Manifold Learning Techniques for Editing Motion Capture Data

by

Christopher Joseph Dean

A thesis submitted to the
Victoria University of Wellington
in fulfilment of the requirements for the degree of
Master of Science
in Computer Science

Victoria University of Wellington
2016

Supervisory Committee

Primary Supervisor: Dr. John P. Lewis

Victoria University of Wellington
Computer Graphics Department
School of Engineering and Computer Science

Secondary Supervisor: Dr. Taehyun Rhee

Victoria University of Wellington
Computer Graphics Department
School of Engineering and Computer Science

Abstract

Streamlining the process of editing motion capture data and keyframe character animation is a fundamental problem in the animation field. This thesis explores new methods for editing character animation.

In this research, we edit motion capture animations by using a data-driven pose distance as a falloff to interpolate new poses seamlessly into the sequence. This pose distance is the measure given by Green's function of the pose space Laplacian. The falloff shape and timing extent are naturally suited to the skeleton's range of motion, replacing the need for a manually customized falloff spline. This data-driven falloff is somewhat analogous to the difference between a generic spline and the "magic wand" selection in an image editor, but applied to the animation domain. It also supports powerful non-local edit propagation, in which edits are applied to all similar poses in the entire animation sequence.

In the second component of this research, we propose a novel application of non-local means denoising to motion capture data. This new adaptation, called non-local pose means, proves to be an effective method for reducing noise without compromising physical accuracy of motion data.

Acknowledgments

My most sincere thanks go to my primary advisor, Dr. John P. Lewis, whose unparalleled passion, knowledge, and pompousness provided me with the encouragement and enthusiasm I needed to finish this thesis. This research would not have been possible without the guidance and academic rigor of my secondary advisor Dr. Taehyun Rhee. You two have created an outstanding academic environment here at the Victoria Computer Graphics lab—I am proud and honored to have been a part of it.

Special thanks are due to all of my friends, family, and parents for their everlasting love and support.

I would especially like to express my eternal gratitude to my partner Diana, the girl who stole my heart halfway around the world. Thank you for inspiring me, I could not have done this without you.

The motion capture data used in this thesis was obtained from mocap.cs.cmu.edu. The database was created with funding from NSF EIA-0196217.

We would also like to thank the Blender foundation for its contributions of the character Sintel©, from the open-source Durian project, *Sintel* the movie. © Blender Foundation — www.sintel.org

Contents

List of Figures

List of Tables

Glossary of Acronyms

mocap	Motion Capture
dofs	Degrees of Freedom
VFX	Visual Effects
2D	2-Dimensional
3D	3-Dimensional
CG	Computer Graphics
CMU	Carnegie Mellon University
IK	Inverse Kinematics
FK	Forward Kinematics
PCA	Principal Component Analysis
MDS	Multi-dimensional Scaling
DSP	Digital Signal Processing
MRI	Magnetic Resonance Imaging
LPF	Low-pass Filter
FFT	Fast Fourier Transform
SVD	Singular Value Decomposition
GPU	Graphical Processing Unit
NLM	Non-local Means
NLPM	Non-local Pose Means
PSNR	Peak signal to Noise Ratio
MSE	Mean Squared Error

Chapter 1

Introduction

"Animation is the art of motion and art in motion"
–Vibeke Sorensen, *Philosophy Statement, DADA at USC*

1.1 Motivation

Animators mould organic gestures to breathe life and soul into digital characters. Their craft enables us to enjoy an adventure in an imaginative and visually immersive digital world. Like any other VFX department in film and games, character animation is an enormous task, demanding the effort of dozens of artists. According to the Bureau of Labor Statistics, the United States is home to over 64,400 animators who instill life into the characters, animals, and objects of our favorite television stories, films, games, commercials, and other multimedia projects [1].

Animation is a time consuming process—the production of a few seconds of animation can take many weeks, requiring a skill that takes years to acquire [2]. Animation artists are trained in stylistic motion, obeying laws of physical accuracy and artistic form conventions [3] [4].

To save money, time, and effort, many animators turn to motion capture (mocap), a method for recording live movement of actors and converting it into mathematical data [5]. Motion capture can be a very cost-

Figure 1.1: Short film character Sintel animated via motion capture, *model provided by The Blender Foundation and the Durian Project* [6].

effective means of mass producing realistic animations.

Among the titans of animation, Pixar, Inc. once boasted of their talented animators' independence from motion capture, skillfully animating their characters frame by frame. Each meticulously subtle twitch of a character in a Pixar movie was the careful design of an animator [7]. The film VFX company Weta Digital, however, revolutionized animated realism using motion capture technology in blockbuster hits Avatar and The Lord of the Rings [8]. Their engineers and technicians are able to create ultra-realistic human (and non-human) motion through state-of-the-art capture techniques.

Although it could be argued that motion capture cheats by removing an element of artistry from the equation, mocap still requires a great deal of supervision by expert eyes. Motion capture data is extremely dense and heavyweight compared to traditional animation data [7]. A time-consuming component of working with motion capture data is during post-processing. The raw capture data must be cleaned, mended, and error-checked before it can be used for animation [9, 10, 11]. Secondly, if a director imposes creative changes late in production, animators must quickly edit the motion capture data by hand. This makes it a very daunting task for an animator to make extensive changes [2].

