easily computed) solution. Readers are referred to [26] for a detailed explanation of Active Set Methods.

Matrix decomposition techniques are the most popular direct optimization approaches to skinning decomposition problems. The matrix-based problem structure of standard LBS models lends itself well to matrix decomposition solutions. In addition to the SSDR work of Le and Deng [57], Hasler et al [34] also use SVD to compute the bone transformations in their LSSP approach. FSD, by Kavan et al, uses Cholesky decomposition to optimize bone matrices[54].

Matrix decomposition is the parameterization (or factoring) of matrices. Matrix decomposition provides a means for solving systems of equations by parameterizing (or factoring) matrices. The parameterization of matrices resulting from decomposition simplifies the problem and provides more efficient solutions. The choice of which matrix decomposition method to use for optimization often depends on the structure of underlying problem for which a solution is being attempted. Readers are referred to *Fundamentals of Matrix Computations* for an in-depth explanation of matrix decomposition [107].

Constraint Adherence - Constraint Adherence is often handled as an element of the optimization algorithm. Because many of the weighting subproblem implementations solve a constrained system of equations using a least-squares approach at some point during the smooth skinning process, bounded or constrained optimization methods are a natural component of skinning decomposition, such as in the work of James and Twigg [49], Rhee et al [83], and Schaefer and Yuksel [87]. Nonnegative least squares (NNLS) is commonly employed to compute weights that adhere to the non-negative bounding conditions of the skinning decomposition problem.

SMA uses NNLS to compute vertex weights in their SMA approach to skinning decomposition [49]. Similarly, FSD uses NNLS as a component of their vertex weight optimization process [54]. Although many optimization algorithms include some constraint adherence functionality, such as ensuring bounded solutions, often basic constraint adherence is not sufficient to address all of the constraints for a given problem.

To offer further assurance of constraint adherence, additional methods are needed, often implemented in conjunction with optimization implementations that include an initial level of constraints. Some implementations are elegant considerations of competing constraints, while other implementations compute optimal solutions without regard to the constraints and then adjust the computed solution to adhere to the constraints ex post facto.

LSSP introduces an additional constraint in the form of the L1-norm to enforce sparsity [34]. While this addresses the limitation on the number of bones that influence a vertex, it does not address the summation constraint. Therefore, the results adhering to the L1-norm constraint are subsequently normalized to achieve a solution that also adheres to the summation constraint.

Constraints may also be considered soft constraints that are not required to be strictly

followed, allowing for more flexibility in the solution. The James and Twigg SMA implementation [49] and the FSD work of Kavan et al [54] both implement soft constraints for vertex weights, which allows loosely bound NNLS and orthonormalized factorization to compute weights that adhere to the non-negative constraint, but may or may not adhere to the summation constraint. Optimization solutions that adhere to all given constraints simultaneously typically require additional computational resources and more complex algorithms to find a solution, but produce better results.

	Skinning Mesh Animations (SMA) James and Twigg 2005	Fast and Efficient Skinning of Animated Meshes (FSD) Kavan et al. 2010	Learning Skeleton for Shape and Pose (LSSP) Hasler et al. 2010	Smooth Skinning Decomposition with Rigid Bones (SSDR) Le and Deng 2012	Example-based Parameterization of Linear Blend Skinning for Skinning Decomposition (EP-LBS)
Problem Structure	Standard LBS Model	Standard LBS Model	Standard LBS Model	Standard LBS Model	Standard LBS Model w/ Quaternion Rotations
Constraints	Non-negative Weights Weight Summation (soft) Sparse Weights	Non-negative Weights Weight Summation Sparse Weights	Non-negative Weights Weight Summation Sparse Weights L-1 Norm of Weights	Non-negative Weights Weight Summation Sparse Weights	Non-negative Weights Weight Summation Quaternion Summation
Preprocessing	None	None	Encode Bind Shapes for All Example Models Compute Mean Shape	None	None
Initialization	Mean Shift Clustering of Rotation Sequences	Multiple-source, Region-growing Clustering	Template-based Spectral Clustering	K-means Clustering	Iterative Patch-based K-means clustering of transformation sequences NNLS
Optimization	TSVD NNLS	Modified Gram-Schmidt Orthonormalization Alternating Least Squares Iterative Coordinate Descent	SVD: Levenberg-Marquardt algorithm Alternating Optimization	Alternating Block Coordinate Descent: Active Set Method SVD	Gradient Descent
Constraint Adherence	Component of Optimization Normalization Soft Constraints	Component of Optimization Discarding Data	Normalization and Renormalization	Component of Optimization	Parameter Mapping

Table 3.1: Comparison of Skinning Decomposition approaches.
(Abbreviations: NNLS - Non-negative Linear Least Squares, SVD - Singular Value Decomposition, TSVD - Truncated SVD)

3.4.2.1 This Research and Popular Skinning Decomposition Methods

Similar to SMA, FSD, LSSP and SSDR, this research formulates the skinning decomposition problem as the minimization of the reconstruction error determined by animation parameters and the LBS deformation model. However, the research presented in this thesis modifies the standard LBS model in skinning decomposition to use quaternion rotations. Quaternion rotations in vector form require the calculation of fewer animation parameters than matrix rotations. This change in rotation representation allows the optimization algorithm to execute more effectively by taking advantage of the reduced dimensions of rotation vectors.

The standard LBS model is restricted by constraints on the animation parameters. Valid vertex weights are defined by non-negativity and summation constraints. Additionally, vertices are often limited in the number of bones that may influence its movement by way of a sparseness or influence constraint. This research adheres to the summation and non-negativity constraints. This research does not implement a sparseness constraint. However, the addition of quaternions requires an additional summation constraint to ensure valid quaternions are used. The quaternion summation constraint is similar to the summation constraint for vertex weights.

The basis of all example-based skinning decomposition initialization is a clustering algorithm, clustering vertices, geometry or rotation sequences. Common methods for clustering include mean shift, spectral and k-means. This research uses k-means clustering similar to SSDR. However, this research chooses to cluster transformations rather than geometry. LSSP also clusters transformations. The major difference between the transform clustering performed on LSSP and that performed for this research is the element that is being transformed. This research clusters vertex transforms while LSSP clusters triangle transforms.

Similar to the preprocessing done in FSD and SDDR, this research scales all models to a unit cube prior to initialization.

Constraint adherence can be handled as an built-in element of the optimization algorithm or the problem can be structured in a way that considers constraints as an added element to the solution, either during optimization or afterwards. NNLS, used by SMA, inherently adheres to the non-negative constraint and the Active Set Method (ASM) and SVD, allow for the inclusion of constraints when defining the optimization problem. Conflicting constraints are often handled by normalization as in the LSSP implementation. Although multiple constraints often make optimization more challenging, this research reduces the impact of the constraints by defining value mappings to convert animation parameters into a space of valid constraint animation parameter values during each optimization iteration. The mapping technique utilized in EP-LBS makes possible the use of basic gradient descent methods to implement a skinning decomposition solution that adheres to standard LBS constraints, yet does not requires complex optimization algorithms to solve.

This research uses basic gradient descent optimization. Many skinning decomposition approaches choose to alternate between optimization of the two basic subproblems computing optimal vertex weights and computing the ideal bone transformations, as is done with SDDR. This research chooses to solve for optimal weights and transforms simultaneously.

This research is a unified solution to computing both transformation and weight animation parameters while adhering to all LBS deformation constraints. This research presents an alternate initialization approach that determines the number of bones using only the example data and introduces new parameter mapping algorithms that guarantee non-conflicting constraint adherence throughout the iterative optimization process.

Chapter 4

Example-based Parameterization of Linear Blend Skinning

Given the basic mathematical model for Linear Blend Skinning, the research presented in this thesis refactors the basic LBS model, defines a new motion-based clustering technique and introduces parameter mapping for constraint adherence. The refactored model and the new clustering and constraint adherence methods address many of the usability and efficiency challenges of the standard LBS model.

This chapter details the development and implementation of the major contributions of this research. First, the development of the parameter mapping method and the alternate LBS model are explained. Next with the introduction of a modified LBS model, a method for motion-based clustering is introduced to identify bones in the model. Finally, the challenge of computing animation parameters from a set of example data is structured as a constrained optimization problem and the chapter closes with a detailed description of the objective function, initialization methods and optimization approach used to solve the constrained optimization problem.

4.1 Process Overview

To guide the discussion for the remainder of the chapter, the process developed and introduced through the work of this thesis is succinctly described as follows:

The method presented in this thesis takes example poses as input to a least-squares non-linear optimization process and institutes a single constrained optimization equation that allows the simultaneous computation of all animation parameters for the model. An iterative clustering methodology is used to construct an initial parameterization estimate for the model, which is then subject to a non-linear optimization system that utilizes parameter mapping to map the constraints to a domain suitable for nonlinear minimization. The result is an iterative example-based LBS solution that improves the fitting of the initial parameterization to a solution that closely models the example data and adheres to standard LBS constraints.

4.2 Development of Methods

To inform the discussion of Example-based Parameterization of Linear Blend Skinning, this section details the development of the modified mathematical model for LBS, the new bone clustering method for skinning decomposition initialization and the newly introduced parameter mapping methods that serve as the underlying mechanisms for improved example-based computation of animation parameters. Each of these methods serve as major contributions of this thesis.

Inputs - A *3D model*, specifically a set of 3D character meshes, each expressed as a set of vertices and faces, as captured in a *series of poses*. One of the poses is selected as the base pose, from which all other poses can be generated.

Outputs - Animation parameters needed to convert the base pose to each of the remaining example poses for the input model. Specifically,

1. *Bone Transformations* for each bone in each pose
2. *Skinning Weights* for each vertex in the model

Process - A Skinning Decomposition Pipeline consisting of three major components, initialization, optimization and constraint adherence, used to compute animation parameters from a set of example data using optimization techniques. The pipeline process introduced by this thesis is as follows:

1. *Iterative Two-stage Clustering Initialization*
2. *Gradient Descent Optimization*
3. *Parameter Mapping for Constraint Adherence*

To inform the optimization process, an *initialization* process is performed to provide an estimation of weights and transforms as computed using patch-based clustering and NNLS. *Gradient descent optimization* with *parameter mapping* is used to compute the weights and transforms needed to recreate the input example poses.

Figure 4.1: Example-based Parameterization of Linear Blend Skinning for 3D Skeletal Animation, a skinning decomposition pipeline process.

4.2.1 Modified Linear Blend Skinning

This research presents a modified LBS model which uses a combination of Euler angles and quaternions to represent rotations, allowing for the expression of both fine and gross rotations of a bone during skinning decomposition. The LBS equation originally presented as Equation (2.10) can be modified as shown in Equation (4.1) to reflect both the large magnitude bone rotations that generally orient the bone for a particular pose and the small bone movements that refine the overall bone position at each iteration of the optimization algorithm during skinning decomposition.

$$\mathbf{v}'_i = \sum_j w_{i,j} (\mathbf{Q}_j \mathbf{e}_j \mathbf{v}_i + \mathbf{s}_j) \quad (4.1)$$

The Euler angle bone rotation for bone j is identified by \mathbf{e}_j . Quaternion rotations for a bone continue to be identified by \mathbf{Q}_j . At each iteration, the modified LBS equation is subject only to the following boundary conditions:

$$0 \leq w_{ij} \leq 1 \quad (4.2)$$

$$-\pi \leq e_\phi, e_\theta, e_\psi \leq \pi \quad (4.3)$$

Equation (4.3) is added to ensure valid Euler values, where the subscripts ϕ , θ and ψ refer to the elements of euler angle vector.

Although the standard LBS summation constraint is not explicitly indicated, the skinning decomposition methods presented in this research maintain adherence to this fundamental constraint. The re-factored LBS model presented in Equations (4.1) - (4.3) are used in

conjunction with the newly introduced weight and rotation parameter mapping to ensure constraint adherence for all LBS parameters.

4.2.2 Parameter Mapping for Constraint Adherence

Computing solutions to an LBS objective function that adheres to a given set of constraints is a major challenge for optimization algorithms. LBS constraints include simple boundary conditions as well as more complex linear and non-linear constraints, such as summation constraints for weights or normalization requirements for quaternions. The most basic constrained optimization algorithms rely heavily on simple boundary conditions to constrain parameter values. More complex algorithms often build on simple bounded optimization approaches to achieve constraint-adherent solutions as well. This is due, in great part, to the relatively cheap computational expense of complying with boundary conditions compared to honoring other linear and nonlinear constraint equations.

Iterative optimization methods, such as gradient descent, present an additional challenge to constraint adherence because iterations that minimize the objective function may result in values that do not adhere to parameter constraints. Standard processes to ensure constraint adherence at each iteration often increase the computational complexity of the optimization algorithm.

Still, constraint equations often infer certain implicit boundary conditions that can be exploited to improve optimization efficiency. Although not every objective function parameter has both explicit upper and explicit lower bounds on the parameter values, the combination of all constraints for a single parameter value often lead to implied upper or lower limits for the missing bounds.

This research leverages implied boundary conditions to restrict the search space for the

optimization algorithm. Complex linear and non-linear constraint equations are removed as explicit constraints and are instead addressed with a new parameter mapping process.

4.2.2.1 Implied Bounds from Competing Constraints

For weights, the sum-to-one constraint combined with a nonnegative boundary condition implies that valid weight values not only have a lower bound of zero, but also an upper bound of one. The weight constraints listed in Chapter 2 are repeated here for convenience.

$$0 \leq w_{ij} \tag{4.4}$$

$$\sum_j w_{ij} = 1 \tag{4.5}$$

Because bounds offer optimization algorithms greater flexibility, this research explicitly enforces the upper bound implied by the sum-to-one constraint. Given the non-negative bounding condition and the sum-to-one constraint, both an upper and a lower bound are enforced for the weights.

$$0 \leq w_{ij} \leq 1 \tag{4.6}$$

This is a necessary condition for weights, but not sufficient to ensure the sum-to-one constraint is satisfied. Although solutions that sum to one are possible within these bounds, optimization algorithms with only upper and lower bound specifications may yield a numerically optimal solution in which the weights for a vertex sum to a value other than one. In fact, only a subset of the weights generated that satisfy Equation (4.6) also satisfy the sum-to-one constraint presented in Equation (4.5). To address this conflict, parameter mapping

of weights is introduced as a new method that maps a simpler range-bounded space to the more complex weight space. The combination of the implicit boundary condition, deductive reasoning and weight mapping eliminate the need for the summation constraint for weights in the mapped space.

4.2.2.2 Weight Mapping

The general approach for iterative optimization methods is to compute a set of potential parameters that adhere to given bounds and constraints and evaluate the objective function for solution convergence given those parameter values. If the resulting solution is not convergent the process is repeated.

The vector of weights computed by an iteration of the optimization algorithm form a set of free variables that can be iteratively mapped to new constraint-adherent values determined by the parameter mapping in (4.7) through (4.9).

$$0 \leq w_{ij} \leq 1 \tag{4.7}$$

$$w'_1 = w_1 \tag{4.8}$$

$$w'_j = w_j \left(1 - \sum_{k=1}^{j-1} w'_k \right) \tag{4.9}$$

Additionally, the final weight is computed via deductive reasoning from the mapped values:

$$w'_J = 1 - \sum_{j=1}^{J-1} w_j \quad (4.10)$$

In these equations, w'_i are the actual weights utilized in the LBS algorithm, while w_i represent the mapped weights subject to non-linear optimization. The presented parameter mapping algorithm proceeds in a progressive, cascading manner. The first free variable remains unchanged as the original computed weight value (Equation (4.8)). For each subsequent free variable, the sum of all previously computed variables is subtracted from one (Equation (4.9)). This ensures adherence to the summation constraint propagates throughout the mapping process. The resulting difference is then multiplied by parameter value to obtain a new constraint-adherent mapped weight value value (Equation (4.9)).

The multiplicative properties of the bounded values, guarantees that the multiplication of any two values within the bounds of zero and one results in a new value that is also bounded by zero and one. Finally, the last free variable is compute as the sum of all previous values subtracted from one (Equation (4.10)). In this manner, free variables representing weights are mapped to a set of values that adhere to all standard LBS constraints.

Index k is introduced to represent weights that have already been mapped and J represents the final weight subject to the save-one weight computation. Using these indices, free variables map bounded weight values to a constrained search space for the optimization algorithm. Still, the mapping alone will not guarantee the sum-to-one condition. It only guarantees a set of weights whose sum is less than or equal to one. Therefore, the save-one technique is used in combination with mapping to compute weights that satisfy both the boundary conditions and the summation constraint.

The mapping process provides a means for transforming the weight values to new val-

ues within a search space of valid weight values during each iteration of the optimization algorithm. Traditional optimization algorithm approaches require the computation of values that simultaneously adhere to all constraints. This can be costly and is difficult when the terms are linearly dependent. Instead, this thesis introduces parameter mapping which employs a modest number of simple linear equations in a cascading fashion to compute new weight values at each iteration. Although it eliminates the need for computation of simultaneous constraint adherence, the presented mapping algorithm uses successive operations to compute a set of values that adhere to all standard LBS constraints.

As the weights are mapped into a search space of valid LBS weights, there must also be operations to convert mapped values back to their respective values in real number space. The vector of mapped weights can be inverted with inverse weight mapping defined in Equations 4.11 and 4.12.

$$w_1 = w'_1 \tag{4.11}$$

$$w_j = \frac{w'_j}{\left(1 - \sum_{k=1}^{j-1} w'_k\right)} \tag{4.12}$$

Implicit boundaries, deductive reasoning and a new value mapping technique that exploits the mathematical properties of the bounded values are combined to transform computed weights to a set of free variables that constrain the optimization search space to a set of values that only map to valid weights, adhering to the summation constraint. Appendix B offers mathematical proof of mapping constraint adherence properties.

As an example, consider the case of three weight values w_1 , w_2 , and w_3 . Weights are constrained to sum to one:

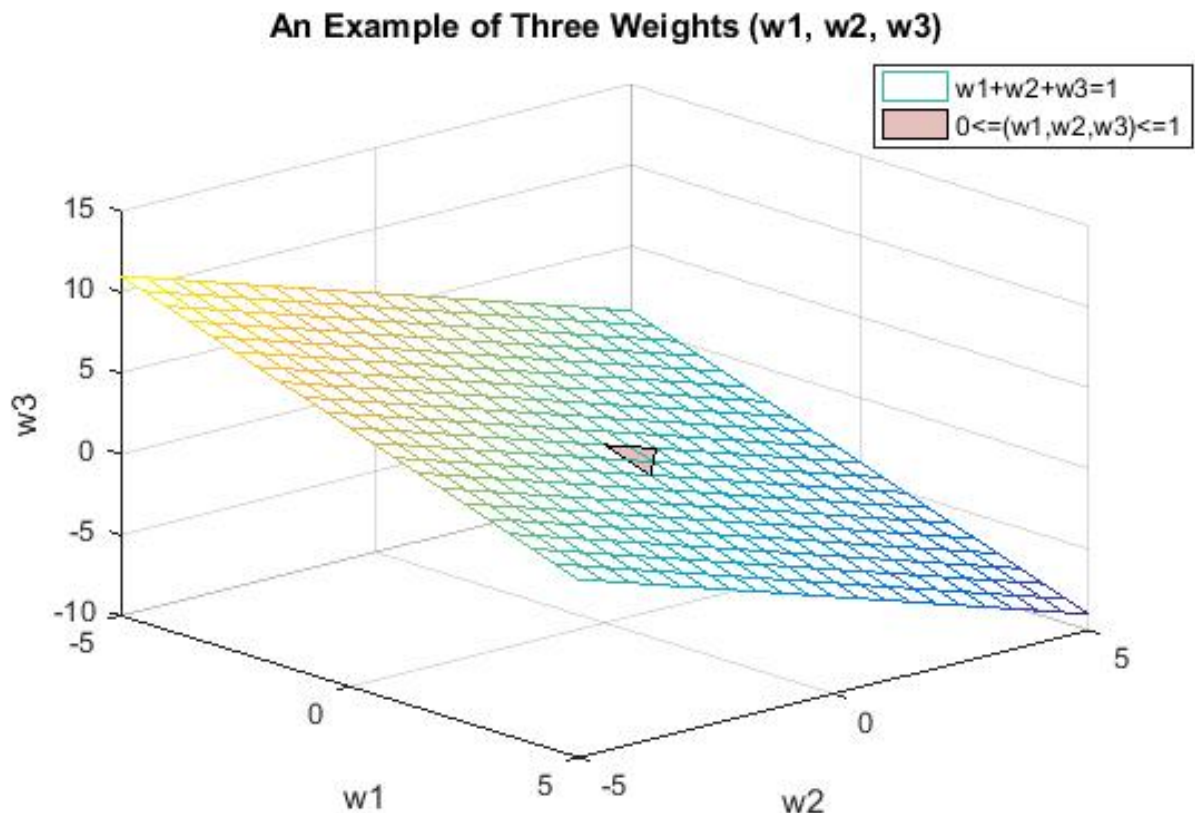


Figure 4.2: Plane containing weights w_1 , w_2 , and w_3 that sum to one with the triangular region of plane bounded to the range $[0, 1]$ that represents valid weights.

$$w_1 + w_2 + w_3 = 1 \quad (4.13)$$

Therefore, all valid weights line on the plane containing w_1 , w_2 , and w_3 (see Figure 4.2). In addition, weights are bounded to the range $[0, 1]$, which limits the valid range to the surface of a triangle with the three vertices: $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$.

It is difficult to manipulate these values during non-linear minimization due to the correlation between the weights. Changing one weight necessarily entails changing at least one other weight to ensure the constraints are satisfied. Some algorithms take the approach of relaxing the constraints during the non-linear minimization, then re-applying them when the minimization is complete, effectively allowing the weights to deviate from the plane,

2D Projection of Three Constrained Weights

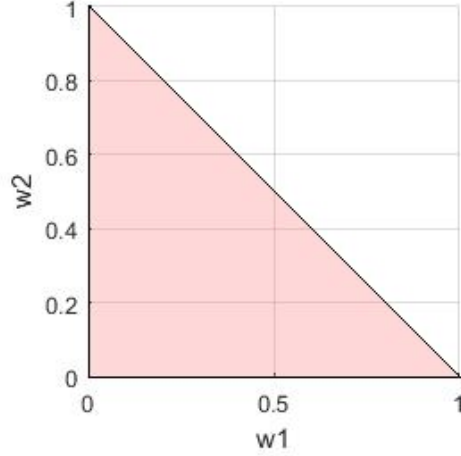


Figure 4.3: The 2D projection of weights w_1 , w_2 , and w_3 representing the region of three valid weights as determined by only two weights w_1 and w_2 .

then renormalizing by finding the nearest point on the plane after the minimization is complete. This is necessarily suboptimal, since the required value is the point on the plane that minimizes the equation.

Since weights sum to one, it is possible to omit one weight from the computation, since $w_3 = 1 - w_1 - w_2$. However, this simply modifies the constraint from finding a point on the three-dimensional triangle in space to finding a point on the two dimensional projection of the triangle:

The coupling between the free variables remains, only the constraint has changed to $w_1 + w_2 \leq 1$. This is evident from the shape of the triangle.

Weight mapping replaces w_1 and w_2 with two new variables u_1 and u_2 that has the relaxed constraint:

$$0 \leq u_1, u_2 \leq 1 \quad (4.14)$$

The actual weights to use are computed from these new terms using these equations, which represent the J=3 cases of equations 4.8 and 4.9:

$$w_1 = u_1 \quad (4.15)$$

$$w_2 = u_2(1 - u_1) \quad (4.16)$$

Now, u_2 in the range $[0,1]$ is multiplied by $(1 - u_1)$, which determines a resulting space that is the triangle from figure 4.3. The space of (u_1, u_2) has a one-to-one mapping onto (w_2, w_1) up to the limit as u_1 tends to one, and is mapping from the square, where the bounds are simply $[0,1]$ to the triangular space, effectively decoupling the two terms and greatly simplifying the constraint.

4.2.2.3 Rotation Mapping

Quaternions also bear implied boundary conditions as a result of the summation constraint (repeated here as Equation (4.17) for convenience).

$$q_w^2 + q_x^2 + q_y^2 + q_z^2 = 1 \quad (4.17)$$

The mathematical properties for valid quaternion values defined by Equation (4.17), implies that values must fall in the range of $(-1, 1)$, which supports the cascading parameter mapping approach introduced for weight mapping. The product of two values in the bounded range for valid quaternions produces a new value that also adheres to the boundary condi-

tions. Unfortunately, applying deductive reasoning to quaternions values does not result in unique solutions due to the non-linear summation constraint. The computation of the final quaternion value would require the computation of a square root term, which results in two potential values, yielding a non-unique solution.

As an alternative, Euler angles are used in composition with quaternions. Euler angle rotations can also be represented by a vector of values. The vector values represent the three coordinate rotations performed about pre-defined frames of reference in succession; an initial rotation ψ about an initial k-axis, a second rotation angle θ about a j-axis, and a final rotation ϕ about a i-axis. The determination of the i, j, k axes in relation to the x, y, z axes are determined by convention. This research uses the $z - x - z$ convention, known as the "x-convention" or proper Euler angles [111].

$$\mathbf{e} = [e_\phi, e_\theta, e_\psi] \quad (4.18)$$

By definition, the elements of the Euler angle vector are bounded by both minimum and maximum values. Each element of Equation (4.18) falls in the range $[-\pi, \pi]$.

$$-\pi \leq e_\phi, e_\theta, e_\psi \leq \pi \quad (4.19)$$

The free variables associated with Euler angles are not subject to any summation constraint and, therefore, can simply be mapped to generate new values for each optimization iteration. First, each Euler angle is mapped to its corresponding value in the range $[0, 1]$, \mathbf{e}' (Equation 4.20).

$$\mathbf{e}' = \frac{\mathbf{e} + \pi}{2\pi} \quad (4.20)$$

Much like the mapped weights, inverse operations are defined for rotation mapping as well. The bounds are enforced (\mathbf{e}_b'') prior to unmapping the Euler values. Equations 4.21 through 4.22 are used to convert mapped Euler values to their original range of $[-\pi, \pi]$.

$$\mathbf{e}'' = \begin{cases} 0 & \text{if } \mathbf{e}' \leq 0 \\ \mathbf{e}' & \text{if } 0 > \mathbf{e}' > 1 \\ 1 & \text{if } \mathbf{e}' \geq 1 \end{cases} \quad (4.21)$$

$$\mathbf{e} = (2\pi\mathbf{e}_b'') - \pi \quad (4.22)$$

The number of values required to represent intermediate bone rotations for EP-LBS is ideal in comparison to matrix or quaternion rotation representations. Although either Euler angles or quaternions alone can be used to fully represent a rotation, there are conditions under which each form of rotation fails to produce the desired results. Euler angles suffer from gimbal lock, particularly when used to compute rotations of large magnitude [24]. However, when used to describe small, incremental rotations, gimbal lock becomes a negligible concern. This is why proper initialization of the bone transforms to a value close to optimum is essential for making this method work. Quaternions, when used with the save-one summation solutions, generate indeterminate solutions as result of the square root calculation required when computing the final value of a save-one solution that satisfies the constraint equation.

Because both the nonnegative square root and the negative square root are both valid, this results in an ambiguous solution. See Appendix C for detailed mathematical proof of rotation mapping constraint adherence.

4.2.3 Iterative Motion-based Clustering for Initialization

Like all optimization algorithms, the ability to find a reasonable minimum for the objective function is determined in great part by the choice of an initial starting point [81]. An ideal initial search space that has been calculated near the solution is more likely to converge on a solution faster and is less likely to terminate at a suboptimal local minimum. For skinning decomposition, initialization amounts to determining animation parameters (vertex-bone associations, bone rotations and skinning weights) that roughly approximate the general shape of each pose.

Most skinning decomposition solutions employ bone clustering as the central component to determining the initial animation parameters for the optimization algorithm. Clustering vertices or polygons is the imperative first step in the initialization process that computes the vertex-bone associations needed to identify initial bone groups which can then be used to compute initial bone rotations and skinning weights. Although the LBS model allows for multiple bones to influence a single vertex, singleton vertex-bone associations are computed for the initial bone clustering.

Bone clustering is not a trivial task in the initialization process, however. A number of features of the skinning decomposition problem, such as the high-dimensionality of multi-pose data sets and interdependent animation parameters, complicate the determination of bone groups using standard clustering methods. This makes clustering across all poses challenging. Although some clustering approaches cluster based on a single pose to mitigate

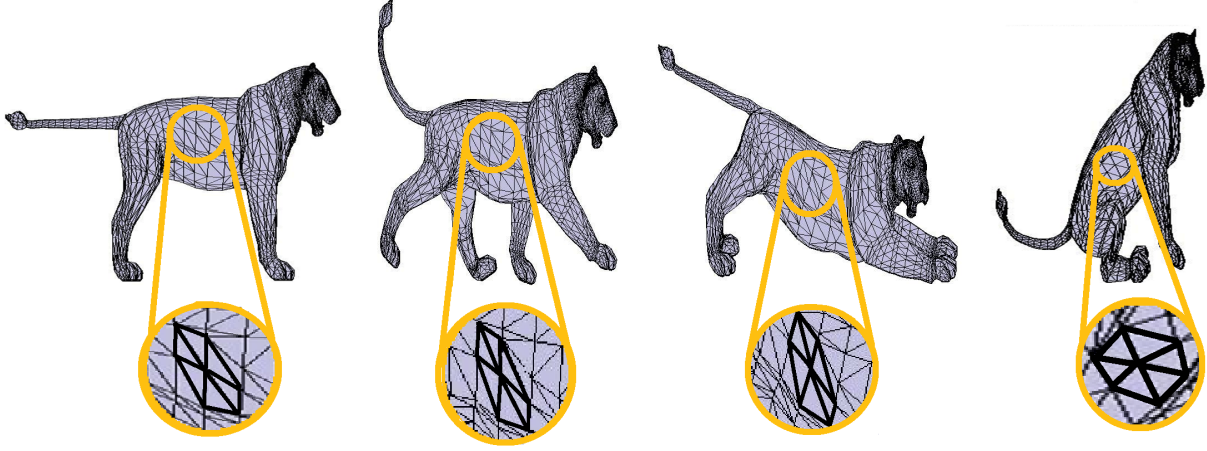


Figure 4.4: 1-ring clusters for a single vertex across multiple poses for a lion model.

this challenge, many initialization approaches are further complicated by the need to know the number of bones in a model prior to clustering the data. This research introduces a bone clustering initialization method for example-based LBS problems that does not require a priori knowledge of the number of bones in the model and offers improved initialization by clustering across all poses.

4.2.3.1 Representing High-dimensional Data Sets as Vectors

The basic approach to the EP-LBS bone clustering method is to compare the movement of vertices, clustering vertices that move in a similar manner across all poses, forming initial vertex-bone associations. The key to clustering across poses is the ability to represent all pose data for a single vertex in a single feature vector. To inform the discussion of feature vectors, some basic set notation for this problem is explored first.

EP-LBS defines \mathcal{V}_p as the set of I vertices in each example pose p of the sequence of P character meshes. \mathcal{V}_p , written as:

$$\mathcal{V}_p = \{v_{1p}, v_{2p}, \dots, v_{ip}, \dots, v_{Ip}, \}, \forall p \quad (4.23)$$

With this basic set notation, EP-LBS defines $\bar{\mathcal{V}}_i$ the set of first-order vertex neighbors of vertex v_i . First-order neighbors are the vertices immediately adjacent to v_i forming a single connected component, the connectivity is indicated by the specified geometry. The position of the vertices of all example poses are given as input to the EP-LBS skinning decomposition problem. As a result the first-order neighborhood for each vertex can be computed via simple traversal of a character mesh. The set of neighbors $\bar{\mathcal{V}}_i$ is written as:

$$\bar{\mathcal{V}}_i = \{(v_{i0}), (v_{i1}), (v_{i2}), \dots, (v_{ik}), \dots, (v_{iK}), \} \quad (4.24)$$

where K is the number of vertices immediately adjacent to v_i and v_{i0} is simply vertex v_i itself. It is important to note that this research assumes point correspondence and identical, non-varying topology across all poses. Because the connectivity remains the same for each example pose in the sequence, the number of vertices I is constant whatever pose p is selected. Additionally, as a result of fixed connectivity, the set $\bar{\mathcal{V}}_i$ is identical for all example poses p .

The set of P first-order vertex neighbors for a vertex i could be chained together and considered a feature vector for v_i across all poses. However, this feature vector would be a collection of vertex positions, whose dot product would represent the rotational motion between normalized vertex positions but would not capture the individual rotational and translational components of movement between poses. To capture both elements of vertex neighborhood movement between poses it is necessary to compute and compare the transformation from the base pose to each example pose for every vertex neighborhood, rather than the vertex positions directly.

The difference in position of a vertex and its first-order neighbors from the base pose to an example pose p can be represented as a standard 4x4 affine transformation matrix \mathbf{M}_{ip} , that can be computed with Least Squares Fitting of Two 3D Point Sets [4]. By decomposing the computed transformation matrix into its quaternion rotation vector $\dot{\mathbf{q}}_{ip}$ and normalized homogeneous translation vector $\bar{\mathbf{s}}_{ip}$ components, the transformation of a vertex neighborhood from the base pose to an pose can be represented in vector form \mathbf{u}_{ip} as a the concatenation of the rotation and translation vectors.

$$\mathbf{u}_{ip} = [\dot{\mathbf{q}}_{ip} \bar{\mathbf{s}}_{ip}] \quad (4.25)$$

Thus, the sets $\bar{\mathcal{V}}_{ip} \forall p$ of first-order neighbors of vertex i across all poses p can be combined and transformed into a single high-dimensional transformation sequence feature vector for vector comparison:

$$\bar{\mathbf{u}}_i = [\mathbf{u}_{i1}, \mathbf{u}_{i2}, \dots, \mathbf{u}_{ip}, \dots, \mathbf{u}_{iP}], \forall p \quad (4.26)$$

The neighborhood transforms are chained together to create a transformation sequence feature vectors that can be clustered and compared with cosine distance measurements.

4.2.3.2 Cosine similarity for Vector Comparison

The trigonometric and geometric properties of vectors can be used to derive meaningful metrics for comparing the similarity of two vectors. One comparison metric, the cosine distance, offers a measurement of similarity of vector orientation for normalized vectors. Cosine distance d_{ab} of two vectors, \mathbf{a} and \mathbf{b} , is defined as one minus the cosine of the angle, θ , between the two vectors. The angle between two vectors can be computed from the dot

product of two normalized vectors and their magnitudes prior to normalization.

$$d_{ab} = 1 - \cos \theta \quad (4.27)$$

$$= 1 - \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}| |\mathbf{b}|} \quad (4.28)$$

This cosine distance metric is neatly bounded in the range $[0, 1]$ inclusive. Vectors with a cosine distance of 0 are perpendicular and vectors with a cosine distance value of 1 represent the same vector (albeit, perhaps, in exact opposite orientations).

EP-LBS leverages the ability of cosine distance metrics to measure vector similarity, even in higher-dimensions, and to compare transformation sequences and identify vertices with similar transformation sequences, indicating similar vertex movement across all poses.

4.2.3.3 Converting Translation Vectors for Cosine Similarity Comparison

Translation vectors in three dimensions must be converted to a form suitable for cosine difference comparison. An additional homogeneous coordinate is used to yield a four element vector that is normalized prior to cosine similarity comparison.

Given a three-dimensional translation vector $\mathbf{s} = [s_x, s_y, s_z]$, the four-dimensional homogeneous coordinate representation of \mathbf{s}' is $[1, s_x, s_y, s_z]$. The normalized translation vector used for cosine similarity comparison is therefore

$$\bar{\mathbf{s}} = \frac{\mathbf{s}'}{|\mathbf{s}|} \quad (4.29)$$

The normalized homogeneous translation coordinates now form a normalized four-dimensional vector, $\bar{\mathbf{s}}$ the can be used with cosine similarity comparison for initialization.

4.2.3.4 Basic Initialization

Given a means for representing all pose data for a single vertex in one feature vector and a method for comparing the similarity of these feature vectors, it is now possible to use basic clustering methods to group similar feature vectors. K -means is a common clustering technique that works for a wide variety of data and similarity metrics. In this case, k -means clusters vertex feature vectors into k groups based on cosine distance of the feature vectors. The computed clusters indicate which vertices move in similar manners across all poses and should therefore be associated with a single bone transformation.

K -means clustering yields the initial bone groups. However, the values needed for initialization of LBS are the bone rotations and skinning weights. Fortunately, the computed bone groups are a critical component to computing bone rotations, which can in turn be used to compute initial skinning weights.

Least Squares Fitting is used once more to compute the bone rotations and translations. Each of the bone groups computed by k -means clustering defines a 3D point set, which persists as identical groups of vertices across both base and example poses, much like the neighborhood groups. The transform between the base pose vertices in each of computed bone groups and the corresponding vertices in each of the poses can be computed with Least Squares Fitting, thereby computing the initial bone rotations for EP-LBS.

Given the base pose vertices, each set of example pose vertices and the initial bone rotations for each of the computed bone groups, the problem of computing the initial skinning weights can be structured as an inverse problem. Standard non-negative least squares (NNLS) methods for solving inverse problems can be used to find loosely bounded solutions for the skinning weights. The computed weights can then be normalized to yield a set of

initial skinning weights for EP-LBS that adhere to all standard LBS constraints.

4.2.3.5 NNLS and Iterative Clustering for EP-LBS Initialization

Together the computed bone groups, bone rotations and skinning weights offer a reasonable initial starting point for the EP-LBS optimization algorithm. Still, experimental evaluation of a variety of models indicated that a single pass of clustering tends to fail to independently cluster smaller groups of vertices that move together. This is especially true for appendages such as fingers and feet (see Table 4.5 and 4.1). The inability to distinguish appendages in a single clustering pass can be exaggerated by the use of a relatively small number of poses or poses with limited range of motion. If the example poses do not demonstrate the full range of all possible deformations for each bone, then the algorithm must infer the bones from the given poses.

Computing all the bones and extremities in one pass is possible if the full range of deformation is included in the given example poses. Because this is often not the case, this research implements re-clustering to divide bone groups with large error into two separate bones, allowing the algorithm to refine the clustering and identify extremities such as hands and feet. Furthermore, a desired feature of any example-based LBS problem is the ability to automatically determine the number of bones in a model from only the base and example pose vertex data. EP-LBS uses this basic initialization process and re-clustering iteratively to achieve this desired feature.

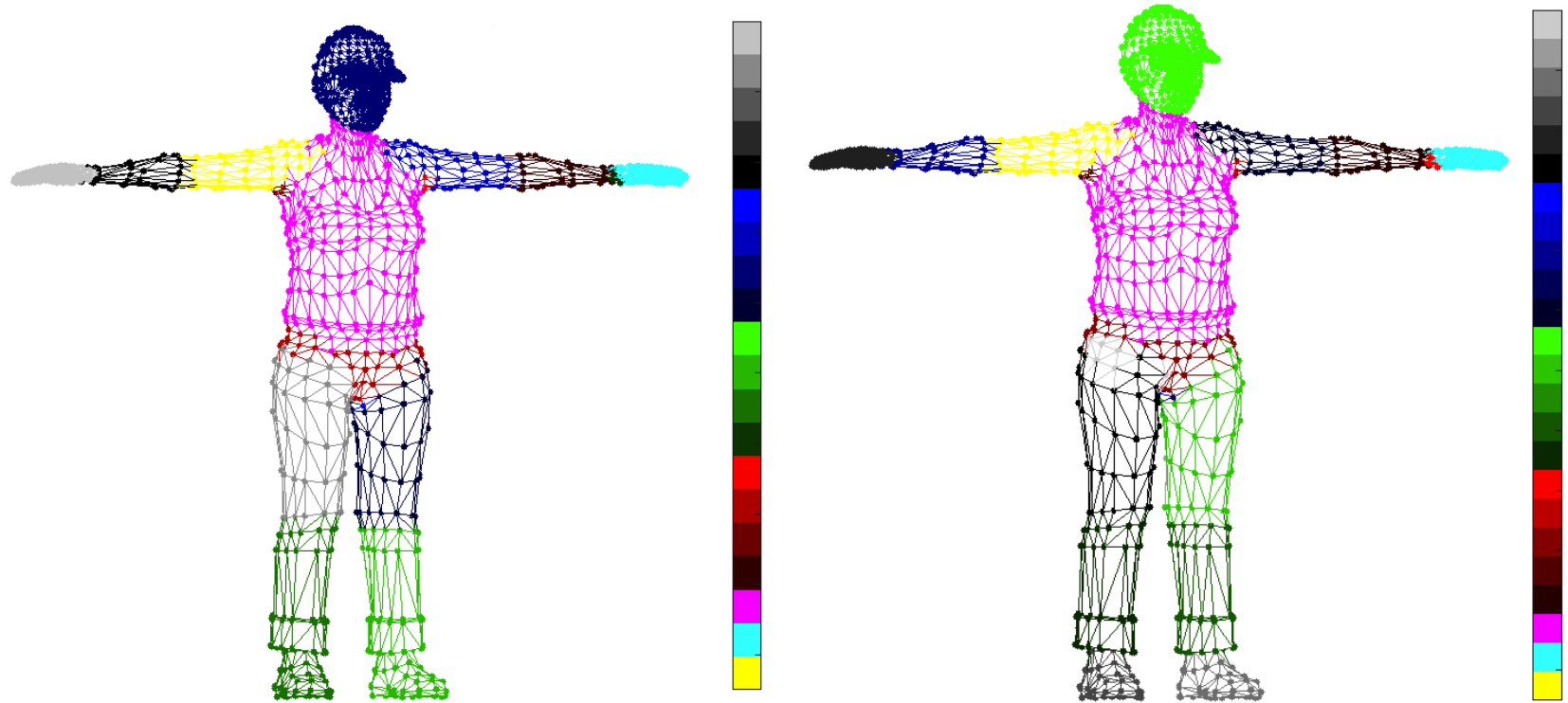


Figure 4.5: Sample Single-pass Reclustering for Initialization for Woman Model. First pass clustering (left) and after reclustering (right). Note that the legs of the first clustering pass group the lower leg and the foot together. After splitting bones and another pass at clustering, the left foot and lower left leg as well as the right leg and right foot are identified by four separate bones instead of two.

Model	Two-Stage Init			Two-Stage + Recluster		
	# Bones	Init Err	Time(s)	# Bones	Init Err	Time(s)
snake	26	2.652×10^{-7}	21.32	37	1.096×10^{-7}	25.11
dance	21	7.607×10^{-6}	19.46	26	4.204×10^{-6}	30.16
horse	29	2.718×10^{-5}	31.57	35	1.223×10^{-5}	61.05
woman	20	5.863×10^{-6}	14.60	24	3.000×10^{-7}	24.15
elephant	24	2.289×10^{-5}	127.5	30	1.486×10^{-5}	360.2
flamingo	10	2.090×10^{-4}	38.68	12	9.502×10^{-5}	71.50
jump	21	1.698×10^{-4}	32.44	25	8.312×10^{-5}	139.9

Table 4.1: Single-pass Reclustering Sample- Demonstration of improvement in error when a single reclustering pass is used.

The value of k used in the k -means clustering determines the number of bones in the model and must be supplied by the user. In effort to generalize the process to work with limited a priori knowledge of the number of bones in the model, EP-LBS establishes an iterative process for computing a skeletal structure with a minimal number of bones, splitting the bones with large vertex error and then re-clustering with additional bones. This patch-based clustering, bone splitting and re-clustering process forms a major contribution of the presented research. The process is detailed in Algorithm 4.2.

This iterative initialization process, combining patch-based clustering and NNLS, provides an approximation of the gross bone transformations and influences. Each iteration of the parameter mapping gradient descent optimization algorithm then refines the deformation parameters until a convergent solution is found. The iterative two-stage initialization serves as one of the major contributions of this research.

4.3 Implementing Example-based Parameterization of Linear Blend Skinning

Section 4.2 introduced a mathematical model for LBS requiring only bounding conditions and relatively inexpensive parameter mappings to constrain the search space of valid animation parameters that serves as a major contribution of this research. The modified LBS equation is used as the objective function for the constrained optimization problem of example-based skinning decomposition. This remainder of this chapter details the use of constrained optimization to determine the animation parameters for a character model.

4.3.1 Pose Nomenclature

Given the vertex mesh of a 3D character in a base pose and in a set of example poses, the modified LBS equation is used as the objective function for a constrained optimization solution to the skinning decomposition problem of computing animation parameters. To inform the discussion of example-based LBS, pose nomenclature is introduced for the example data. In addition to being referenced by index i , vertices from an example pose are also indexed by pose. For example, the i^{th} vertex in its transformed position as determined by the observed example pose, p , is represented as \mathbf{v}_{ip}'' . In example-based approaches, transformation variables are also indexed by pose. Generic transformation variables, without regard to their representation, are indicated by \mathbf{T}_{pj} . Rotations are identified by \mathbf{R}_{pj} in matrix form, by \mathbf{q}_{pj} in quaternion vectors and by \mathbf{e}_{pj} in Euler angle vector representations. Translation vectors are indicated by \mathbf{s}_{pj} . The nomenclature listed in Table 2.1 is updated to include pose indexed notation and listed here as Table 4.2.