The goal of this research is to create a motion capture editing system that allows animators to artistically manipulate mocap animations using

familiar animation techniques. We aim to build a more solid bridge between motion capture animation and conventional digital character animation. To do this, we designed an editing system that is inspired by artist-preferred animation methods, and we improve upon the existing tools for processing mocap data.

1.2 Research Objectives

The objective of this thesis is to create an animation editing tool for editing character movement. To this end, we offer two solutions for the motion capture animation pipeline:

First, the artist needs a means of editing motion capture data easily and swiftly, such that it generates realistic motion, a necessity in film CG. To fit our needs, the animation editing system must meet the following requirements:

- An artist can specify a new pose for a character at a given point in time, and the system intelligently interpolates the surrounding frames.
- The animator should retain artistic control over the timings and positions of the poses.
- The solution should operate at interactive speeds.
- Newly synthesized motion should be stylistically similar to other movement in the motion capture data.

Secondly, we require a method for post-processing faulty motion capture data. This calls for the following system requirements:

- The denoiser should smooth noisy data without removing the subtle detail that makes animation realistic.

- It should clean mocap data without altering the physics of the animation.

- It should handle different types of interference, such as sudden and unwanted impulses, outliers, random noise, and Gaussian noise.
- It should recover missing motion capture marker data.

- It should be intuitive for a seasoned animator, with minimal user parameters and without relying on large motion database libraries.
- Like the editing system, the algorithm should be aware of the stylistic animation behaviour and characteristics.

1.3 Research Methodology

The thesis objectives require the following steps to be completed:

1. Investigate current mocap editing techniques to identify an artist-preferred workflow to emulate in motion capture editing.

2. Investigate manifold learning methods, and consider machine learning for intelligent editing.

3. Investigate state of the art techniques in signal, image, and motion denoising for comparison.

4. Identify a method for interpolating poses using the above findings, which *should* assist in denoising and post-processing motion capture data.

5. Implement a viable and reasonable solution, preferably in a commercial 3D software used by animators in the industry, such as Blender or Maya.

6. Test and compare each method against other related works, and evaluate the pose editing against traditional keyframe-based animation.

1.4 Thesis Structure

This thesis is presented as follows:

Chapter 2 Summarizes the background concepts and mathematics for computerized character animation, motion capture performance, and manifold learning techniques.

Chapter 3 Presents our primary research contribution (forthcoming publication) entitled, "Manifold Pose Distance for Intelligent Motion Capture Editing" [12].

This chapter begins with an introduction to motion capture editing and a presentation of previous works for character animation and manifold learning techniques in motion capture.

We describe our novel method for finding a natural data-driven falloff distance between character poses, along with our process for interpolating poses using this falloff.

We discuss our results for comparing the pose-distance interpolated motion to other manifold techniques and to traditional spline-based methods. Finally, we discuss the implications, limitations, and future work regarding this research.

Chapter 4 Presents the secondary research contribution (forthcoming publication) entitled, "Non-local Pose Means for Motion Capture Noise Reduction" [13]. This research focuses on removing noisy artifacts and signal interference from motion capture data.

After a review of contemporary works in the field, it demonstrates our novel application of non-local means image denoising to motion capture data.

We reveal our results compared to the results of other common denoising methods, and conclude with a summary and discussion of limitations and future work.

Chapter 5 Discusses our holistic approach to designing a motion capture editing system. This chapter details the limitations and future work for our proposed tools in the context of a robust and unified animation editing system.

Chapter 2

Background and Theory

In this chapter, we introduce the mechanics of motion capture and computerized character animation and identify key components for creating an animator-driven motion capture editing tool. We then define the relevant background and theory needed to understand the animation algorithms and animation manifold in Chapter 3.

2.1 Animation

2.1.1 Animating Digital Characters

Digital characters such as Sintel (Figure 1.1) are brought to life by animators using software packages such as Blender3D or Autodesk Maya [14, 15]. Beneath Sintel's digital skin are invisible "bones", analogous to a real human being. Mathematically, a CG bone is a line connected by two 3D coordinates, and these bones are parented to one another in a dependent hierarchy. The animator poses Sintel like a doll, entering x, y, and z angular Euler rotation values for every joint in her body. To animate Sintel, we need to understand the concept of *tweening* [16].

In traditional 2D drawn animation, the primary artists draw the most important shots and images in the scene first. These are the character's

primary action poses, and they occur once every few seconds in the animation. Other artists, called *tweeners*, 'connect the dots' and fill in the remaining motion by drawing the interpolated characters between these initial frames. A majority of hand-drawn animated content is comprised of these 'in-between' shots [16].

The same process is used in computer animation. The artist instructs Sintel to enter a specific pose at a precise time. The artist sets a *keyframe*, constraining Sintel's pose at this time. After multiple keyframes are placed over a few seconds of time, Blender automatically tweens the animation, interpolating the joints between these constraints [16]. The result is an animated figure.

2.1.2 What is motion capture?

In the 1932 Disney classic, *Snow White and the Seven Dwarves*, animators used real-life figures to inspire their animated work. Using a method called rotoscoping, they traced the outlines of live subjects to translate human movements into the characters of *Snow White* [5]. The subject of motion transfer is not new, as artists have been emulating live subjects throughout the entirety of art history. Rotoscoping is a rudimentary form of motion capture, but today we have the assistance of computers and sensing technology to aid computer animation.