Object	Variable	Subscript	Range	Element Indices
Base Vertex	\mathbf{v}	i,p		w, x, y, z
Computed Vertex	\mathbf{v}'	i,p		w, x, y, z
Example Vertex	\mathbf{v}''	i,p		w, x, y, z
Quaternion Rotation Matrix	\mathbf{Q}	p,j		w, x, y, z
Quaternion Rotation Vector	$\dot{\mathbf{q}}$	p,j	$(0,1)$	w, x, y, z
Euler Angle Rotation	\mathbf{e}	p,j	$(-\pi, \pi)$	ϕ, θ, ψ
Translation	\mathbf{s}	p,j		w, x, y, z
Transformation	\mathbf{T}	p,j		w, x, y, z
Weight	w	i,j	$(0,1)$	

Table 4.2: Updated Symbols and Terminology

4.3.2 Objective Function: Modified Linear Blend Skinning

Given the Euclidean distance between two vertices as:

$$|\mathbf{v}_{i,p}'' - \mathbf{v}_{i,p}'| = \sqrt{(\mathbf{v}_{i,p,x}'' - \mathbf{v}_{i,p,x}')^2 + (\mathbf{v}_{i,p,y}'' - \mathbf{v}_{i,p,y}')^2 + (\mathbf{v}_{i,p,z}'' - \mathbf{v}_{i,p,z}')^2} \quad (4.30)$$

the skinning decomposition problem is expressed as the minimization of the sum of squared differences between the observed vertex positions in each example pose and the computed vertex positions as determined by the optimization process:

$$\min \sum_{i=1}^I \sum_{p=1}^P |\mathbf{v}_{i,p}'' - \mathbf{v}_{i,p}'| \quad (4.31)$$

Example poses are assumed to represent ground truth and the ideal deformations for each pose. The goal of the skinning decomposition problem is, therefore, to compute the deformation parameters that make the deformation of the base model into the example poses possible. The viability of the computed parameters is determined by the normalized sum squared error for the model. A successful solution will minimize this error for the model.

Combining the re-factored LBS equation and implicit boundaries, the skinning decomposition is formulated as error function E (Equation (4.32)) which serves as the objective function for optimization:

$$\min_{w, \mathbf{e}, \mathbf{s}} E = \min_{w, \mathbf{e}, \mathbf{s}} \sum_{i=1}^I \sum_{p=1}^P |\mathbf{v}_{ip} - \sum_j^J w_{ij} (\mathbf{Q}_{pj} \mathbf{v}_i \mathbf{e}_{pj} + \mathbf{s}_{pj})| \quad (4.32)$$

Subject to :

$$0 \leq w_{ij} \leq 1 \quad (4.33)$$

$$-\pi \leq e_{\phi_{pj}}, e_{\theta_{pj}}, e_{\psi_{pj}} \leq \pi \quad (4.34)$$

4.3.3 Initialization: NNLS + Iterative k -means Clustering

Assuming example poses with identical mesh topology (such that vertex connectivity remains unchanged from pose to pose), the presented technique is a loosely constrained two-step initialization process.

Computing Initial Bone Transformations and Vertex-Bone Associations -

As described in detail in Section 4.2.3 1-ring patches are identified and their transforms are computed and chained together for each vertex. K -means clustering and least-squares fitting are used to compute the initial singleton bone groups. Least-squares fitting is used once again to determine the initial joint transformations, including a rotation and translation value for each bone in each pose.

Computing Initial Weights - Initial singleton vertex-bone associations and bone rotations are only part of the LBS model. Initial values for the skinning weights must

also be determined. Initial weights are computed using standard nonnegative least squares (NNLS) methods. Given the initial transforms computed from the patched-based clustering process and the vertex positions in a base pose and a set of example of poses, NNLS is used to compute the skinning weights for the model.

Mathematically, the process for computing initial weights can be modeled as follows. For some parameterized model \mathbf{C} and estimated parameters \mathbf{x} , standard nonnegative least squares optimization problem is expressed in the form:

$$\min_{\mathbf{x}} |\mathbf{C} \cdot \mathbf{x} - \mathbf{d}| \text{ where } \mathbf{x} \geq 0 \quad (4.35)$$

where the goal is to find the values for \mathbf{x} that minimize error when fitting model $\mathbf{C} \cdot \mathbf{x}$ to some data \mathbf{d} . For the purposes of this research, the nonnegative least squares problem of computing weights for a single vertex is structured as follows.

$$\min_{\mathbf{w}_i} |(\mathbf{Q}_p \mathbf{v}_i + \mathbf{s}_p) \cdot \mathbf{w}_i - \mathbf{v}_i''| \text{ where } \mathbf{w}_i \geq 0 \quad (4.36)$$

The NNLS problem for initial weight computation computes the set of weights, \mathbf{w} , that minimizes the sum square error of fitting vertices from example poses, \mathbf{v}'' , to the LBS model, $\mathbf{Q}\mathbf{v} + \mathbf{s}$. The initial weights are only loosely constrained by the nonnegative bounds on NNLS. Therefore, as the final step to the initialization process, the weights are normalized to ensure constraint adherence.

4.3.4 Optimization & Constraint Adherence: Gradient Descent with Parameter Mapping

A basic gradient descent algorithm is used to drive the optimization of the objective function (Equation 5.2). The gradient descent algorithm is augmented to perform value mapping as described in Section 4.2.2. During each iteration the rotation and weight mapping algorithms are used to map the deformation parameters to new values within the range of valid LBS values prior to taking a step in the direction of the negative gradient for subsequent iterations.

Finite differences are used to compute an estimate of the gradient. At each iteration of the optimization process, a step is taken in the direction of the gradient. A momentum buffer is an adaptive step size for the gradient descent algorithm that allows the algorithm to gather momentum and take larger steps when possible, modestly speeding up convergence at the minimal cost of an additional function evaluation when the step size needs to be reduced. For this research a momentum buffer of five was used.

The step size is increased after some number of successful iterations and decreased if the the current step would result in an increase in the objective function value. The number of iterations is referred to as the momentum buffer and takes the value of five for this research. This basic adaptive step size approach allows the algorithm to gather momentum and take larger steps when possible, modestly speeding up convergence at the minimal cost of an additional function evaluation when the step size needs to be reduced.

The gradient descent and parameter mapping algorithm is outlined in Algorithm 4.3.

Algorithm 4.1: Basic LBS Initialization

Input: vertices in a base pose \mathbf{v} , vertices in P example poses \mathbf{v}'' , vertex topology \mathbf{M}

Output: initial rotations \mathbf{q}_{pj} , initial translations \mathbf{s}_{pj} , initial weights \mathbf{w}_{ij}

```
1: {Loop through vertices and compute first-order neighbors}
2: for every vertex do
3:   Set  $\mathbf{A}$  to the first-order neighborhood for the vertex in the base pose
4:   Create an empty feature vector for the vertex
5:   for every pose do
6:     Set  $\mathbf{B}$  to the first-order neighborhood for the vertex in the current pose
7:     Compute the transformation matrix to transform  $\mathbf{A}$  to  $\mathbf{B}$  using least squares
        fitting of two 3D point sets
8:     Compute the quaternion form of the transformation rotation component
9:     Normalize the transformation translation component
10:    Concatenate the quaternion rotation-translation vector sub-sequence
        and the vertex feature vector
11:   end for
12: end for
13: Use k-means clustering to group vertex feature vectors into bone clusters
14: {Loop through bone clusters and poses to compute transform from base bone
    cluster to example bone cluster}
15: for every pose do
16:   for every bone do
17:     Set  $\mathbf{AA}$  to all vertices in base pose assigned to the current bone
18:     Set  $\mathbf{BB}$  to all vertices in current example pose assigned to the current bone
19:     Compute the initial bone transformation matrix to transform  $\mathbf{AA}$  to  $\mathbf{BB}$ 
        using least squares fitting of two 3D point sets
20:     Compute the quaternion form of the transformation rotation component
21:   end for
22: end for
22: {Compute Weights by solving system of linear equations in the form  $\mathbf{Ax}=\mathbf{b}$  }
23: for every vertex do
24:   Create an empty vector for transformed vertices  $\mathbf{AAA}$ 
25:   for every pose do
26:     for every joint do
27:       Compute the transformed vertex position determined by the current bone
28:     end for
29:     Add the transformed vertex positions to the vector  $\mathbf{AAA}$ 
30:   end for
31:   Set  $\mathbf{BBB}$  to the current vertex in each example pose
32:   Compute initial weights for current vertex using  $\mathbf{AAA}$ ,  $\mathbf{BBB}$  and NNLS
33: end for
34: return  $\mathbf{w}, \mathbf{q}, \mathbf{s}$ 
```

Algorithm 4.2: NNLS and Iterative Clustering EP-LBS Initialization

Input: model error tolerance $MTOL$

objective function $F(\mathbf{x})$

vertices in a base pose \mathbf{v}

vertices in P example poses \mathbf{v}''

vertex topology \mathbf{M}

Output: animation parameters \mathbf{x} (consisting of \mathbf{q}_{pj} , \mathbf{s}_{pj} , \mathbf{s}_{ij})

updated bone count J

- 1: Compute initial LBS animation parameters \mathbf{x} using Algorithm 4.1
 - 2: **while** $F(\mathbf{x}) > MTOL$ **do**
 - 3: Compute average error for each bone
 - 4: Set N to number of bones with error greater than avg err
 - 5: **for** $n = 1$ **to** N **do**
 - 6: Use k-means, with $k=2$, to cluster vertices for each bone with high error
 - 7: Increment bone count by one
 - 8: **end for**
 - 9: Compute the centroids of the all bone clusters
 - 10: Set \mathbf{x} = results of reclustering all vertices in manner of **Algorithm 4.1**,
using computed bone cluster centroids and new bone count to initialize
the clustering algorithm
 - 11: Compute $F(\mathbf{x})$
 - 12: **end while**
 - 13: **return** \mathbf{x}
-

Algorithm 4.3: Gradient Descent with Parameter Mapping

Input: objective function $F(\mathbf{x})$, derivative of function $\nabla F(\mathbf{x})$,
new point \mathbf{x} , initialized using methods described in **Section 4.3.3**
function value tolerance TOL , step size γ , step size tolerance $STOL$
maximum iterations $MAXITER$
momentum variable m , momentum buffer $MBUFF$

Output: Point at which function minimum occurs \mathbf{x}_{curr}

```
1: Set momentum counter  $g = 0$ 
2: for  $itr := 1$  to  $MAXITER$  do
3:   Set  $\mathbf{x}_{prev} = \mathbf{x}$ 
4:   Set  $f_{prev} = F(\mathbf{x}_{prev})$ 
5:   Map weights according to Equation 4.9
6:   Map Euler angle rotations according to Equation 4.20
7:   Compose mapped point  $\mathbf{x}_{mapped}$ ,
   from computed mapped weights and rotations
8:   Compute gradient,  $\nabla F$ , based on mapped point,  $\mathbf{x}_{mapped}$ 
9:   Compute new point in direction of negative gradient
    $\mathbf{x}_{curr} = \mathbf{x}_{prev} - \gamma \nabla F(\mathbf{x}_{mapped})$ 
10:  Compute function value at new point,  $f_{temp} = F(\mathbf{x}_{curr})$ 
11:  while  $f_{temp} > f_{prev}$  do
12:    Cut step size in half  $\gamma = \frac{\gamma}{2}$ 
13:    Restart momentum counter  $g = 0$ 
14:    if  $\gamma < STOL$  then
15:      break
16:    end if
17:    Compute new point in direction of negative gradient
    $\mathbf{x}_{temp} = \mathbf{x}_{prev} - \gamma \nabla F(\mathbf{x}_{mapped})$ 
18:    Compute function value at new point,  $f_{temp} = F(\mathbf{x}_{temp})$ 
19:  end while
20:  if  $\mathbf{x}_{temp} - \mathbf{x}_{prev} < TOL$  then
21:    Decrement  $itr$ 
22:    break
23:  end if
24:  if  $\gamma < STOL$  then
25:    Decrement  $itr$ 
26:    break
27:    Update  $\mathbf{x} = \mathbf{x}_{temp}$ 
28:    Update  $f = f_{temp}$ 
29:    Increment momentum counter  $g$ 
30:    if  $g > MBUFF$  then
31:      Increase momentum  $\gamma = 1.75\gamma$ 
32:    end if
33:  end if
34: end for
35: return  $\mathbf{x}$ 
```

Chapter 5

Results

Computer animation and 3D graphics are often subject to both quantitative and qualitative analysis as a result of the need for both subjective and objective evaluation of the final animation sequence. For instance, the objective (and very practical) concern of whether or not a character’s feet are placed correctly so as not to intersect the ground is equally as important as the style of movement in the animation portraying a feeling of giddiness. As such, this research employs both qualitative and quantitative metrics for evaluation of the presented Example-based Parameterization of Linear Blend Skinning for Skinning Decomposition (EP-LBS) process.

This chapter analyzes the results of testing the presented EP-LBS method for skinning decomposition using parameter mapping. To inform analysis of EP-LBS results, this chapter first begins by detailing the methods used to evaluate parameter mapping for skinning decomposition and its ability to recreate observed character deformation. The results are then presented both quantitatively and qualitatively. Finally, the results are discussed and analyzed in the broader context of skinning decomposition processes and work-flows.

5.1 Methods of Evaluation

Both quantitative and qualitative analysis of results are used to evaluate EP-LBS. Visual comparison of expected and computed vertex positions yields qualitative analysis that pro-

vide subjective evaluation of EP-LBS results. Numeric measurements of the difference between the expected and observed vertex positions provide objective quantitative evaluation. Both means of evaluation have practical value for analyzing example-based animation and skinning decomposition solutions.

5.1.1 Quantitative Analysis

Two means of quantitative analysis are use to evaluate the merits of this research objectively: model error and execution time.

5.1.1.1 Model Error

Quantitative results for model error are presented as normalized root mean square error (RMSE) values. Most example-based skinning decomposition solutions use some form of normalized RMSE to evaluate models quantitatively [34]. This error metric works well for example-based LBS and skinning decomposition problems because the goal of the underlying optimization is generally to compute parameter values that minimize the RMSE of the computed vertices.

$$\min_{w, \mathbf{e}, \mathbf{s}} E = \min_{w, \mathbf{e}, \mathbf{s}} \sum_{i=1}^I \sum_{p=1}^P |\mathbf{v}_{ip}'' - \sum_{j=1}^J w_{ij} (\mathbf{Q}_{pj} \mathbf{v}_i \mathbf{e}_{pj} + \mathbf{s}_{pj})| \quad (5.1)$$

The objective function, E , of the EP-LBS skinning decomposition problem used for this research is an RMSE function (Equation 5.1.1.1). Structured as a constrained optimization problem with the objective of minimizing the difference between the observed vertex positions and the corresponding computed vertices after deformation, the sum squared error of all

vertices is normalized by the number of vertices, I , and the number of poses, P , to yield model error Err (Equation (5.2)) which serves as a common error metric for comparison of skinning decomposition processes.

$$Err = \frac{E}{IP} \quad (5.2)$$

A final model error value nearer zero is preferred and typically represents a solution that more effectively recreates the observed vertex positions, than larger model error values.

Additionally, the standard deviation of vertex error (σ) is measured for each pose. Vertex error is determined by the Euclidean distance between the example vertex and the computed vertex. Defined as the square root of the average the square differences from the mean, standard deviation provides a measurement of the range of vertex error, which is useful when dealing with a large number of vertices across multiple poses.

For skinning decomposition the mean vertex distance is only meaningful on a per pose basis. Therefore the mean vertex distance, \bar{v} , for a pose, p , is defined as:

$$\bar{v}_p = \frac{1}{I} \sum_{i=1}^I \left| \mathbf{v}_{ip}'' - \mathbf{v}_{ip}' \right| \quad (5.3)$$

The standard deviation for a pose, σ_p , is then:

$$\sigma_p = \sqrt{\frac{1}{I} \sum_{i=1}^I \left(\left| \mathbf{v}_{ip}'' - \mathbf{v}_{ip}' \right| - \bar{v}_p \right)^2} \quad (5.4)$$

The average overall standard deviation for a model, σ_{avg} , is computed as:

$$\sigma_{avg} = \frac{1}{P} \sum_{p=1}^P \sigma_p \quad (5.5)$$

A lower standard deviation value indicates the vertex error is relatively close to the average vertex error for all vertices in the pose. Larger standard deviation values indicate some vertices had very large error while others had small error. A low-valued σ is ideal.

5.1.1.2 Execution Time

In addition to vertex error, execution time is a useful performance metric. As such, execution time is also presented in the results data. Attempts to optimize running time have only been implemented at the highest-level. Aggregate efforts to improve individual components of this process are beyond the scope of this research and are left as areas for future exploration.

EP-LBS is an offline pre-processing method for linear blend skinning systems. While, real-time computation is not needed for one-time pre-processing or offline systems, it is still reasonable to expect results to be available in a few minutes, or about the time it takes to grab a cup of coffee, for pre-production stand-in models of a few thousand vertices and in a few hours for high-resolution, production-quality models of tens of thousands of vertices. Performed once at the onset of using an LBS model, EP-LBS is executed to compute skeletal structure and bone rotations that recreate existing example poses, as well as compute the skinning weights needed to generate new poses.

5.1.2 Qualitative Analysis

Visually, humans are rather adept at detecting visual imperfections and inconsistencies and often have an idea of what they expect to see and the quality of that visual image, even before laying eyes on it. Images that differ even slightly from the expected image are quickly detected by the human eye. As a result, qualitative analysis has long been the dominant method of analysis and quality determination for this field. The prominent use of animation

has been as a medium for visual entertainment, where aesthetics tend to prevail over numeric accuracy. As the computer graphics industry moves toward higher-resolution and better quality character models, it will be more important to have both qualitative and quantitative accuracy. For now, the look of a model still is perhaps the most important tool.

Models used in this research are qualitatively evaluated by visual comparison. Both the expected model and the computed model are displayed simultaneously and visually evaluated for differences between expected and computed vertex positions. The error is qualitatively visualized with adjacent models, the gray model representing the expected example pose and the green model represented the computed pose. In addition to side-by side comparisons, the expected and computed models are also superimposed on one another to offer another means of comparison in the form of a heat map. Differences in vertex positions are presented as a heat map, with blue areas indicating little or no error and red areas indicating the largest error. The error scale indicating values for blue through red regions are evaluated per model. Thus, red values represent the largest error across all poses for a model.

5.1.3 Characteristics of Desired Results

Successful implementation of EP-LBS yields modest error results, with objective function, E_{RMS} , and standard deviation, σ_p , values near zero, and recreated poses that are near indistinguishable from the examples used to create them as indicated by cool (blue) heat maps.

5.2 Models

To test the presented process for skinning decomposition, EP-LBS was used on a collection of nine publicly available 3D models commonly used for skinning decomposition testing and evaluation: camel, cat, dance, elephant, flamingo, horse, jump, lion and snake. The dance and jump models represent human-like models and were obtained from the collection associated with Khodakovsky’s *Wavelet Compression of Parametrically Coherent Mesh Sequences* [33]. The original jump model does not include datapoints for the model’s head. The camel, cat, elephant, flamingo, horse and lion models are from the *Deformation Transfer for Triangle Meshes* dataset [98]. The snake model was obtained James and Twigg’s *Skinning Mesh Animations* data [49]. In addition to the nine models from the de facto skinning decomposition testing dataset, a new humanoid woman model was introduced. The woman model was used with permission from the Media and Entertainment Technologies Lab (METLAB) at Michigan State University.

An important criteria for the use of example data to drive computation is the need for correspondence between examples, particularly vertex correspondence. Point correspondence is an important aspect of any example-based methods and is beyond the scope of this dissertation. The data used for this research has non-varying topology between poses. Therefore, the question of point correspondence between examples is presumed resolved and is not a concern for the datasets used in this research.

The model datasets and associated pose data used in this research were published as a series of triangulated meshes and have been used as presented in the referenced publications. For the purpose of comparison, all models were scaled to fit within a unit cube. No additional processing of the mesh data was performed. In particular, no noise reduction was applied.

The models used in this research are listed in Tables 5.1 and ?? along with basic dimensions for each dataset.

5.3 Example-based Parameterization of Linear Blend Skinning Results

Given the characteristics for the testing dataset and the methods of evaluation, as well as an explanation of potential results and desired outcomes, the results are presented for evaluation and analysis in visual and numeric form. For each model, EP-LBS initialization and optimization were performed on the model and results were evaluated both quantitatively and qualitatively. Presented in this section are the EP-LBS results for each model. The results include initial singleton bone clusters identified by the EP-LBS initialization algorithm, a plot of the evolution of model error, E , analyzed at each iteration of EP-LBS optimization and a visual comparison of the expected versus the computer vertex positions in the form of a heat map.



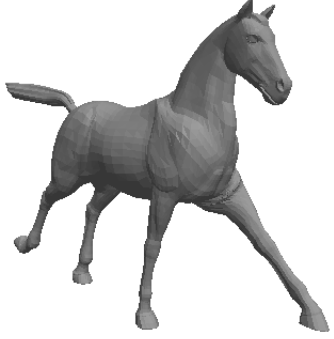

Model		#Vertices	# Bones	# Poses
Cat [98]		7207	22	10
Lion[98]		5000	22	10
Horse[98]		8431	29	11
Woman		3003	20	25

Table 5.1: Models

Table 5.1 (cont'd)

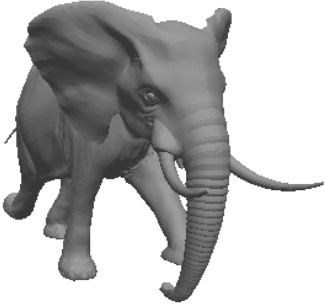


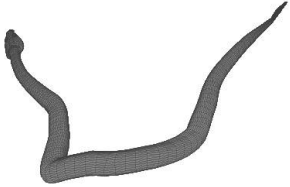
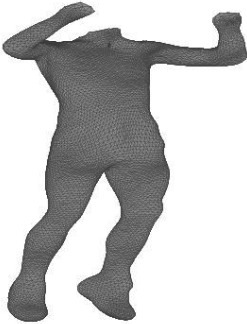
Model		#Vertices	# Bones	# Poses
Elephant[98]		42321	24	11
Flamingo[98]		26907	10	11
Camel[98]		21887	22	11
Snake[98]		9179	26	5

Table 5.1 (cont'd)

Model		#Vertices	# Bones	# Poses
Dance [33]		7061	21	11
Jump [33]		15830	21	8

5.3.1 Model: Cat



(a)



(b)

Figure 5.1: **Initialization Results for Cat Model** - 5.1a Model. 5.1b Initialization Results: Singleton bone groups.

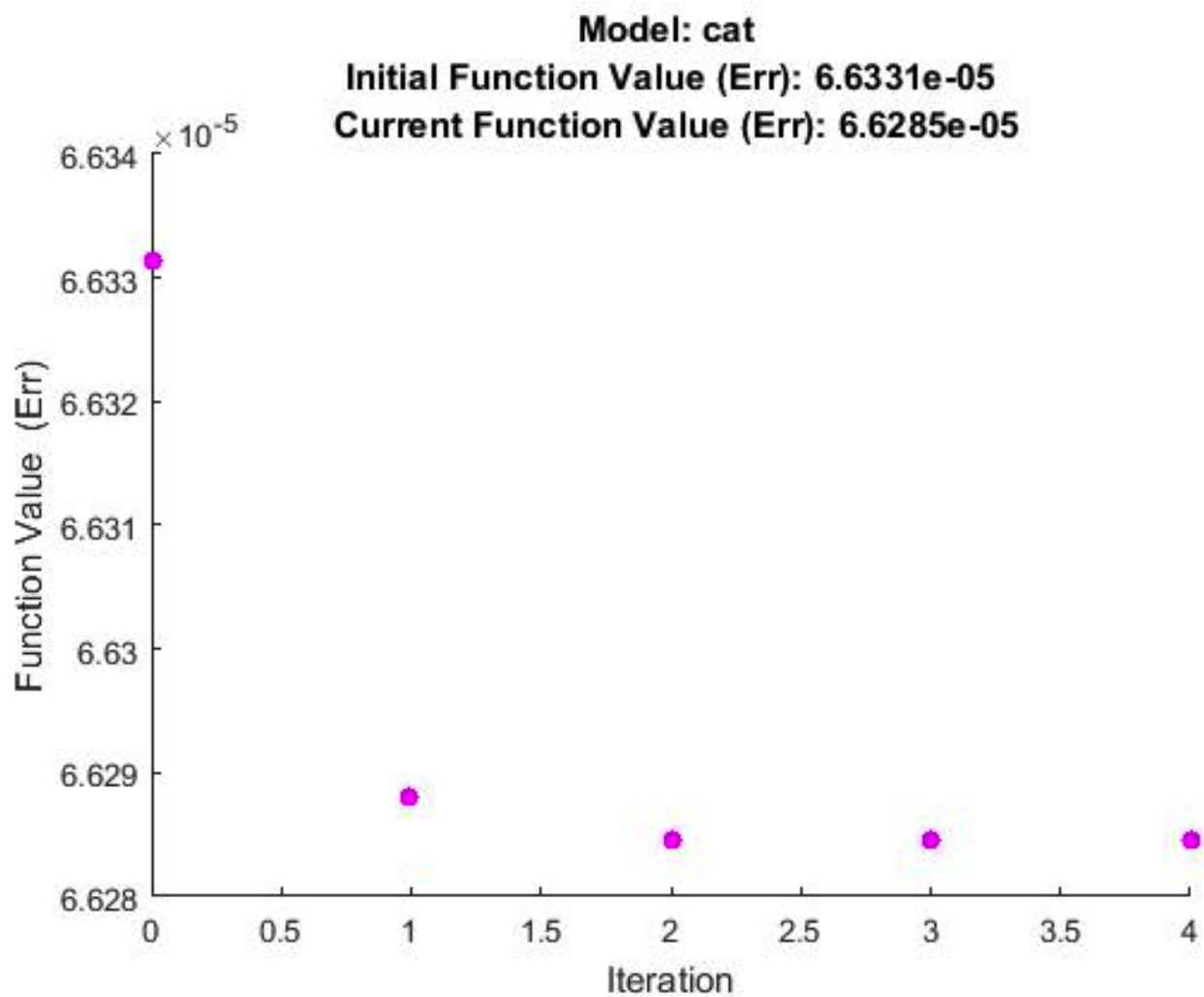
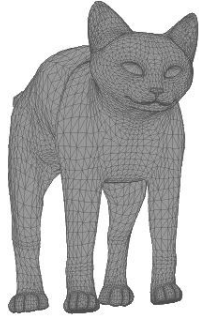
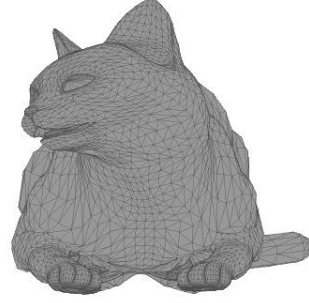


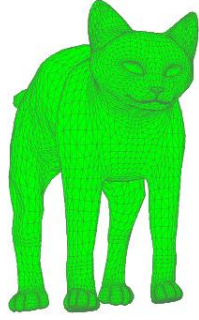
Figure 5.2: **Optimization Results for Cat Model** - Progress of EP-LBS minimization of the objective function.



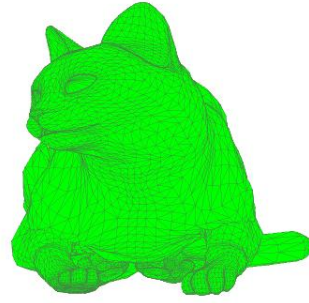
(a) Pose 01 - Expected Vertices



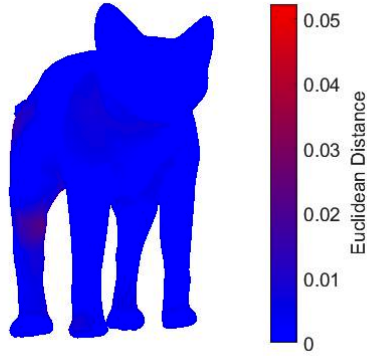
(b) Pose 02 - Expected Vertices



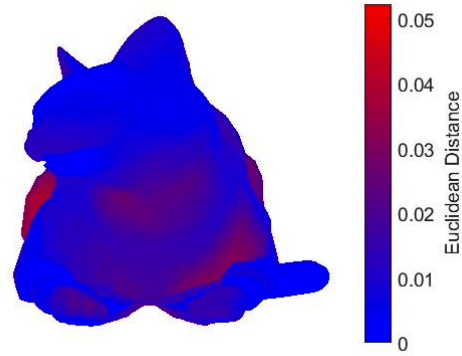
(c) Pose 01 - Computed Vertices



(d) Pose 02 - Computed Vertices

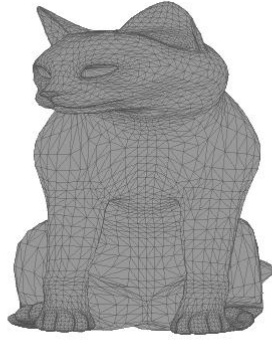


(e) Pose 01 - Error Heat Map
Pose Error: 4.5301×10^{-6}
Std Dev: 0.00184

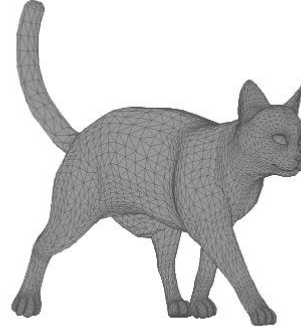


(f) Pose 02 - Error Heat Map
Pose Error: 6.697×10^{-5}
Std Dev: 0.00453

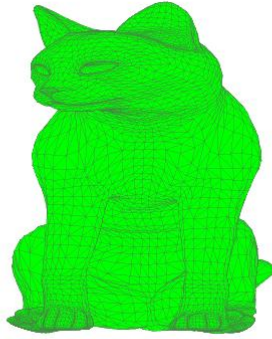
Figure 5.3: **Optimization Results for Cat Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 01 [left column] and Pose 02 [right column].



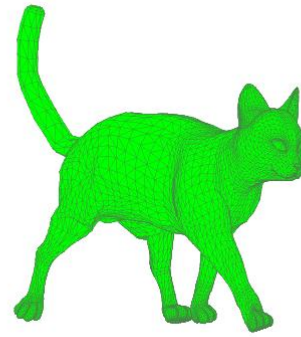
(a) Pose 03 - Expected Vertices



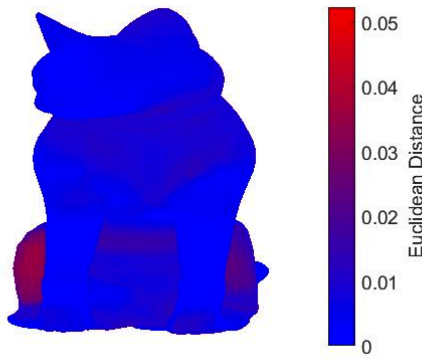
(b) Pose 04 - Expected Vertices



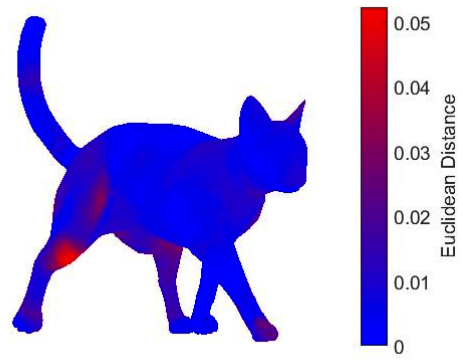
(c) Pose 03 - Computed Vertices



(d) Pose 04 - Computed Vertices

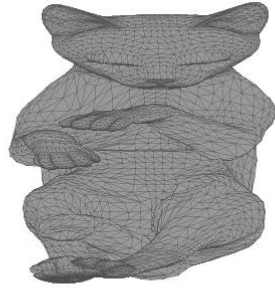


(e) Pose 03 - Error Heat Map
Pose Error: 3.2281×10^{-5}
Std Dev: 0.00315

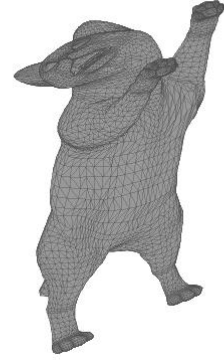


(f) Pose 04 - Error Heat Map
Pose Error: 5.9654×10^{-5}
Std Dev: 0.00448

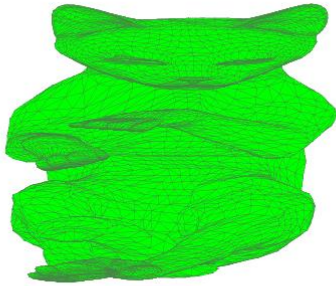
Figure 5.4: **Optimization Results for Cat Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 03 [left column] and Pose 04 [right column].



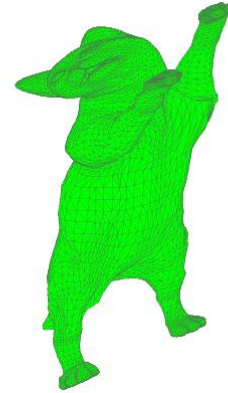
(a) Pose 05 - Expected Vertices



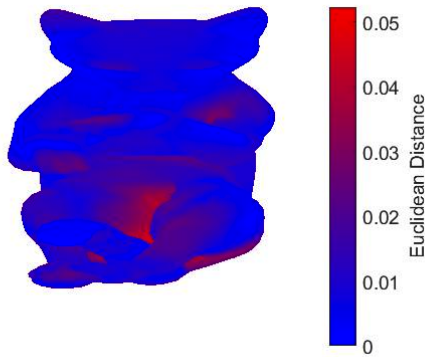
(b) Pose 06 - Expected Vertices



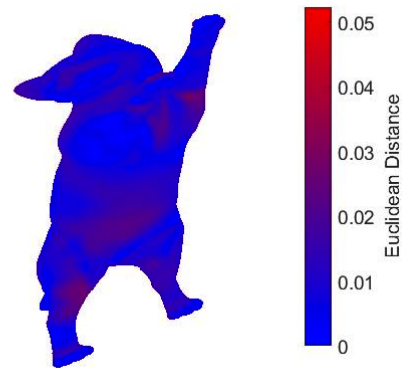
(c) Pose 05 - Computed Vertices



(d) Pose 06 - Computed Vertices

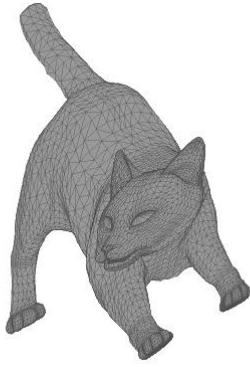


(e) Pose 05 - Error Heat Map
Pose Error: 7.2916×10^{-5}
Std Dev: 0.00474

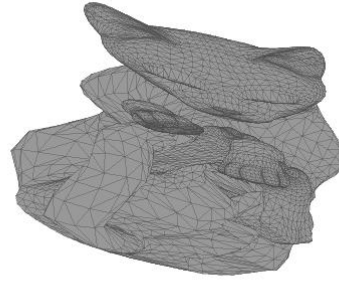


(f) Pose 06 - Error Heat Map
Pose Error: 5.997×10^{-5}
Std Dev: 0.00389

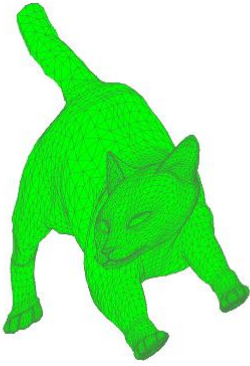
Figure 5.5: **Optimization Results for Cat Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 05 [left column] and Pose 06 [right column].



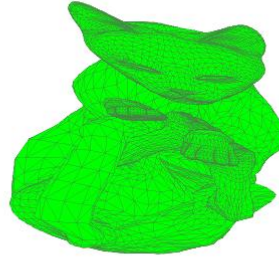
(a) Pose 07 - Expected Vertices



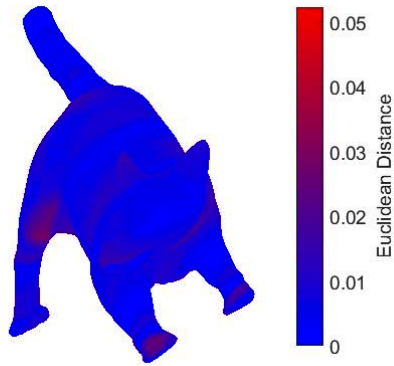
(b) Pose 08 - Expected Vertices



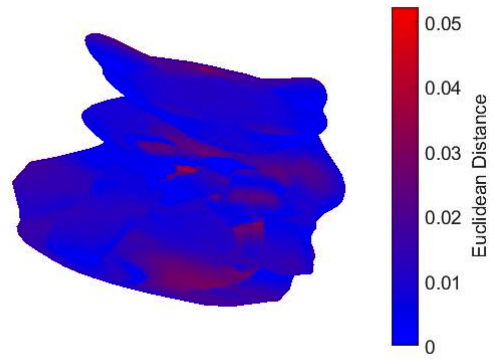
(c) Pose 07 - Computed Vertices



(d) Pose 08 - Computed Vertices

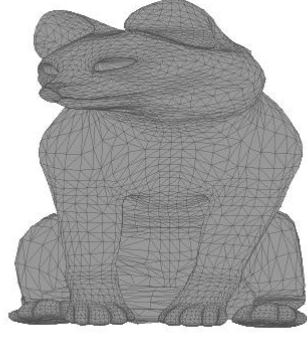


(e) Pose 07 - Error Heat Map
Pose Error: 6.1526×10^{-5}
Std Dev: 0.00402

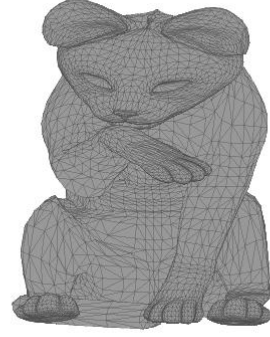


(f) Pose 08 - Error Heat Map
Pose Error: 7.6439×10^{-5}
Std Dev: 0.00463

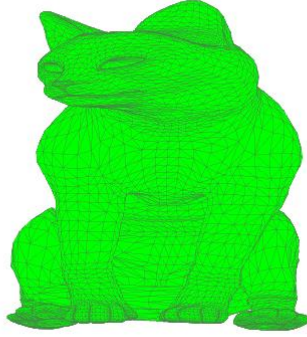
Figure 5.6: **Optimization Results for Cat Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 07 [left column] and Pose 08 [right column].



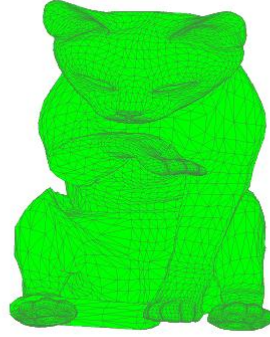
(a) Pose 09 - Expected Vertices



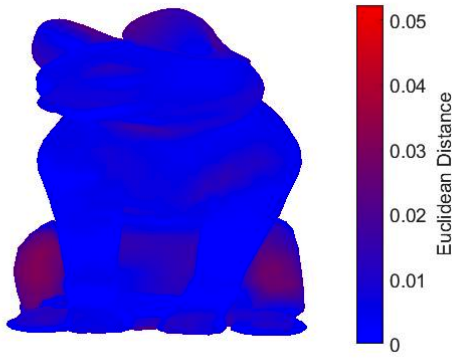
(b) Pose 10 - Expected Vertices



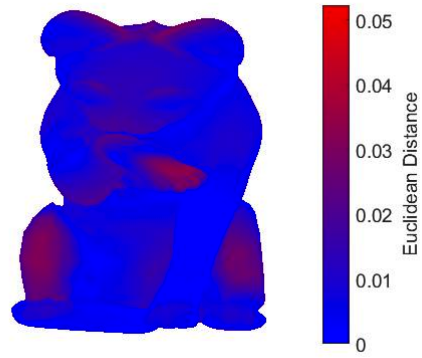
(c) Pose 09 - Computed Vertices



(d) Pose 10 - Computed Vertices



(e) Pose 09 - Error Heat Map
Pose Error: 3.828×10^{-5}
Std Dev: 0.00344



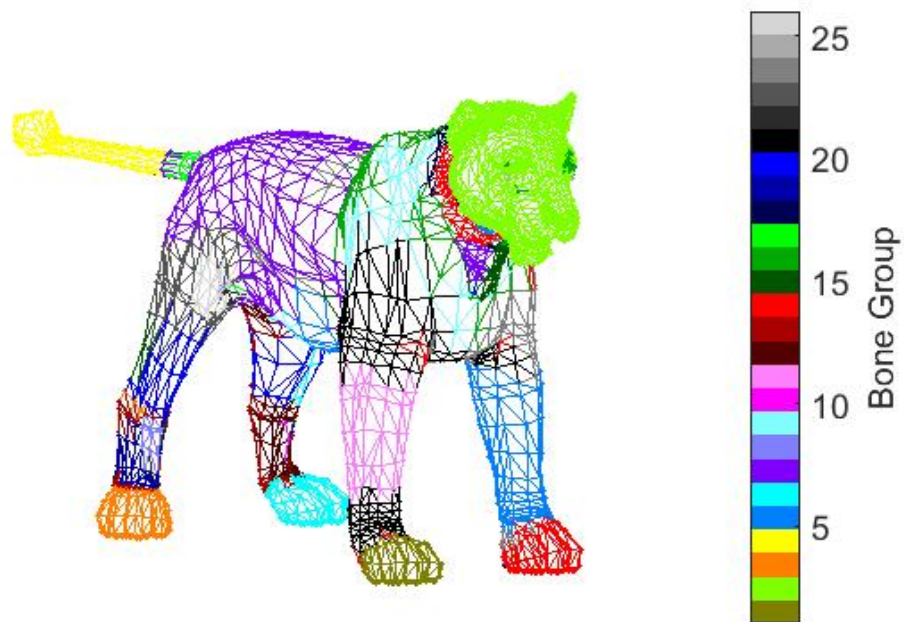
(f) Pose 10 - Error Heat Map
Pose Error: 6.2253×10^{-5}
Std Dev: 0.0045

Figure 5.7: **Optimization Results for Cat Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 09 [left column] and Pose 10 [right column].

5.3.2 Model: Lion



(a)



(b)

Figure 5.8: **Initialization Results for Lion Model** - 5.8a Model. 5.8b Initialization Results: Singleton bone groups.

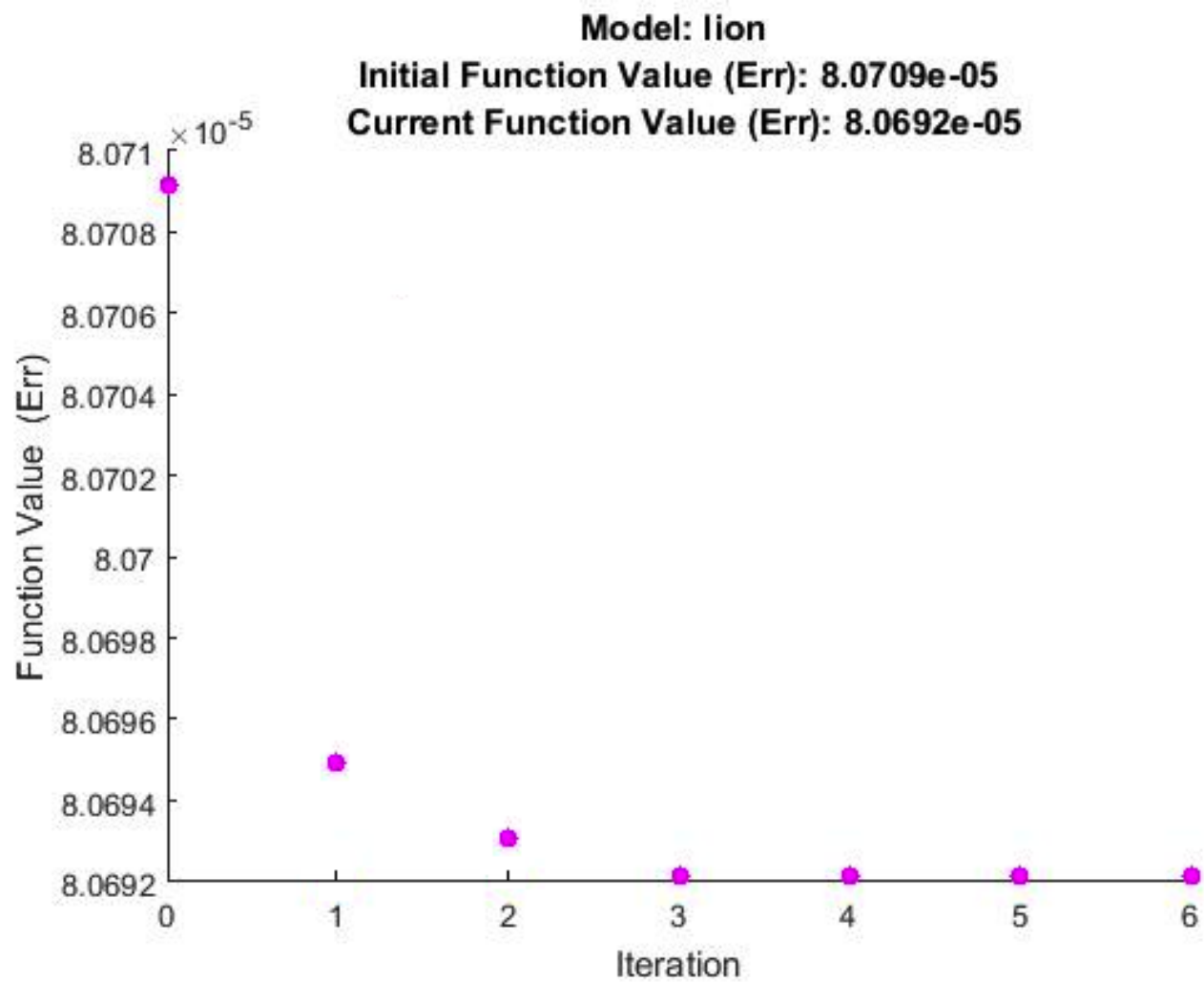
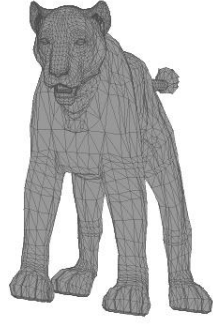
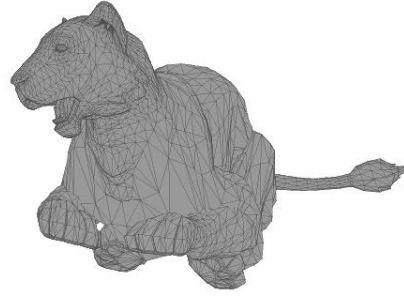


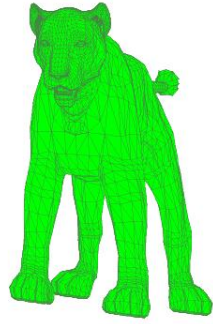
Figure 5.9: **Optimization Results for Lion Model** - Progress of EP-LBS minimization of the objective function.