Motion capture animation is a means of recording motion from live actors and mathematically applying the same motions to a digital model, like Sintel. There are many techniques for achieving motion capture. Table 2.1 lists the most common means of data capture, among which there are three major categories: active markers, passive markers, and markerless [5]. Active marker systems (Figure 2.2) will place powered electronics (LEDs, accelerometers, etc.) on the motion capture actor, which communicate with the computer or sensing devices. Passive marker systems (Figure 2.1) are less invasive. They are coated with reflective materials, which can

Figure 2.1: Passive infrared markers and infrared motion capture cameras. Image licensed by creative commons [17].

Figure 2.2: Active marker mocap system. Light-emitting markers are tracked by video cameras. Image licensed by creative commons [18].

Type of Mocap Markers		Characteristics
Active	Optical	4-32 Cameras work in conjunction to track the motion of LEDs; the different cameras are registered to solve a 3D orientation.
	Inertial	Gyros, accelerometers, and similar sensors are embedded in the actor's clothing to track physical movement.
	Magnetic	Magnetic transmitters and receivers are planted on the subject and around the studio for magnetic tracking.
	Mechanical	Potentiometers and angular measurement devices embedded in the suit gather joint orientation data.
Passive	Optical	Similar to active opticals, the cameras emit infra-red light and capture reflections from passive infra-red markers on body suits.
	Pattern	Fiducial markers, colors, bar codes, and other patterns can be tracked via camera feed and computer vision algorithms.
Markerless	Depth	Microsoft Kinect [19], Leap Motion [20], and other infrared depth sensors create a 3D image of the scene before them, and use machine learning methods to identify human body parts.
	Video	An active research area in computer vision, this type of tracking is handled on a case-by-case basis. Faces and hands can be recognized using classification.

Table 2.1: Common Types of Motion Capture Sensing [5].

be detected via tracking algorithms from a camera feed. Recently, markerless systems have been popularized for commercial user interaction. The markerless systems are the least accurate tracking alternative, but are often cost effective, non-invasive, and easy to set up [5, 19].

We use skeletal motion capture data provided by Carnegie Mellon University. CMU's passive optical tracking system recorded the movements of actors performing a large number of common human activities, such as running, climbing, jumping, walking, turning, crawling, and sneaking [21].

2.1.3 Defining a Pose

Our digital character model Sintel is posed much like one would pose an action figure. A plastic toy has physical limitations. Her legs should not be able to spin in $360°$, nor should her head or other body parts. The same is true in animation. Artists create intricate skeletal rigs with hundreds of such constraints that prevent the character from entering unrealistic poses.

A skeletal pose may be defined as a list of its joint rotations. Artists commonly designate these angles by Euler rotations: x-, y-, and z-axis rotations that range from $0 - 360°$ [16]. To pose this character, an artist sets Euler rotations for the torso, spine, shoulders, arms, hands, and fingers, sequentially toward the last children in the hierarchy. Such a skeletal configuration is called a Forward-Kinematics rig [16, 22].

FK rigs become difficult to configure when the character needs to place her hand at a specific point in space. Inverse-Kinematics (IK) corrects this problem by posing the child joints first, and then it works out the parents' locations in an inverse order. This is accomplished by solving kinematics equations for a series of constrained joints [16, 22]. The equations find the best joint angles required to reach a desired position. The artist tells the rig the position of the hand, and the computer works out what angles the arm and torso should form to reach that position. In this system, a pose

may be defined as a list of positional IK constraints.

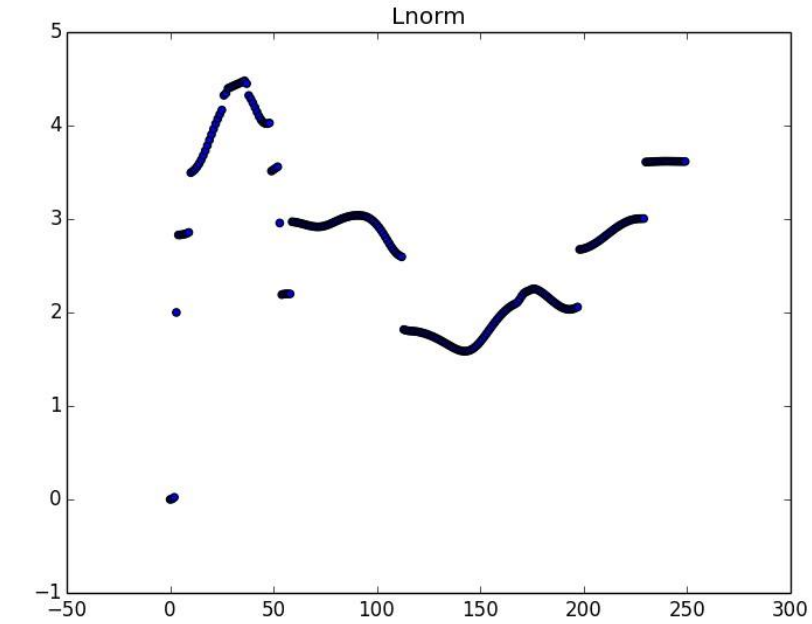
Skeletons are not the only posable structure in animation. An animator often poses facial expressions with numerical sliders. The values of the collection of facial sliders constitute a facial pose. For the purposes of this research, we will define a pose to be the configuration of any pose controls, joints, or abstract manipulators that define a character's position, shape, or orientation [23].