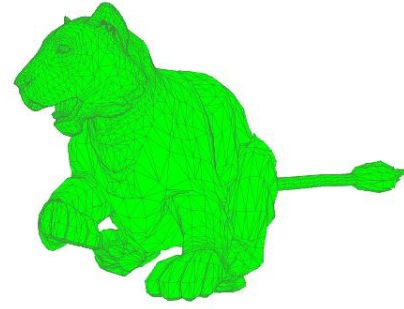
(a) Pose 01 - Expected Vertices



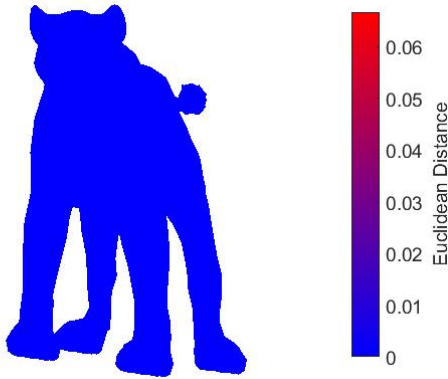
(b) Pose 02 - Expected Vertices



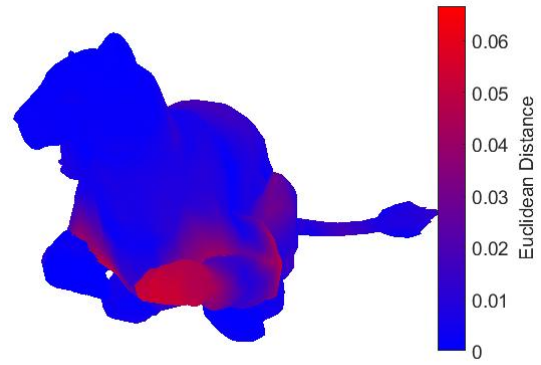
(c) Pose 01 - Computed Vertices



(d) Pose 02 - Computed Vertices



(e) Pose 01 - Error Heat Map
Pose Error: 1.2272×10^{-7}
Std Dev: 1.74×10^{-6}

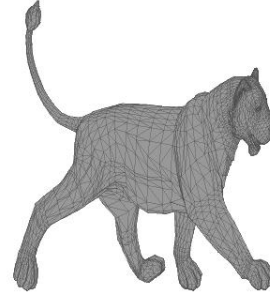


(f) Pose 02 - Error Heat Map
Pose Error: 2.1179×10^{-4}
Std Dev: 0.0121

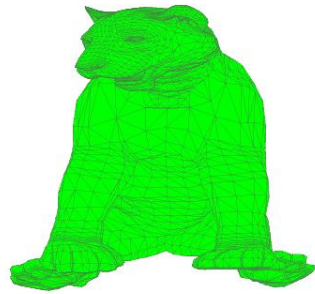
Figure 5.10: **Optimization Results for Lion Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 01 [left column] and Pose 02 [right column].



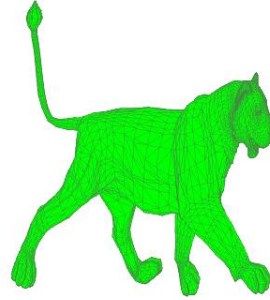
(a) Pose 03 - Expected Vertices



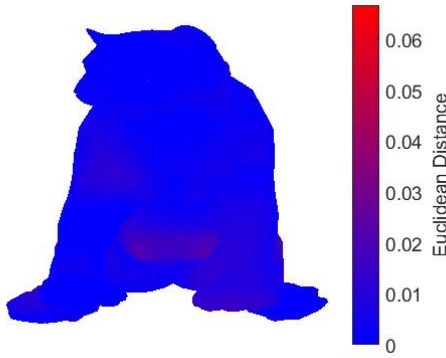
(b) Pose 04 - Expected Vertices



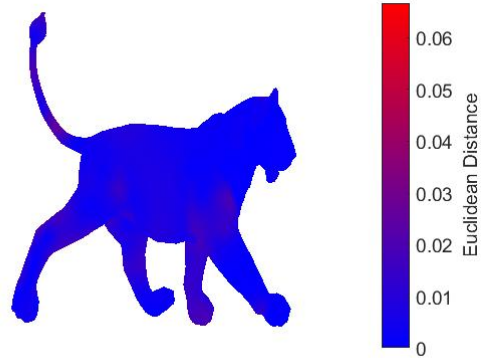
(c) Pose 03 - Computed Vertices



(d) Pose 04 - Computed Vertices

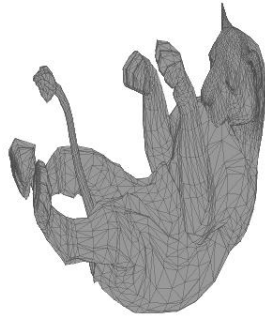


(e) Pose 03 - Error Heat Map
Pose Error: 3.1695×10^{-5}
Std Dev: 0.00395

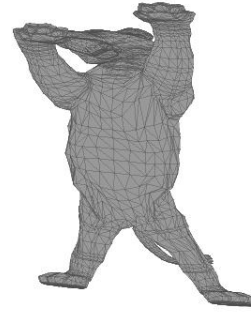


(f) Pose 04 - Error Heat Map
Pose Error: 5.8528×10^{-5}
Std Dev: 0.00571

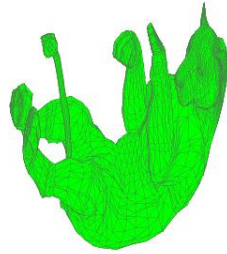
Figure 5.11: **Optimization Results for Lion Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 03 [left column] and Pose 04 [right column].



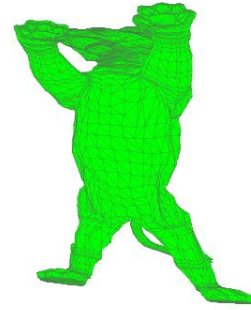
(a) Pose 05 - Expected Vertices



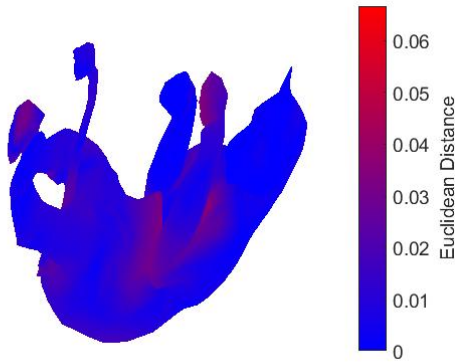
(b) Pose 06 - Expected Vertices



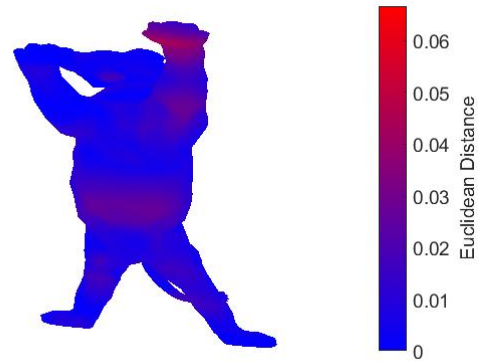
(c) Pose 05 - Computed Vertices



(d) Pose 06 - Computed Vertices

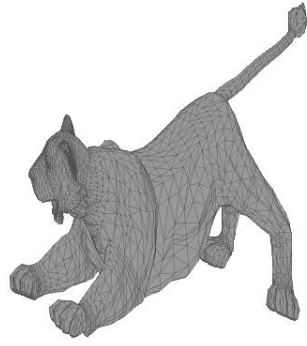


(e) Pose 05 - Error Heat Map
Pose Error: 1.1267×10^{-4}
Std Dev: 0.00809

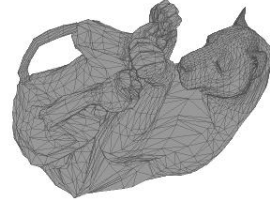


(f) Pose 06 - Error Heat Map
Pose Error: 1.0278×10^{-4}
Std Dev: 0.00744

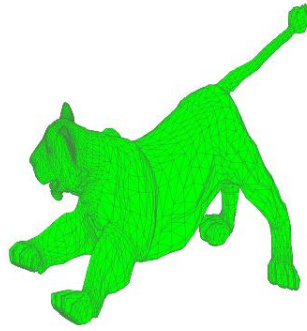
Figure 5.12: **Optimization Results for Lion Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 05 [left column] and Pose 06 [right column].



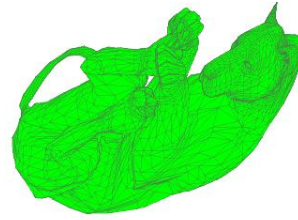
(a) Pose 07 - Expected Vertices



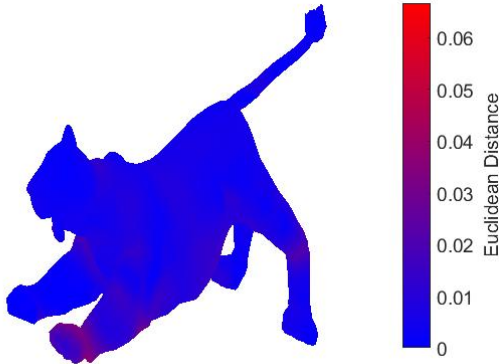
(b) Pose 08 - Expected Vertices



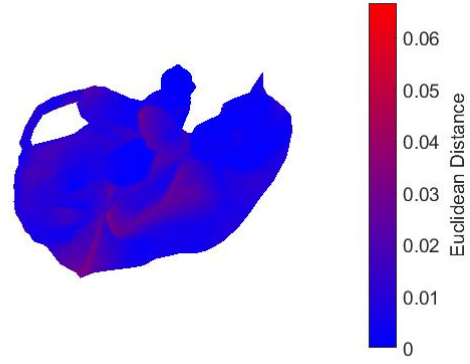
(c) Pose 07 - Computed Vertices



(d) Pose 08 - Computed Vertices



(e) Pose 07 - Error Heat Map
Pose Error: 8.6674×10^{-5}
Std Dev: 0.00699

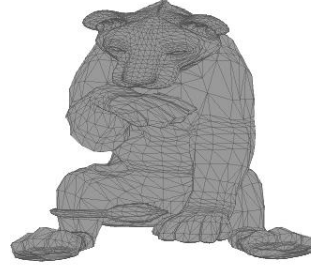


(f) Pose 08 - Error Heat Map
Pose Error: 8.691×10^{-5}
Std Dev: 0.00683

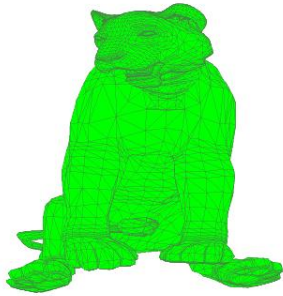
Figure 5.13: **Optimization Results for Lion Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 07 [left column] and Pose 08 [right column].



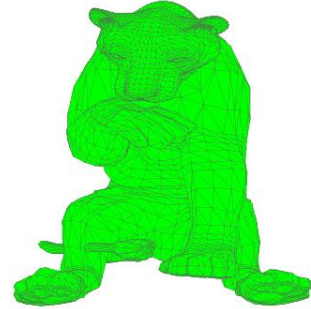
(a) Pose 09 - Expected Vertices



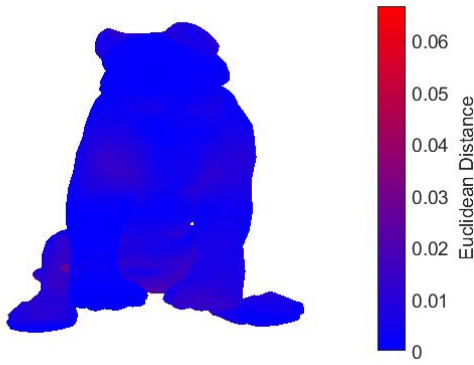
(b) Pose 10 - Expected Vertices



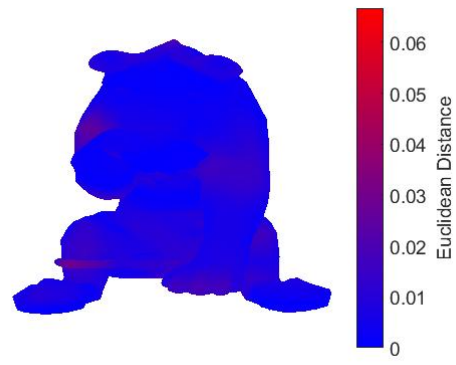
(c) Pose 09 - Computed Vertices



(d) Pose 10 - Computed Vertices



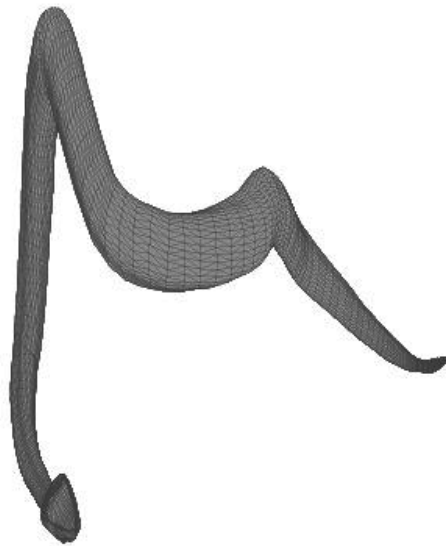
(e) Pose 09 - Error Heat Map
Pose Error: 5.5694×10^{-5}
Std Dev: 0.00502



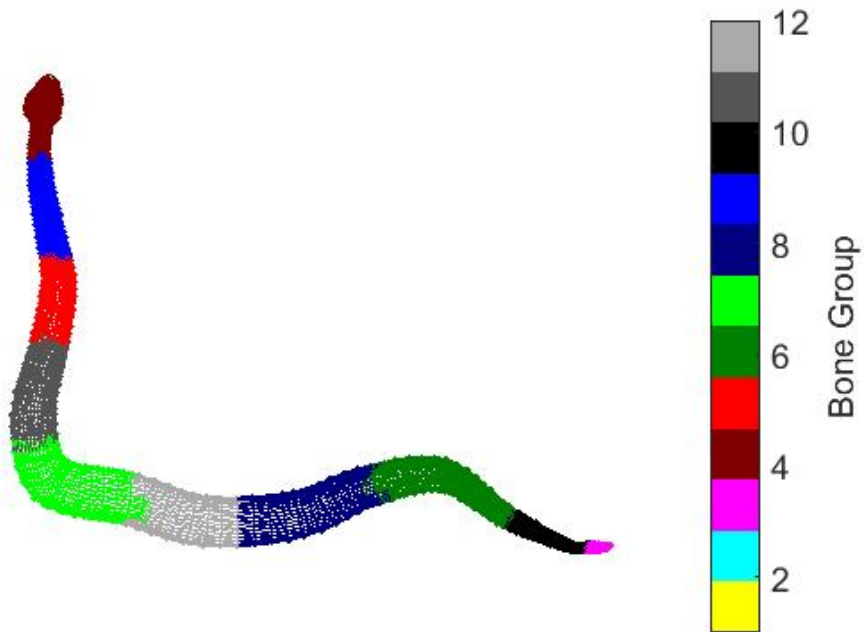
(f) Pose 10 - Error Heat Map
Pose Error: 6.0052×10^{-5}
Std Dev: 0.00474

Figure 5.14: **Optimization Results for Lion Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 09 [left column] and Pose 10 [right column].

5.3.3 Model: Snake



(a)



(b)

Figure 5.15: **Initialization Results for Snake Model** - 5.15a Model. 5.15b Initialization Results: Singleton bone groups.

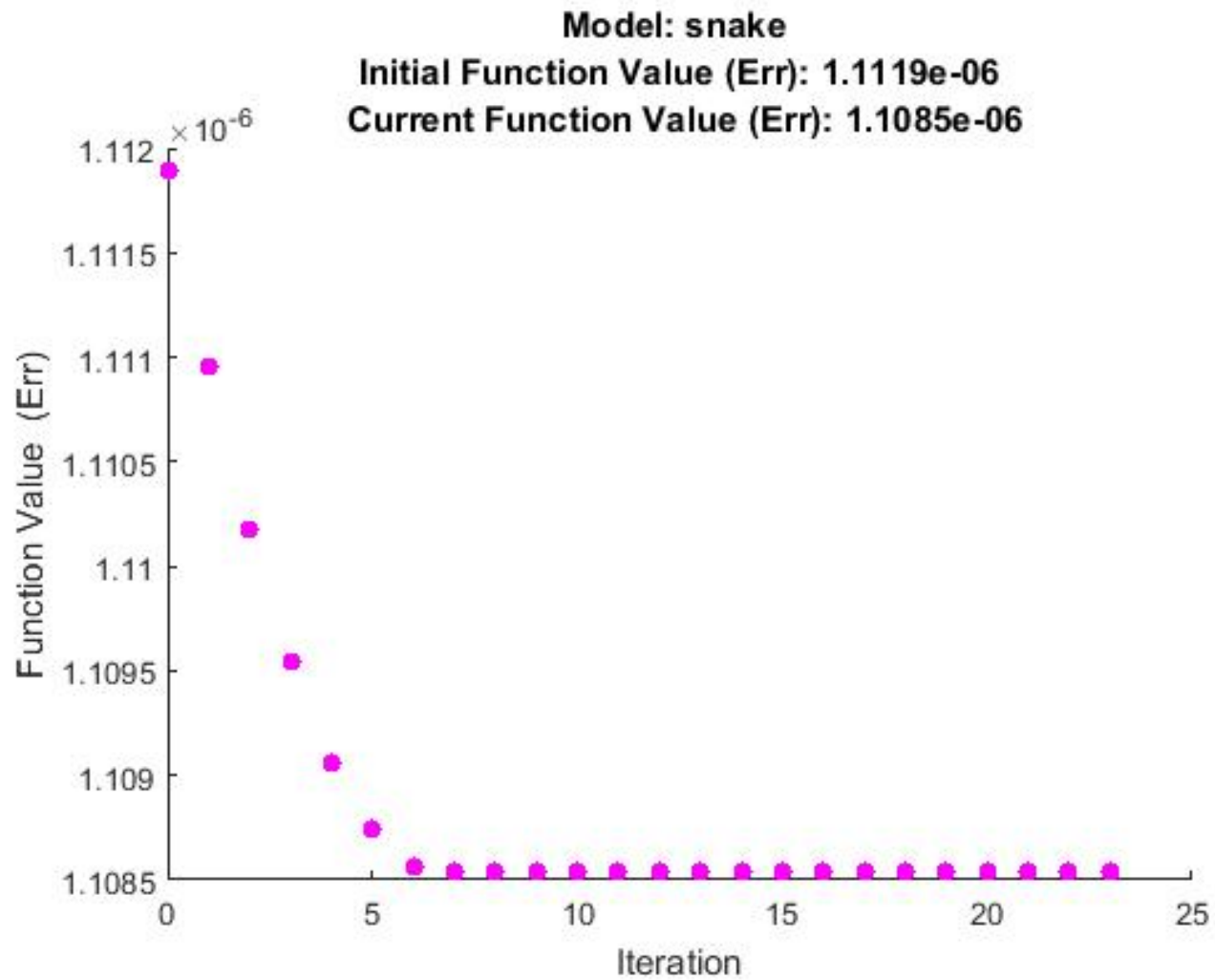
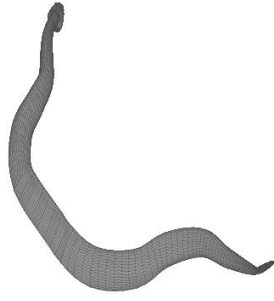
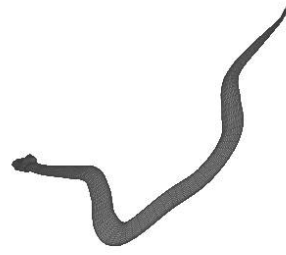


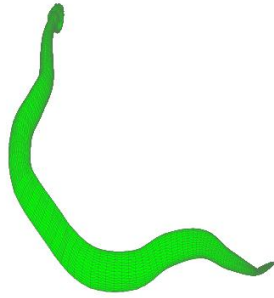
Figure 5.16: **Optimization Results for Snake Model** - Progress of EP-LBS minimization of the objective function.



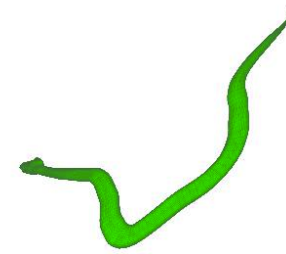
(a) Pose 01 - Expected Vertices



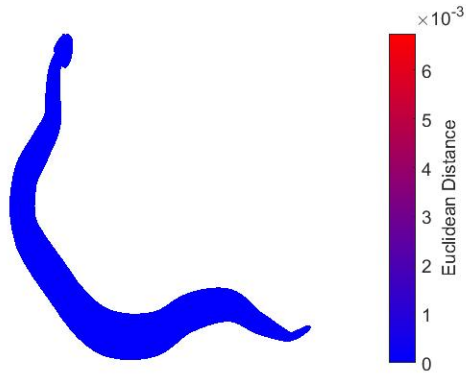
(b) Pose 02 - Expected Vertices



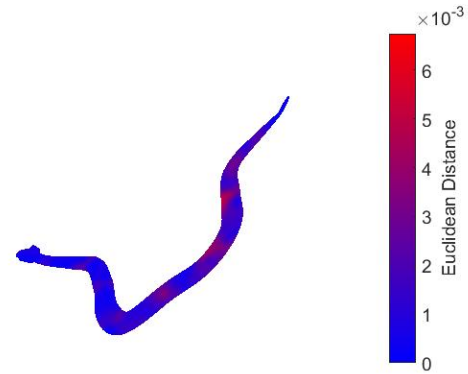
(c) Pose 01 - Computed Vertices



(d) Pose 02 - Computed Vertices

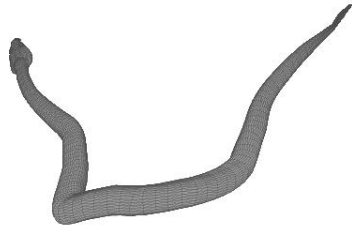


(e) Pose 01 - Error Heat Map
 Pose Error: 3.7356×10^{-9}
 Std Dev: 1.05×10^{-6}

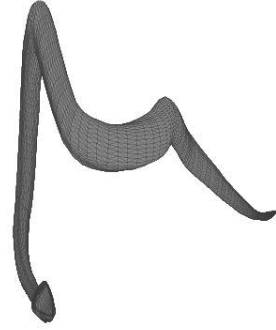


(f) Pose 02 - Error Heat Map
 Pose Error: 1.0597×10^{-6}
 Std Dev: 0.000745

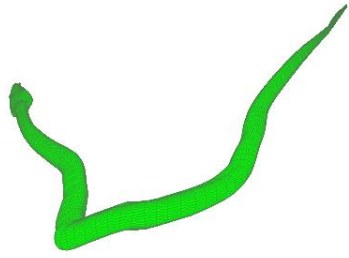
Figure 5.17: **Optimization Results for Snake Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 01 [left column] and Pose 02 [right column].



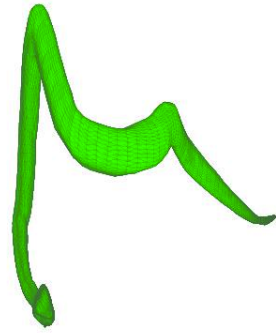
(a) Pose 03 - Expected Vertices



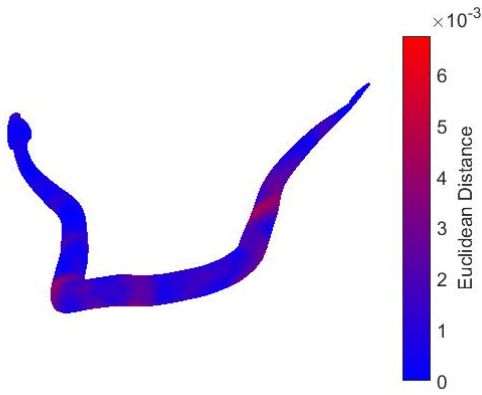
(b) Pose 04 - Expected Vertices



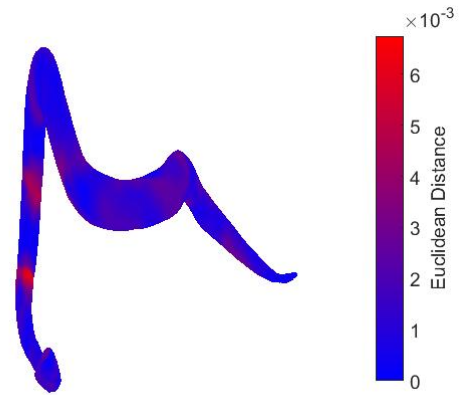
(c) Pose 03 - Computed Vertices



(d) Pose 04 - Computed Vertices

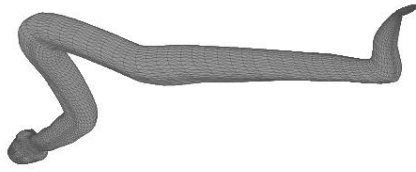


(e) Pose 03 - Error Heat Map
Pose Error: 1.0419×10^{-6}
Std Dev: 0.000733

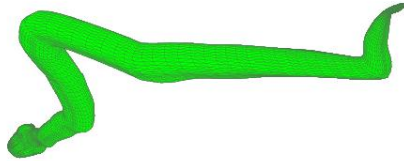


(f) Pose 04 - Error Heat Map
Pose Error: 1.8518×10^{-6}
Std Dev: 0.000881

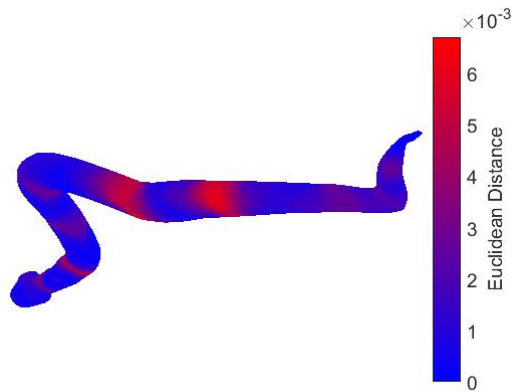
Figure 5.18: **Optimization Results for Snake Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 03 [left column] and Pose 04 [right column].



(a) Pose 05 - Expected Vertices



(b) Pose 05 - Computed Vertices



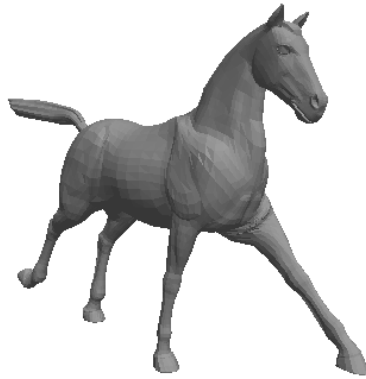
(c) Pose 05 - Error Heat Map

Pose Error: 1.5855×10^{-6}

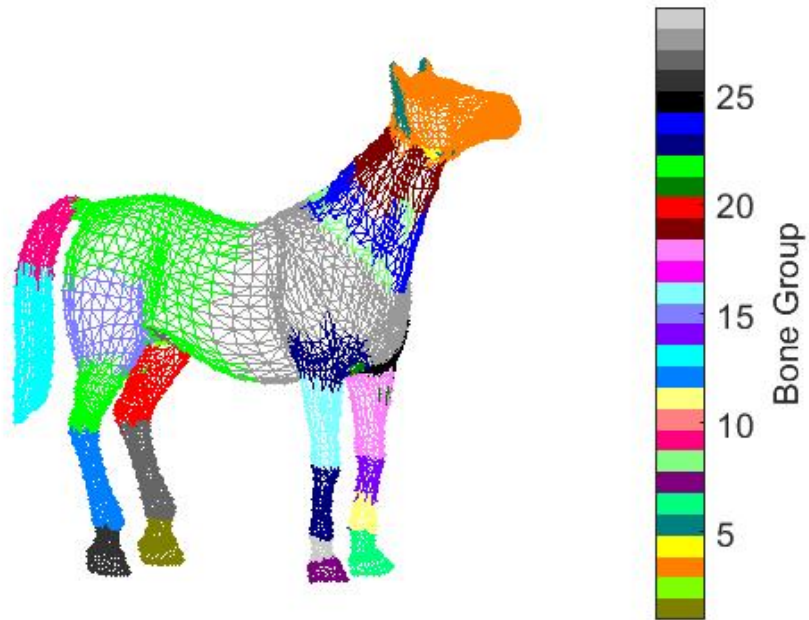
Std Dev: 0.000941

Figure 5.19: **Optimization Results for Snake Model** - Expected vertex positions [top:(a)], Computed vertex positions [middle:(b)] and Error Heat Map [bottom:(c)] for Pose 05.

5.3.4 Model: Horse



(a)



(b)

Figure 5.20: **Initialization Results for Horse Model** - 5.20a Model. 5.20b Initialization Results: Singleton bone groups.

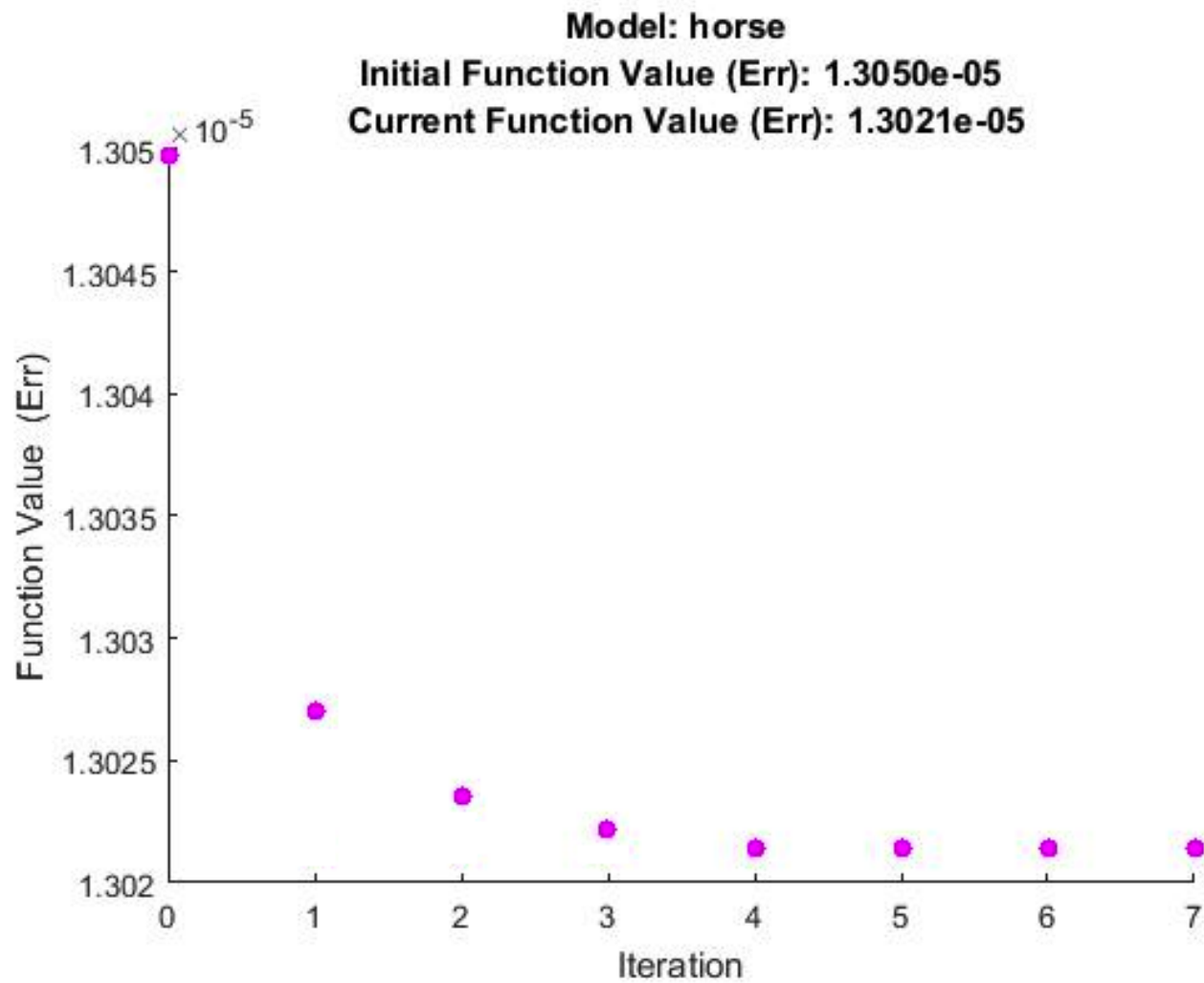
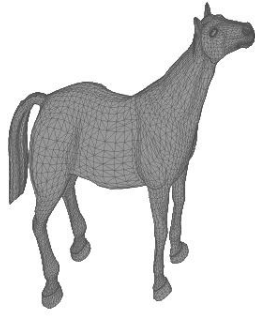
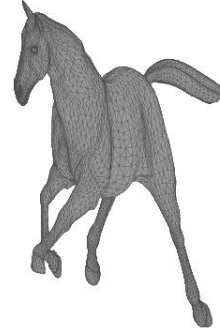


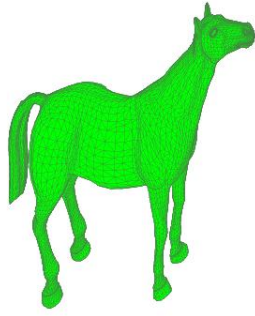
Figure 5.21: **Optimization Results for Horse Model** - Progress of EP-LBS minimization of the objective function.



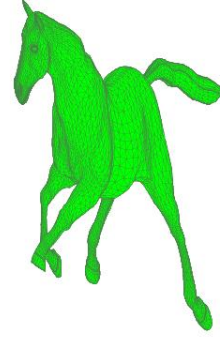
(a) Pose 01 - Expected Vertices



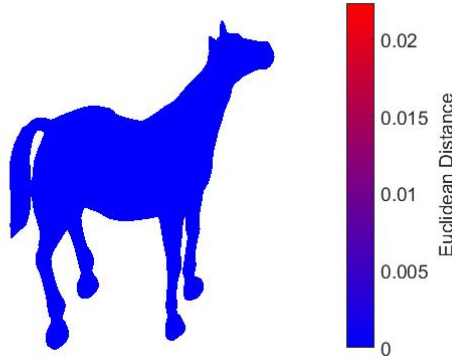
(b) Pose 02 - Expected Vertices



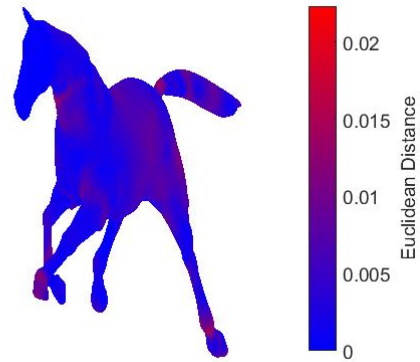
(c) Pose 01 - Computed Vertices



(d) Pose 02 - Computed Vertices

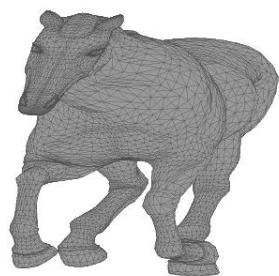


(e) Pose 01 - Error Heat Map
Pose Error: 4.3962×10^{-8}
Std Dev: 1.798×10^{-6}

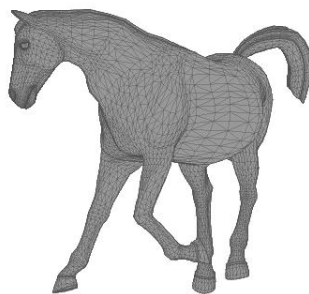


(f) Pose 02 - Error Heat Map
Pose Error: 1.8738×10^{-5}
Std Dev: 0.00297

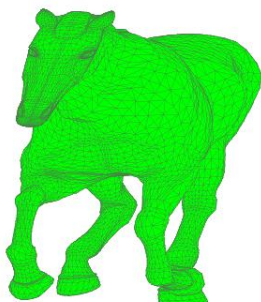
Figure 5.22: **Optimization Results for Horse Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 01 [left column] and Pose 02 [right column].



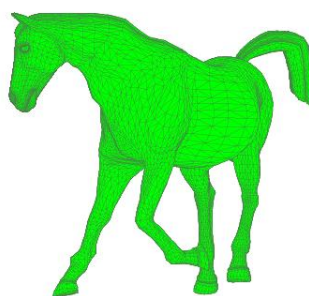
(a) Pose 03 - Expected Vertices



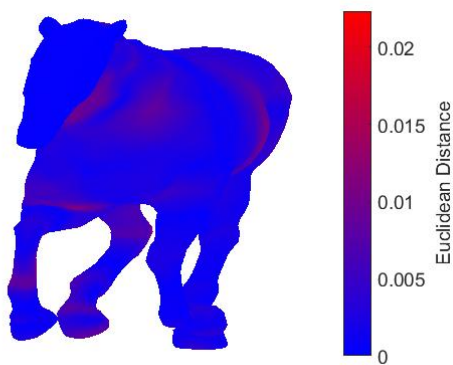
(b) Pose 04 - Expected Vertices



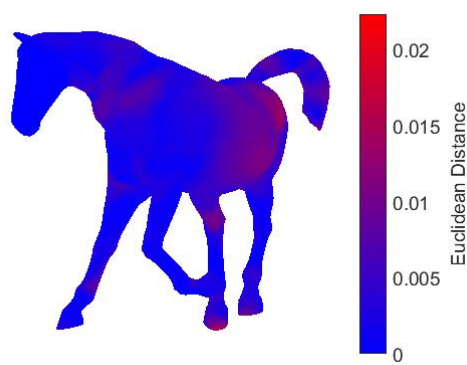
(c) Pose 03 - Computed Vertices



(d) Pose 04 - Computed Vertices

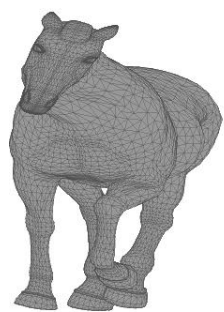


(e) Pose 03 - Error Heat Map
Pose Error: 1.6002×10^{-5}
Std Dev: 0.00292

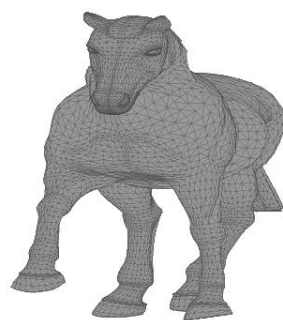


(f) Pose 04 - Error Heat Map
Pose Error: 1.6564×10^{-5}
Std Dev: 0.00294

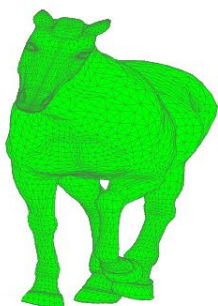
Figure 5.23: **Optimization Results for Horse Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 03 [left column] and Pose 04 [right column].



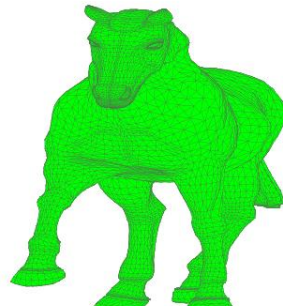
(a) Pose 05 - Expected Vertices



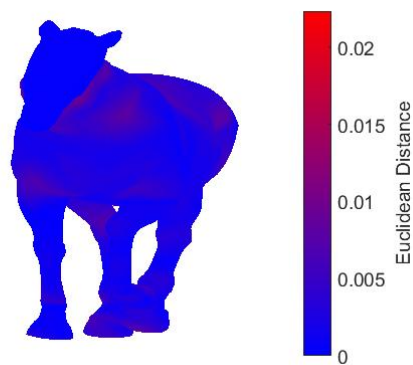
(b) Pose 06 - Expected Vertices



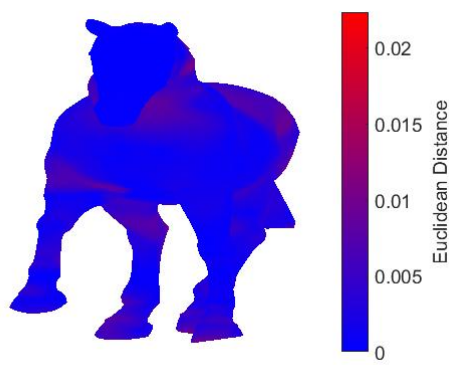
(c) Pose 05 - Computed Vertices



(d) Pose 06 - Computed Vertices

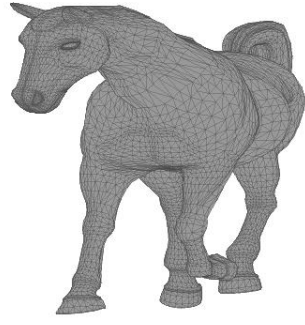


(e) Pose 05 - Error Heat Map
Pose Error: 1.1176×10^{-5}
Std Dev: 0.00231

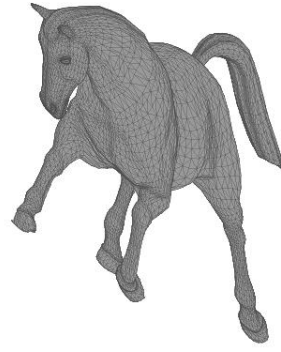


(f) Pose 06 - Error Heat Map
Pose Error: 1.4662×10^{-5}
Std Dev: 0.00267

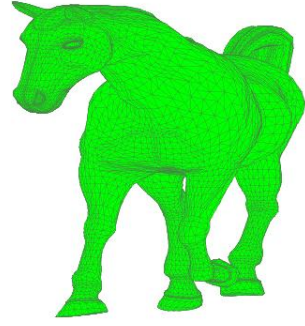
Figure 5.24: **Optimization Results for Horse Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 05 [left column] and Pose 06 [right column].



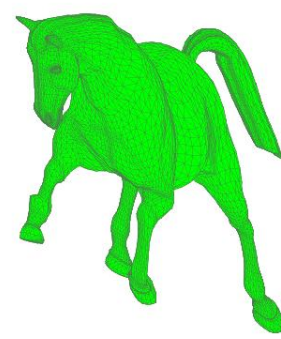
(a) Pose 07 - Expected Vertices



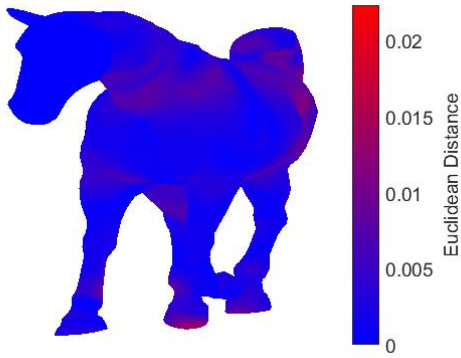
(b) Pose 08 - Expected Vertices



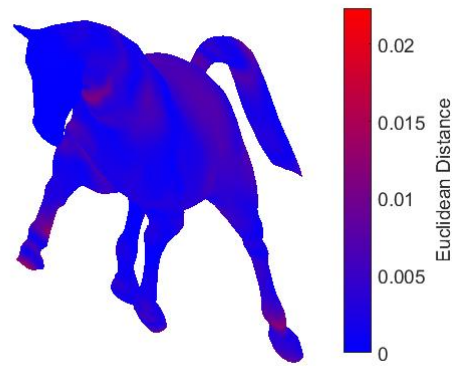
(c) Pose 07 - Computed Vertices



(d) Pose 08 - Computed Vertices

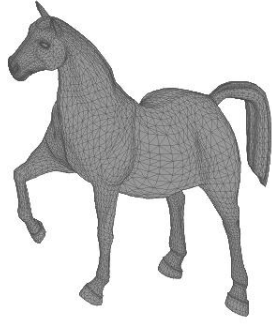


(e) Pose 07 - Error Heat Map
Pose Error: 1.7544×10^{-5}
Std Dev: 0.00303

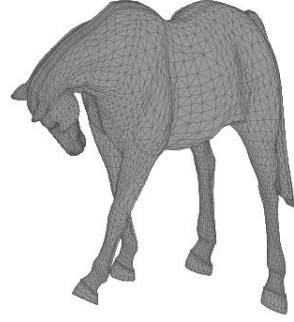


(f) Pose 08 - Error Heat Map
Pose Error: 1.5033×10^{-5}
Std Dev: 0.00276

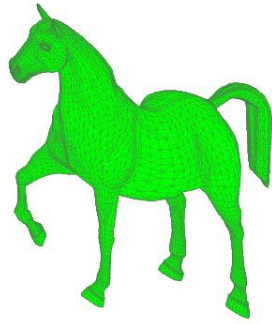
Figure 5.25: **Optimization Results for Horse Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 07 [left column] and Pose 08 [right column].



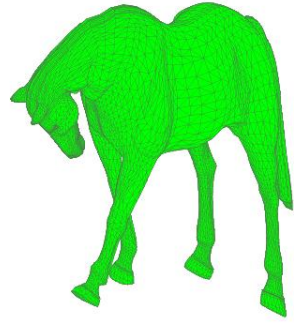
(a) Pose 09 - Expected Vertices



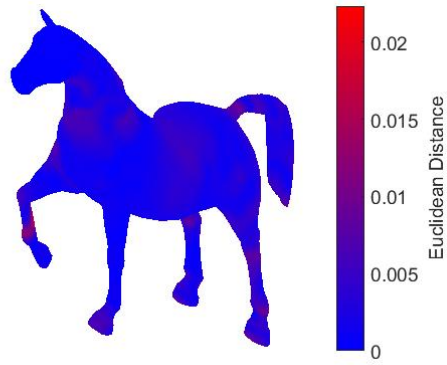
(b) Pose 10 - Expected Vertices



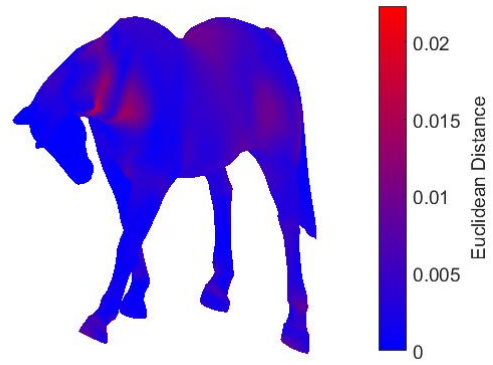
(c) Pose 09 - Computed Vertices



(d) Pose 10 - Computed Vertices

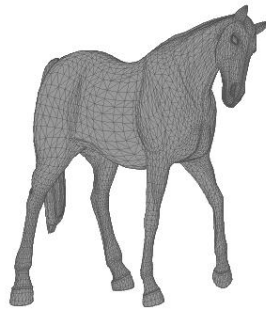


(e) Pose 09 - Error Heat Map
Pose Error: 1.0281×10^{-5}
Std Dev: 0.00221

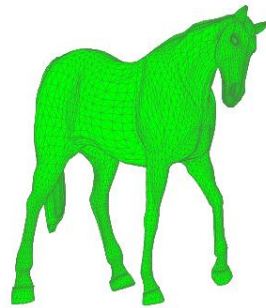


(f) Pose 10 - Error Heat Map
Pose Error: 1.3649×10^{-5}
Std Dev: 0.0026

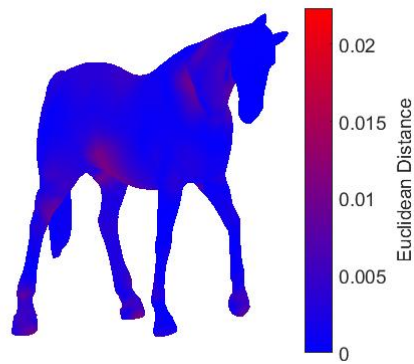
Figure 5.26: **Optimization Results for Horse Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 09 [left column] and Pose 10 [right column].



(a) Pose 11 - Expected Vertices



(b) Pose 11 - Computed Vertices



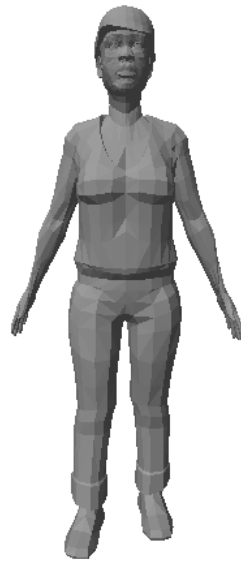
(c) Pose 11 - Error Heat Map

Pose Error: 9.5436×10^{-6}

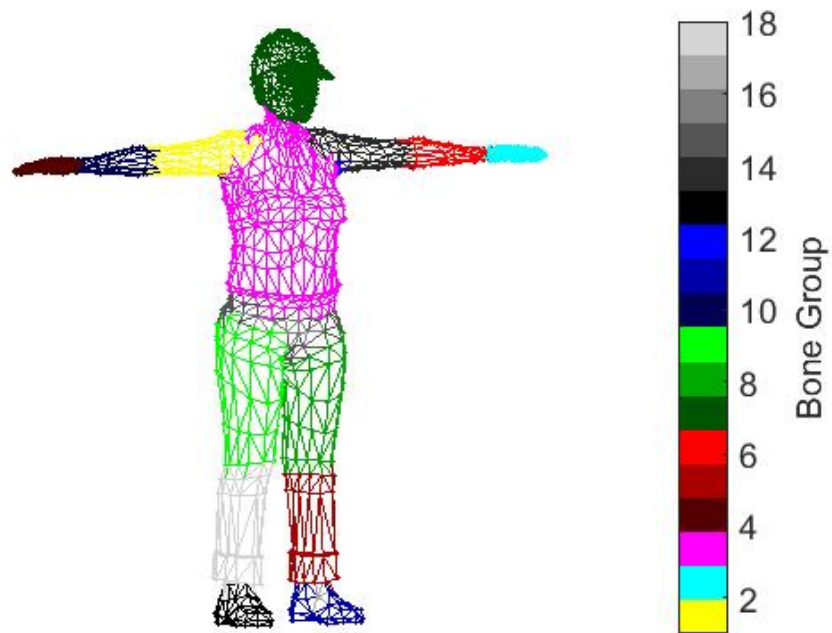
Std Dev: 0.00218

Figure 5.27: **Optimization Results for Horse Model** - Expected vertex positions [top:(a)], Computed vertex positions [middle:(b)] and Error Heat Map [bottom:(c)] for Pose 11.

5.3.5 Model: Woman



(a)



(b)

Figure 5.28: **Initialization Results for Woman Model** - 5.28a Model. 5.28b Initialization Results: Singleton bone groups.

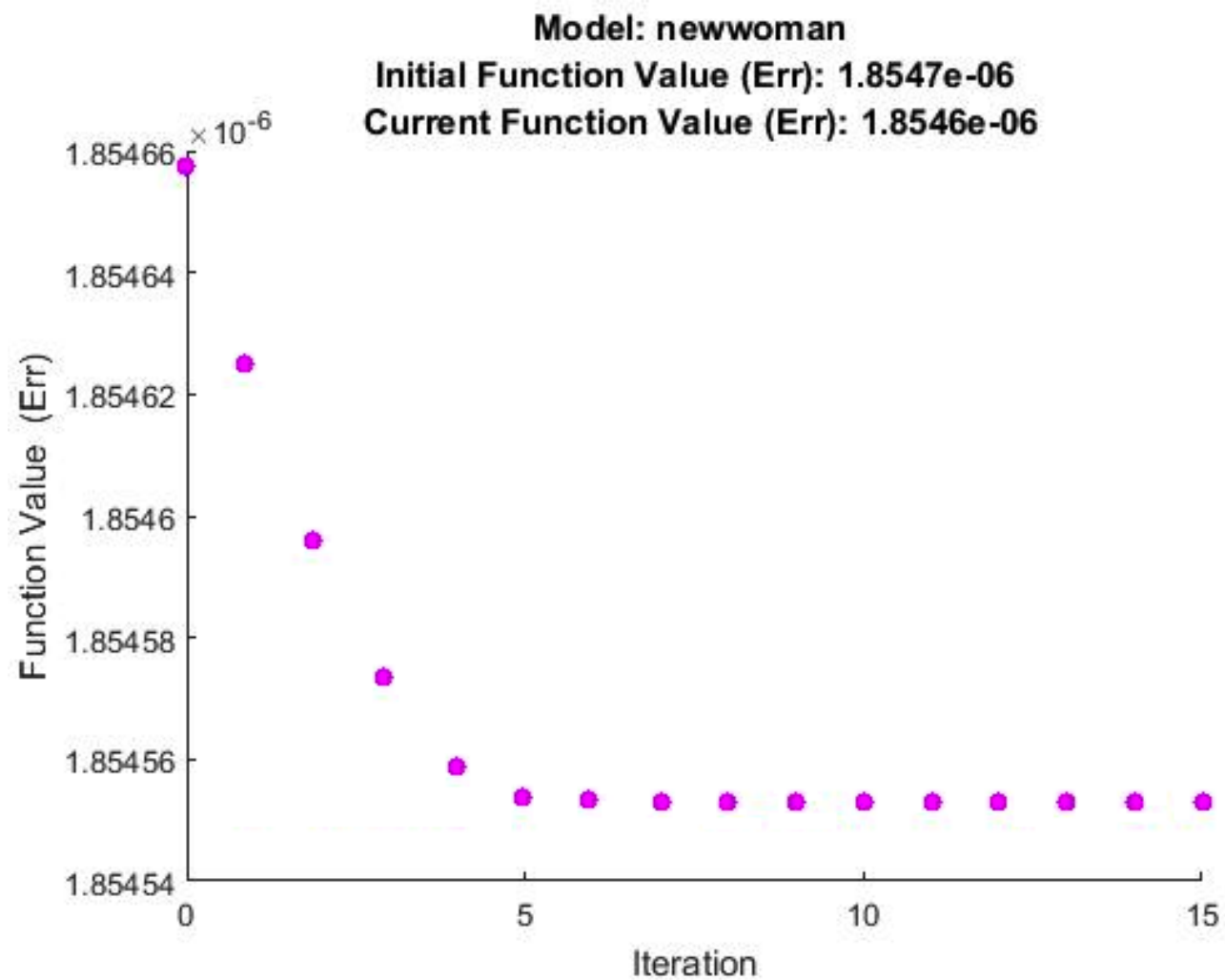
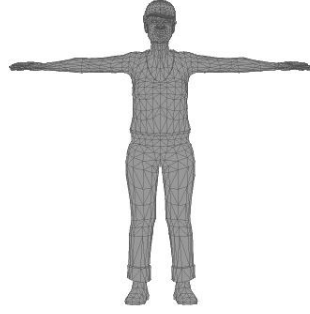
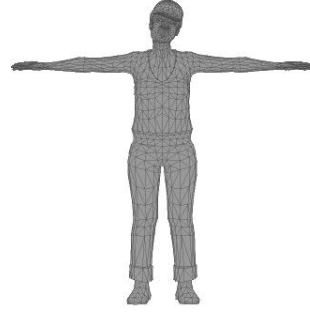


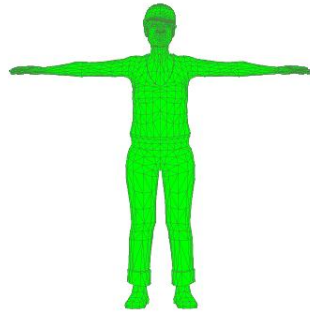
Figure 5.29: **Optimization Results for Woman Model** - Progress of EP-LBS minimization of the objective function.



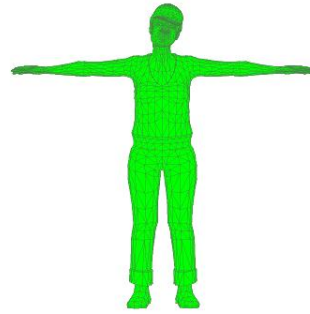
(a) Pose 01 - Expected Vertices



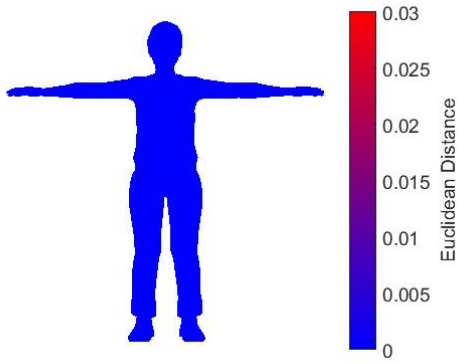
(b) Pose 02 - Expected Vertices



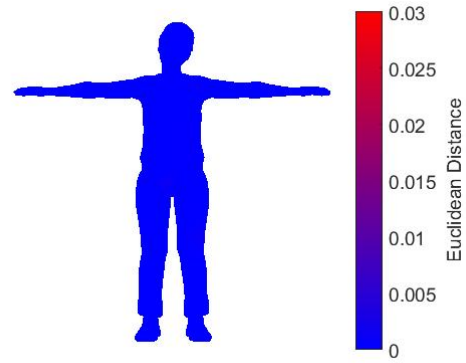
(c) Pose 01 - Computed Vertices



(d) Pose 02 - Computed Vertices

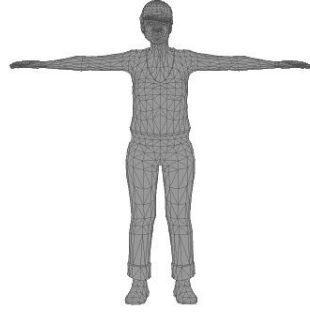


(e) Pose 01 - Error Heat Map
Pose Error: 6.263×10^{-10}
Std Dev: 1.42×10^{-6}

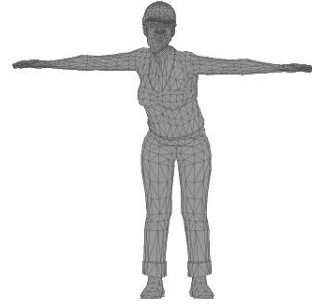


(f) Pose 02 - Error Heat Map
Pose Error: 1.602×10^{-8}
Std Dev: 0.000113

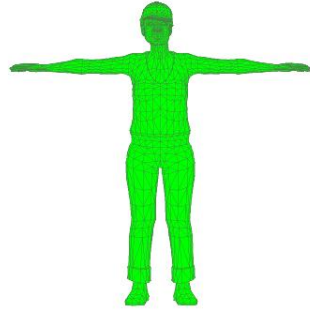
Figure 5.30: **Optimization Results for Woman Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 01 [left column] and Pose 02 [right column].



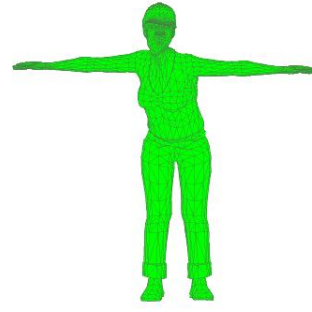
(a) Pose 03 - Expected Vertices



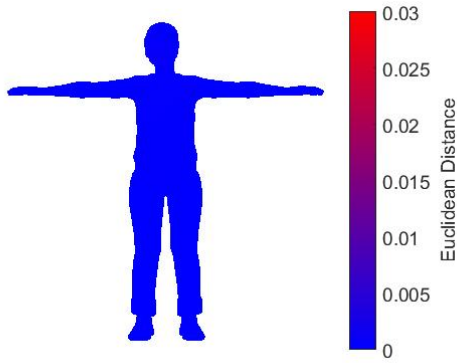
(b) Pose 04 - Expected Vertices



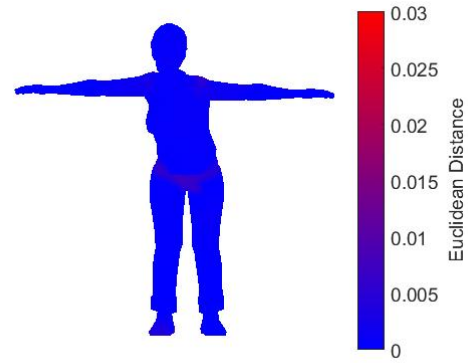
(c) Pose 03 - Computed Vertices



(d) Pose 04 - Computed Vertices

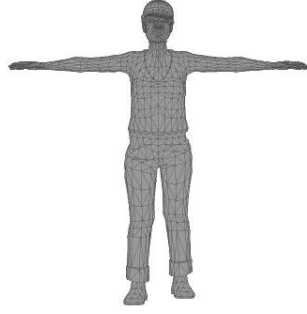


(e) Pose 03 - Error Heat Map
Pose Error: 1.245×10^{-7}
Std Dev: 0.000268

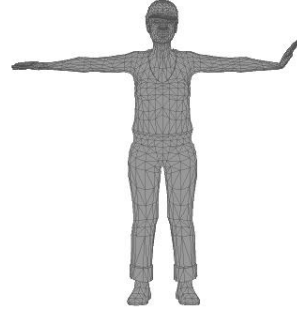


(f) Pose 04 - Error Heat Map
Pose Error: 7.8832×10^{-7}
Std Dev: 0.000762

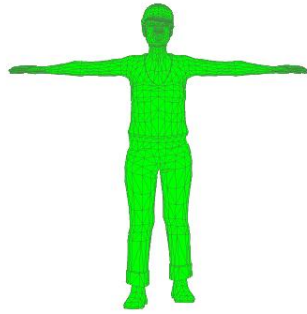
Figure 5.31: **Optimization Results for Woman Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 03 [left column] and Pose 04 [right column].



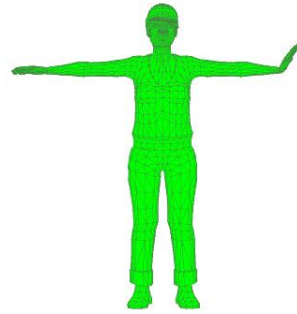
(a) Pose 05 - Expected Vertices



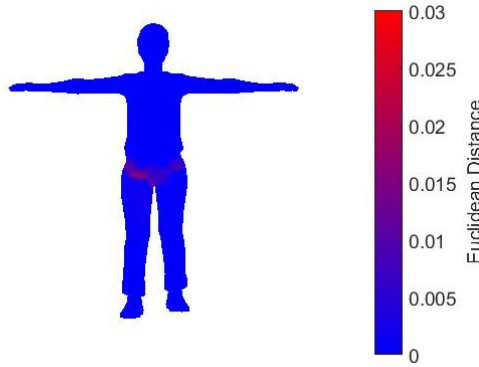
(b) Pose 06 - Expected Vertices



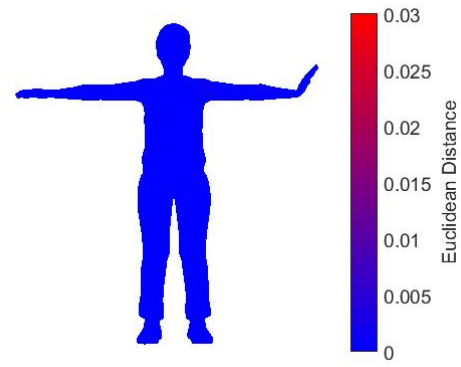
(c) Pose 05 - Computed Vertices



(d) Pose 06 - Computed Vertices

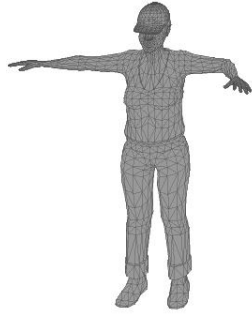


(e) Pose 05 - Error Heat Map
Pose Error: 1.8677×10^{-6}
Std Dev: 0.00134

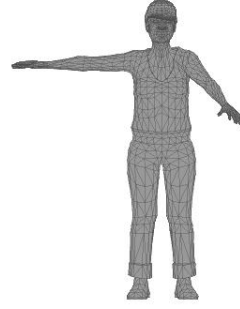


(f) Pose 06 - Error Heat Map
Pose Error: 7.9948×10^{-8}
Std Dev: 0.000273

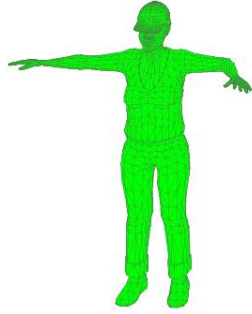
Figure 5.32: **Optimization Results for Woman Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 05 [left column] and Pose 06 [right column].



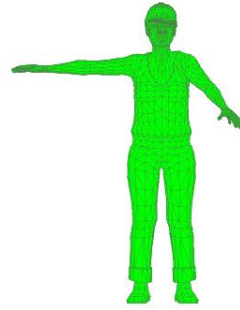
(a) Pose 07 - Expected Vertices



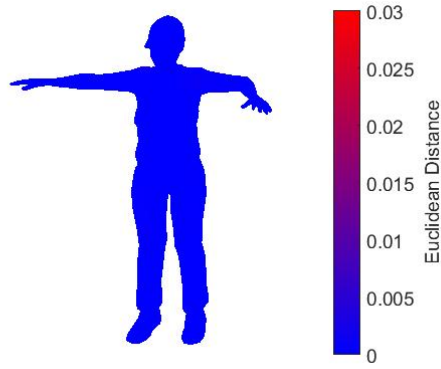
(b) Pose 08 - Expected Vertices



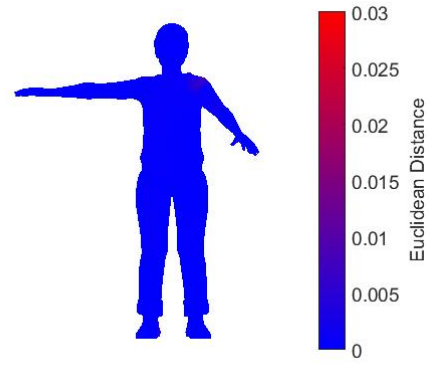
(c) Pose 07 - Computed Vertices



(d) Pose 08 - Computed Vertices

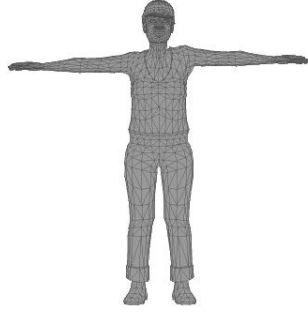


(e) Pose 07 - Error Heat Map
Pose Error: 2.6813×10^{-8}
Std Dev: 0.000142

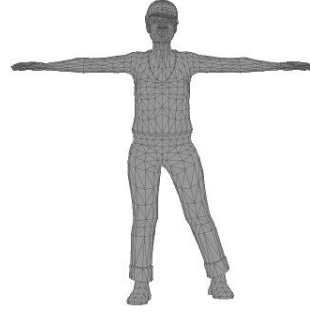


(f) Pose 08 - Error Heat Map
Pose Error: 3.3986×10^{-7}
Std Dev: 0.000547

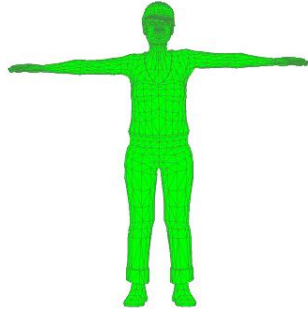
Figure 5.33: **Optimization Results for Woman Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 07 [left column] and Pose 08 [right column].



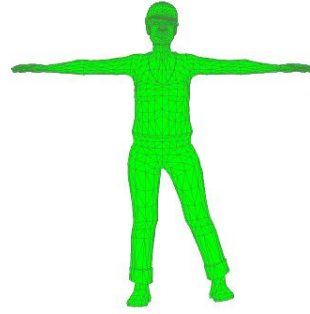
(a) Pose 09 - Expected Vertices



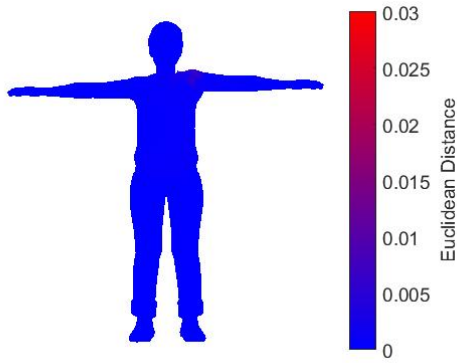
(b) Pose 10 - Expected Vertices



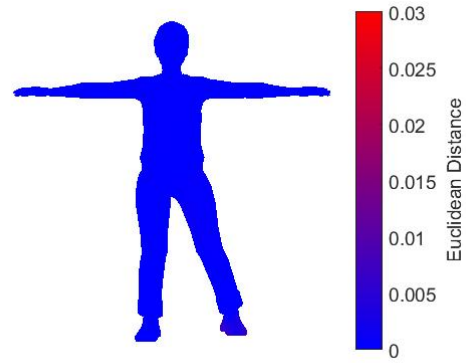
(c) Pose 09 - Computed Vertices



(d) Pose 10 - Computed Vertices

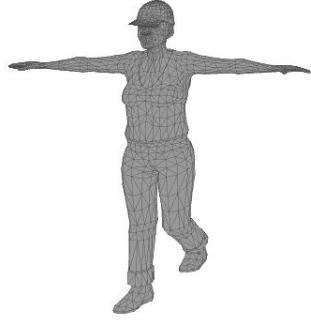


(e) Pose 09 - Error Heat Map
Pose Error: 1.9921×10^{-7}
Std Dev: 0.000423

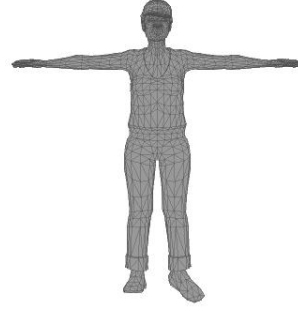


(f) Pose 10 - Error Heat Map
Pose Error: 3.802×10^{-7}
Std Dev: 0.000603

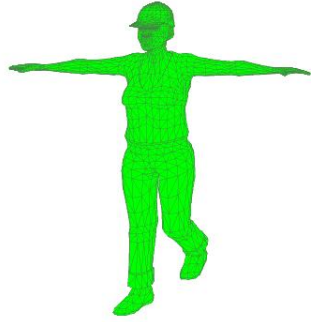
Figure 5.34: **Optimization Results for Woman Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 09 [left column] and Pose 10 [right column].



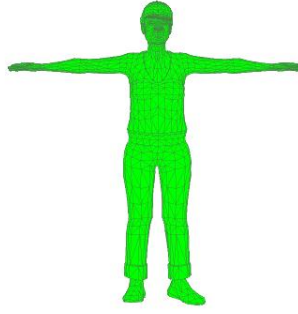
(a) Pose 11 - Expected Vertices



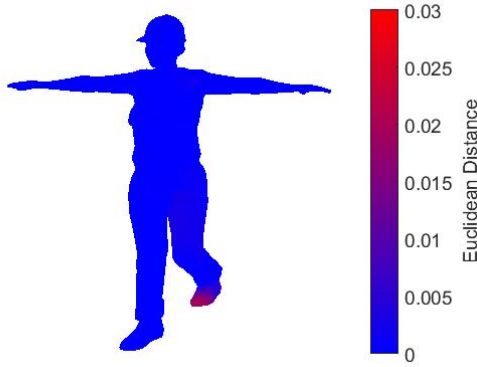
(b) Pose 12 - Expected Vertices



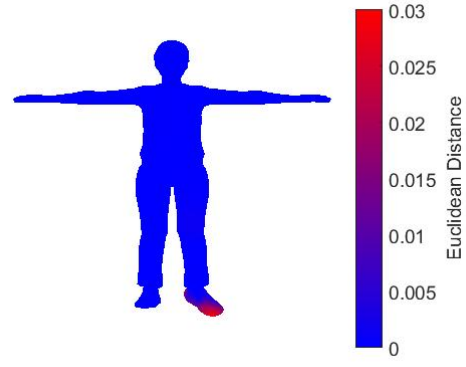
(c) Pose 11 - Computed Vertices



(d) Pose 12 - Computed Vertices

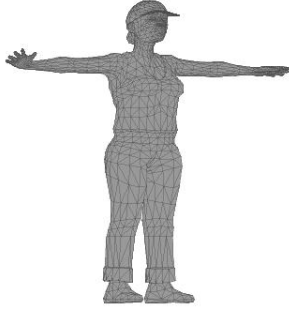


(e) Pose 11 - Error Heat Map
Pose Error: 2.1106×10^{-6}
Std Dev: 0.00143

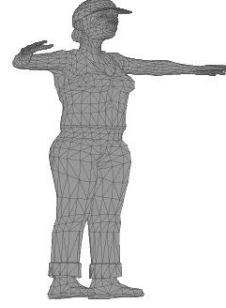


(f) Pose 12 - Error Heat Map
Pose Error: 4.2109×10^{-6}
Std Dev: 0.00203

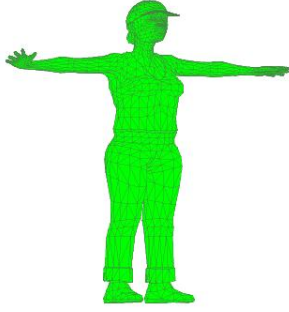
Figure 5.35: **Optimization Results for Woman Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 11 [left column] and Pose 12 [right column].



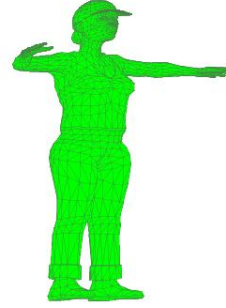
(a) Pose 13 - Expected Vertices



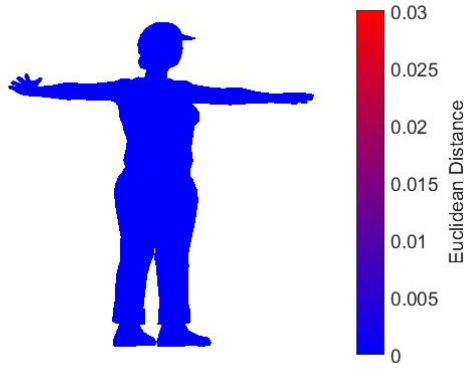
(b) Pose 14 - Expected Vertices



(c) Pose 13 - Computed Vertices



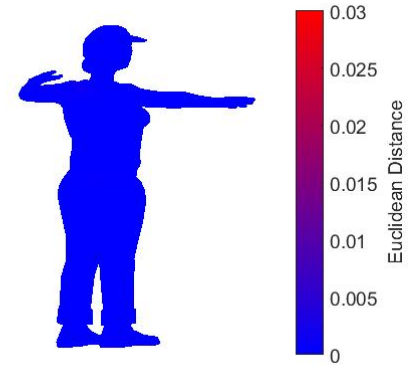
(d) Pose 14 - Computed Vertices



(e) PPose 13 - Error Heat Map

Pose Error: 1.1195×10^{-8}

Std Dev: 9.23×10^{-5}

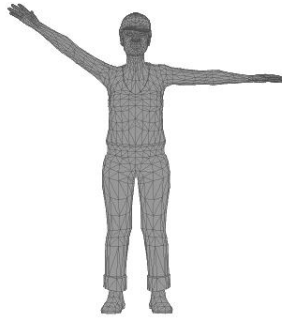


(f) Pose 14 - Error Heat Map

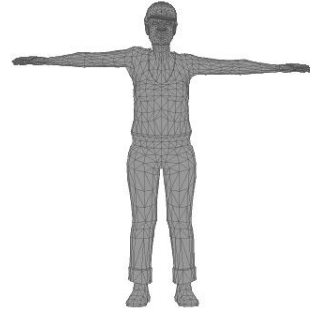
Pose Error: 1.8168×10^{-8}

Std Dev: 0.000124

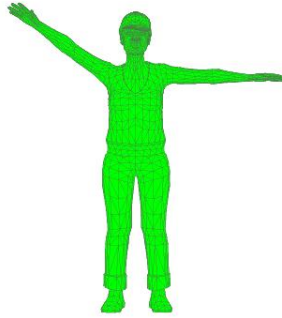
Figure 5.36: **Optimization Results for Woman Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 13 [left column] and Pose 14 [right column].



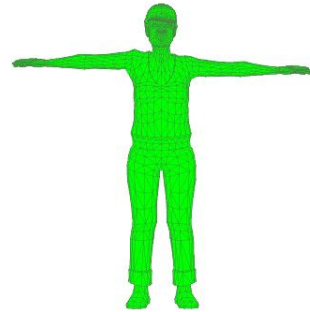
(a) Pose 15 - Expected Vertices



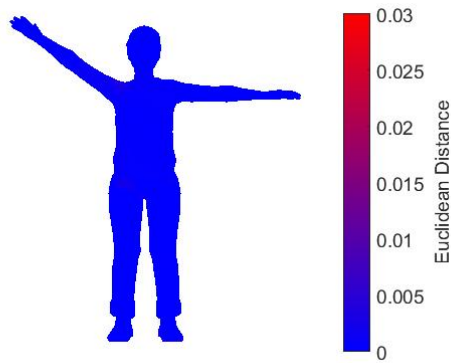
(b) Pose 16 - Expected Vertices



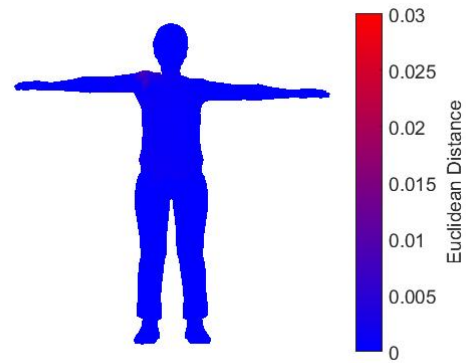
(c) Pose 15 - Computed Vertices



(d) Pose 16 - Computed Vertices

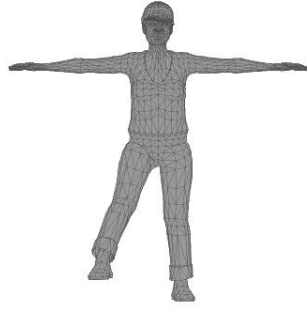


(e) Pose 15 - Error Heat Map
Pose Error: 1.1575×10^{-7}
Std Dev: 0.000312

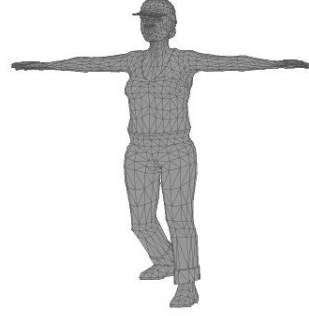


(f) Pose 16 - Error Heat Map
Pose Error: 2.9547×10^{-7}
Std Dev: 0.000516

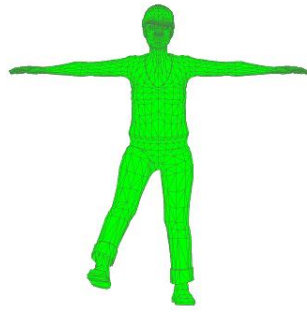
Figure 5.37: **Optimization Results for Woman Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 15 [left column] and Pose 16 [right column].



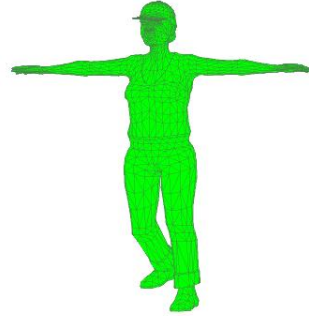
(a) Pose 17 - Expected Vertices



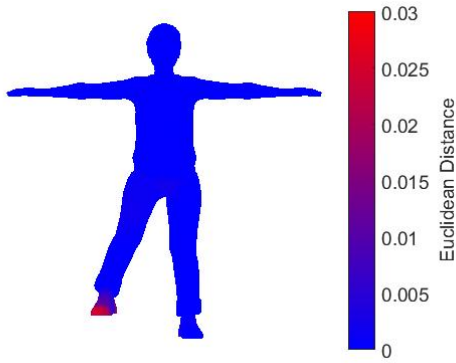
(b) Pose 18 - Expected Vertices



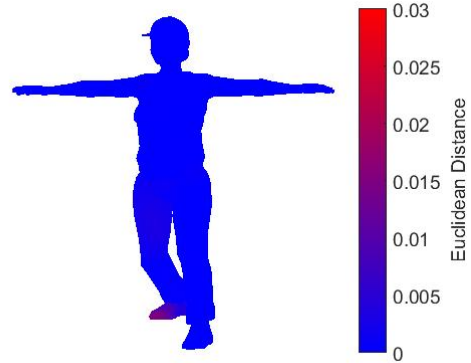
(c) Pose 17 - Computed Vertices



(d) Pose 18 - Computed Vertices

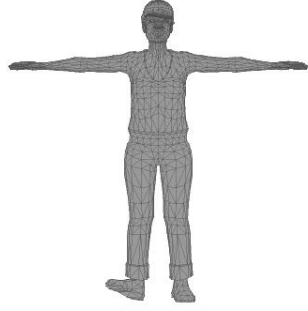


(e) Pose 17 - Error Heat Map
Pose Error: 3.95×10^{-6}
Std Dev: 0.00196

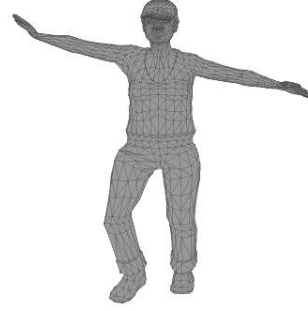


(f) PPose 18 - Error Heat Map
Pose Error: 1.5105×10^{-6}
Std Dev: 0.0012

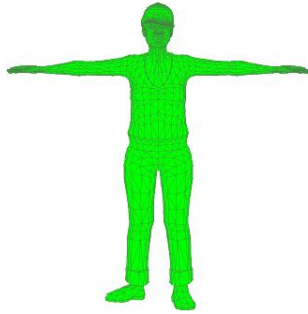
Figure 5.38: **Optimization Results for Woman Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 17 [left column] and Pose 18 [right column].



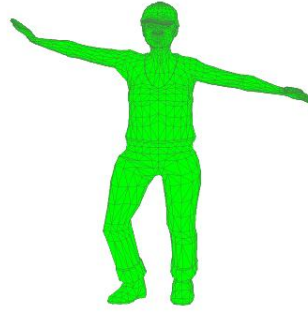
(a) Pose 19 - Expected Vertices



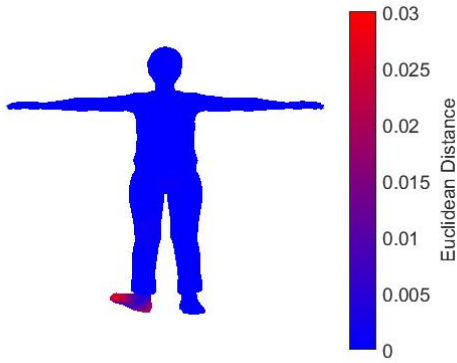
(b) Pose 20 - Expected Vertices



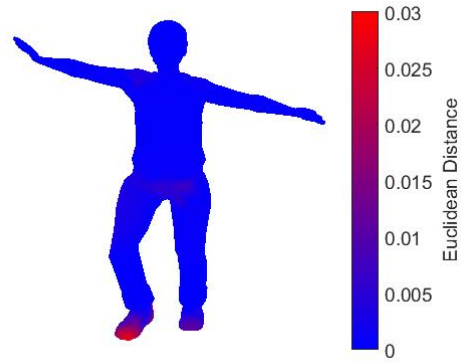
(c) Pose 19 - Computed Vertices



(d) Pose 20 - Computed Vertices

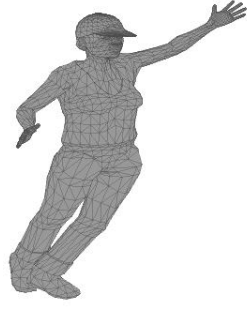


(e) Pose 19 - Error Heat Map
Pose Error: 3.9259×10^{-6}
Std Dev: 0.00196

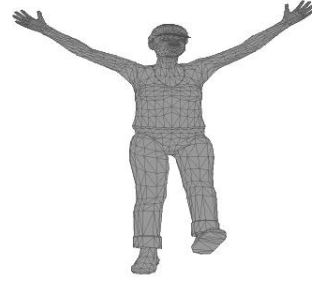


(f) PPose 20 - Error Heat Map
Pose Error: 5.3775×10^{-6}
Std Dev: 0.00222

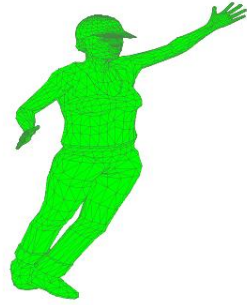
Figure 5.39: **Optimization Results for Woman Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 19 [left column] and Pose 20 [right column].



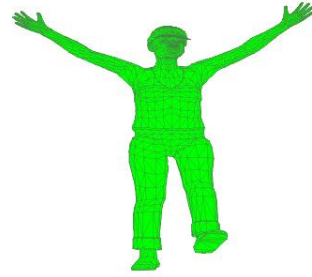
(a) Pose 21 - Expected Vertices



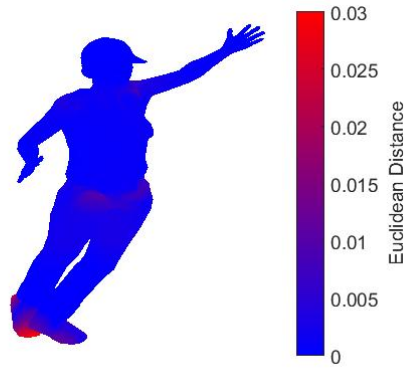
(b) Pose 22 - Expected Vertices



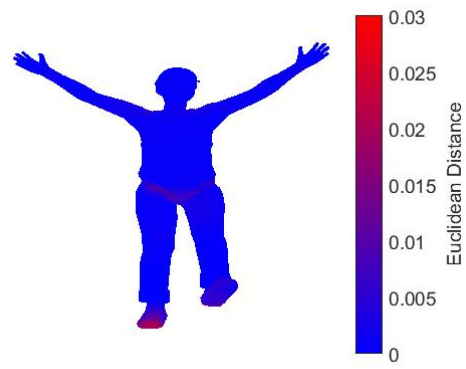
(c) Pose 21 - Computed Vertices



(d) Pose 22 - Computed Vertices

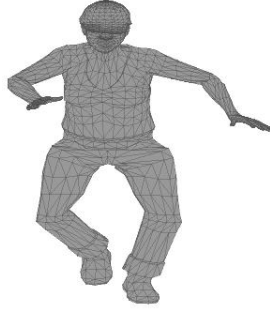


(e) Pose 21 - Error Heat Map
Pose Error: 7.0971×10^{-6}
Std Dev: 0.00251

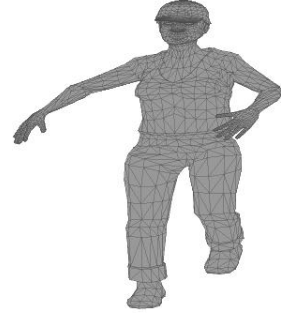


(f) Pose 22 - Error Heat Map
Pose Error: 4.5842×10^{-6}
Std Dev: 0.00195

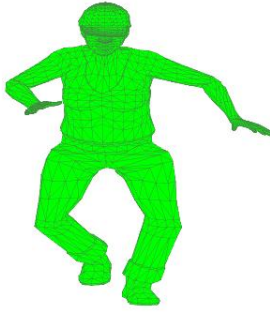
Figure 5.40: **Optimization Results for Woman Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 21 [left column] and Pose 22 [right column].