Computationally, it is reasonable to structure the pose as a vector of configuration values. In the case of positional or rotational data, we concatenate the vectors together into one long vector. Take a skeleton with n bones, each having an Euler rotation, (x, y, z). We create a pose vector, $\mathbf{p}$:

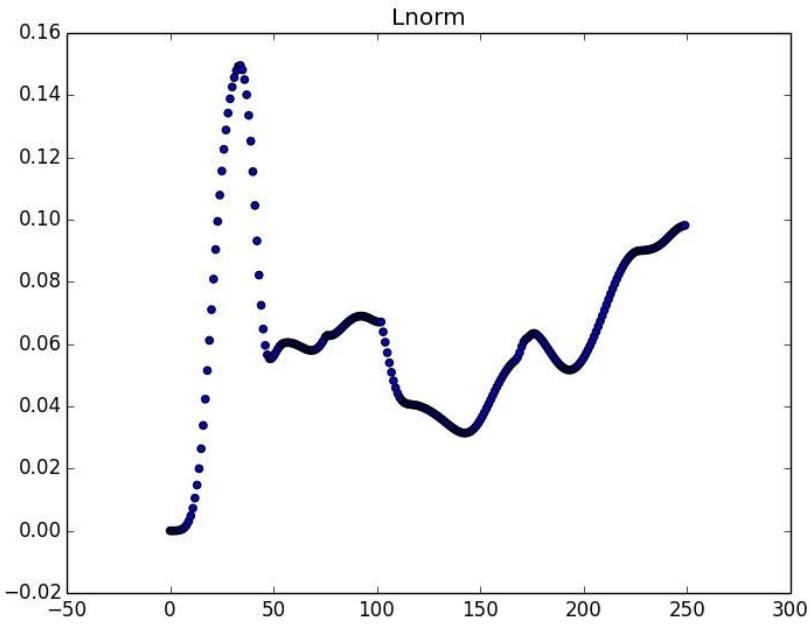
$$\mathbf{P} = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ . \\ . \\ . \\ x_n \\ y_n \\ z_n \end{bmatrix} \tag{2.1}$$

2.1.4 Quaternions

Euler angles are user-friendly, intuitive coordinates for describing rigid rotation. Yaw, pitch, and roll are important when flying an airplane that generally does not rotate beyond $\pi/2$ radians in more than two axes simultaneously. When the y-axis turns $90°$, for example, the rotation gimbal aligns perfectly with one of the other axes. Rotating either axis results in the same rotation, meaning a degree of freedom 'locks up'. This gimbal lock and other angular problems, such as angle flipping, order-dependency, and non-unique interpolations, render Euler rotations computationally



(a)



(b)

Figure 2.3: Figure 2.3a shows the L_2 Euclidean distance between quaternions over time during an animation. Figure 2.3b corrects the gaps by compensating for the multiple quaternion interpretations.

unfeasible for most animation algorithms [24].

Quaternions were invented to mitigate this complexity, formulating rotations such that they could be added, subtracted, and otherwise operated without complication. A rotation (x, y, z) can be transformed into a quaternion of form (w, ai, bj, ck), where complex numbers i, j, k have the property:

$$i^2 = j^2 = k^2 = ijk = -1 \quad [25]. \tag{2.2}$$

Conversion between Euler and quaternion notation is generally implemented as built-in functionality for developers and animators in software such as Blender and Maya. Distance metrics and interpolation for quaternions, however, are less straightforward.

The distance between two vectors, **q** and **r**, is the L_2 norm $||\mathbf{q} - \mathbf{r}||$. This formulation partially works for quaternion vectors (Figure 2.3a), but quaternions redundantly represent rotations in 4-dimensional space. Just as $720° = 360°$ in Euler rotation, multiple quaternions exist for a given rotation at 2π intervals. Rearranging the terms to exploit complex number cancellation, we can correct this issue with an absolute value [26]:

$$\begin{aligned} d(\mathbf{q},\ \mathbf{r}) &= ||\mathbf{q}\text{-}\mathbf{r}|| && (2.3)\\ &= \mathbf{q}^2 + \mathbf{r}^2 - 2\mathbf{qr} \\ &= 1 + 1 - 2|\mathbf{qr}| \\ &= 2(1 - |\mathbf{qr}|). && (2.4) \end{aligned}$$

Figure 2.3b shows the corrected result.

When animating, spherical rotations do not appear natural under linear interpolation. For this reason, Shoemake [27] introduced spherical linear interpolation for quaternions, which interpolates rotations across a sphere at unit-radius. This procedure computes inefficiently, but paved the way for other similar concepts.