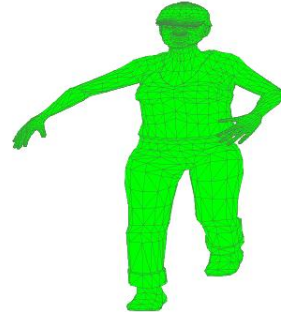
(a) Pose 23 - Expected Vertices



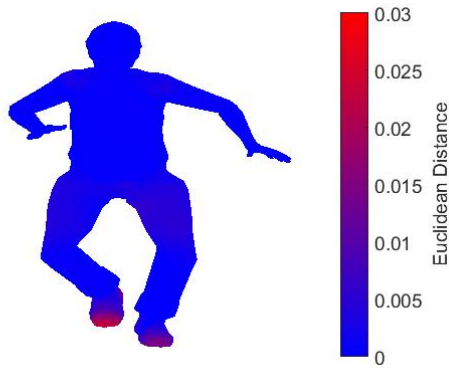
(b) Pose 24 - Expected Vertices



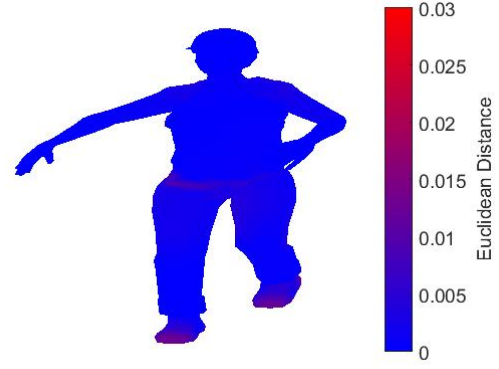
(c) Pose 23 - Computed Vertices



(d) Pose 24 - Computed Vertices



(e) Pose 23 - Error Heat Map
Pose Error: 4.7154×10^{-6}
Std Dev: 0.00206



(f) Pose 24 - Error Heat Map
Pose Error: 2.7975×10^{-6}
Std Dev: 0.00156

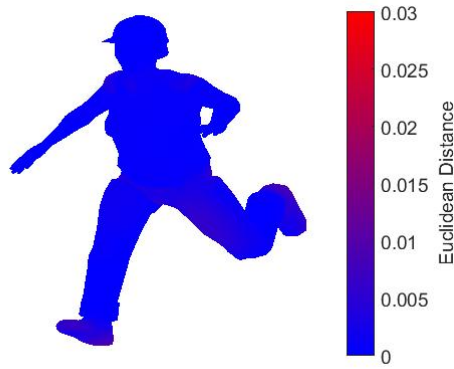
Figure 5.41: **Optimization Results for Woman Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 23 [left column] and Pose 24 [right column].



(a) Pose 25 - Expected Vertices



(b) Pose 25 - Computed Vertices



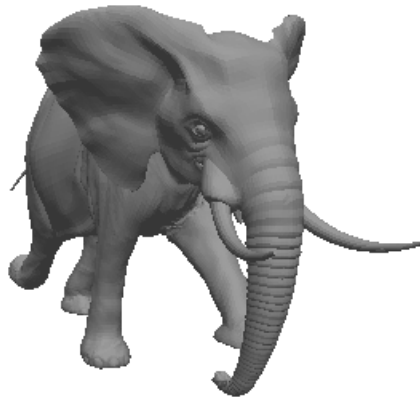
(c) Pose 25 - Error Heat Map

Pose Error: 1.8203×10^{-6}

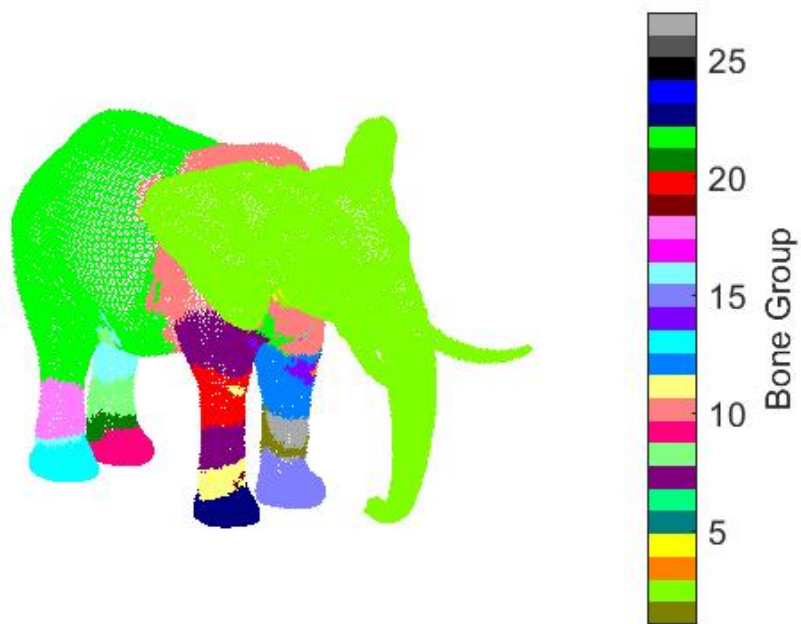
Std Dev: 0.00122

Figure 5.42: **Optimization Results for Woman Model** - Expected vertex positions [top:(a)], Computed vertex positions [middle:(b)] and Error Heat Map [bottom:(c)] for Pose 15 [left column].

5.3.6 Model: Elephant



(a)



(b)

Figure 5.43: **Initialization Results for Elephant Model** - 5.43a Model. 5.43b Initialization Results: Singleton bone groups.

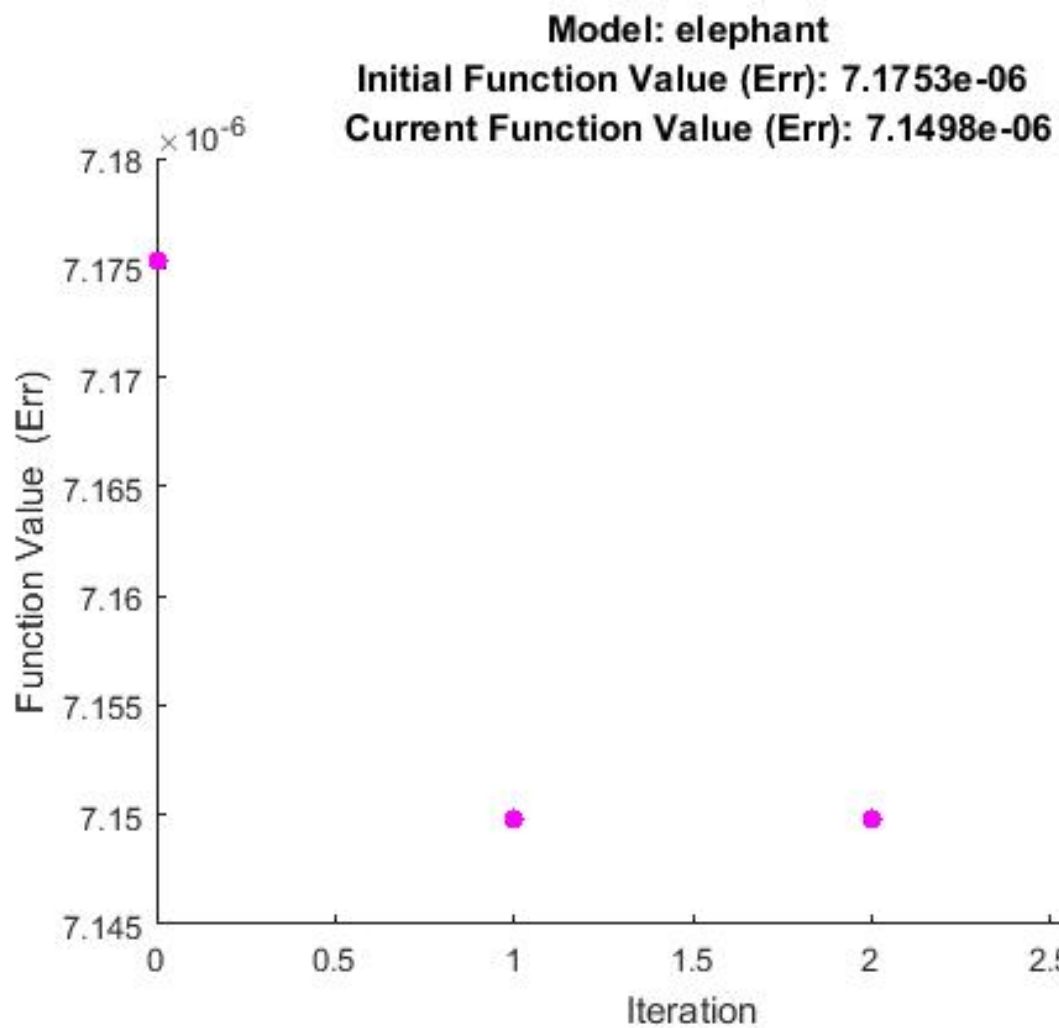
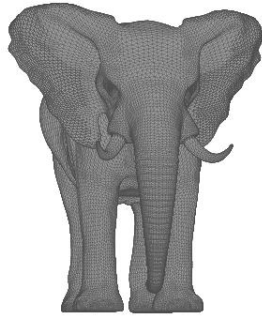
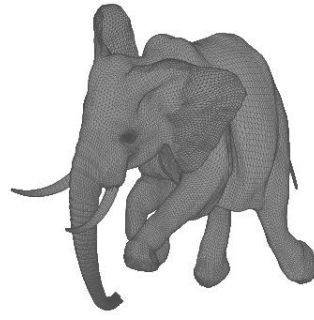


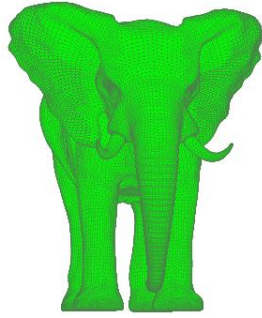
Figure 5.44: **Optimization Results for Elephant Model** - Progress of EP-LBS minimization of the objective function.



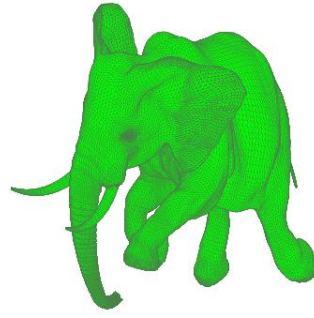
(a) Pose 01 - Expected Vertices



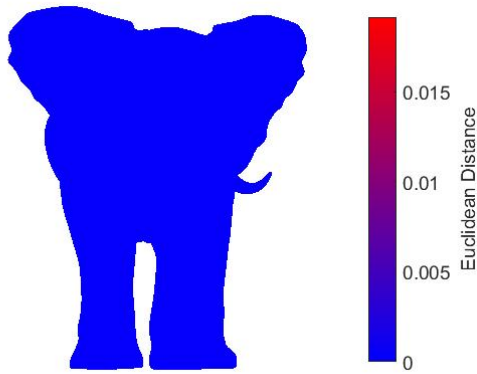
(b) Pose 02 - Expected Vertices



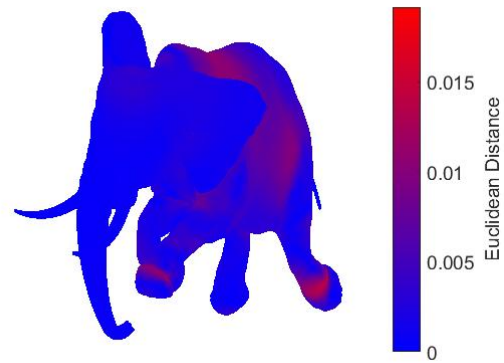
(c) Pose 01 - Computed Vertices



(d) Pose 02 - Computed Vertices

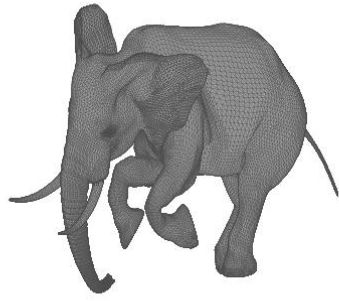


(e) Pose 01 - Error Heat Map
 Pose Error: 8.4215×10^{-8}
 Std Dev: 1.76×10^{-6}

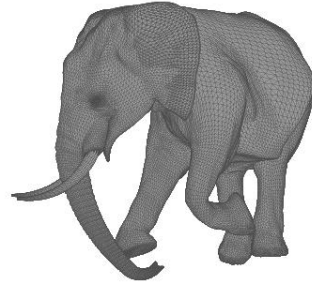


(f) Pose 02 - Error Heat Map
 Pose Error: 8.7393×10^{-6}
 Std Dev: 0.00203

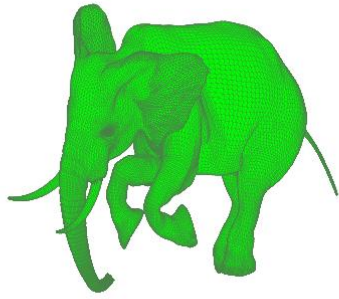
Figure 5.45: **Optimization Results for Elephant Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 01 [left column] and Pose 02 [right column].



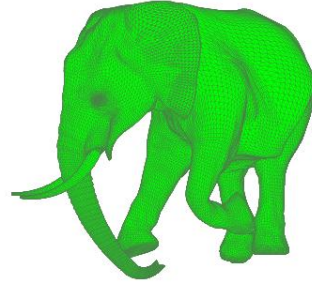
(a) Pose 03 - Expected Vertices



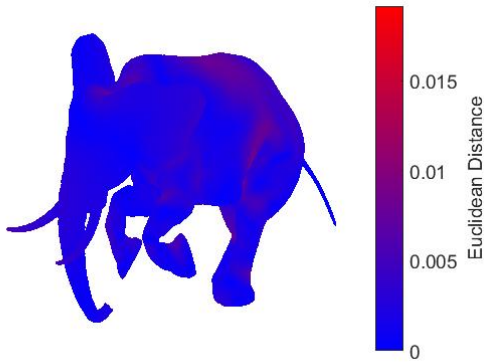
(b) Pose 04 - Expected Vertices



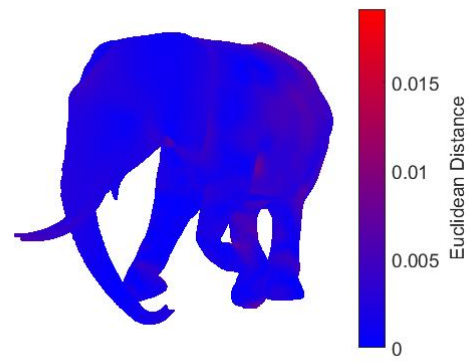
(c) Pose 03 - Computed Vertices



(d) Pose 04 - Computed Vertices

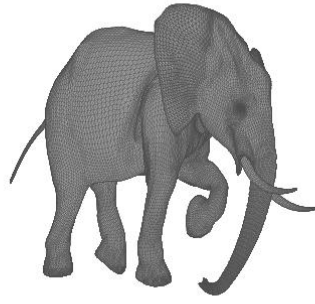


(e) Pose 03 - Error Heat Map
Pose Error: 6.5827×10^{-6}
Std Dev: 0.00137

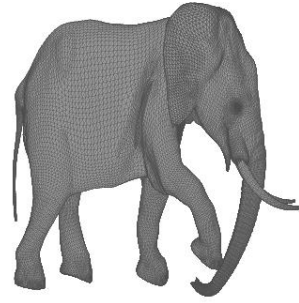


(f) Pose 04 - Error Heat Map
Pose Error: 6.9776×10^{-6}
Std Dev: 0.00148

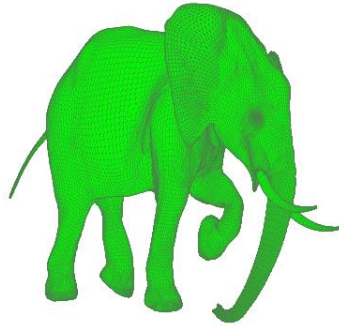
Figure 5.46: **Optimization Results for Elephant Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 03 [left column] and Pose 04 [right column].



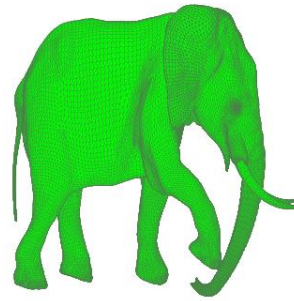
(a) Pose 05 - Expected Vertices



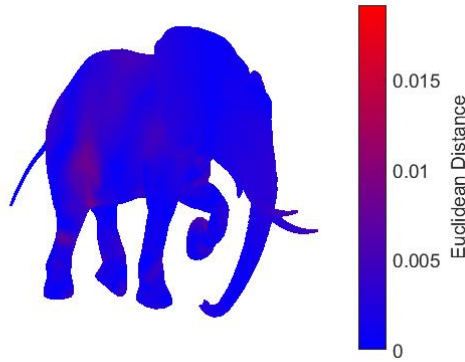
(b) Pose 06 - Expected Vertices



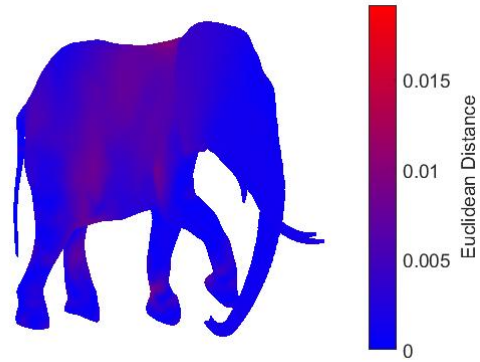
(c) Pose 05 - Computed Vertices



(d) Pose 06 - Computed Vertices

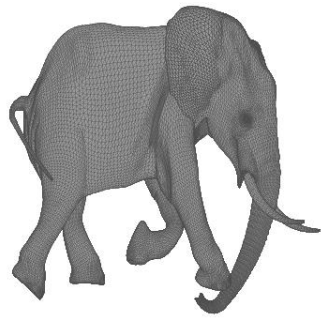


(e) Pose 05 - Error Heat Map
Pose Error: 5.7952×10^{-6}
Std Dev: 0.00123

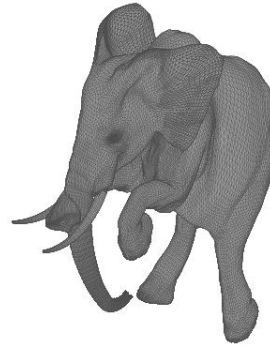


(f) Pose 06 - Error Heat Map
Pose Error: 7.7641×10^{-6}
Std Dev: 0.00169

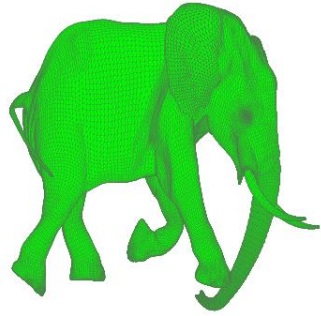
Figure 5.47: **Optimization Results for Elephant Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 05 [left column] and Pose 06 [right column].



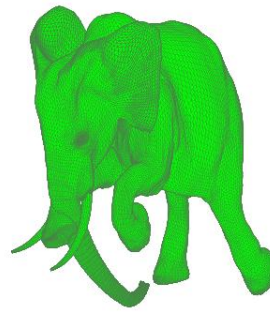
(a) Pose 07 - Expected Vertices



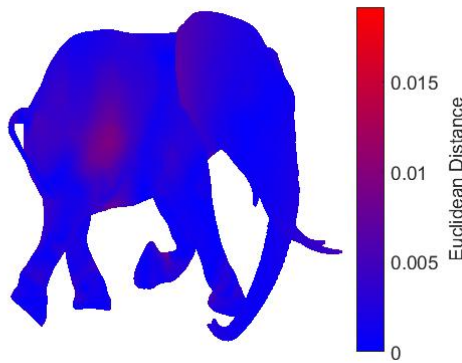
(b) Pose 08 - Expected Vertices



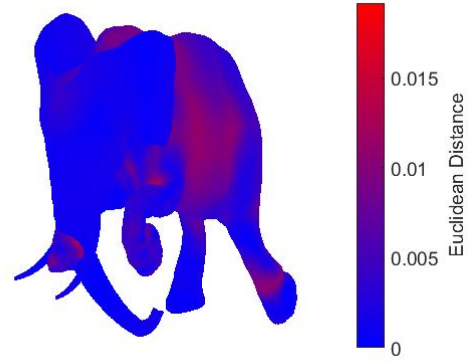
(c) Pose 07 - Computed Vertices



(d) Pose 08 - Computed Vertices

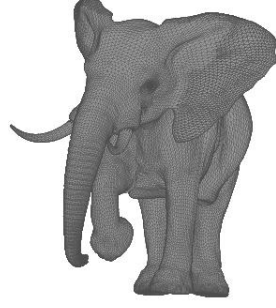


(e) Pose 07 - Error Heat Map
Pose Error: 6.5512×10^{-6}
Std Dev: 0.00141

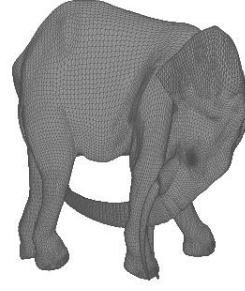


(f) Pose 08 - Error Heat Map
Pose Error: 1.0876×10^{-5}
Std Dev: 0.00223

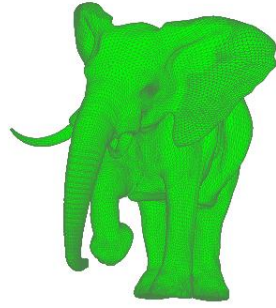
Figure 5.48: **Optimization Results for Elephant Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 07 [left column] and Pose 08 [right column].



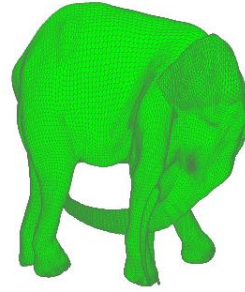
(a) Pose 09 - Expected Vertices



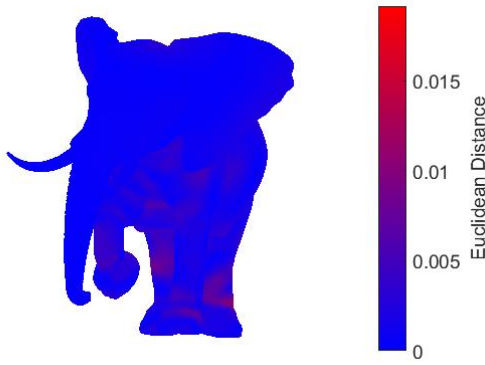
(b) Pose 10 - Expected Vertices



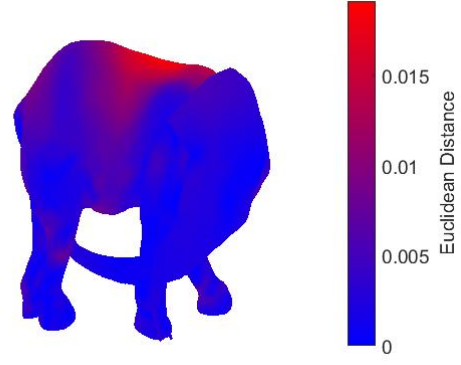
(c) Pose 09 - Computed Vertices



(d) Pose 10 - Computed Vertices

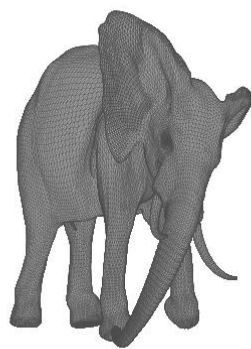


(e) Pose 09 - Error Heat Map
Pose Error: 4.7429×10^{-6}
Std Dev: 0.00145

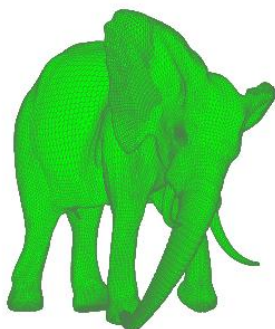


(f) Pose 10 - Error Heat Map
Pose Error: 1.502×10^{-5}
Std Dev: 0.00249

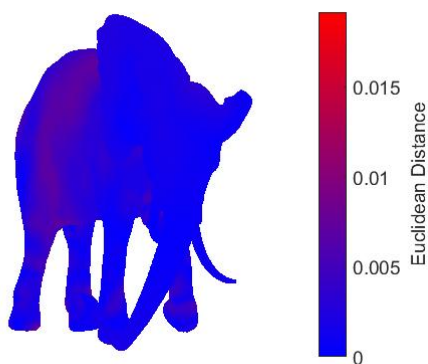
Figure 5.49: **Optimization Results for Elephant Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 09 [left column] and Pose 10 [right column].



(a) Pose 11 - Expected Vertices



(b) Pose 11 - Computed Vertices



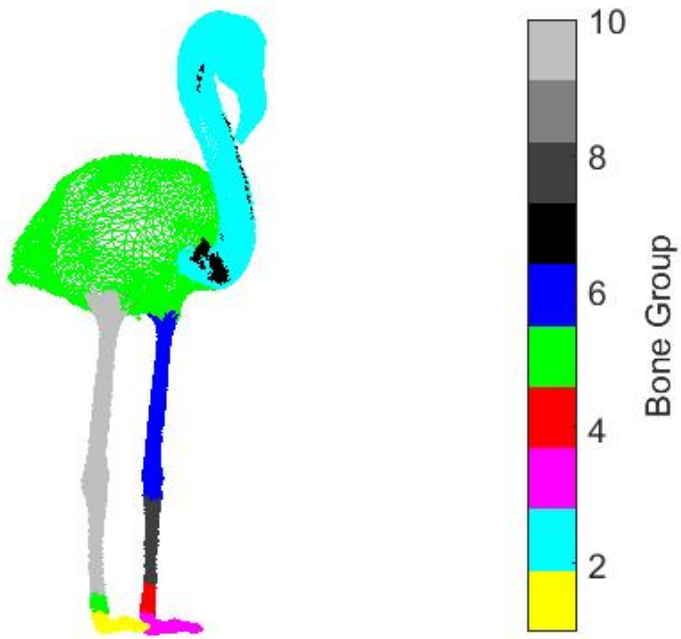
(c) Pose 11 - Error Heat Map
 Pose Error: 5.5141×10^{-6}
 Std Dev: 0.00136

Figure 5.50: **Optimization Results for Elephant Model** - Expected vertex positions [top:(a)], Computed vertex positions [middle:(b)] and Error Heat Map [bottom:(c)] for Pose 11.

5.3.7 Model: Flamingo



(a)



(b)

Figure 5.51: **Initialization Results for Flamingo Model** - 5.51a Model. 5.51b Initialization Results: Singleton bone groups.

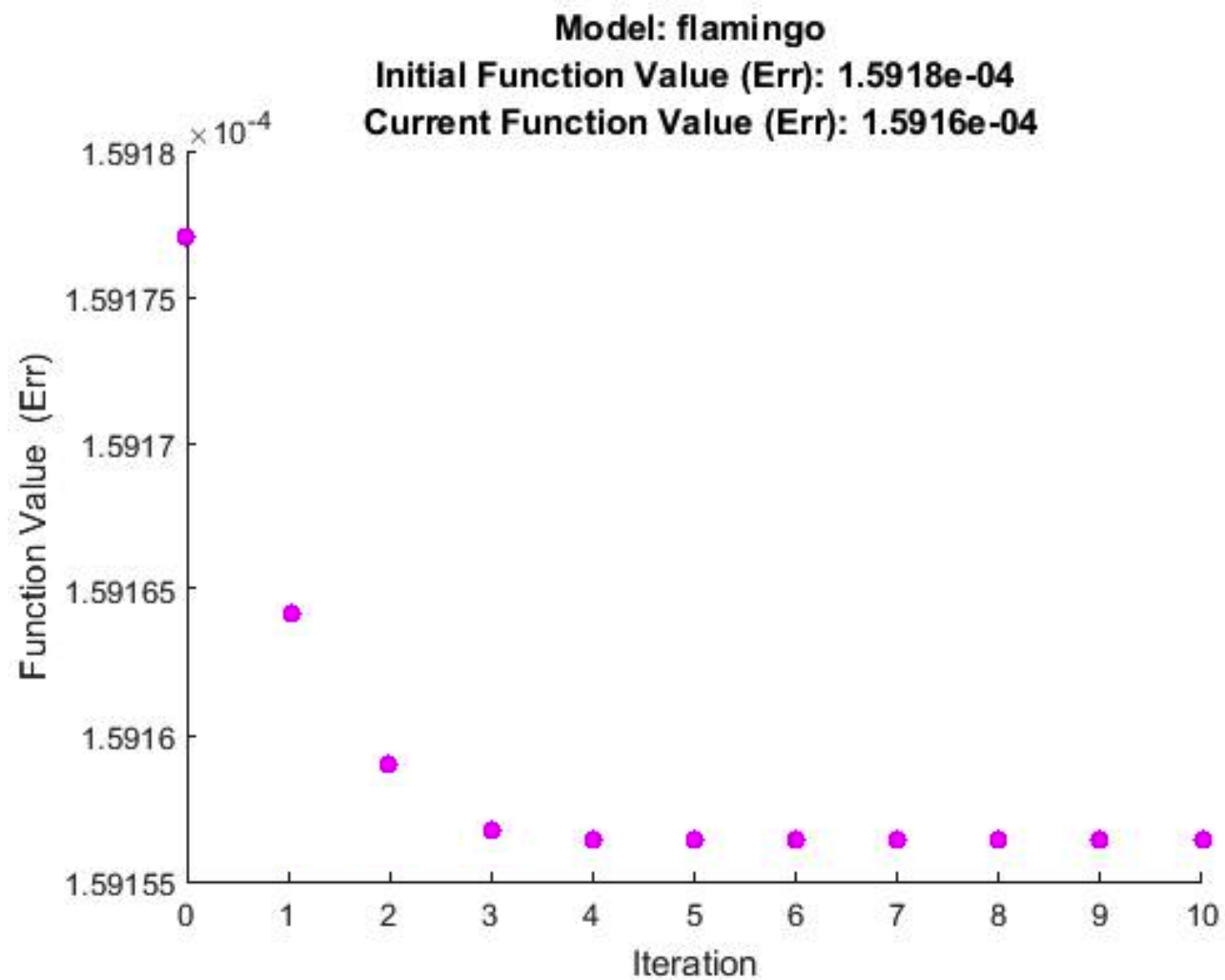
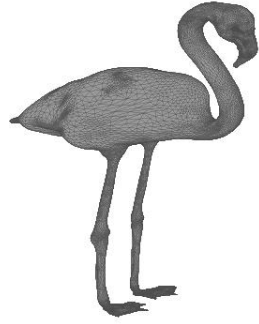
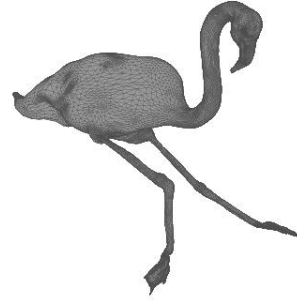


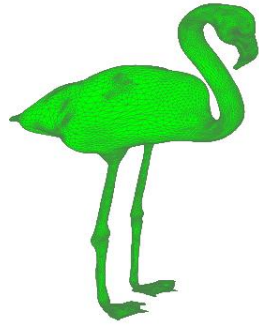
Figure 5.52: **Optimization Results for Flamingo Model** - Progress of EP-LBS minimization of the objective function.



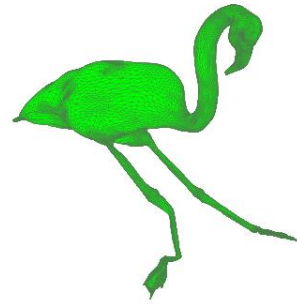
(a) Pose 01 - Expected Vertices



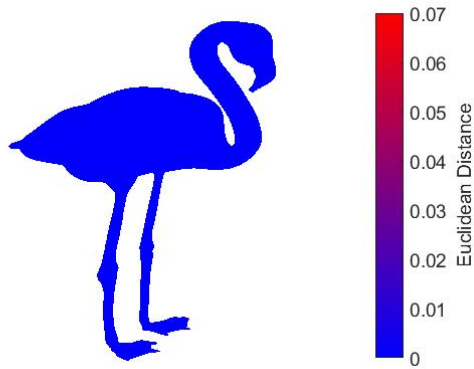
(b) Pose 02 - Expected Vertices



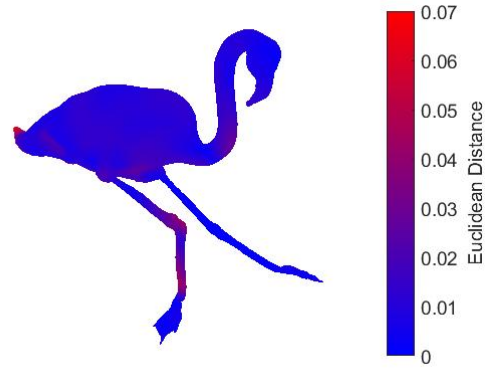
(c) Pose 01 - Computed Vertices



(d) Pose 02 - Computed Vertices



(e) Pose 01 - Error Heat Map
Pose Error: 2.6478×10^{-7}
Std Dev: 1.22×10^{-6}

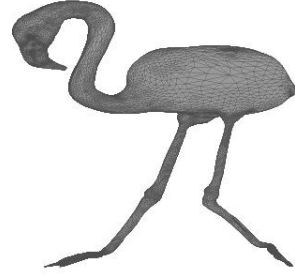


(f) Pose 02 - Error Heat Map
Pose Error: 1.7839×10^{-4}
Std Dev: 0.00942

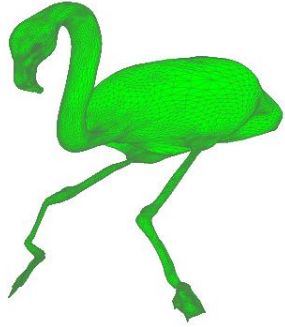
Figure 5.53: **Optimization Results for Flamingo Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 01 [left column] and Pose 02 [right column].



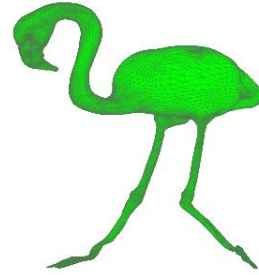
(a) Pose 03 - Expected Vertices



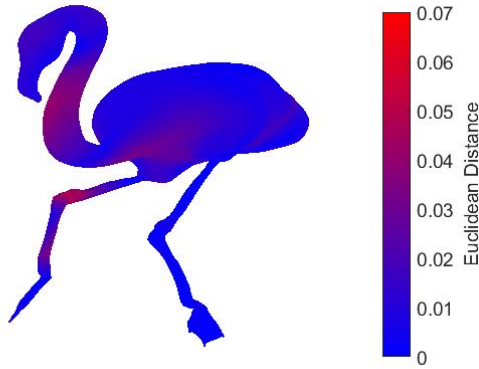
(b) Pose 04 - Expected Vertices



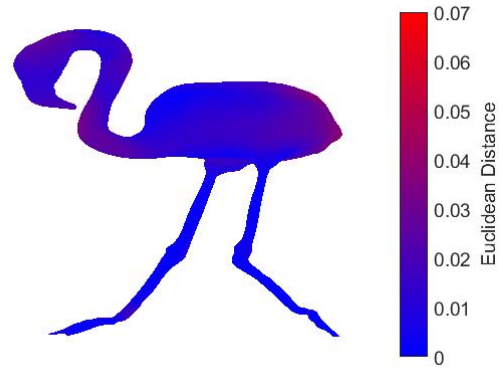
(c) Pose 03 - Computed Vertices



(d) Pose 04 - Computed Vertices

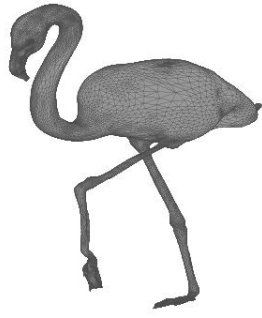


(e) Pose 03 - Error Heat Map
Pose Error: 1.6126×10^{-4}
Std Dev: 0.0087

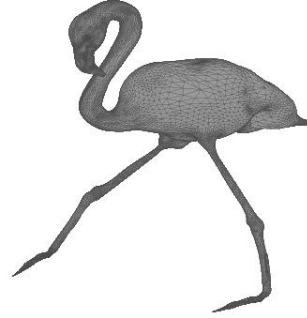


(f) Pose 04 - Error Heat Map
Pose Error: 2.2696×10^{-4}
Std Dev: 0.0101

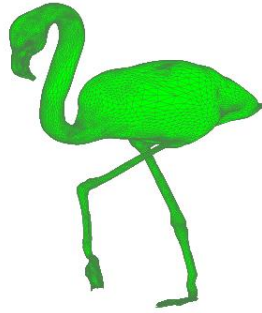
Figure 5.54: **Optimization Results for Flamingo Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 03 [left column] and Pose 04 [right column].



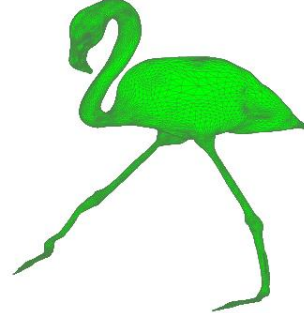
(a) Pose 05 - Expected Vertices



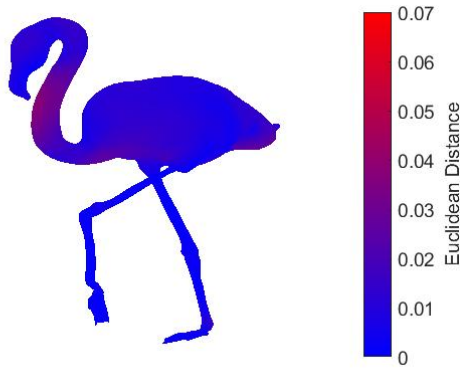
(b) Pose 06 - Expected Vertices



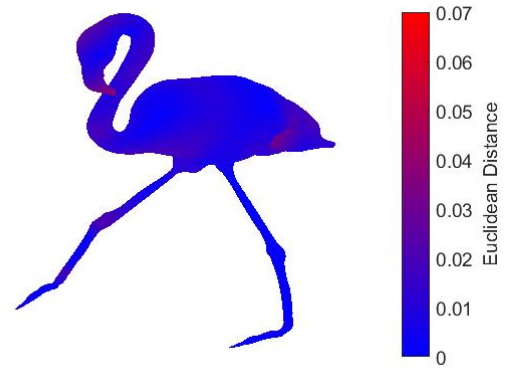
(c) Pose 05 - Computed Vertices



(d) Pose 06 - Computed Vertices

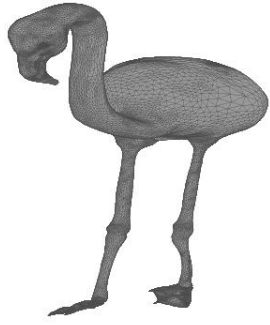


(e) Pose 05 - Error Heat Map
Pose Error: 7.6959×10^{-5}
Std Dev: 0.00586

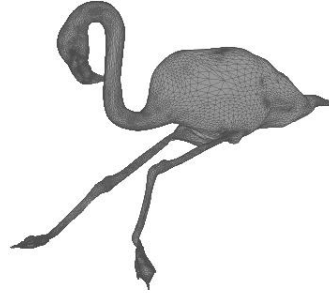


(f) Pose 06 - Error Heat Map
Pose Error: 1.3685×10^{-4}
Std Dev: 0.00739

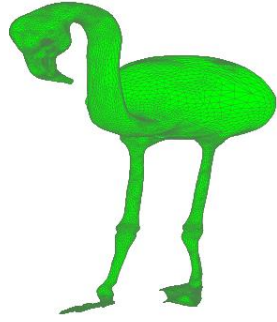
Figure 5.55: **Optimization Results for Flamingo Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 05 [left column] and Pose 06 [right column].



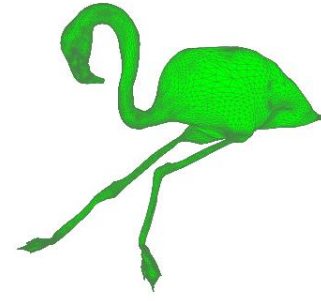
(a) Pose 07 - Expected Vertices



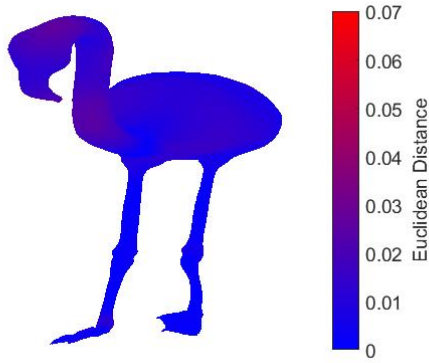
(b) Pose 08 - Expected Vertices



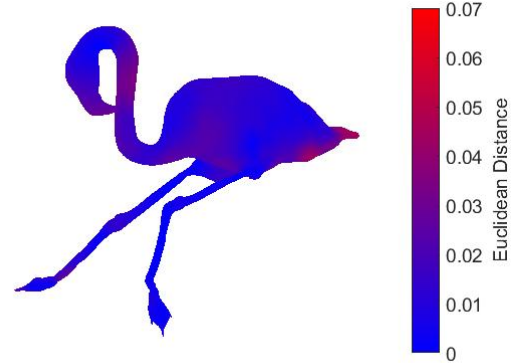
(c) Pose 07 - Computed Vertices



(d) Pose 08 - Computed Vertices

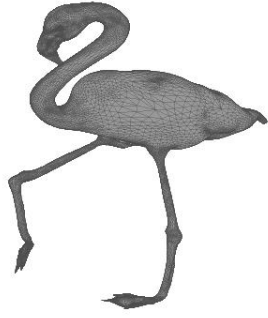


(e) Pose 07 - Error Heat Map
Pose Error: 1.1169×10^{-4}
Std Dev: 0.00647

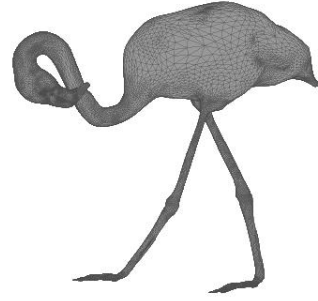


(f) Pose 08 - Error Heat Map
Pose Error: 2.3156×10^{-4}
Std Dev: 0.00891

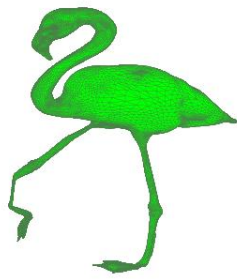
Figure 5.56: **Optimization Results for Flamingo Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 07 [left column] and Pose 08 [right column].



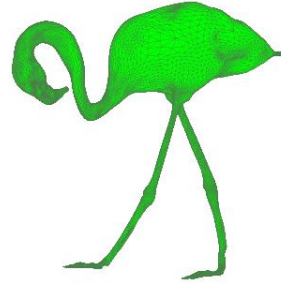
(a) Pose 09 - Expected Vertices



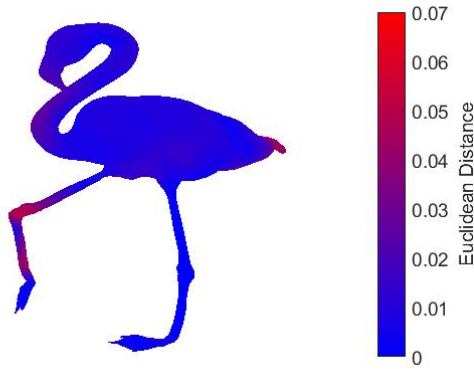
(b) Pose 10 - Expected Vertices



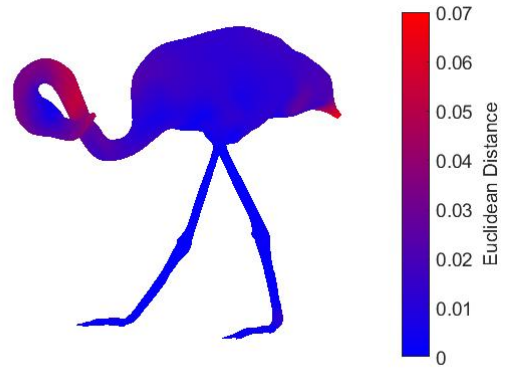
(c) Pose 09 - Computed Vertices



(d) Pose 10 - Computed Vertices



(e) Pose 09 - Error Heat Map
Pose Error: 2.0149×10^{-4}
Std Dev: 0.00999

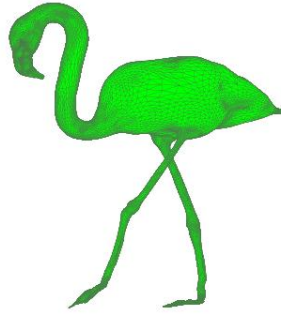


(f) Pose 10 - Error Heat Map
Pose Error: 3.2464×10^{-4}
Std Dev: 0.0126

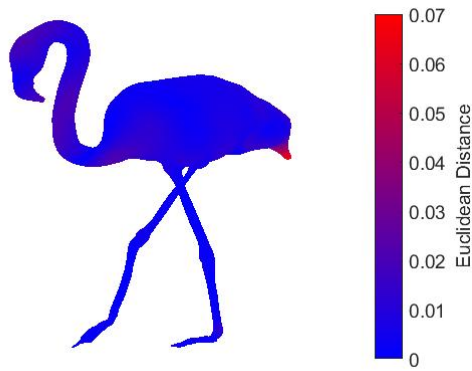
Figure 5.57: **Optimization Results for Flamingo Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 09 [left column] and Pose 10 [right column].



(a) Pose 11 - Expected Vertices



(b) Pose 11 - Computed Vertices



(c) Pose 11 - Error Heat Map

Pose Error: 1.0066×10^{-4}

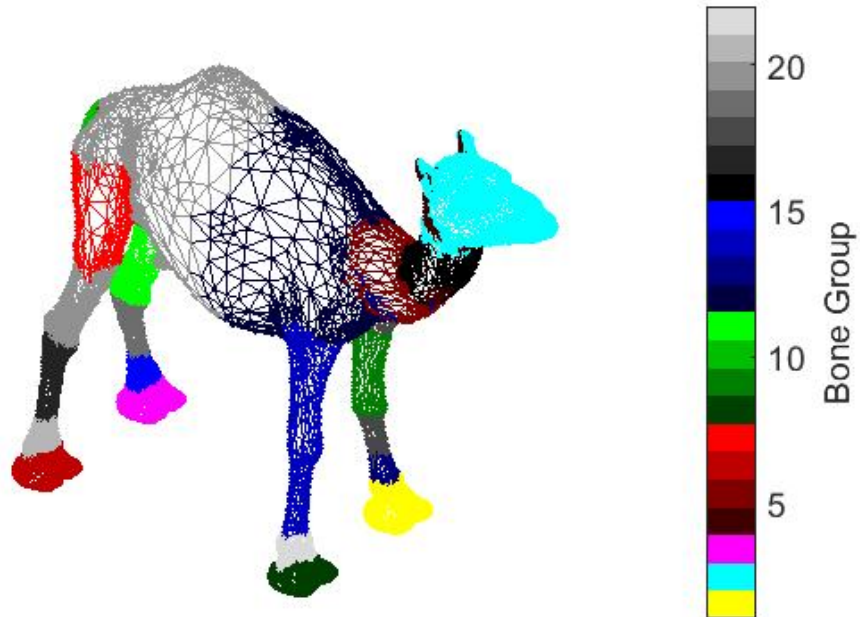
Std Dev: 0.00731

Figure 5.58: **Optimization Results for Flamingo Model** - Expected vertex positions [top:(a)], Computed vertex positions [middle:(b)] and Error Heat Map [bottom:(c)] for Pose 11.

5.3.8 Model: Camel



(a)



(b)

Figure 5.59: **Initialization Results for Camel Model** - 5.59a Model. 5.59b Initialization Results: Singleton bone groups.

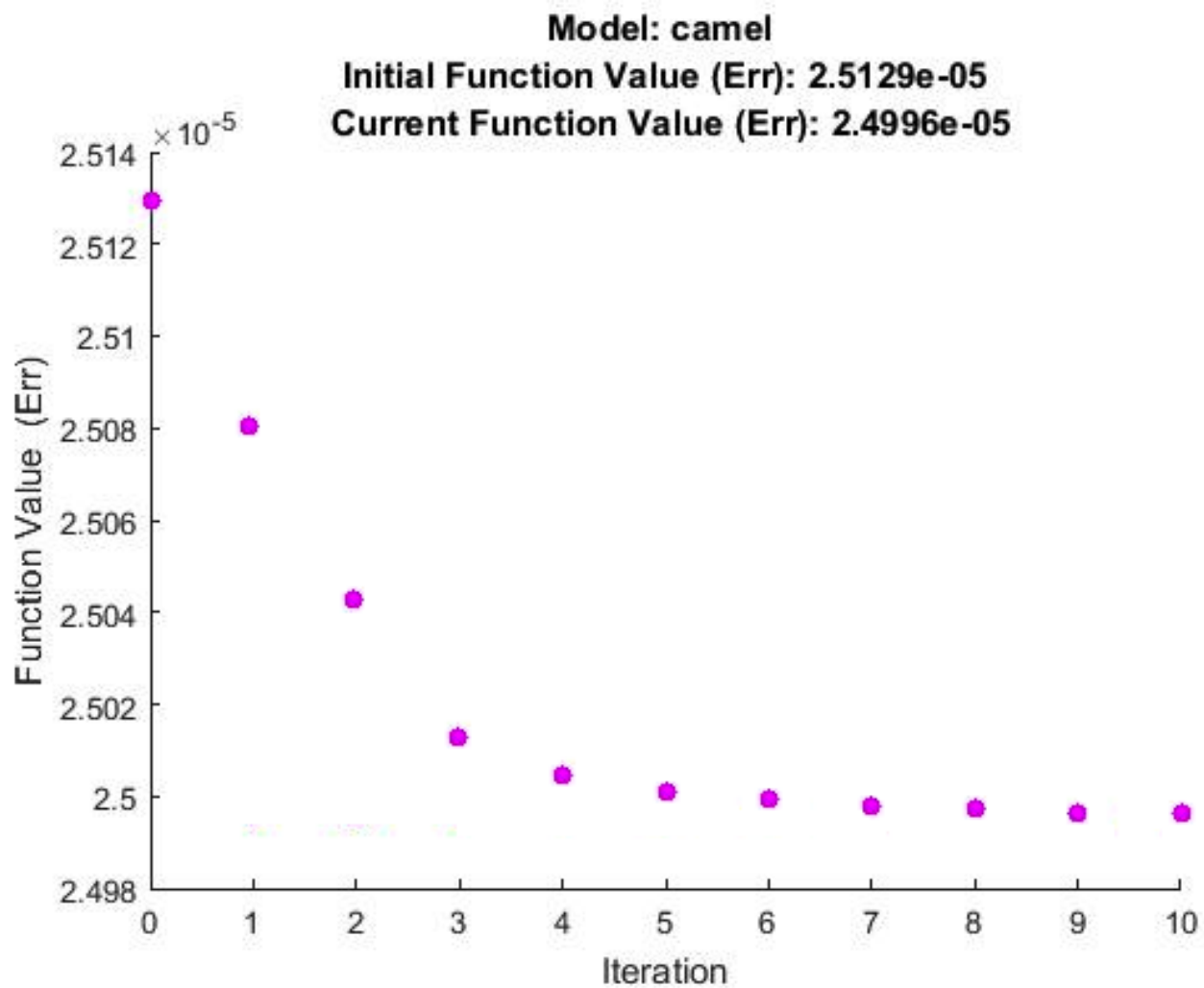
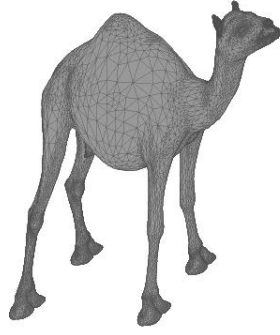
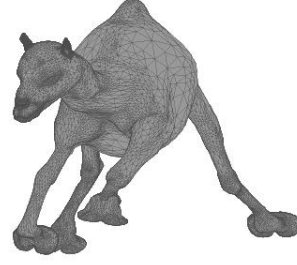


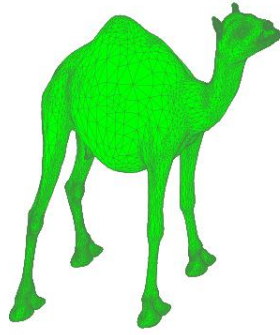
Figure 5.60: **Optimization Results for Camel Model** - Progress of EP-LBS minimization of the objective function.



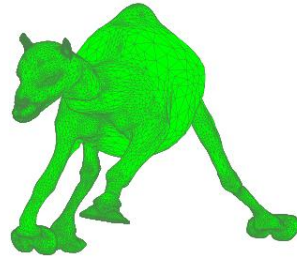
(a) Pose 01 - Expected Vertices



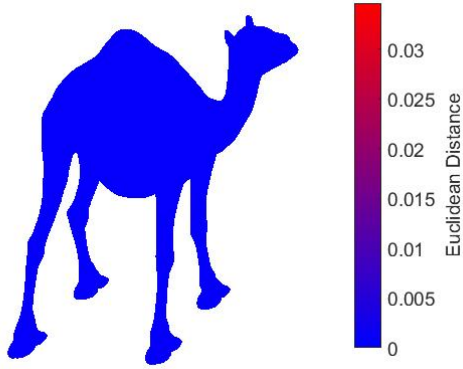
(b) Pose 02 - Expected Vertices



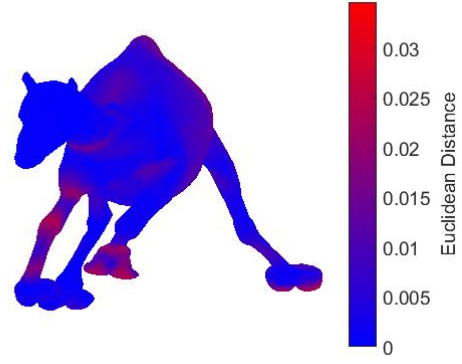
(c) Pose 01 - Computed Vertices



(d) Pose 02 - Computed Vertices

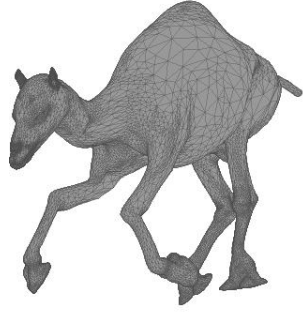


(e) Pose 01 - Error Heat Map
Pose Error: 2.7474×10^{-7}
Std Dev: 1.95×10^{-6}

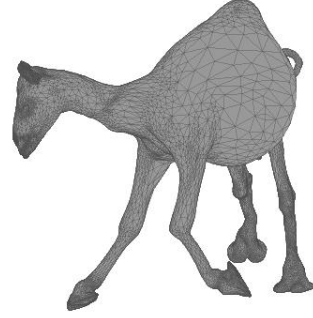


(f) Pose 02 - Error Heat Map
Pose Error: 4.4779×10^{-5}
Std Dev: 0.00502

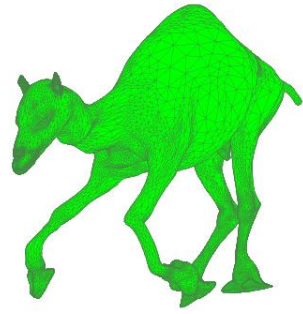
Figure 5.61: **Optimization Results for Camel Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 01 [left column] and Pose 02 [right column].



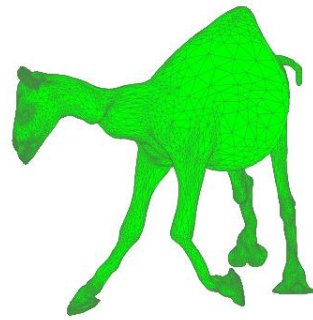
(a) Pose 03 - Expected Vertices



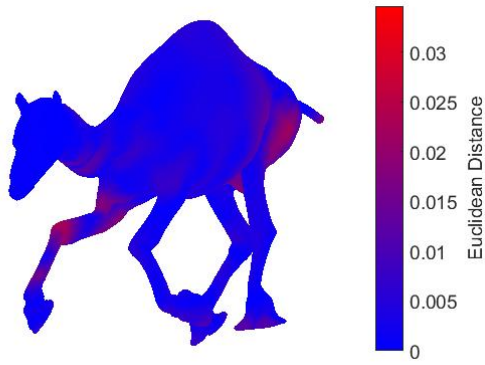
(b) Pose 04 - Expected Vertices



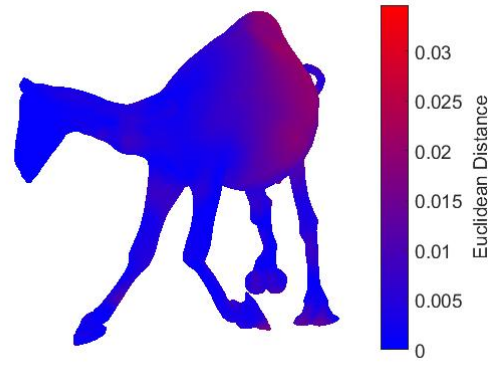
(c) Pose 03 - Computed Vertices



(d) Pose 04 - Computed Vertices

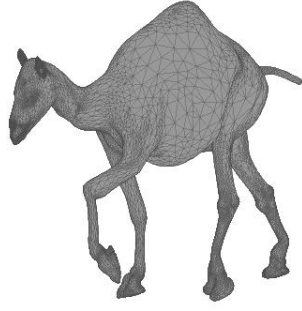


(e) Pose 03 - Error Heat Map
Pose Error: 2.5622×10^{-5}
Std Dev: 0.00381

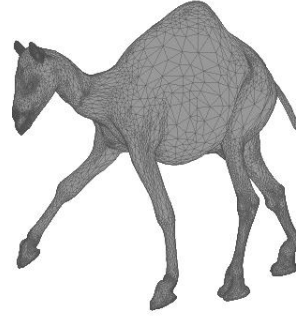


(f) Pose 04 - Error Heat Map
Pose Error: 3.4501×10^{-5}
Std Dev: 0.00448

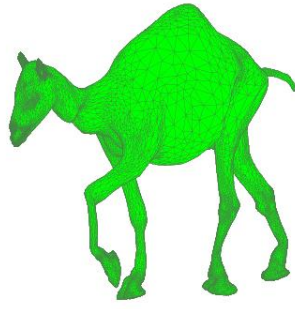
Figure 5.62: **Optimization Results for Camel Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 03 [left column] and Pose 04 [right column].



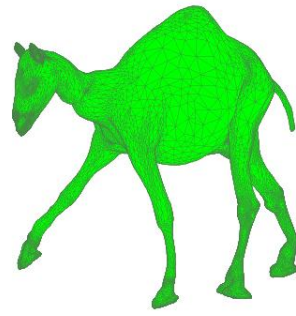
(a) Pose 05 - Expected Vertices



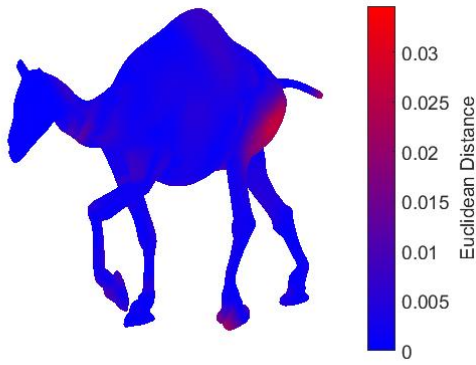
(b) Pose 06 - Expected Vertices



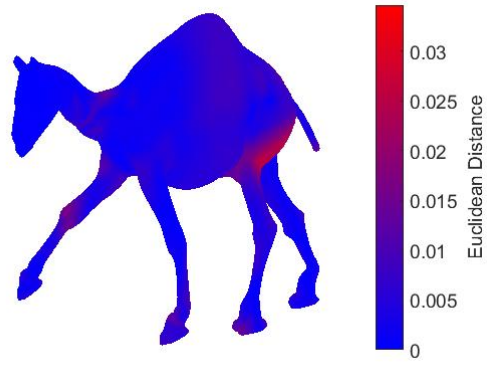
(c) Pose 05 - Computed Vertices



(d) Pose 06 - Computed Vertices

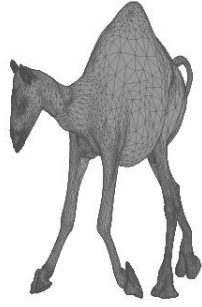


(e) Pose 05 - Error Heat Map
Pose Error: 2.2168×10^{-5}
Std Dev: 0.00355

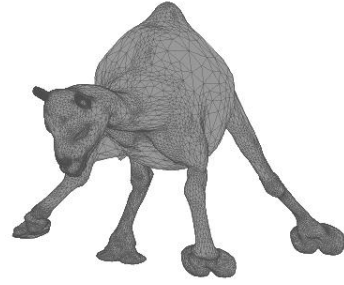


(f) Pose 06 - Error Heat Map
Pose Error: 2.6566×10^{-5}
Std Dev: 0.00384

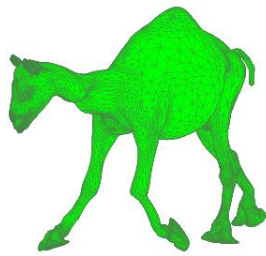
Figure 5.63: **Optimization Results for Camel Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 05 [left column] and Pose 06 [right column].



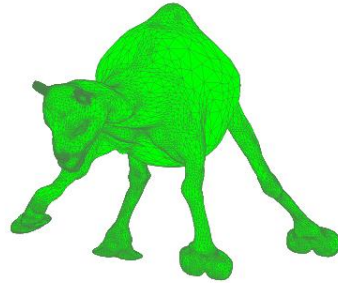
(a) Pose 07 - Expected Vertices



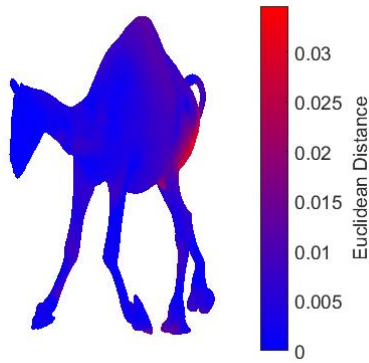
(b) Pose 08 - Expected Vertices



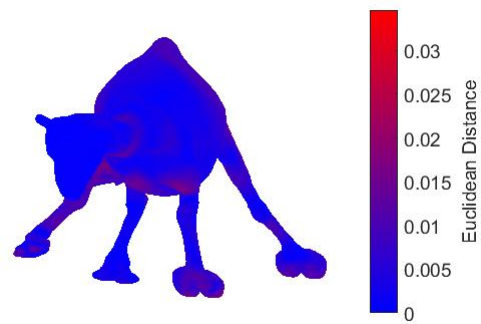
(c) Pose 07 - Computed Vertices



(d) Pose 08 - Computed Vertices

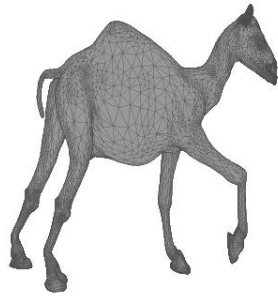


(e) Pose 07 - Error Heat Map
Pose Error: 3.6836×10^{-5}
Std Dev: 0.00465

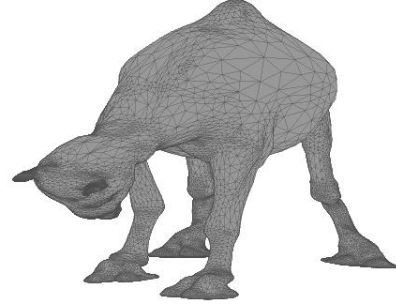


(f) Pose 08 - Error Heat Map
Pose Error: 3.05×10^{-5}
Std Dev: 0.00388

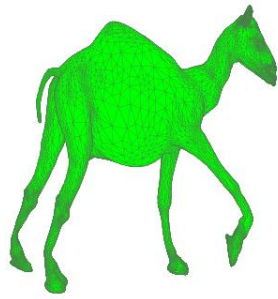
Figure 5.64: **Optimization Results for Camel Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 07 [left column] and Pose 08 [right column].



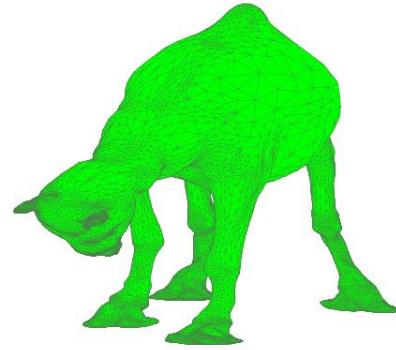
(a) Pose 09 - Expected Vertices



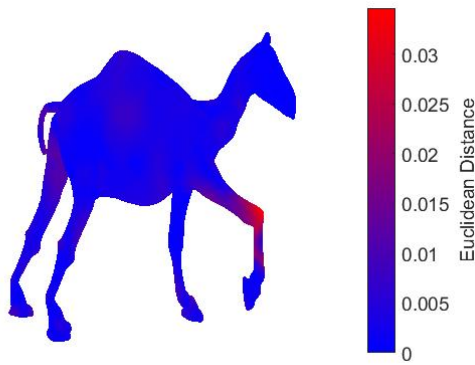
(b) Pose 10 - Expected Vertices



(c) Pose 09 - Computed Vertices



(d) Pose 10 - Computed Vertices

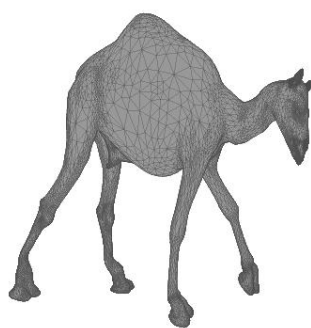


(e) Pose 09 - Error Heat Map
Pose Error: 1.7508×10^{-5}
Std Dev: 0.00314

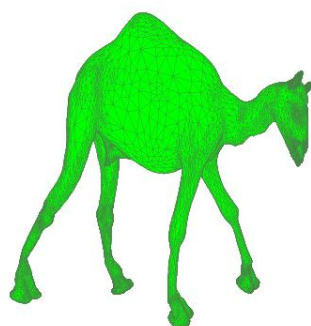


(f) Pose 10 - Error Heat Map
Pose Error: 2.0463×10^{-5}
Std Dev: 0.0033

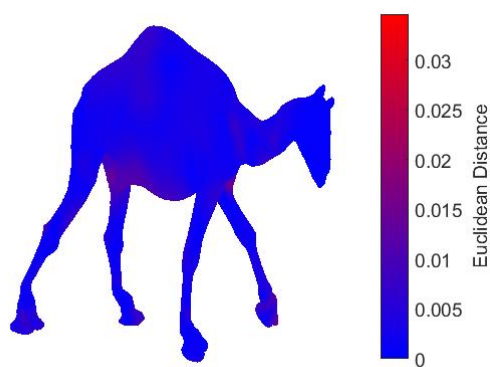
Figure 5.65: **Optimization Results for Camel Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 09 [left column] and Pose 10 [right column].



(a) Pose 11 - Expected Vertices



(b) Pose 11 - Computed Vertices



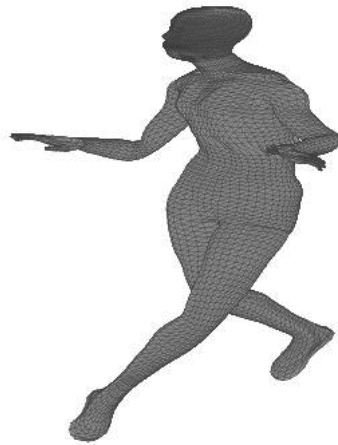
(c) Pose 11 - Error Heat Map

Pose Error: 1.5744×10^{-5}

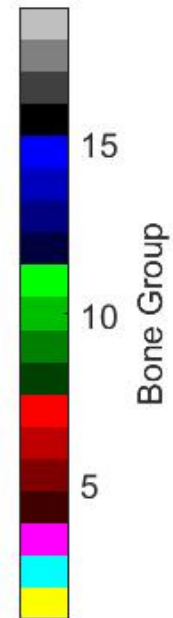
Std Dev: 0.00285

Figure 5.66: **Optimization Results for Camel Model** - Expected vertex positions [top:(a)], Computed vertex positions [middle:(b)] and Error Heat Map [bottom:(c)] for Pose 11.

5.3.9 Model: Dance



(a)



(b)

Figure 5.67: **Initialization Results for Dance Model** - 5.67a Model. 5.67b Initialization Results: Singleton bone groups.

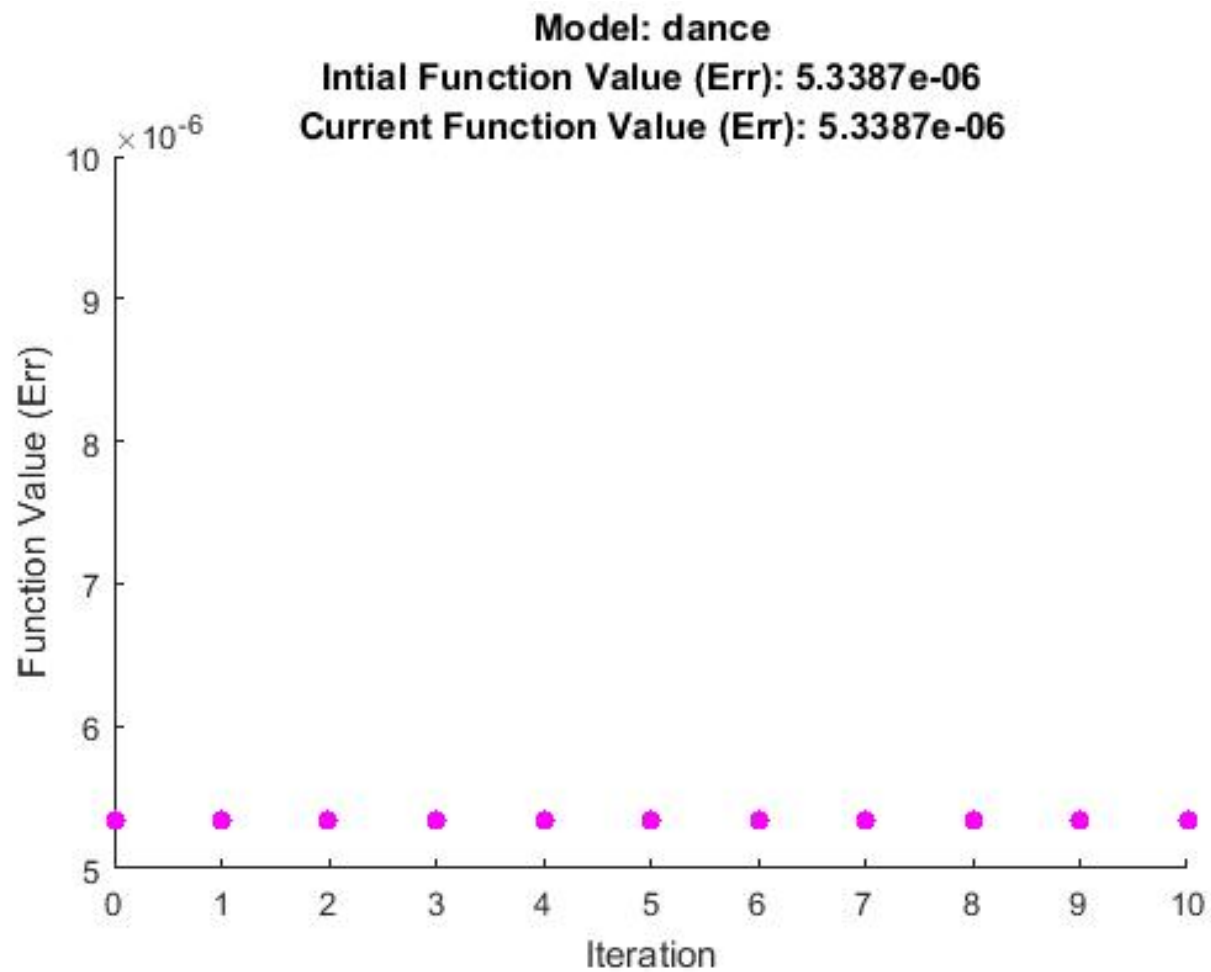
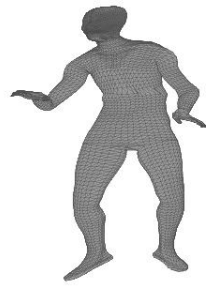
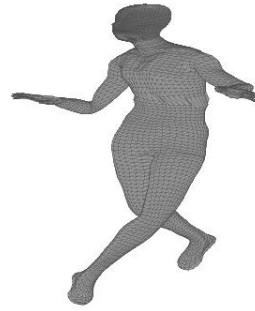


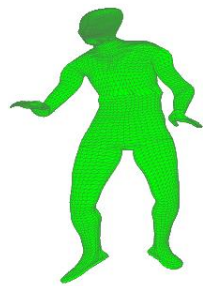
Figure 5.68: **Optimization Results for Dance Model** - Progress of EP-LBS minimization of the objective function.



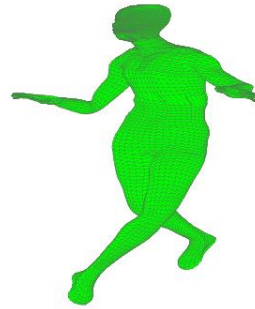
(a) Pose 01 - Expected Vertices



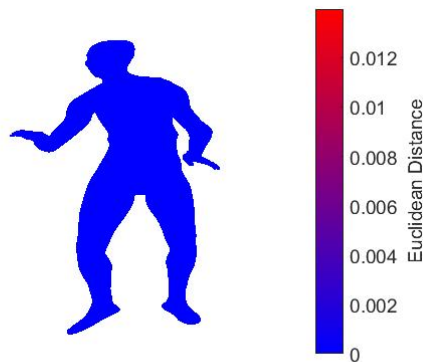
(b) Pose 02 - Expected Vertices



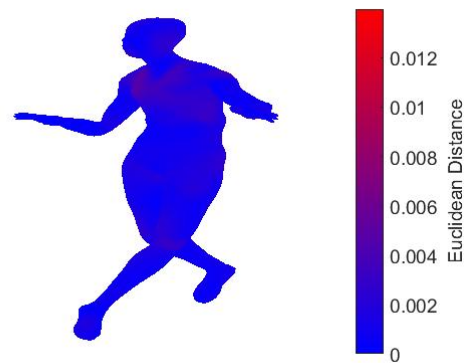
(c) Pose 01 - Computed Vertices



(d) Pose 02 - Computed Vertices

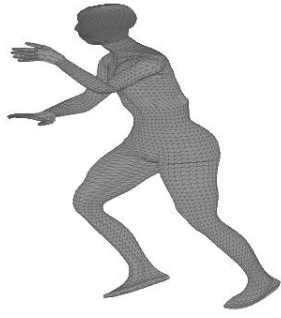


(e) Pose 01 - Error Heat Map
Pose Error: 1.5816×10^{-11}
Std Dev: 1.58×10^{-6}



(f) Pose 02 - Error Heat Map
Pose Error: 1.9627×10^{-6}
Std Dev: 0.000726

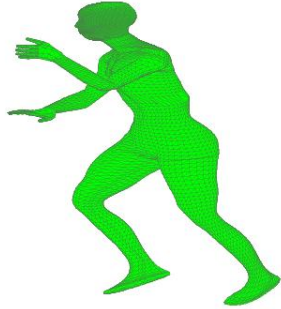
Figure 5.69: **Optimization Results for Dance Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 01 [left column] and Pose 02 [right column].



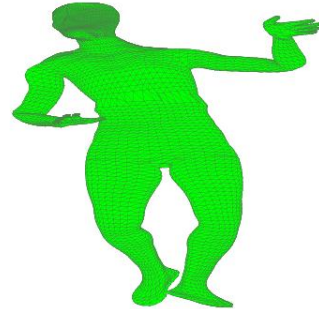
(a) Pose 03 - Expected Vertices



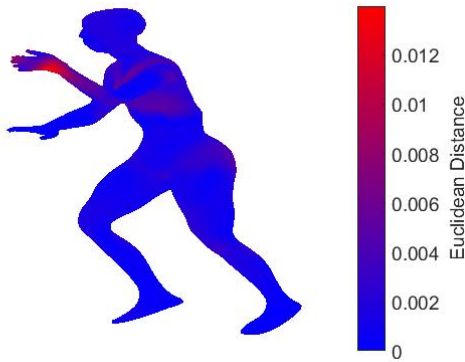
(b) Pose 04 - Expected Vertices



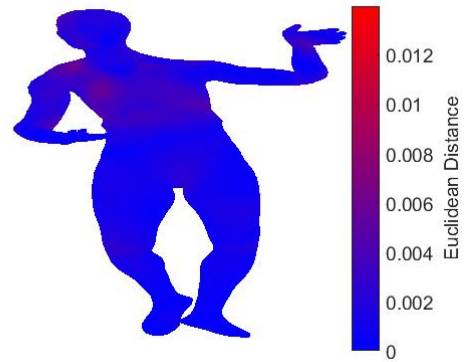
(c) Pose 03 - Computed Vertices



(d) Pose 04 - Computed Vertices

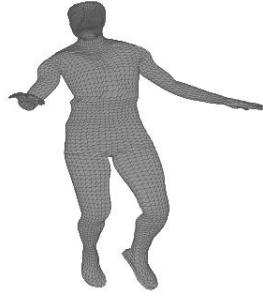


(e) Pose 03 - Error Heat Map
Pose Error: 5.3557×10^{-6}
Std Dev: 0.00148

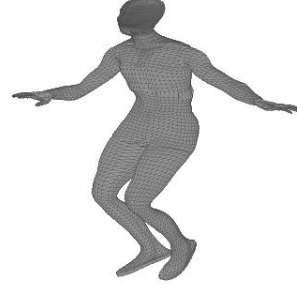


(f) Pose 04 - Error Heat Map
Pose Error: 3.8826×10^{-6}
Std Dev: 0.00117

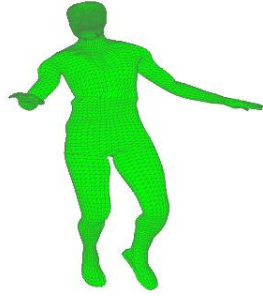
Figure 5.70: **Optimization Results for Dance Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 03 [left column] and Pose 04 [right column].



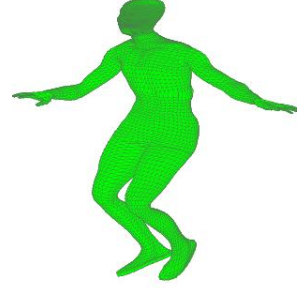
(a) Pose 05 - Expected Vertices



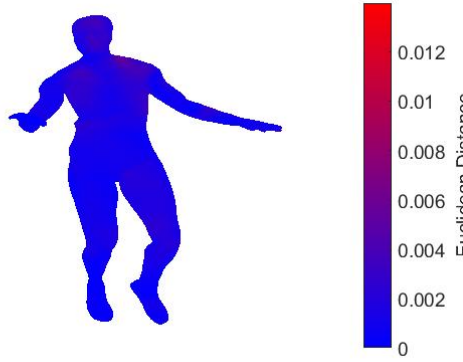
(b) Pose 06 - Expected Vertices



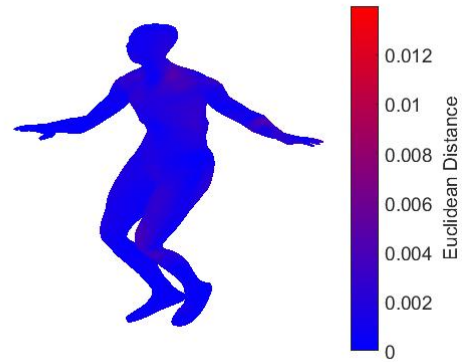
(c) Pose 05 - Computed Vertices



(d) Pose 06 - Computed Vertices

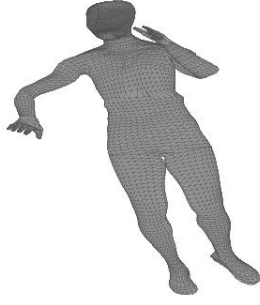


(e) Pose 05 - Error Heat Map
Pose Error: 2.9509×10^{-6}
Std Dev: 0.000943

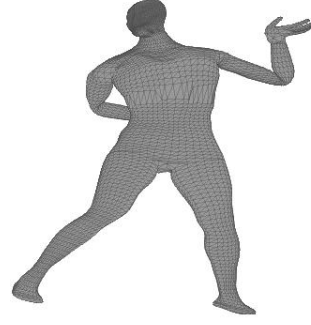


(f) Pose 06 - Error Heat Map
Pose Error: 2.0488×10^{-6}
Std Dev: 0.000805

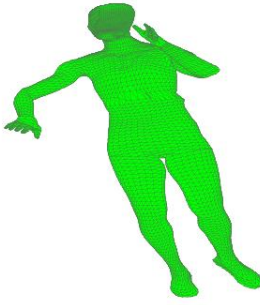
Figure 5.71: **Optimization Results for Dance Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 05 [left column] and Pose 06 [right column].



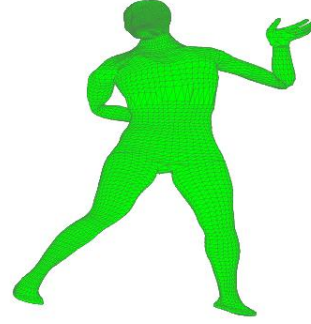
(a) Pose 07 - Expected Vertices



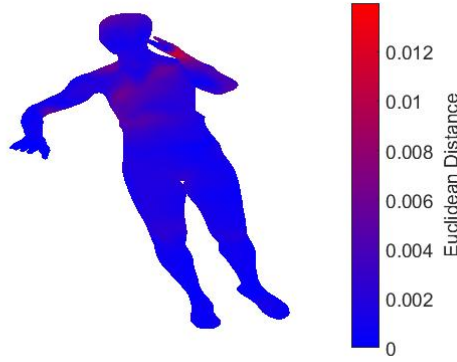
(b) Pose 08 - Expected Vertices



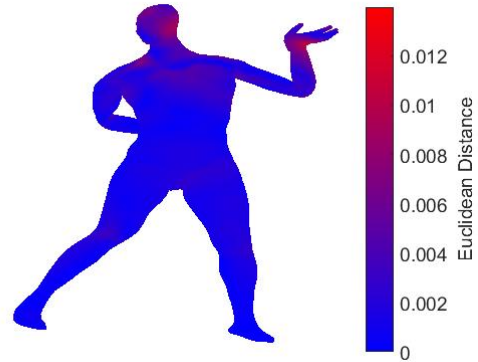
(c) Pose 07 - Computed Vertices



(d) Pose 08 - Computed Vertices

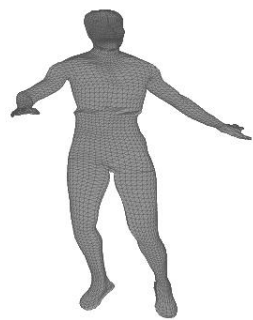


(e) Pose 07 - Error Heat Map
Pose Error: 6.097×10^{-6}
Std Dev: 0.00155

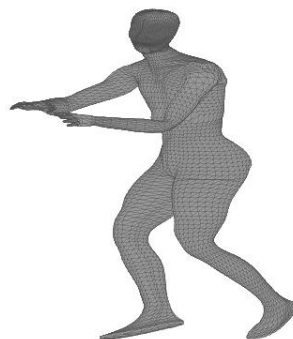


(f) Pose 08 - Error Heat Map
Pose Error: 1.4605×10^{-5}
Std Dev: 0.00246

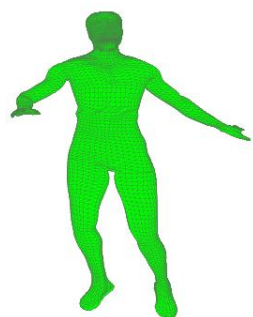
Figure 5.72: **Optimization Results for Dance Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 07 [left column] and Pose 08 [right column].



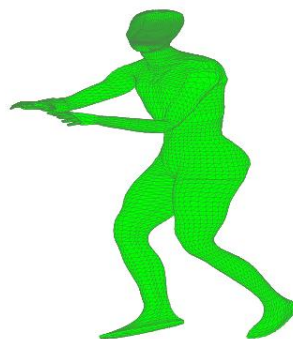
(a) Pose 09 - Expected Vertices



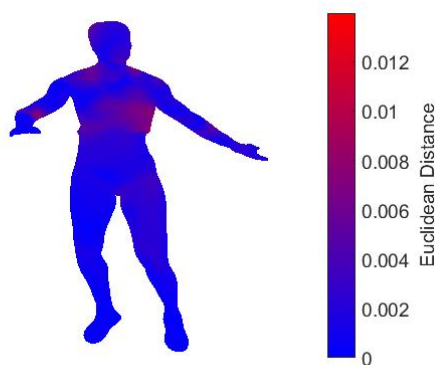
(b) Pose 10 - Expected Vertices



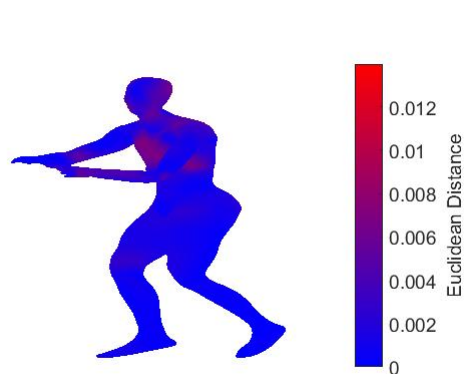
(c) Pose 09 - Computed Vertices



(d) Pose 10 - Computed Vertices

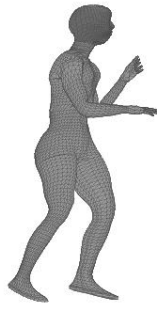


(e) Pose 09 - Error Heat Map
Pose Error: 6.5719×10^{-6}
Std Dev: 0.00157

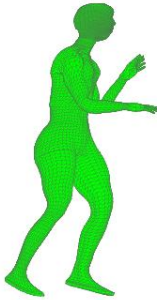


(f) Pose 10 - Error Heat Map
Pose Error: 8.5102×10^{-6}
Std Dev: 0.00183

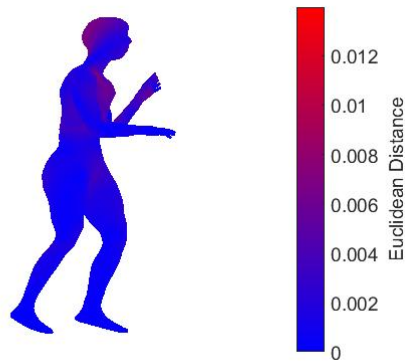
Figure 5.73: **Optimization Results for Dance Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 09 [left column] and Pose 10 [right column].



(a) Pose 11 - Expected Vertices



(b) Pose 11 - Computed Vertices



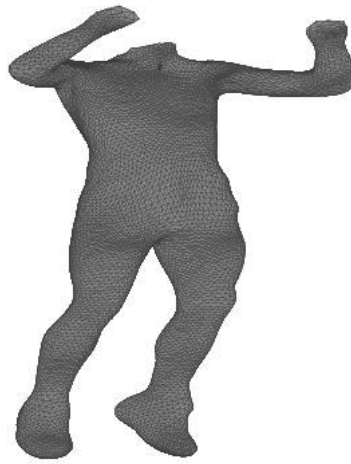
(c) Pose 11 - Error Heat Map

Pose Error: 6.7409×10^{-6}

Std Dev: 0.00155

Figure 5.74: **Optimization Results for Dance Model** - Expected vertex positions [top:(a)], Computed vertex positions [middle:(b)] and Error Heat Map [bottom:(c)] for Pose 11.

5.3.10 Model: Jump



(a)



(b)

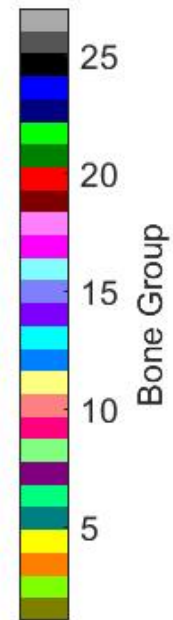


Figure 5.75: **Initialization Results for Jump Model** - 5.75a Model. 5.75b Initialization Results: Singleton bone groups.