One such improvement is exponential mapping. In exponential mapping, we linearly interpolate quaternions on a tangent plane to the

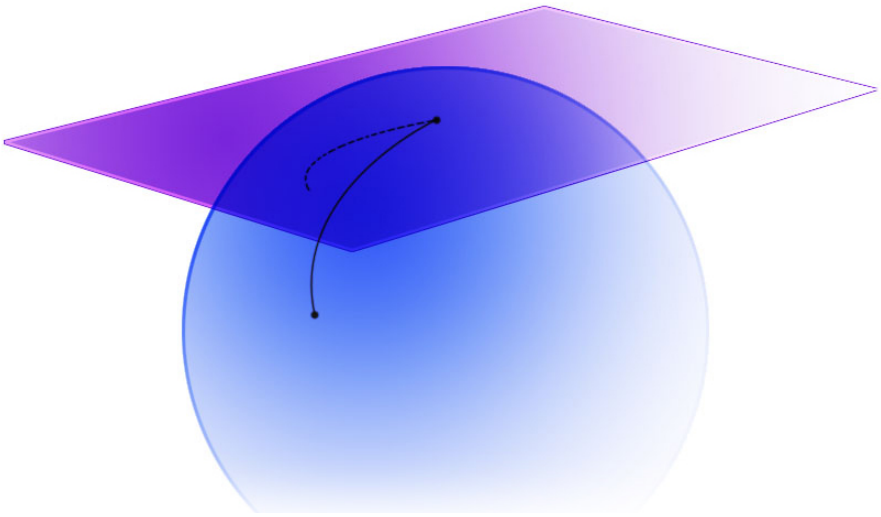


Figure 2.4: This is an example of a non-linear spherical rotation being mapped to a tangent plane. This sort of motion is not easily expressed in linear interpolation for Euler or quaternion rotations.

rotational sphere [28]. Figure 2.4 illustrates the exponential mapping. To enter the tangent plane, we take $log(q)$ for *unit* quaternion q. The inverse mapping, $e^{log(q)}$, returns the vector to quaternion space. Taking advantage of Euler's formula [28], this can be simplified as follows:

Quaternion notation:

$$\begin{aligned} \mathbf{q} &= a + bi + cj + dk \\ &= a + \mathbf{v} \end{aligned} \tag{2.5}$$

Exponential mapping:

$$\begin{aligned} \mathbf{v}' &= \log(\mathbf{q}) \\ &= \cos^{-1}(a)\frac{\mathbf{v}}{||\mathbf{v}||} \end{aligned} \tag{2.6}$$

Inverse mapping:

$$\begin{aligned} \mathbf{q'} &= e^{\mathbf{v'}} \\ &= \cos(||\mathbf{v'}||) + \sin(||\mathbf{v'}||)\frac{\mathbf{v'}}{||\mathbf{v'}||} \end{aligned} \tag{2.7}$$

Exponential mapping is exploited extensively in Chapters 3 and 4, where the pose algorithms take place within this mapping.

2.1.5 Animated Splines

To animate a character between poses, the animation software interpolates each joint position and rotation for all keyframes in the timeline. This interpolation should be smooth unless noted otherwise by an artist. Controls should be made available to sharpen or flatten the "easing" of the movement over time. This is done with splines.

Splines are graph interpolation functions that connect the dots between constrained points. They are piecewise defined by polynomial curve segments, joined together by imperceptibly smooth seams [29]. There are many types of splines, but Bézier curves are the most useful for animation [29]. Pose data points are interpolated smoothly, and the segments are joined together by a flat tangent line. The tangent line is divided in two pieces, and tangent control handles are exposed to the artist.

Figure 2.5 illustrates Bézier curve splines in Blender's motion graph editor. An artist can manipulate the tangent, and therefore the curve, to the left or right of a keyframe. The length of the tangent handle determines the extent of the motion falloff and the angle of the tangent handle determines the falloff shape. The curves represent motion over time, where the slope of these curves represent positional or angular velocity. An animator will manipulate these curves to animate position and angle taking into account a specific velocity, acceleration, and force.

Figure 2.5: A spline motion graph of Bézier curves. These motion graphs depict time *(x-axis)* vs. rotation or position value *(y-axis)*. In this graph, X and Y rotation channels *(red and green lines)* are represented as splines. The keyframes *orange dots* are interpolated via the spline, guided by the tangent controls *(orange circles with handle bar)* for a simple animation created in Blender3D.

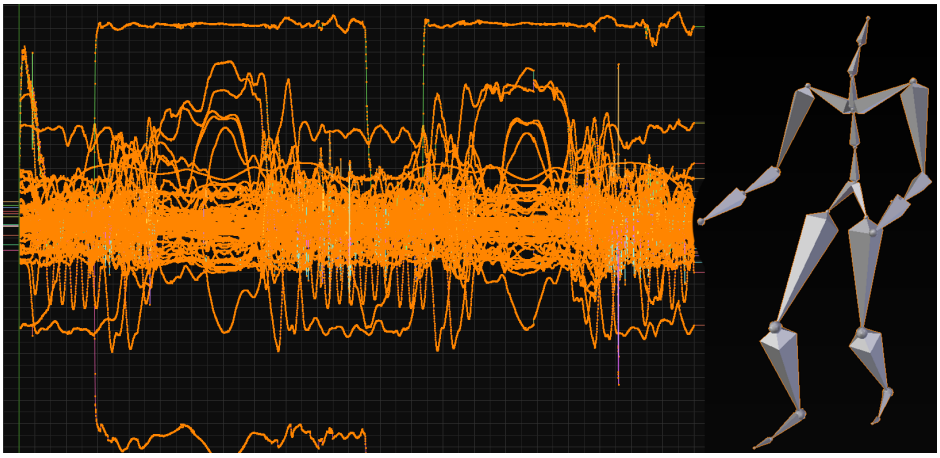
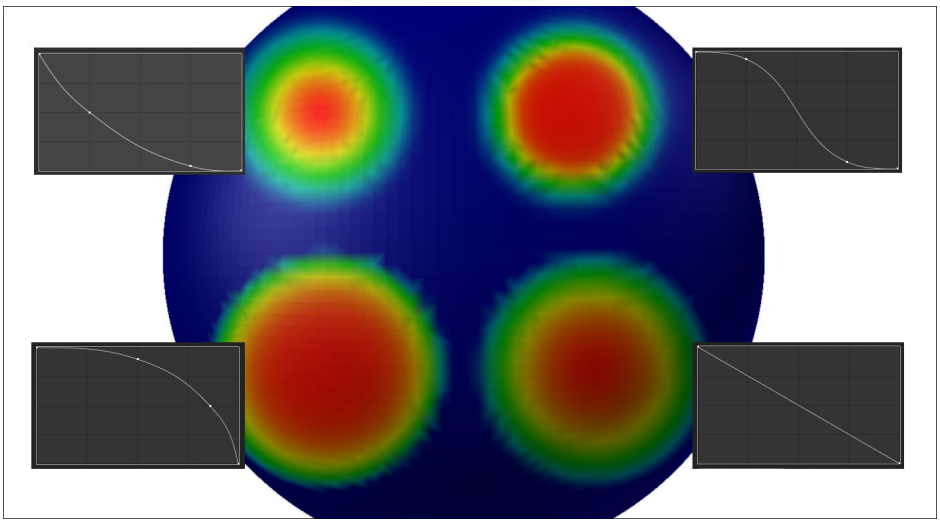