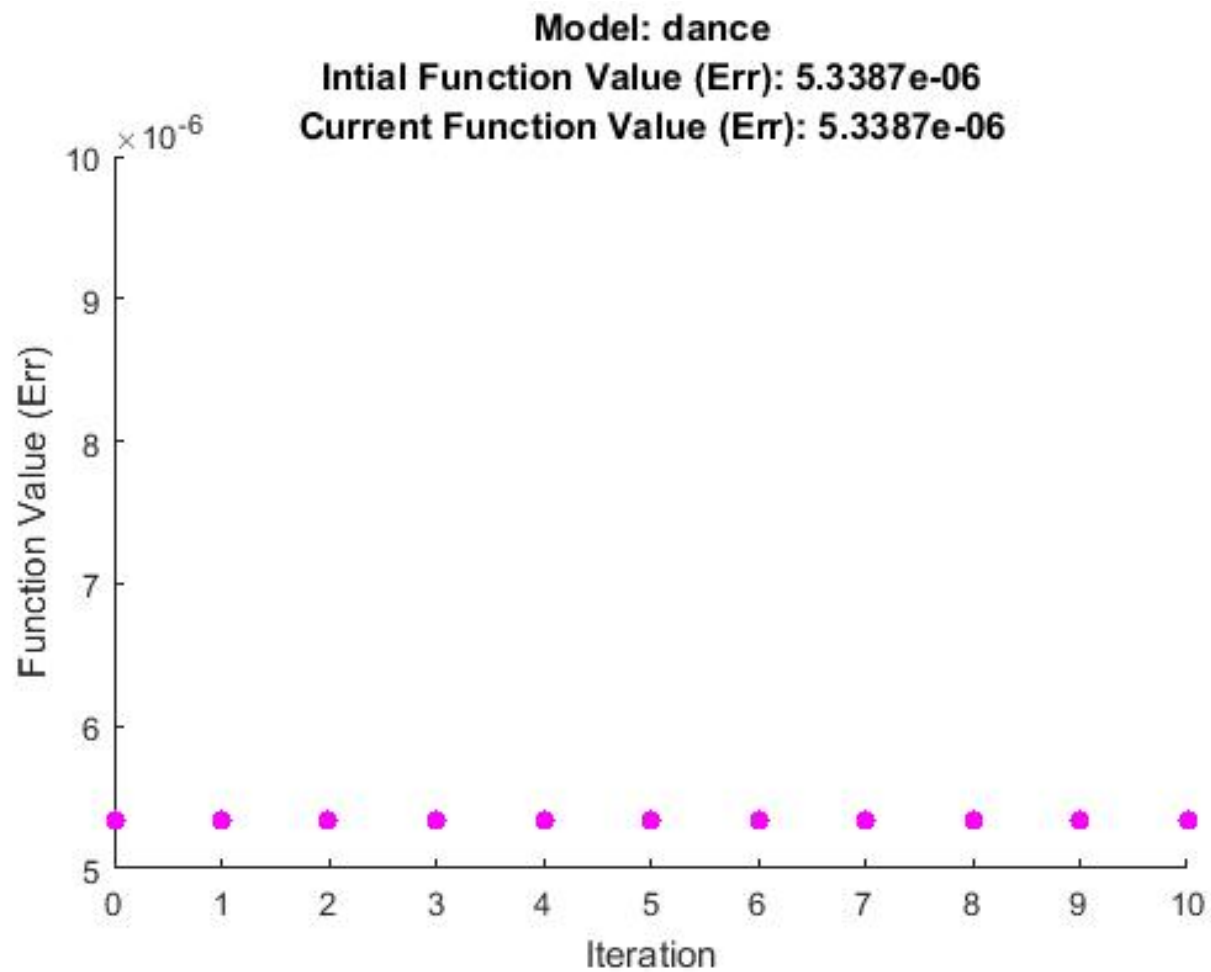
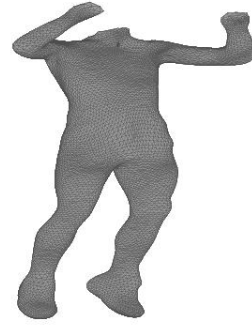


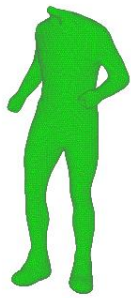
Figure 5.76: **Optimization Results for Jump Model** - Progress of EP-LBS minimization of the objective function.



(a) Pose 01 - Expected Vertices



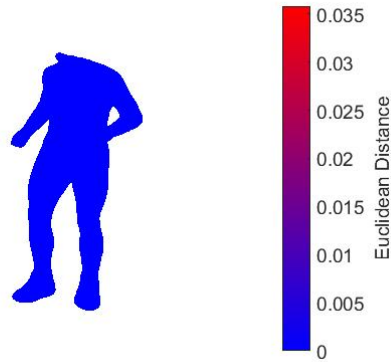
(b) Pose 02 - Expected Vertices



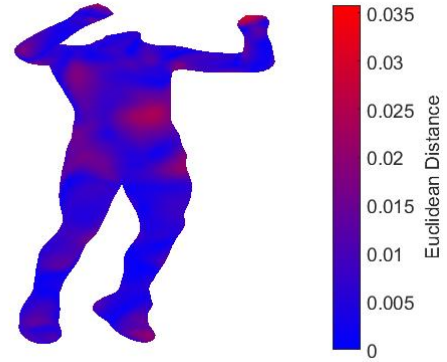
(c) Pose 01 - Computed Vertices



(d) Pose 02 - Computed Vertices



(e) Pose 01 - Error Heat Map
Pose Error: 3.4418×10^{-11}
Std Dev: 2.22×10^{-6}



(f) Pose 02 - Error Heat Map
Pose Error: 9.0657×10^{-5}
Std Dev: 0.00469

Figure 5.77: **Optimization Results for Jump Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 01 [left column] and Pose 02 [right column].



(a) Pose 03 - Expected Vertices



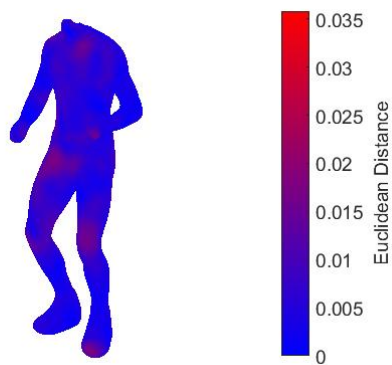
(b) Pose 04 - Expected Vertices



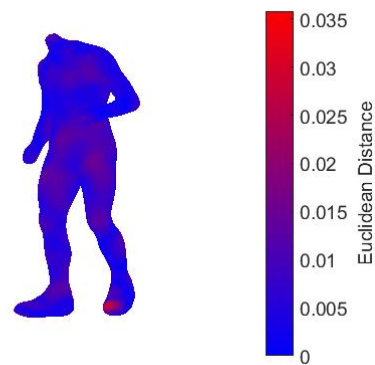
(c) Pose 03 - Computed Vertices



(d) Pose 04 - Computed Vertices

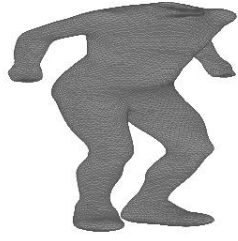


(e) Pose 03 - Error Heat Map
Pose Error: 5.1322×10^{-5}
Std Dev: 0.00342

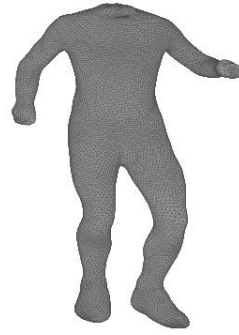


(f) Pose 04 - Error Heat Map
Pose Error: 6.6075×10^{-5}
Std Dev: 0.00398

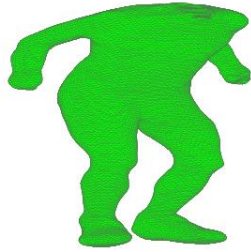
Figure 5.78: **Optimization Results for Jump Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 03 [left column] and Pose 04 [right column].



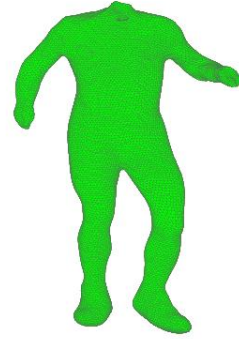
(a) Pose 05 - Expected Vertices



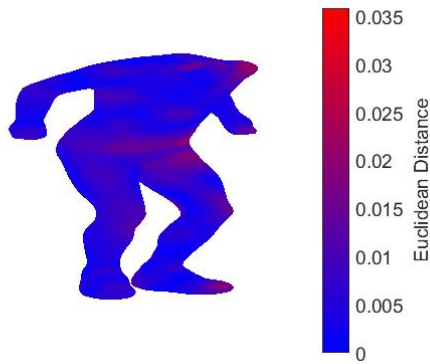
(b) Pose 06 - Expected Vertices



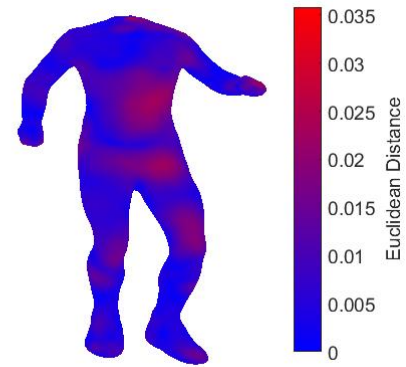
(c) Pose 05 - Computed Vertices



(d) Pose 06 - Computed Vertices

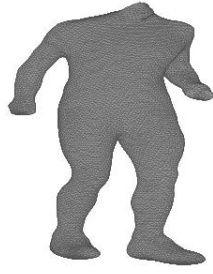


(e) Pose 05 - Error Heat Map
Pose Error: 6.3239×10^{-5}
Std Dev: 0.00403

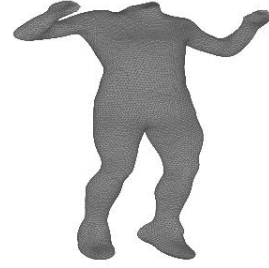


(f) Pose 06 - Error Heat Map
Pose Error: 8.0184×10^{-5}
Std Dev: 0.00444

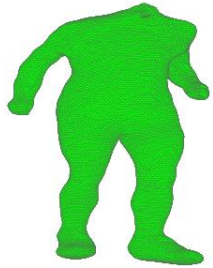
Figure 5.79: **Optimization Results for Jump Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 05 [left column] and Pose 06 [right column].



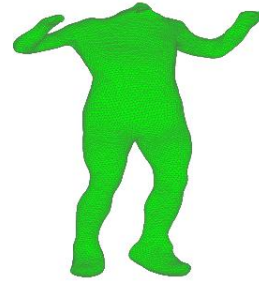
(a) Pose 07 - Expected Vertices



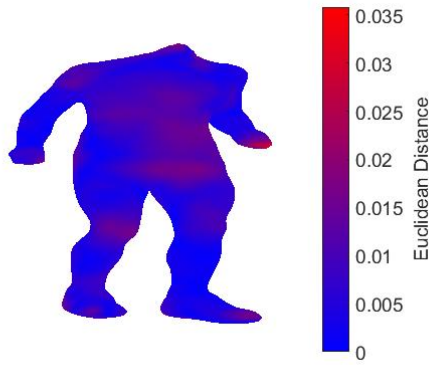
(b) Pose 08 - Expected Vertices



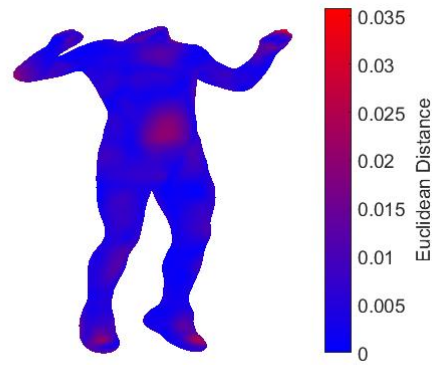
(c) Pose 07 - Computed Vertices



(d) Pose 08 - Computed Vertices



(e) Pose 07 - Error Heat Map
Pose Error: 4.9889×10^{-5}
Std Dev: 0.00358



(f) Pose 08 - Error Heat Map
Pose Error: 5.7242×10^{-5}
Std Dev: 0.00399

Figure 5.80: **Optimization Results for Jump Model** - Expected vertex positions [top:(a)(b)], Computed vertex positions [middle:(c)(d)] and Error Heat Map [bottom:(e)(f)] for Pose 07 [left column] and Pose 08 [right column].

5.4 Discussion

Successful skinning decomposition solutions address the three major components of the decomposition pipeline: initialization, optimization and constraint adherence. The results exhibited in Figures 5.7 through 5.80 demonstrate the success of EP-LBS across all components of the skinning decomposition workflow, indicating a viable solution for computing animation parameters needed to recreate the given example poses. Initialization and optimization are easily visualized with bone group plots and compared using adjacent images of both expected and computed poses and error heat maps for each model. Additionally, analysis of the expected and computed models and convergence of the objective function confirm adherence to LBS constraints.

Initialization Results					
Model	Expected # Bones	Computed # Bones	# Initialization Iterations	Initialization Time (min)	Initialization Model Error (E)
Cat [98]	22	31	7	4	6.3310×10^{-5}
Woman	20	18	5	2.8	1.8547×10^{-6}
Snake [49]	26	12	3	1	1.1119×10^{-6}
Lion [98]	22	26	5	1.7	8.0709×10^{-5}
Horse [98]	29	29	4	5.8	1.3050×10^{-5}
Flamingo [98]	10	10	4	8.1	1.5918×10^{-4}
Camel [98]	22	22	6	11.2	2.5129×10^{-5}
Elephant [98]	24	27	8	31.8	7.1753×10^{-6}
Dance [33]	21	19	3	2.7	5.3387×10^{-6}
Jump [33]	21	27	2	6.4	5.7326×10^{-5}

Table 5.2: Initialization Results

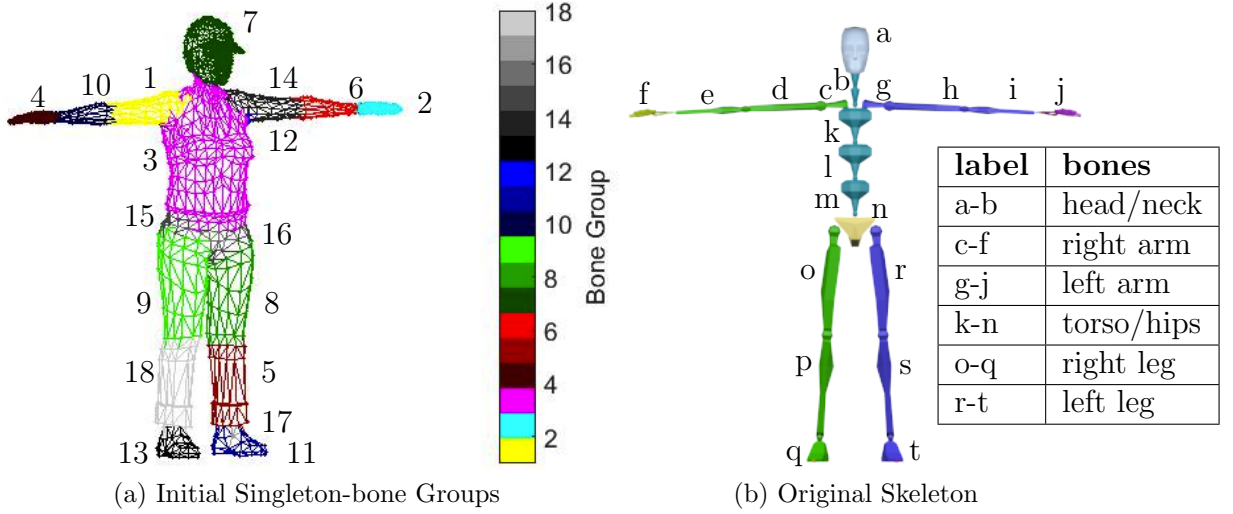


Figure 5.81: The computed singleton bone groups for the woman model after initialization (5.81a) and the original bones and skeletal structure (5.81b) used to generate the poses. The initial skeleton contained 20 bones, including three bones for the torso. The initialization process computed 18 bones. Because the example poses did not include articulation of each of the torso bones, the initialization step grouped the three bones together as one bone for the entire torso.

5.4.1 Initialization

The quaternion-based progressive clustering of vertex motion introduced by EP-LBS, paired with loosely constrained NNLS weight computation provides a strong initialization approach for example-based skinning decomposition solutions. For each of the models, the EP-LBS initialization process was able to determine the number of bones for the model, such that the initial pose deformations and initial model error provided optimal starting points for the EP-LBS optimization process. Using this initialization method the optimization algorithm was able to converge on a solution for each model, without a priori knowledge of the number of bones or skeletal structure for the model. For three models, the horse, flamingo and camel, the EP-LBS initialization process was able to compute the exact number of bones that were expected for the model (see Table 5.2). The remaining models were a mixture of over-counting and under-counting the number of bones for a model.

Results show that the initial clustering and determination of singleton bone groups is highly dependent on the quality and variety of the input example models. For models with fewer computed bones than expected, the missing or under-counted bones reflect areas of the model where the example poses did not demonstrate variety (if any) in the deformation of the bones in that area of the model. This is confirmed by the woman model, for which both the example poses and the skeletal structure used to generate the example poses were available. The original skeletal structure included 20 bones, including separate bones for the head, neck and clavicles as well as multiple spine links. The computed singleton-bone groups clustered these seven bones together as a single torso region because across the given example poses, these seven bones all moved in a similar manner (see Figure 5.81).

So, what happened to the five other bones (neck, clavicle and additional spine links) that should have been found in the torso region? Some were moved to other areas of the body, especially those areas that experience a wide variety of deformation across poses, and other bones were left out completely, hence fewer computed bones. In this case the left foot received an extra bone in the ankle area, the pelvis received an extra bone and there is a bone in the left armpit. That leaves two less bones than expected, which aligns with the 18 computed bones and 20 expected bones.

Although, the flamingo computed the correct number of bones for the model, the location of the bones was not correct as indicated by the error heat maps. Specifically the vertices in the head and neck of the flamingo were largely considered a single bone group, where the heat map indicates a greater difference between the expected and computed vertex positions (see Figure 5.82). Greater deformation of the head and neck in the example poses, would likely lead to the initialization process computing two or more regions which would offer better results. Additionally, more examples of deformation for the right upper leg of the

flamingo could yield a better distribution of vertices in the bone clusters for the right leg (see Figure 5.82a).

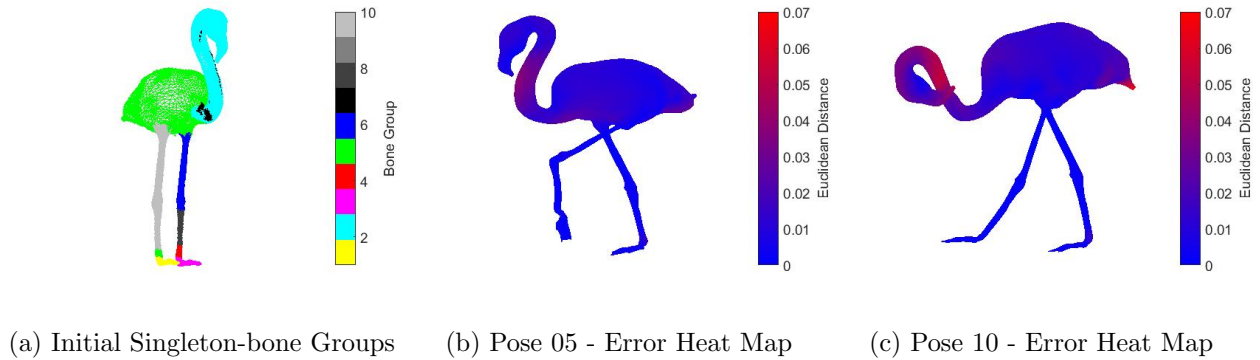
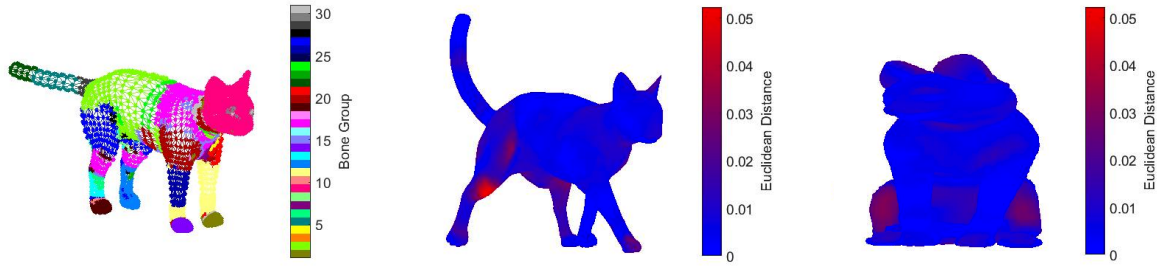


Figure 5.82: Lack of variation of head and neck poses leads to clustering the head and neck regions into a single bone group, causing large errors in that region of the model as indicated by the heat map.

The cat model demonstrates one of the outcomes of models with a lack of variety and unintentionally correlated motion in the example pose deformations. For many of the examples poses the upper hind legs moved in a similar manner relative to one another. As such the upper hind legs were clustered together despite the vertices in the cluster in fact being in two distinct disparate groups. This clustering leads to large errors in those regions (see Figure 5.83c). This can occur when there is symmetry in the skeletal structure and the example poses include many mirrored or reflective movements of symmetric bones causing implied correlation between the movement of the symmetric bones. To address this concern, connected components or some other adjacency constraint can be imposed during the initialization process to ensure that the computed bone groups.

As demonstrated by the initial singleton bone groupings and the error heat maps, the EP-LBS initialization process is able to effectively compute bone groups without a priori knowledge of the number bones or the structure of the skeleton. Where the bone groups



(a) Initial Singleton-bone Groups (b) Pose 04 - Error Heat Map (c) Pose 09 - Error Heat Map

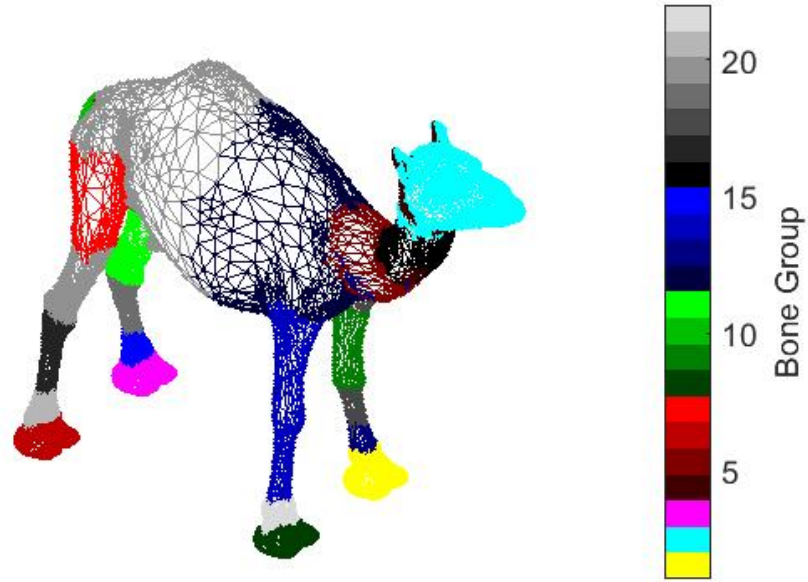
Figure 5.83: Incorrect clustering of bones that move in parallel to one another, such as the clustering of the upper hind legs into one bone group (bone 17), leads to higher model area in those areas, as indicated by the corresponding areas of the error heat map being marked hot.

compute more or less than the expected number of bones and where bones are missing in expected areas and present in unexpected areas, the EP-LBS model is still able to effectively model the character, as indicated by the near-zero model error value for all models after initialization.

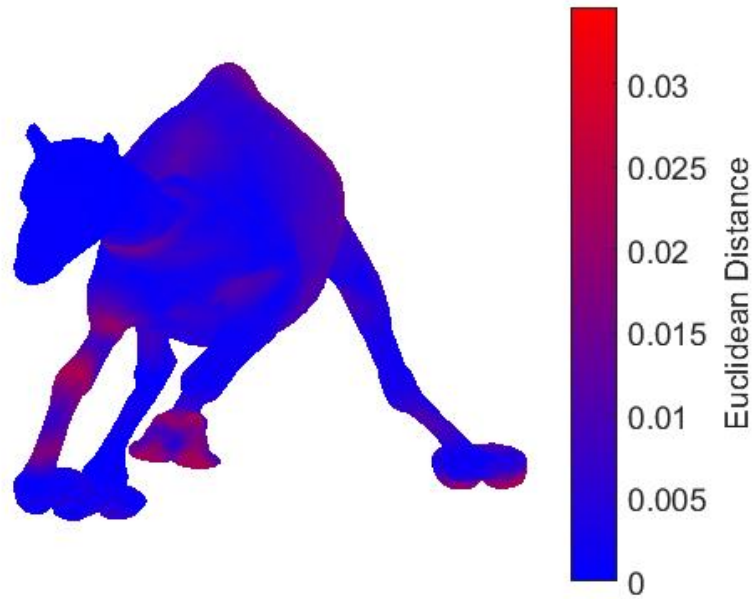
To resolve this clustering artifact, EP-LBS will perform better if input example poses include uncorrelated deformation of each bone in the model. Given a wide range of example poses spanning the range of motion for all bones, EP-LBS is able to precisely estimate the animation parameters that recreate the example pose deformations. Additionally, even when example poses are limited or do not demonstrate the distinct articulation of each bone, EP-LBS is still able to generate the skeletal structure and skinning weights that will recreate the given example data.

5.4.2 Optimization and Constraint Adherence

Results show that EP-LBS optimization is able to compute the constrained animation parameters needed to recreate the example poses and minimize the model error. The results are



(a) Camel Initial Bones

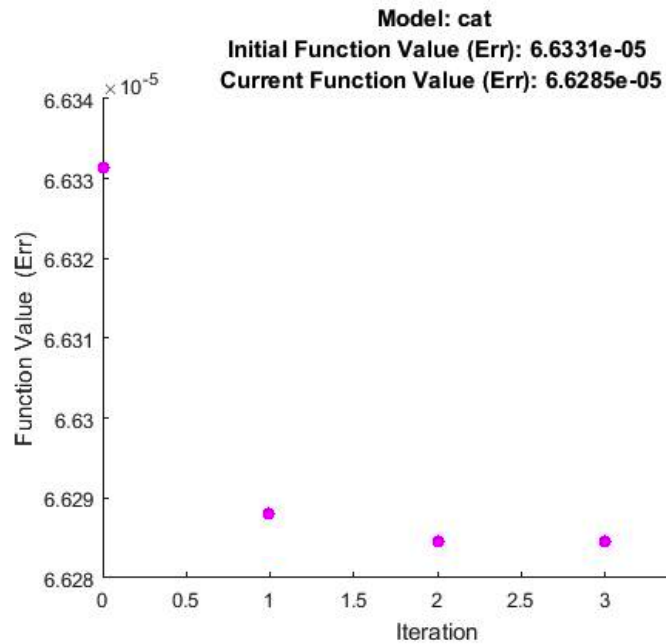


(b) Camel Pose 02 Error Heat Map

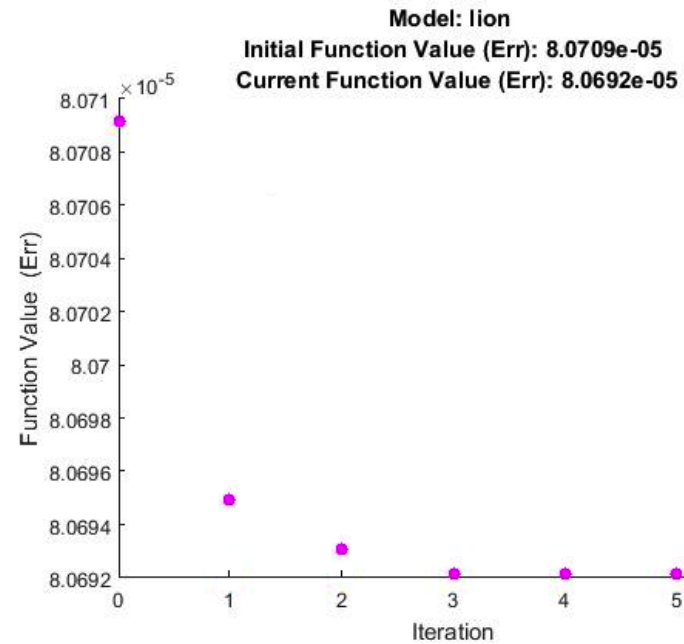
Figure 5.84: **Expected Bones for Camel Model** - The front right leg of the camel is incorrectly clustered into one large bone group. The heat map for pose 02 gives indication of where bones should be.

demonstrated by the heat maps and error values for each pose. Each model has been scaled to fit in a unit cube and the maximum error indicated by red on the heat map represents the maximum error across all poses. The average maximum error for all models is 0.0328 or roughly three hundredths of a unit for the scaled models. The large areas of cool blue for each pose across all models indicate the computed animation parameters applied to the base pose result in very little or no error for the vast majority of models and poses. Additionally, the maximum error, indicated by red areas on the heat map are near zero and do not create significant visual differences.

There is often greater error at joints or regions where the influence of two or more bones overlap. This is the result of error in the computation of weights. The EP-LBS results presented in this research are more likely to indicate larger areas as joints because there is no sparsity constraint implemented for the EP-LBS models. Similarly, for vertices incorrectly clustered as a single bone, the heat map indicates where additional joints should have been, such as with the front right leg of the camel in pose 02 (see Figure 5.4.1). At the expense of computational time, beyond including more example poses demonstrating a broad range of motion for each joint, implementing the sparsity constraint for bones could improve these results.



(a) Cat Optimization

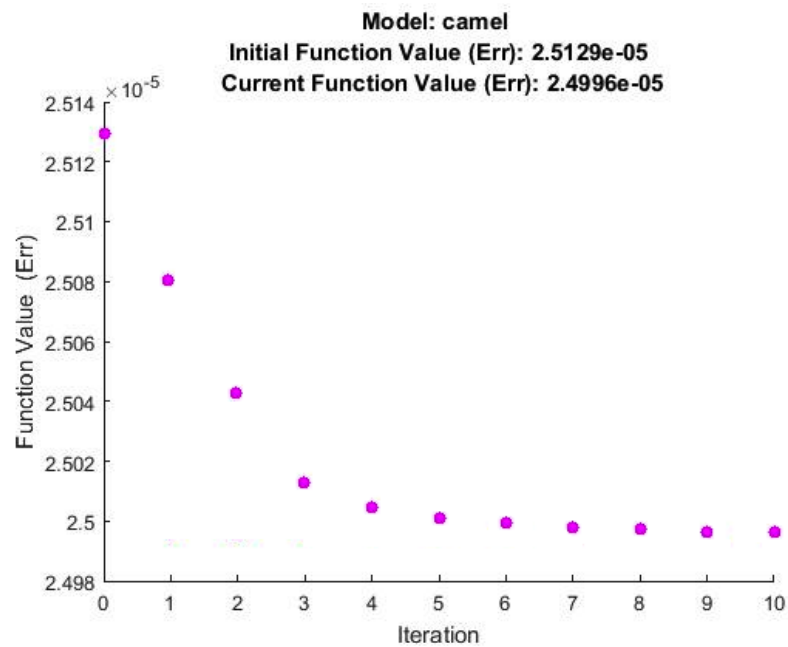


(b) Lion Optimization

Figure 5.85: **General Patterns in Optimization Results** - Most models begin with an initial objective function value near the solution. In a few iterations the EP-LBS optimization process minimizes the objective function value and converges on a near-zero solution.

Figure 5.85 (cont'd)

(a) Camel Optimization



(b) Elephant Optimization

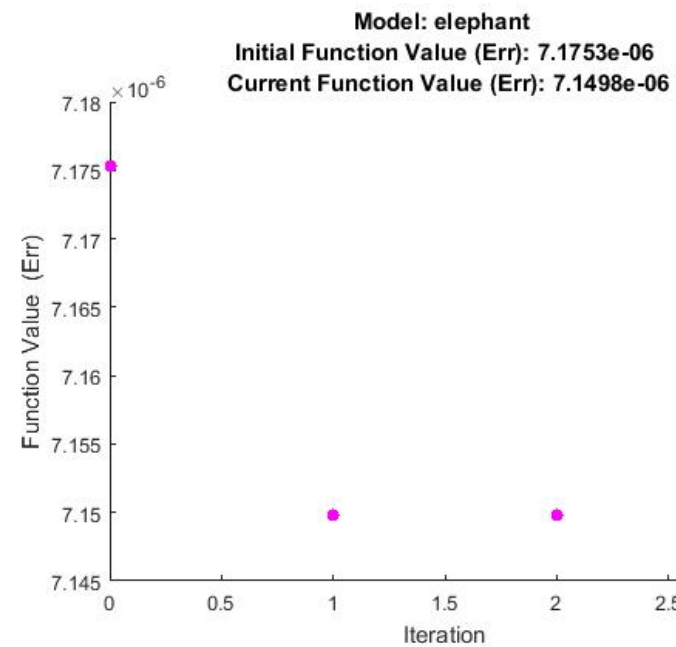
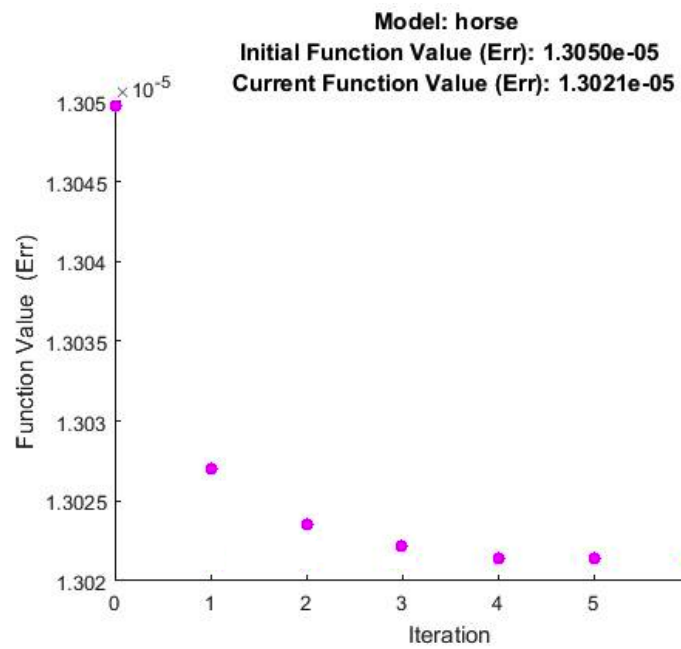


Figure 5.85 (cont'd)

(a) Horse Optimization



(b) Flamingo Optimization

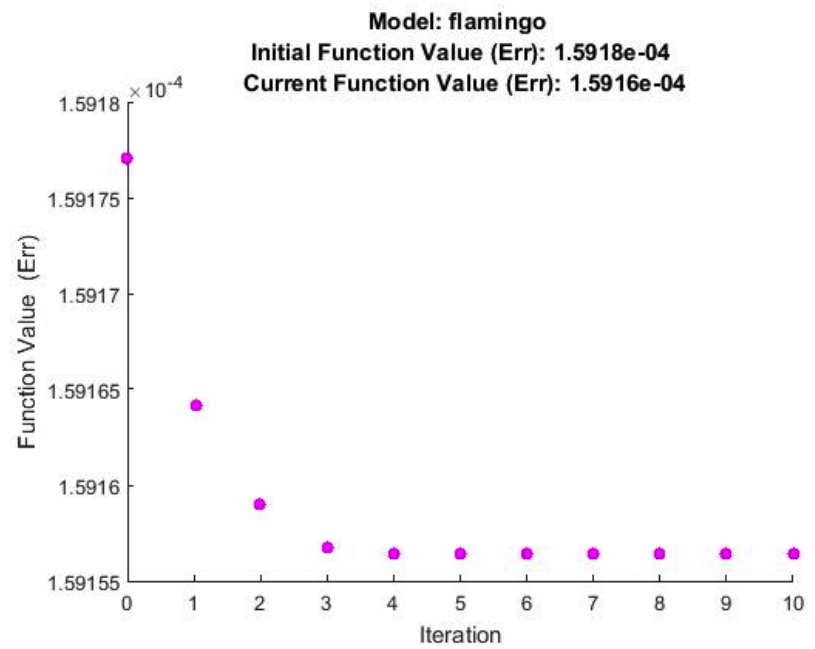
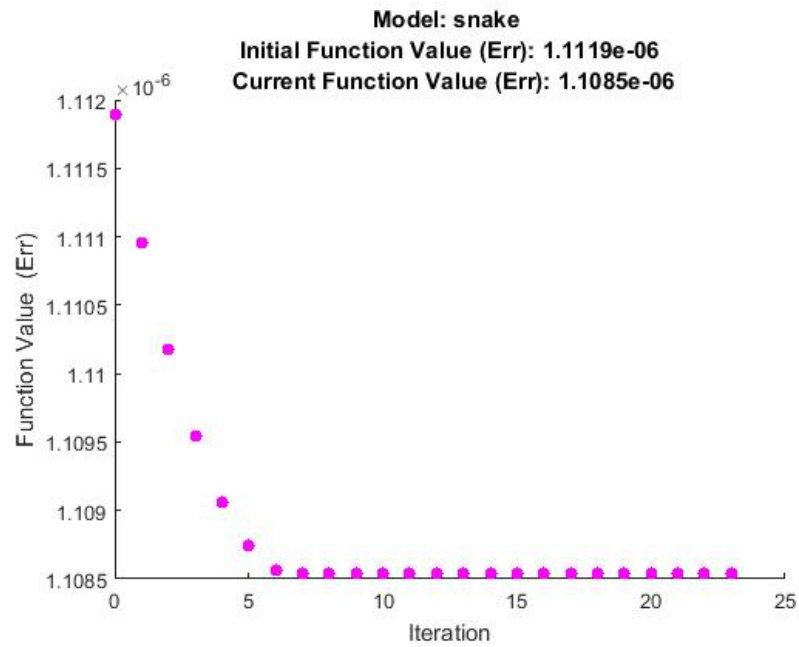


Figure 5.85 (cont'd)

(a) Snake Optimization



(b) Woman Optimization

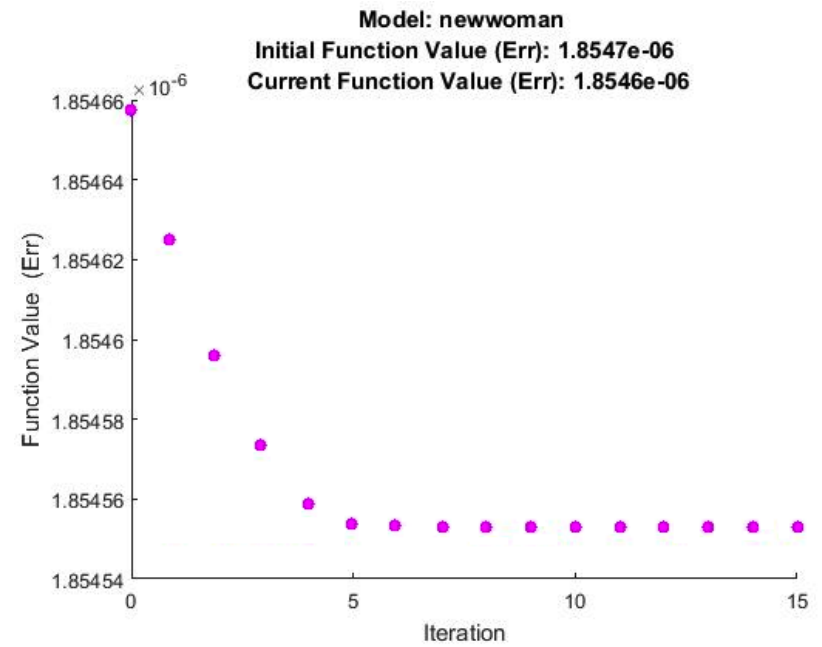
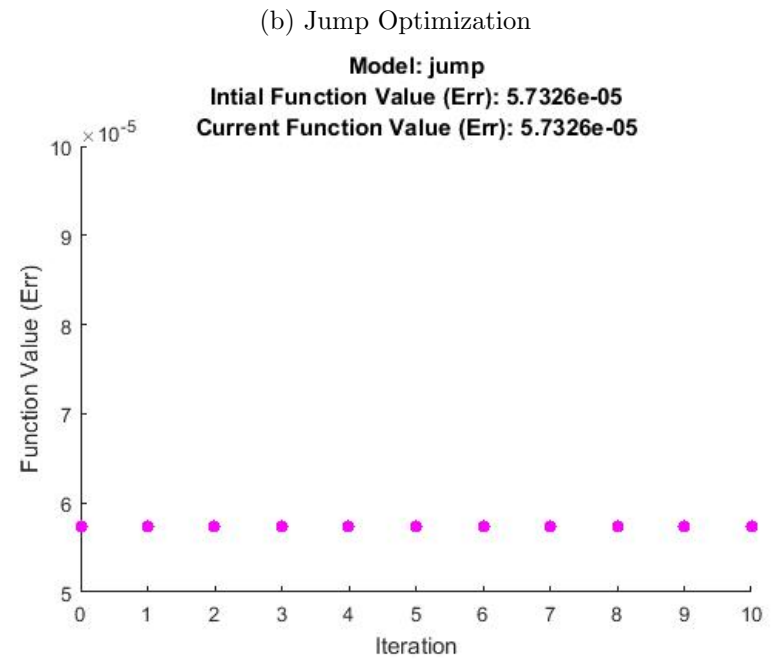
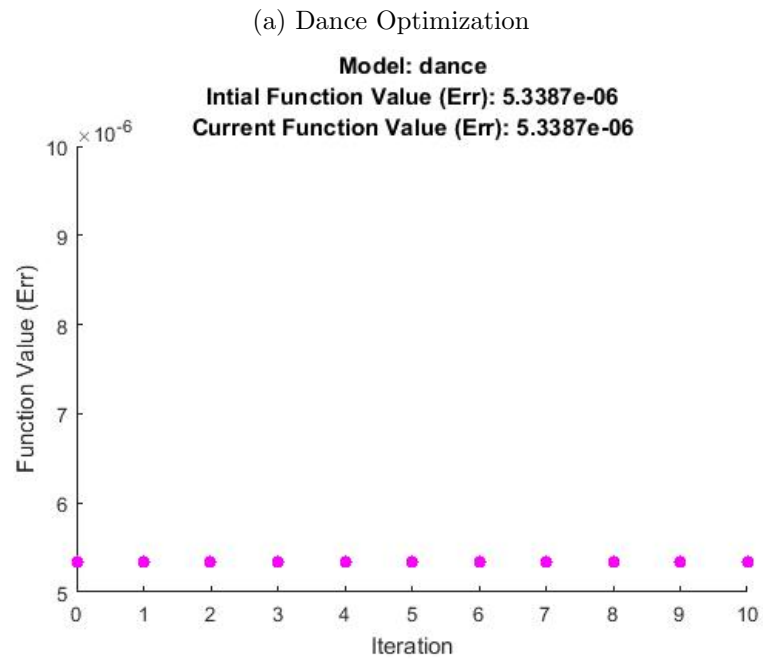


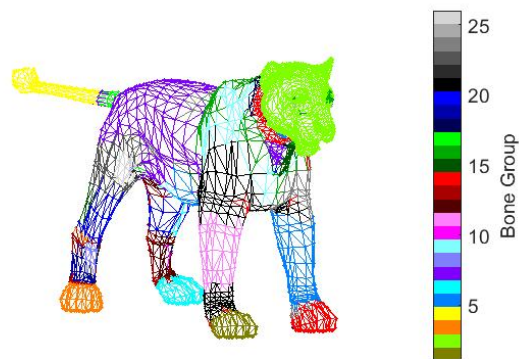
Figure 5.85 (cont'd)



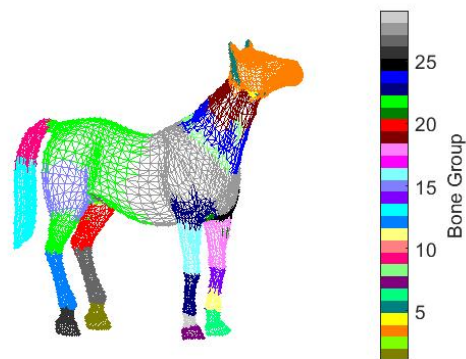
Additionally, large undefined bodies of mass are difficult to cluster appropriately and subsequently optimize. Often the heat map of the torso, back and belly of models indicate some error, such as with the elephant, lion and horse (see Figure 5.86). Typically, skeletal structures and skinning weights would define the torso as a banded series of bones following the spinal structure of the model. However, EP-LBS models are typically clustered such the the torso is one large bone group with a few isolated patches of vertices as islands within the bone group, representing additional bones in areas with large vertex movement. Adjacency constraints during initialization and sparsity constraints for weights during optimization could address the differences in the expected and computed torso vertex positions.

Optimization and constraint adherence are effectively demonstrated by the decrease and subsequent convergence on the objective function to near-zero values. Most models saw modest decrease in objective function value over the course of roughly half a dozen iterations. The limited number of iterations prior to convergence indicate an effective EP-LBS initialization process. More iterations would imply an initial parameter set further from the eventual local minimum, which would result in greater computation time.

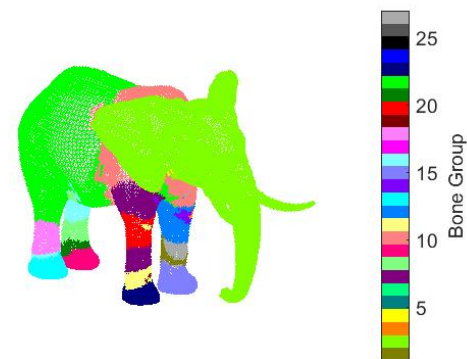
Two models, jump and dance, saw no movement in the objective function after ten EP-LBS iterations. This is a result of the characteristics of input poses and the effectiveness of EP-LBS initialization, as well as the lack of sparsity constraint. More poses, spanning a wide range of motion for each bone would yield a better initial clustering for the dance and jump models. Additionally, because the EP-LBS initialization process computes a highly optimal starting parameter set for the given clustering, very little movement is seen for the objective function optimization, resulting in a flat optimization plot. Finally, the implementation of the sparsity constraint would offer additional opportunities for optimization that may results in more movement on the plot, showing minimization of the objective function.



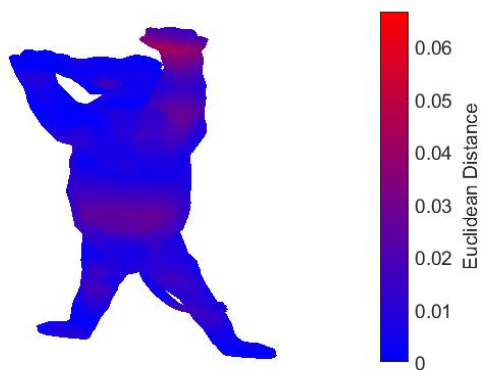
(a) Lion Initial Bones



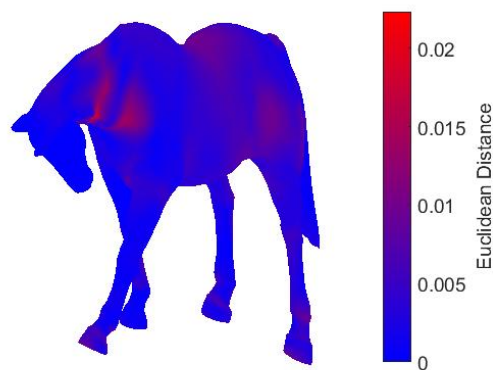
(b) Horse Initial Bones



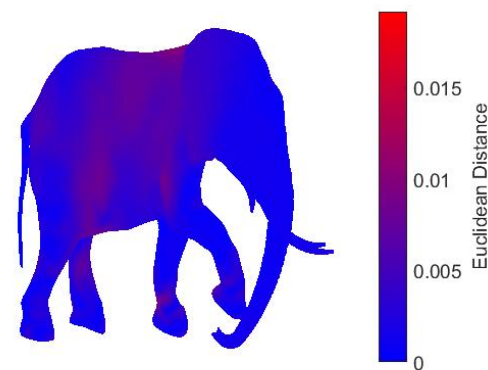
(c) Elephant Initial Bones



(d) Lion Pose 06



(e) Horse Pose 10



(f) Elephant Pose 06

Figure 5.86: Large undefined groups of vertices, such as the torso regions of biped and quadruped models, may include island patches of vertices within a larger bone group or may not identify additional bones in the region, causing errors in the initial clustering that can not be effectively optimized.

	Optimization Results			Additional Stats		
Model	# Opt Iters	Opt Time (min)	Model Error (E)	Max Vertex Distance	Std Dev Vertex Distance	Total Execution Time (min)
Cat [98]	4	18.1	6.629×10^{-5}	0.0406	0.0039	22.2
Woman	15	65.9	1.855×10^{-6}	0.0301	0.0010	67.3
Snake [49]	23	63.2	1.109×10^{-6}	0.0067	0.0007	66.9
Lion [98]	6	26.8	8.069×10^{-5}	0.0667	0.0061	28.1
Horse [98]	7	97	1.302×10^{-5}	0.0223	0.0024	100
Flamingo [98]	10	325.1	1.592×10^{-4}	0.0701	0.0079	330
Camel [98]	10	575	2.5×10^{-5}	0.0346	0.0035	580.2
Elephant [98]	3	1210.4	7.15×10^{-6}	0.0191	0.0015	1228.2
Dance [33]	10	70.3	5.339×10^{-6}	0.0140	0.0013	72.6
Jump [33]	10	27.2	5.733×10^{-5}	0.0358	0.0035	27.6

Table 5.3: Optimization Results

5.4.3 Algorithm Complexity

The runtime complexity of this algorithm is presented in terms of the number of vertices, I , the number of bones or joints, J , and the number of poses, P , where the number of vertices is significantly larger than the number of bones and poses.

The complexity of initialization is determined by the computation of immediate neighbors for each vertex, IP , the computation of transforms for every bone in every pose and the solving of a linear system of equations to compute the weights, IPJ . This process is repeated for c iterations.

$$O(c(IP + PJ + IPJ)) \tag{5.6}$$

Complexity of optimization is determined by the number of values ($7PJ + IP$) the optimization algorithm must compute each iteration and the number of iterations, d , performed in total.

$$O(d(7PJ + IP)) \tag{5.7}$$

The overall complexity of the algorithm is ultimately a function of the number of vertices, joints and poses.

$$O(2IP + 8PJ + IPJ) \tag{5.8}$$

The overall complexity of the entire EP-LBS algorithm can simply be expressed as a function of the number of vertices, joints and poses, where the number of vertices is significantly greater than the number of bones or poses for the model, as shown in Equation (5.9).

$$O(IPJ) \tag{5.9}$$

where

$$I \gg J \tag{5.10}$$

$$I \gg P \tag{5.11}$$

The overall complexity of the entire EP-LBS algorithm can simply be expressed as a function of the number of vertices, joints and poses, where the number of vertices is significantly greater than the number of bones or poses for the model, as shown in Equation (5.9).

5.4.4 Execution Time

As expected, the execution time of EP-LBS is most heavily influenced by the number of vertices in the model. Total execution time for models of a few thousand vertices in this research was 20-100 minutes, averaging less than an hour per model. In contrast the three models with the highest resolution models of 20-40,000 vertices resulted in 5-20 hours of execution time. Being a pre-process performed once at the onset of the animation process,

single-threaded serial execution times of up to a few hours are reasonable. EP-LBS running times (comparable to published times of other skinning decomposition methods for skeletal animation) demonstrate the viability of EP-LBS as a method for LBS-based skinning decomposition.

Chapter 6

Conclusion

This research presents Example-based Parameterization of Linear Blend Skinning for Skinning Decomposition (EP-LBS) as a method for computing the LBS animation parameters, in the form of skinning weights and bone transformations needed to recreate a set of example character deformations. The EP-LBS solution to skinning decomposition introduces a new mathematical model for LBS that uses quaternion rotations and defines an objective function constrained only by upper and lower bounds on the animation parameters that generates an optimal set of initial animation parameters for optimization.

As shown in results, EP-LBS is able to compute the number and placement of bones from a collection of example poses with no a priori knowledge of the skeletal structure. Once initial singleton-bone groups have been computed, initial weights are determined using NNLS to give an initial parameterization for EP-LBS optimization. The optimization process proceeds with a bounded gradient descent implementation with parameter mapping for additional constraint adherence. Results are presented indicating that the EP-LBS initialization and optimization workflows minimize the objective function error (often in less than a dozen iterations) and is able to reproduce the example poses with minimal error.

EP-LBS Optimization extends basic gradient descent methods with parameter mapping to achieve constraint adherence for skinning decomposition optimization. In this manner, this research has demonstrated a new method for example-based skinning decomposition that uses standard skinning decomposition problem structure and addresses each of the three

major sub-components to the skinning decomposition workflow: initialization, optimization and constraint adherence.

EP-LBS is a unique approach to skinning decomposition because it is able to compute the bone structure using only a set of example poses consisting of a series of character meshes. Additionally, EP-LBS introduces parameter mapping, which offers a distinct approach to constraint adherence that allows for use of fundamental optimization techniques that require little overhead. By using parameter mapping, EP-LBS can use basic gradient descent optimization, which offers computationally efficient minimization of the LBS objective function.

The objectives of this research were to offer an comprehensive automated optimization method for skinning decomposition that was scalable and adhered to standard LBS constraints. The presented EP-LBS research introduces a process that requires minimal pre- and post-processing, limited heuristics and manual adjustment. The EP-LBS optimization process simultaneously optimizes the solution both the skinning weights and bone transformations, while adhering to standard LBS constraints. Additionally, it is demonstrated that this process is scalable, viable for models of a few hundred vertices and a half-dozen poses to models with tens of thousands of vertices and dozens of bones and poses. As an offline method for animation systems, EP-LBS is shown to be a viable solution for the computation of standard LBS animation parameters from a set of example poses.

6.1 Contributions of this Thesis

This thesis presents Example-based Parameterization of Linear Blend Skinning for Skinning Decomposition and offers these significant contributions to the skinning decomposition

process:

- ***Parameter Mapping for LBS Constraint Adherence***: The standard LBS model results in conflicting constraints on animation parameters. EP-LBS addresses interdependent and conflicting constraints by mapping parameter values to alternate domains ideal for nonlinear minimization. This modification results in constraint adherence at minimal computational cost.
- ***Unified Least-Squares Non-linear Optimization for LBS*** : Most standard skinning decomposition approaches repeatedly alternate the optimization of skinning weights. Given a set of example poses, the skinning decomposition problem is formulated into a unified least-squares objective function that is minimized to solve for both skinning weights and joint transforms simultaneously. Parameter mapping and gradient descent are used to define the optimization algorithm. The result is a comprehensive example-based non-linear optimization solution for automated, computation of all LBS parameters, simultaneously.
- ***Iterative Motion-based Clustering for Automated LBS Initialization***: Many skinning decomposition implementations require the number of bones as input to the optimization system. EP-LBS uses motion-based clustering of estimated bone transforms to compute initial transforms. The initial transforms are combined with non-negative least-squares to compute initial weights. This process is executed iteratively, each time splitting bones in areas with large error into two separate bone groups, until the model converges on a basic skeletal structure and initial skinning weights that generate a model with minimal error. The result is an automatic estimation of joints and bone influence for a model that requires no a priori knowledge of the model's skeletal

structure.

In all, EP-LBS defines a skinning decomposition process that takes only a series of example poses as input and computes the number of bones and animation parameter values needed to recreate the initial example poses. The process introduces a new example-based LBS problem structure and initialization process, a new constraint adherence method and a new bounded optimization approach to determine the skinning decomposition solution. The results show that EP-LBS can, without a priori knowledge of any bone structure, achieve all of the standard objectives of successful skinning decomposition solutions, as well as the additional rigorous objectives defined by this research.

Analysis of EP-LBS experiment results presented in this thesis lead to exciting potential applications of this research and interesting areas for future work.

6.2 Applications for this Research

EP-LBS defines a skinning decomposition process for computing animation parameters for LBS animation systems. As such, the most straight-forward use for the workflow presented in this research is as a pre-processing task, the results of which can be exported to any LBS-supported 3D animation system or game engine for use in production.

One advantage of the newly introduced EP-LBS process is the ability to optimize the computational efficiency of an offline process. Skinning weights can be pre-computed and transformations can be derived from example data. As offline pre-processing for any 3D engine, EP-LBS can leverage motion capture and other 3D scanning and imaging technology more effectively to record data that can generate baseline LBS models with minimal artist effort, allowing greater opportunities for more artistic styling and animation rather than

tedious LBS skinning and rigging tasks.

Although, not intended to be a real-time system, an interesting avenue of future study could be exploring ways in which EP-LBS techniques could be integrated with or used as a pre-processing task for interactive animation interfaces [25, 37, 86], real-time systems [55, 45, 90, 102, 94, 47] and advanced non-linear deformation systems [104, 84], each of which are open areas of research for example-based LBS that constitute relevant potential applications and avenues of future work for this research. It may also be interesting to explore how EP-LBS methods can be used to improve modeling and animation for mobile systems, such as phones and gaming devices.

Perhaps the most exciting applications are a result of the recent widespread introduction of virtual and augmented reality into mainstream entertainment. The predicted everyday, common use of VR and AR technologies (as indicated by the development and wide release of a variety of VR content for mobile phones and gaming systems in 2015 and 2016), anticipates the need for large quantities of additional 3D content. This scenario presents an opportunity to find ways to generate 3D content more efficiently and EP-LBS can be a tool for increased efficiency in articulated model creation. Where a variety of human motion can be captured and subsequently processed with EP-LBS, massive amounts of varied background characters can be produced quickly with the level of assortment of shape and movement that makes VR characters immersive rather than distracting. There is great potential for EP-LBS to be used as a means for rapid prototyping and content generation in a variety of 3D environments, including VR, film and gaming.

6.2.1 Opportunities for Additional Analysis in Related Fields

The EP-LBS problem appears to take on a machine learning approach to training animation parameter data, however, machine learning techniques are not applicable to EP-LBS. Although machine learning approaches are easily applied to computation of skinning weights or bone count (if considered independent of bone transformations), training data cannot be used for the computation of bone transformations because the bone positions vary between poses. The nature of varying poses means the training data will not yield data relevant to the testing data. This is a direct result of the unified problem structure of EP-LBS.

Still, as an additional qualitative metric, it is possible to perform cognitive perception and decision making experiments derived from communications and human computer interaction fields, to measure the level at which errors in the EP-LBS methods are discernible. This information could be used to both evaluate the success of EP-LBS and also to fine tune the termination metrics for the optimization algorithm.

6.3 Future Work

While this thesis presents a full skinning decomposition pipeline that generates viable animation parameters for standard LBS systems, additional research and exploration could yield further improvements to the process, as is indicated by results and analysis presented in this thesis. Little work has been done to optimize EP-LBS computations. Parallel computing implementations have the potential to provide additional time savings during initialization and optimization. The structure of the EP-LBS objective function and the nature of gradient descent optimization is ideal for exploiting parallel computing and multi-threaded implementations. The majority of execution time is spent on initialization, indicating performance

improvements to the initialization process have potential to significantly improve performance. Further, the implementation of EP-LBS uses finite difference estimation for gradient (derivative) calculations. Direct computation of the derivative would also likely improve that computation time for EP-LBS optimization.

This research opens up avenues for related and tangential research, such as use of EP-LBS initialization as a means for optimizing existing skeletal structures. Given a model in a series of poses that span the entire range of motion desired for model, the EP-LBS process could be used to compute potentially fewer bone groups than the initial complex skeletal structure used to generate the example poses, eliminating unused bones. The initialization process could be further extended to take skeletal structure as input and work backwards to combine unused bones, creating a more efficient skeletal structure.

One promising research direction lies within the analysis of the tradeoffs between adding more bones, using more weights and increasing the complexity of the optimization algorithm. Skinning weights dominate the computational resources consumed during the EP-LBS process as a result of being the most numerous of the animation parameters. Limitations on the number of weights that may influence a vertex (sparsity constraints) can reduce the amount of memory and quantity of animation parameters being computed. While limiting the number of bone influences can reduce the amount of data that must be computed, there is also the potential to increase the complexity of the optimization algorithm.

The EP-LBS implementation does not produce sparse weights, which indicates there are opportunities to explore the addition of a cardinality or sparseness constraint to limit the number of influence bones to a small number. Historically, vertices have sparseness constraint of four, limiting the influence of the movement of a single vertex to at most four bones. The four-bone constraint is chosen in great part due to the optimization of hardware

implementations of LBS calculations. As an alternative, a post-processing step could be used to compress and reduce weights, potentially yielding a more efficient model.

For continuity of bone groups in bone clustering during initialization, implementing some connectivity constraint, may yield better initialization results faster as a result of fewer animation parameters for which to solve. Initialization solutions with a connectivity constraint should converge in fewer iterations, but the iteration count savings must be balanced with the computational time expense of the more complex connectivity-constrained initialization and optimization algorithms.

The research results presented in this thesis indicate that the EP-LBS cluster initialization is suitable for both determining the number of bones needed to animate a character mesh and initializing the bone groups and bone transformations for example-based LBS methods. This research introduces the combination of basic gradient descent algorithms with the parameter mapping to create an effective constrained optimization algorithm that can handle conflicting constraints. This thesis presents a full skinning decomposition pipeline that generates viable LBS animation parameters that adhere to standard LBS constraints and opens the doors to various avenues of extended and related research.

APPENDICES

Appendix A

Coordinate Systems

Chapter 2 introduced the Linear Blend Skinning model for vertex deformation as:

$$\mathbf{v}'_i = \sum_j w_{i,j} \mathbf{T}_j \mathbf{v}_i \quad (\text{A.1a})$$

Subject to constraints:

$$0 \leq w_{ij} \quad (\text{A.1b})$$

$$\sum_j w_{ij} = 1 \quad (\text{A.1c})$$

In the LBS model, a skeletal structure embedded in a character and the movement of the skeleton drives the movement of the character geometry. The geometry of a character is defined using a model coordinate system, or model space. In addition to the model coordinate system, each bone in the skeletal structure has its own coordinate system. For every bone there is typically a matrix that transforms the bone in the bone coordinate system, or bone space, to model space. For hierarchical models, bones are transformed from the bone space to the parent bone space.

In many graphics systems the transformation from local bone coordinate system to the model coordinate system is computed and stored as a bind matrix, \mathbf{B}_j . The inverse of the

bind matrix, \mathbf{B}_j^{-1} , converts from model coordinates to bone coordinates. Vertex positions are expressed in a model coordinate system. The transformation \mathbf{T}_j implies a global transformation expressed in the model coordinates for each bone. In practice, bone transformations are expressed in local bone coordinates. A more detailed representation of vertex deformation requires transforming vertices into the appropriate bone coordinate system, applying the bone transformation and transforming the vertex back into model space.

Incorporating conversions between coordinate spaces, the deformation of a vertex from Equation A.1a can be represented with the following equations:

$$\mathbf{T}_j \mathbf{v}_i = \mathbf{B}_j \mathbf{M}_j \mathbf{B}_j^{-1} \mathbf{v}_i \quad (\text{A.2})$$

$$\mathbf{v}'_i = \sum_j w_{ij} \left(\mathbf{B}_j \mathbf{M}_j \mathbf{B}_j^{-1} \right) \mathbf{v}_i \quad (\text{A.3})$$

In Equation A.2 the vertex transformation by one bone is defined. \mathbf{M}_j represents the transformation of bone j that is responsible for the vertex deformation in the current pose. The bind matrix and its inverse convert between model coordinate system used by the vertices and the bone coordinate system used by the joints.

The use of multiple coordinate systems allow the clear definition of all points in space with respect to the origin for that coordinate system. In this manner, complex models can be created with intuitive transformation representations. However, for this research we use the global model space representation and than local parent spaces. As previously noted, in addition to operating in different coordinate frames, bone transformations are defined by the parent bone transformations as well as the local bone transformation. A child bone is said to rotate around a parent joint's coordinate system. In practice, the model space

bone transformation abstraction we used is ultimately the product of the local parent bone representations.

Appendix B

Weight Mapping

The chapter mathematically demonstrates that mapping operations performed on the bounded LBS weight parameters yield new values also within the bounded range. Additionally, it is demonstrated that the mapped values also adhere to the summation constraint.

Proof of Weight Mapping Constraint Adherence

Lemma B.0.1 *Let w be a real value on the closed unit interval $[0, 1]$ such that $0 \leq w \leq 1$.*

Then $0 \leq 1 - w \leq 1$.

Proof Assume $0 \leq w \leq 1$. Then by multiplying by -1, $0 \geq -w \geq -1$. Adding 1 yields $1 \geq 1 - w \geq 0$. It follows that $0 \leq 1 - w \leq 1$. ■

Lemma B.0.2 *Let a and b be real values on the closed unit interval $[0, 1]$ such that $0 \leq a \leq 1$ and $0 \leq b \leq 1$. Then the product of a and b is also a value on the closed unit interval $[0, 1]$ such that $0 \leq ab \leq 1$.*

Proof Suppose $0 \leq a \leq 1$ and $0 \leq b \leq 1$. Then $b*0 = 0$ and $b*1 = b$. By $b*0 \leq b*a \leq b*1$, it follows that $0 \leq ba \leq b$. Since $0 \leq b \leq 1$, $0 \leq ab \leq 1$. ■

Theorem B.0.3 *Let $w'_1 = w_1$ and $w'_j = w_j \left(1 - \sum_{k=1}^{j-1} w'_k\right) = w_j \prod_{k=1}^{j-1} (1 - w_k)$ and $w'_J = 1 - \sum_{j=1}^{J-1} w_j$. If $0 \leq w_j \leq 1$, then $0 \leq w'_j \leq 1$ and $\sum_{j=1}^J w_j = 1$ for all j .*

Proof (i) Suppose $0 \leq w_1 \leq 1$ and $w'_1 = w_1$. Then $0 \leq w'_1 \leq 1$.

(ii) Let $j = 2$. Suppose $w'_j = w_j \left(1 - \sum_{k=1}^{j-1} w'_k\right)$. Thus $w'_2 = w_2 \left(1 - \sum_{k=1}^1 w'_k\right)$. From (i), $0 \leq w'_1 \leq 1$. Therefore, $0 \leq 1 - w'_1 \leq 1$. Let $a = w_2$ and $b = \left(1 - \sum_{k=1}^1 w'_k\right)$. By Lemma B.0.2 it follows $0 \leq w_2 \left(1 - \sum_{k=1}^1 w'_k\right) \leq 1$. Therefore $0 \leq w'_2 \leq 1$.

(iii) Let $j = 3$. Then from $w'_3 = w_3 \left(1 - \sum_{k=1}^2 w'_k\right)$ it follows $w'_3 = w_3 (1 - w'_1 - w'_2)$ which can be rewritten as $w'_3 = w_3 (1 - w_1)(1 - w_2)$. Let $a = (1 - w_1)$ and $b = (1 - w_2)$. Then by Lemma B.0.2 $0 \leq ab \leq 1$. Let $c = ab$ and $d = w_3$, therefore $0 \leq cd \leq 1$. Thus, $0 \leq w_3 \left(1 - \sum_{k=1}^2 w'_k\right) \leq 1$. Therefore $0 \leq w'_3 \leq 1$.

Suppose $0 \leq w'_1 \leq 1$ and $w'_j = w_j \left(1 - \sum_{k=1}^{j-1} w'_k\right)$. Thus $w'_2 = w_2 \left(1 - \sum_{k=1}^1 w'_k\right)$. From (i), $0 \leq w'_1 \leq 1$. Therefore, $0 \leq 1 - w'_1 \leq 1$. Let $a = w_2$ and $b = \left(1 - \sum_{k=1}^1 w'_k\right)$. By Lemma B.0.2 it follows $0 \leq w_2 \left(1 - \sum_{k=1}^1 w'_k\right) \leq 1$. Therefore $0 \leq w'_2 \leq 1$.

(iv) Let $w'_j = w_j \left(1 - \sum_{k=1}^{j-1} w'_k\right) = w_j \prod_{k=1}^{j-1} (1 - w'_k)$.

Then, by Lemma B.0.2 $0 \leq w'_j \leq 1$ for all j .

(v) Let $j = J$ and $w_J = 1 - \sum_{k=1}^{J-1} w'_k$.

By (i)-(iv), $0 \leq \sum_{k=1}^{j-1} w'_k \leq 1$. Thus, $0 \leq w_J \leq 1$.

(vi) Therefore, by (i),(ii), (iii), (iv) and (v), $0 \leq w'_j \leq 1$ for all j . ■

Appendix C

Rotation Mapping

The chapter mathematically demonstrates that mapping operations performed on the bounded LBS Euler angles yield new values also within the bounded range.

Proof of Bounded Euler Values

Proposition C.0.1 *Suppose \mathbf{e} is the unmapped Euler value such that $-\pi \leq \mathbf{e} \leq \pi$ and \mathbf{e}'' is the bounded Euler value computed from the mapped Euler value $\mathbf{e}' = \frac{\mathbf{e} + \pi}{2\pi}$.*

$$\text{Let } \mathbf{e}'' = \begin{cases} 0 & \text{if } \mathbf{e}' \leq 0 \\ 1 & \text{if } \mathbf{e}' \geq 1 \\ \mathbf{e}' & \text{if } 0 < \mathbf{e}' < 1 \end{cases}$$

$$\text{Then } -\pi \leq \mathbf{e}'' \leq \pi$$

Proof Suppose $-\pi \leq \mathbf{e} \leq \pi$ and $\mathbf{e}' = \frac{\mathbf{e} + \pi}{2\pi}$.

- (i) Let $\mathbf{e}' \leq 0$. Then $\mathbf{e}'' = 0$. Thus $-\pi \leq \mathbf{e}'' \leq \pi$.
- (ii) Let $\mathbf{e}' \geq 1$. Then $\mathbf{e}'' = 1$. Thus $-\pi \leq \mathbf{e}'' \leq \pi$.
- (iii) Let $0 < \mathbf{e}' < 1$. Then $\mathbf{e}'' = \mathbf{e}' = \frac{\mathbf{e} + \pi}{2\pi}$. Then $0 \leq \mathbf{e}'' \leq 1$. Therefore, $-\pi \leq \mathbf{e}'' \leq \pi$.

REFERENCES

REFERENCES

- [1] Avatar(2009) - Box Office Mojo. <http://www.boxofficemojo.com/movies/?id=avatar.htm>. Accessed: 2014-04-26.
- [2] Brett Allen, Brian Curless, and Zoran Popovic. Articulated body deformation from range scan data. *ACM Trans. Graph.*, 21(3):612–619, July 2002.
- [3] Brett Allen, Brian Curless, and Zoran Popović. The space of human body shapes: reconstruction and parameterization from range scans. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 587–594, New York, NY, USA, 2003. ACM.
- [4] K.S. Arun, T.S. Huang, and S.D. Blostein. Least-squares fitting of two 3-d point sets. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (5):698–700, 1987.
- [5] Oscar Kin-Chung Au, Chiew-Lan Tai, Hung-Kuo Chu, Daniel Cohen-Or, and Tong-Yee Lee. Skeleton extraction by mesh contraction. *ACM Trans. Graph.*, 27(3):1–10, 2008.
- [6] G. Aujay, F. Hétroy, F. Lazarus, and C. Depraz. Harmonic skeleton for realistic character animation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 151–160. Eurographics Association, 2007.
- [7] Ilya Baran and Jovan Popovic. Automatic rigging and animation of 3d characters. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 72, New York, NY, USA, 2007. ACM.
- [8] G. Bharaj, T. Thormählen, H.P. Seidel, and C. Theobalt. Automatically rigging multi-component characters. In *Computer Graphics Forum*, volume 31, pages 755–764. Wiley Online Library, 2012.
- [9] blender.org. Blenderwiki - doc:2.4/manual/modeling/meshes/weight paint, 2013.
- [10] J. Bloomenthal. Medial-based vertex deformation. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 147–151. ACM, 2002.

- [11] Jules Bloomenthal and Chek T Lim. Skeletal methods of shape manipulation. In *Shape Modeling International*, volume 99, pages 44–47, 1999.
- [12] Harry Blum. A transformation for extracting new descriptors of shape. 19:362–380, 1967.
- [13] Jonathan Cameron and Joan Lasenby. Estimating human skeleton parameters and configuration in real-time from marked optical motion capture. In *AMDO '08: Proceedings of the 5th international conference on Articulated Motion and Deformable Objects*, pages 92–101, Berlin, Heidelberg, 2008. Springer-Verlag.
- [14] Junjie Cao, Andrea Tagliasacchi, Matt Olson, Hao Zhang, and Zhinxun Su. Point cloud skeletons via laplacian based contraction. In *Shape Modeling International Conference (SMI), 2010*, pages 187–197. IEEE, 2010.
- [15] Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. Interactive skeleton-driven dynamic deformations. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 586–593, New York, NY, USA, 2002. ACM.
- [16] E. Chaudhry, LH You, and J.J. Zhang. Character skin deformation: A survey. In *Computer Graphics, Imaging and Visualization (CGIV), 2010 Seventh International Conference on*, pages 41–48. IEEE, 2010.
- [17] 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, Washington, DC, USA, 2011. IEEE Computer Society.
- [18] Wang Cheng, Ren Cheng, Lei Xiaoyong, and Dai Shuling. Automatic skeleton generation and character skinning. In *VR Innovation (ISVRI), 2011 IEEE International Symposium on*, pages 299 –304, march 2011.
- [19] Jen-Hui Chuang, Narendra Ahuja, Chien-Chou Lin, Chi-Hao Tsai, and Cheng-Hui Chen. A potential-based generalized cylinder representation. *Computers & Graphics*, 28(6):907–918, 2004.
- [20] J.H. Chuang, C.H. Tsai, and M.C. Ko. Skeletonisation of three-dimensional object using generalized potential field. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(11):1241–1251, 2000.

- [21] N.D. Cornea, D. Silver, and P. Min. Curve-skeleton properties, applications, and algorithms. *Visualization and Computer Graphics, IEEE Transactions on*, 13(3):530–548, 2007.
- [22] Tamal K Dey and Jian Sun. Defining and computing curve-skeletons with medial geodesic function. In *Symposium on geometry processing*, volume 6, pages 143–152, 2006.
- [23] H. Doraiswamy, A. Sood, and V. Natarajan. Constructing reeb graphs using cylinder maps. In *Proceedings of the 2010 annual symposium on Computational geometry*, pages 111–112. ACM, 2010.
- [24] Fletcher Dunn and Ian Parberry. *3D math primer for graphics and game development*. CRC Press, 2015.
- [25] Wei-Wen Feng, Byung-Uck Kim, and Yizhou Yu. Real-time data driven deformation using kernel canonical correlation analysis. In *ACM Transactions on Graphics (TOG)*, volume 27, page 91. ACM, 2008.
- [26] Roger Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2013.
- [27] Sven Forstmann and Jun Ohya. Fast skeletal animation by skinned arc-spline based deformation. *EG 2006 Short Papers*, pages 1–4, 2006.
- [28] Sven Forstmann, Jun Ohya, Artus Krohn-Grimberghe, and Ryan McDougall. Deformation styles for spline-based skeletal animation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '07, pages 141–150, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.
- [29] Ivo Zoltan Frey and Ivo Herzeg. Spherical skinning with dual quaternions and qtan-gents. *ACM SIGGRAPH Talks*, 11:1–11, 2011.
- [30] J. Gain and D. Bechmann. A survey of spatial deformation from a user-centered perspective. *ACM Transactions on Graphics (TOG)*, 27(4):107, 2008.
- [31] Sarah F. F. Gibson and Brian Mirtich. A survey of deformable modeling in computer graphics. Technical report, Mitsubishi Electric Research Laboratories, 1997.
- [32] Tanya Grigorishin, Gamal Abdel-Hamid, and Y-H Yang. Skeletonisation: An electro-static field-based approach. *Pattern Analysis and Applications*, 1(3):163–177, 1998.

- [33] Igor Guskov and Andrei Khodakovsky. Wavelet compression of parametrically coherent mesh sequences. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '04, pages 183–192, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.
- [34] N. Hasler, T. Thorm
”ahlen, B. Rosenhahn, and H.P. Seidel. Learning skeletons for shape and pose. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 23–30. ACM, 2010.
- [35] M Sabry Hassouna and Aly A Farag. Robust centerline extraction framework using level sets. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 458–465. IEEE, 2005.
- [36] Jim Hejl. Hardware skinning with quaternions. *Game Programming Gems*, 4:487–495, 2004.
- [37] Robert Held, Ankit Gupta, Brian Curless, and Maneesh Agrawala. 3d puppetry: a kinect-based interface for 3d animation. In *UIST*, pages 423–434. Citeseer, 2012.
- [38] DM Henderson. Euler angles, quaternions, and transformation matrices—working relationships. *NASA TM-74839, JSC-12960*, 1977.
- [39] Masaki Hilaga, Yoshihisa Shinagawa, Taku Kohmura, and Tosiyasu L Kunii. Topology matching for fully automatic similarity estimation of 3d shapes. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 203–212. ACM, 2001.
- [40] S. G. Hoggar. *Mathematics for computer graphics*. Cambridge tracts in theoretical computer science: 14. Cambridge [England] ; New York : Cambridge University Press, 1992., 1992.
- [41] Kayra M. Hopkins. Data driven smooth skin weight computation. Master’s thesis research, Michigan State University, 2011.
- [42] Berthold KP Horn. Some notes on unit quaternions and rotation. *Lecture handouts*, 2001. <http://people.csail.mit.edu/bkph/articles/Quaternions.pdf>, Accessed: 2014-04-26.
- [43] Hui Huang, Shihao Wu, Daniel Cohen-Or, Minglun Gong, Hao Zhang, Guiqing Li, and Baoquan Chen. L1-medial skeleton of point cloud. *ACM Trans. Graph.*, 32(4):65–1, 2013.

- [44] Qi-Xing Huang, Martin Wicke, Bart Adams, and Leonidas Guibas. Shape decomposition using modal analysis. *Computer Graphics Forum*, 28(2):407–416, April 2009.
- [45] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, et al. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 559–568. ACM, 2011.
- [46] A. Jacobson, I. Baran, L. Kavan, J. Popović, and O. Sorkine. Fast automatic skinning transformations. *ACM Transactions on Graphics (TOG)*, 31(4):77, 2012.
- [47] Alec Jacobson, Zhigang Deng, Ladislav Kavan, and JP Lewis. Skinning: Real-time shape deformation. In *ACM SIGGRAPH 2014 Courses*, 2014.
- [48] Alec Jacobson and Olga Sorkine. Stretchable and twistable bones for skeletal shape deformation. In *ACM Transactions on Graphics (TOG)*, volume 30, page 165. ACM, 2011.
- [49] Doug L. James and Christopher D. Twigg. Skinning mesh animations. *ACM Trans. Graph.*, 24:399–407, 2005.
- [50] Tao Ju, Scott Schaefer, and Joe Warren. Mean value coordinates for closed triangular meshes. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 561–566. ACM, 2005.
- [51] S. Katz and A. Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 954–961. ACM, 2003.
- [52] Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O’Sullivan. Skinning with dual quaternions. In *I3D ’07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 39–46, New York, NY, USA, 2007. ACM.
- [53] Ladislav Kavan, Steven Collins, Jiri Zara, and Carol O’Sullivan. Geometric skinning with approximate dual quaternion blending. volume 27, page 105, New York, NY, USA, 2008. ACM Press.
- [54] Ladislav Kavan, P-P Sloan, and Carol O’Sullivan. Fast and efficient skinning of animated meshes. In *Computer Graphics Forum*, volume 29, pages 327–336. Wiley Online Library, 2010.
- [55] Byung-Uck Kim, Wei-Wei Feng, and Yizhou Yu. Real-time data driven deformation with affine bones. *The Visual Computer*, 26(6-8):487–495, 2010.

- [56] 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, New York, NY, USA, 2002. ACM.
- [57] Binh Huy Le and Zhigang Deng. Smooth skinning decomposition with rigid bones. *ACM Trans. Graph.*, 31(6):199:1–199:10, November 2012.
- [58] Binh Huy Le and Zhigang Deng. Robust and accurate skeletal rigging from mesh sequences. *ACM Transactions on Graphics (TOG)*, 33(4):84, 2014.
- [59] J. P. Lewis, Matt Cordner, and Nickson Fong. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 165–172, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.
- [60] Jyh-Ming Lien, John Keyser, and Nancy M. Amato. Simultaneous shape decomposition and skeletonization. In *SPM '06: Proceedings of the 2006 ACM symposium on Solid and physical modeling*, pages 219–228, New York, NY, USA, 2006. ACM.
- [61] H. Liu, X. Wei, J. Chai, I. Ha, and T. Rhee. Realtime human motion control with a small number of inertial sensors. In *Symposium on Interactive 3D Graphics and Games*, pages 133–140. ACM, 2011.
- [62] P. Liu, F. Wu, W. Ma, R. Liang, and M. Ouhyoung. Automatic animation skeleton using repulsive force field. In *Proc. IEEE 11th Pacific conference on Computer Graphics and Applications PG '03*, pages 409–413, 2003.
- [63] L. Lu, F. Hétroy, C. Gérot, B. Thibert, et al. Atlas-based character skinning with automatic mesh decomposition. 2008.
- [64] M. Madaras, R. Ďurikovič, T. Ágošton, and T. Nishita. Skeleton extraction from a mesh for easy skinning animation. In *Proceedings of the 13th International Conference on Humans and Computers*, pages 37–41. University of Aizu Press, 2010.
- [65] Grégoire Malandain and Sara Fernández-Vidal. Euclidean skeletons. *Image and vision computing*, 16(5):317–327, 1998.
- [66] Markets & Markets. Computer graphics market by software (cad/cam, visualization/simulation, digital video, imaging, modeling/animation), service (consulting,

training & support, integration), end-user (enterprise and smb) - worldwide forecasts & analysis (2014-2019). Online, June 2014.

- [67] Autodesk 3DS Max. 3ds max documentation set, 2013.
- [68] Autodesk Maya. *Autodesk Maya 2013 User Guide*, 2013.
- [69] Bruce Merry, Patrick Marais, and James Gain. Animation space: A truly linear framework for character animation. volume 25, pages 1400–1423, New York, NY, USA, 2006. ACM.
- [70] Laurent Moccozet, Fabien Dellas, Nadia Magnenat-thalmann, Silvia Biasotti, Michela Mortara, Bianca Falcidieno, Patrick Min, and Remco Veltkamp. Animatable human body model reconstruction from 3d scan data using templates. In *In Proceedings of Workshop on Modelling and Motion Capture Techniques for Virtual Environments, CAPTECH 2004*, pages 73–79, 2004.
- [71] A. Mohr, L. Tokheim, and M. Gleicher. Direct manipulation of interactive character skins. In *Proceedings of the 2003 symposium on Interactive 3D graphics*, pages 27–30. ACM, 2003.
- [72] Alex Mohr and Michael Gleicher. Building efficient, accurate character skins from examples. *ACM Trans. Graph.*, 22:562–568, July 2003.
- [73] L. Nash Information Services. Box office history for digital animation, 2016.
- [74] LLC Nash Information Services. Box office history for production method - animation/live action, 2016.
- [75] R. Ogniewicz and M. Ilg. Voronoi skeletons: Theory and applications. In *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR’92., 1992 IEEE Computer Society Conference on*, pages 63–69. IEEE, 1992.
- [76] R.L. Ogniewicz and O. Kübler. Hierarchic voronoi skeletons. *Pattern recognition*, 28(3):343–359, 1995.
- [77] A Cengiz Öztireli, Ilya Baran, Tiberiu Popa, Boris Dalstein, Robert W Sumner, and Markus Gross. Differential blending for expressive sketch-based posing. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 155–164. ACM, 2013.

- [78] J.J. Pan, X. Yang, X. Xie, P. Willis, and J.J. Zhang. Automatic rigging for animation characters with 3d silhouette. *Computer Animation and Virtual Worlds*, 20(2-3):121–131, 2009.
- [79] Giuseppe Patane, Michela Spagnuolo, and Bianca Falcidieno. Reeb graph computation based on a minimal contouring. In *Shape Modeling and Applications, 2008. SMI 2008. IEEE International Conference on*, pages 73–82. IEEE, 2008.
- [80] Martin Poirier and Eric Paquette. Rig retargeting for 3d animation. In *Proceedings of Graphics Interface 2009*, pages 103–110. Canadian Information Processing Society, 2009.
- [81] W.H. Press. *Numerical recipes: the art of scientific computing*. Cambridge Univ Pr, 2007.
- [82] Dennie Reniers and Alexandru Telea. Skeleton-based hierarchical shape segmentation. In *Shape Modeling and Applications, 2007. SMI’07. IEEE International Conference on*, pages 179–188. IEEE, 2007.
- [83] T. Rhee, JP Lewis, and U. Neumann. Real-time weighted pose-space deformation on the gpu. In *Computer Graphics Forum*, volume 25, pages 439–448. Wiley Online Library, 2006.
- [84] Taehyun Rhee, Youngkyoo Hwang, James Dokyoon Kim, and Changyeong Kim. Real-time facial animation from live video tracking. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA ’11, pages 215–224, New York, NY, USA, 2011. ACM.
- [85] C. Rose, M.F. Cohen, and B. Bodenheimer. Verbs and adverbs: multidimensional motion interpolation. *Computer Graphics and Applications, IEEE*, 18(5):32–40, sep/oct 1998.
- [86] Andrea Sanna, Fabrizio Lamberti, Gianluca Paravati, and Felipe Domingues Rocha. A kinect-based interface to animate virtual characters. *Journal on Multimodal User Interfaces*, 7(4):269–279, 2013.
- [87] S. Schaefer and C. Yuksel. Example-based skeleton extraction. In *Proceedings of the fifth Eurographics symposium on Geometry processing*, pages 153–162. Eurographics Association, 2007.
- [88] Hyewon Seo, Frederic Cordier, and Nadia Magnenat-Thalmann. Synthesizing animatable body models with parameterized shape modifications. In *SCA ’03: Proceedings of*

- the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 120–125, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.
- [89] J. Seo, Y. Seol, D. Wi, Y. Kim, and J. Noh. Rigging transfer. *Computer Animation and Virtual Worlds*, 21(3-4):375–386, 2010.
 - [90] Jaewoo Seo, Geoffrey Irving, J. P. Lewis, and Junyong Noh. Compression and direct manipulation of complex blendshape models. *ACM Trans. Graph.*, 30(6):164:1–164:10, December 2011.
 - [91] Jonathan Richard Shewchuk. An introduction to the conjugate gradient method without the agonizing pain, 1994.
 - [92] Xiaohan Shi, Kun Zhou, Yiying Tong, Mathieu Desbrun, Hujun Bao, and Baining Guo. Example-based dynamic skinning in real time. *ACM Trans. Graph.*, 27(3):29:1–29:8, August 2008.
 - [93] Philip Shilane, Patrick Min, Michael Kazhdan, and Thomas Funkhouser. The princeton shape benchmark. In *Shape modeling applications, 2004. Proceedings*, pages 167–178. IEEE, 2004.
 - [94] Jamie Shotton, Toby Sharp, Alex Kipman, Andrew Fitzgibbon, Mark Finocchio, Andrew Blake, Mat Cook, and Richard Moore. Real-time human pose recognition in parts from single depth images. *Communications of the ACM*, 56(1):116–124, 2013.
 - [95] K Siddiqi and SM Pizer. Medial representations: Mathematics, algorithms & applications, vol. 37 of comp. *Imaging & Vision series*, 2008.
 - [96] Peter-Pike J. Sloan, Charles F. Rose, III, and Michael F. Cohen. Shape by example. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, I3D ’01, pages 135–143, New York, NY, USA, 2001. ACM.
 - [97] Steven Spielberg. Jurassic park. DVFD, 1993.
 - [98] Robert W. Sumner and Jovan Popović. Deformation transfer for triangle meshes. *ACM Trans. Graph.*, 23(3):399–405, August 2004.
 - [99] Andrea Tagliasacchi, Ibraheem Alhashim, Matt Olson, and Hao Zhang. Mean curvature skeletons. In *Computer Graphics Forum*, volume 31, pages 1735–1744. Wiley Online Library, 2012.

- [100] Marek Teichmann and Seth Teller. Assisted articulation of closed polygonal models. In *Computer Animation and Simulation '98*, pages 87–101. Springer, 1999.
- [101] Julien Tierny, Jean-Philippe Vandeborre, and Mohamed Daoudi. 3d mesh skeleton extraction using topological and geometrical analyses. In *14th Pacific Conference on Computer Graphics and Applications (Pacific Graphics 2006)*, page s1poster, 2006.
- [102] Rodolphe Vaillant, Loïc Barthe, Gaël Guennebaud, Marie-Paule Cani, Damien Rohmer, Brian Wyvill, Olivier Gourmel, Mathias Paulin, LJK-Grenoble Universités-Inria, and CPE Lyon-Inria. Implicit skinning: Real-time skin deformation with contact modeling. *ACM Transactions on Graphics*, 32(4), 2013.
- [103] Oliver van Kaick, Noa Fish, Yanir Kleiman, Shmuel Asafi, and Daniel Cohen-Or. Shape segmentation by approximate convexity analysis. *ACM Transactions on Graphics (TOG)*, 34(1):4, 2014.
- [104] Robert Y. Wang, Kari Pulli, and Jovan Popović. Real-time enveloping with rotational regression. *ACM Trans. Graph.*, 26, July 2007.
- [105] Xiaohuan Corina Wang and Cary Phillips. Multi-weight enveloping: least-squares approximation techniques for skin animation. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 129–138, New York, NY, USA, 2002. ACM.
- [106] Y.S. Wang and T.Y. Lee. Curve-skeleton extraction using iterative least squares optimization. *Visualization and Computer Graphics, IEEE Transactions on*, 14(4):926–936, 2008.
- [107] David S Watkins. *Fundamentals of matrix computations*, volume 64. John Wiley & Sons, 2004.
- [108] Alan H. Watt and Mark Watt. *Advanced animation and rendering techniques : theory and practice*. New York, N.Y. : ACM Press ; Wokingham, England ; Reading, Mass. : Addison-Wesley Pub., 1992., 1992.
- [109] Wazim. Step by step skeletal animation in c++ and opengl, using collada part 2, 2010.
- [110] Eric W Weisstein. Finite difference. 2005.
- [111] Eric W Weisstein. Euler angles. 2009.

- [112] Michael White. Cameron’s ‘Avatar’ opens with \$73 million, 3-D record (update2). http://www.bloomberg.com/apps/news?pid=newsarchive&sid=aBwgFxRW_xkA, December 2009. Accessed: 2014-04-26.
- [113] Xiaosong Yang, Arun Somasekharan, and Jian J. Zhang. Curve skeleton skinning for human and creature characters: Research articles. *Comput. Animat. Virtual Worlds*, 17(3-4):281–292, 2006.
- [114] Sang Min Yoon and H. Graf. Automatic skeleton extraction and splitting of target objects. In *Image Processing (ICIP), 2009 16th IEEE International Conference on*, pages 2421 –2424, November 2009.
- [115] Shin Yoshizawa, Alexander Belyaev, and Hans-Peter Seidel. Skeleton-based variational mesh deformations. *Computer Graphics Forum*, 26(3):255–264, September 2007.