Figure 2.6: Motion Capture data converted into splines in the Blender motion graph. Each spline is associated with a single dof in a skeletal joint. The density and chaos of the keyframes (*orange dots*) highlight the challenge of mocap editing [14].



.

Figure 2.7: This is an example of an artist-defined falloff spline used in modeling and animation. The artist-weight-painting defines the influence an operation has on vertices. An animated deformation may move the central vertex to a new location *(center of red)*. Nearby vertices would be interpolated using these falloff weights *(blue is 0 influence, red is maximum influence)*

It is typical to have upwards 150 degrees of freedom (dofs) in a production-quality skeletal rig in the film industry [23]. A dof is assigned for each unique movement parameterization. For example, a single rigid object can rotate about the yaw, pitch, and roll axes while moving in the x, y, and z directions, yielding 6 dofs [30]. Each dof has its own motion channel spline for animation. The matter is complicated further in motion capture, where each dof is keyframed for every single frame in the timeline. Figure 2.6 shows the scrambled web of splines for a relatively simple rig from the CMU motion library (30 joints $\times$ 3 rotation dofs). A skeleton of n unconstrained bones contains $3(n-1)$ dofs [30]. This poses a challenging problem for designing an interactive editing system.

2.1.6 Falloff Splines

In addition to smoothly interpolating curves, artists use splines to specify falloff weights. Falloff weights are an additive function that interpolate some entity from an activated state to an inert state. In other words, it is a slow on-off switch. The falloff weights range from 0 to 1, with 1 being full activation. These weights determine the influence that a given operation has on an animation, deformation, or some other activity. For example, in facial animation, the face muscles are separated into regions, and expressions are applied to these muscles through falloff weights. Blend shape sliders or splines control the activation of each muscle, and the falloff controls the extent of the effect that muscle's movement has on the skin [31].

Because linear interpolation is unnatural, splines are frequently used to achieve organic falloffs [32]. Figure 2.7 shows falloff weight paintings on a sphere. Four different interpolation schemes are demonstrated, each determined by a falloff spline. The functions blend from red (maximum weight) to blue (minimum weight) as determined by the spline. Professional animators are familiar with spline falloffs, which appear frequently in animation software [33]. For this reason, our editing system seeks to

create a spline-like solution for animators.

Chapter 3 proposes the use of falloff splines for inserting edits into motion capture data. A manifold-based distance is used to calculate these splines. The next section will explain this manifold, as well as similar geodesic distance techniques.

2.2 Manifolds

A pose, **p**, was previously defined as the n-dimensional vector of values describing the configuration of controls that positions a character. Poses are points that lie in very high dimensional spaces and can easily reach vectors of $n > 150$ elements. When a character animates in time, the values of this vector change, forming a line in n-dimensional space.

Humans do not move robotically and cannot perfectly execute actions. It is incredibly difficult to perform an action twice in exactly the same way—each movement will be slightly different [34]. If a mocap actor is recorded for an extended length of time and continuously hits similar poses repeatedly, the line that the vectors make in pose space start to form concentric loops. That is, the lines are close together, but do not overlap perfectly.

If the actor keeps performing the same motions over and over, eventually the points in this high dimensional space become so dense that they form a surface between the lines [34]. This is the animation manifold—a 3D surface in the n-dimensional animation space, representing the range of achievable motion along which our character animates.

2.2.1 The Graph Laplacian

To mathematically approximate the animation manifold, we will use the graph-Laplacian to make a connected network of poses.

For a connected graph of n nodes, the graph Laplacian is expressed as

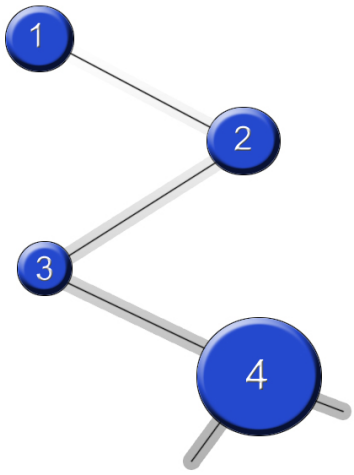


Figure 2.8: An example of a connected graph of nodes.

$$\mathbf{L} = \mathbf{D}_{\boldsymbol{A}} - \boldsymbol{A}, \tag{2.8}$$

where D_A is the diagonal degree matrix, and A is the adjacency matrix. The degree matrix tells us the number of neighbors shared by a given node. So, the ith diagonal entry is the number of connections node i makes in the graph. The adjacency matrix tells us which nodes are neighbors i, placing a value of 1 at the location of each neighbor j in A_{ij}.

The node graph in Figure 2.8 gives rise to a Laplacian of the form

$$\mathbf{A} = \begin{bmatrix} 1 & -1 & 0 & 0 & \dots \\ -1 & 2 & -1 & 0 & \dots \\ 0 & -1 & 2 & -1 & \dots \\ & & \dots & & \end{bmatrix}$$

in which each row sums to 0. In this example, the first point has one neighbor, the second and third points have two neighbors, and so on.

The Laplacian has a number of useful applications, namely as a differential operator in multivariable calculus [35]. The Laplace-Beltrami operator is a more advanced version of the Laplacian that weights the

adjacency matrix elements for structural cohesion. This is useful for situations where the shape of the graph is just as important as the connectivity. It is known that in some circumstances the graph Laplacian converges to the Laplace-Beltrami operator [36]. We will take advantage of this property in the formulation of our manifold for Green's function calculation.

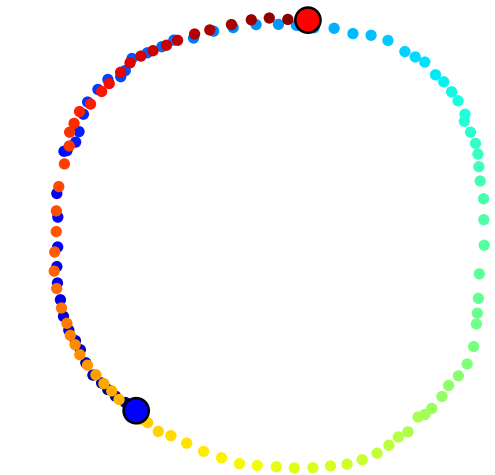
In our work, instead of an adjacency matrix, we construct a normalized probability distribution that describes the likelihood of one pose shifting into its adjacent poses. This connectivity graph is the foundation for a number of mathematics techniques that we can apply to the animation manifold.

2.2.2 Dimensionality Reduction Techniques

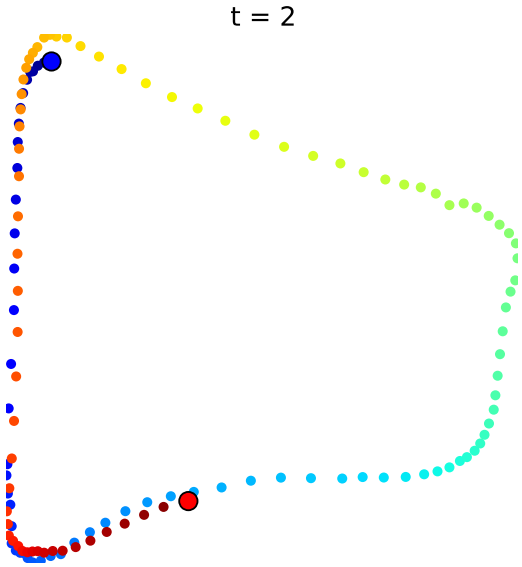
In animation, a character's movements will often comprise a continuous subset of the pose space. Therefore, the animation space is a lower dimensional manifold within the pose space. Because these spaces are high dimensional and unintuitive, this thesis will draw on dimensionality reduction techniques to define the manifold. Editing animations becomes a practice of navigating this lower dimensional manifold and determining geodesic pose distances on its surface.

Dimensionality Reduction is commonly used in computer science to compress high dimensional data or to expose low dimensional patterns that lie in high dimensional spaces. Just as its possible to draw a 1D line or 2D circle in 3D space, high dimensional poses can be part of a simple animation structure with few dimensions. It is useful to visualize animations with these lower dimensional manifolds [37].

Figure 2.9 shows the dimensionality reduction of an animation manifold for a walk cycle. A person who is walking in a repetitive cycle over and over is stuck in a loop; the low dimensional manifold can be seen as a circle or a square. This mapping is done with two different techniques, diffusion mapping and isomapping, each of which identify different char-



(a)



(b)

Figure 2.9: Dimensionality reduction in the embedded Euclidean space. Results from Isomap (a) and diffusion map (b) dimensionality reduction of a run cycle. The large red dot is the first animation frame, the blue dot is the last. The Euclidean distances measured in the two-dimensional plot do not generally reflect distances between poses.

acteristics in the embedded representation. The isomapping sees the repetition as a perfect circle, whereas the diffusion mapping spots distinct moments of change, such as a stomping foot or slight mid-air pause in the gait.

Porte *et al.* [38] describe four major techniques for dimensionality reduction: principal component analysis (PCA), multidimensional scaling (MDS), isometric feature maps (isomap), and diffusion maps.

- PCA is a simple subspace projection that finds a linear mapping from a high dimensional data set into a subspace of principal components that best capture the structure of the original data. PCA fails to capture the non-linear qualities of many real-world data sets, but is useful in machine learning, where PCA expresses trained data as a linear sum of weights [38]. It is also very efficient to add new data, and use PCA to contextualize it in the new subspace.

- MDS finds a lower-dimensional data set that minimizes strain within the data. By minimizing strain, we mean that the pairwise distances between points in the original high dimensional space is retained in the lower dimensional MDS space. Euclidean-distance reliance in MDS provides little information on the global structure of a data set, but it is good at dealing with localized information [38].

- Isomapping is a technique similar to MDS, however it preserves geodesic distances by accounting for graph-based path lengths. Isomap is, therefore, representative of the manifold information of the data set. Unfortunately, it is sensitive to noise—a problem that has profound implications in home-consumer motion capture devices [38].

- Diffusion maps consider the underlying connectivity of a data set and reorganize the set to achieve non-linear dimensionality reduction. Initially, only local information is used in the similarity measure. A time-dependent diffusion process slowly reveals more of the

data. The diffusion time can be interactively changed to encompass more of the manifold structure [38].

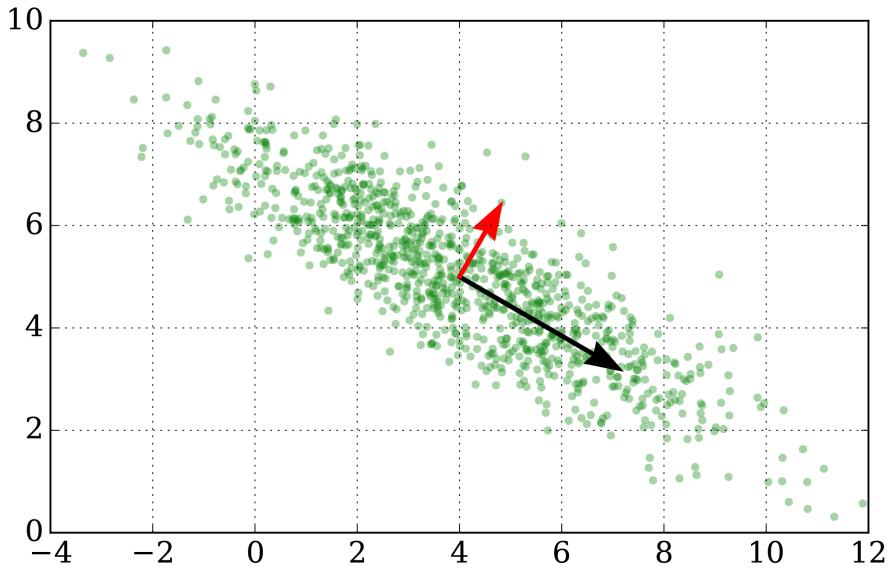


Figure 2.10: Principal Component Analysis finds the two basis vectors that best describe this data set. The graph axes describe the original Euclidean coordinates. The red and black arrows represent the new basis discovered through PCA.

These dimensionality reduction techniques are also useful for describing data in other ways. Figure 2.10 shows a scattered data set, offset from the Euclidean axis coordinate system. It is difficult to mathematically express the primary dimensions of the data. PCA employs a spectral or eigen-decomposition on the data to find a new mathematical basis for describing the information. Figure 2.10 shows the new basis vectors (red and black arrows) gained by transforming the data to a new coordinate frame. In PCA, dimensionality reduction is performed by removing unnecessary dimensions in this way [39].

2.3 Chapter Summary

In this chapter, we discussed the origins and types of motion capture recording techniques. We defined common animation practices, such as keyframe and spline-based character animation. We discovered mathematical formulations for high dimensional character poses as quaternions and methods for interpolating these poses. When enough of these poses accumulate in a data pool, this large collection of poses in animation space forms a surface called a manifold, which can be approximated mathematically using the Laplacian.

Animation space is massive, and motion-capture data adds even more density and bulk to the equation. To make this information more manageable computationally, we employ dimensionality reduction algorithms to view or operate on the data in a low-dimensional space.

The original intent of this thesis research was to apply diffusion mapping to motion capture in a novel manner; in doing so, we discovered another method that offers more value to motion capture editing. Chapter 3 focuses on this method, the Green's function of the Laplacian. We will draw on the dimensionality reduction techniques from Chapter 2 to describe the animation manifold, and we will compare them with our own method for finding geodesic manifold distance (or similarity) between poses.

Chapter 3

Manifold Pose Distance for Intelligent Motion Capture Editing

3.1 Introduction

Creating believable motion for digitally animated characters is an important problem in film and games. Motion capture performances provide detailed and realistic animations, but only after a tremendous amount of effort in post-processing the data.

Suppose, for instance, that an animation director imposes a creative change late in production, and the character needs to step over objects that were not originally present on the motion capture set. An animator would have to re-animate hundreds of steps individually to make the character walk with higher steps. Alternatively, perhaps the actor's skeletal motion needs to be retargeted to a creature with a physiology different to humans. An entire animation database would be off kilter, needing countless changes. These scenarios frequently arise, as creative changes or complications require extensive and repeated changes to a character's entire animation style.

A typical skeletal rig may have 150 degrees of freedom across hundreds of frames. If changes are needed, editing this data on a local or global scale is a daunting task. Re-purposing or reusing these animation sequences is equally burdensome, and the work falls largely on the shoulders of an animator, with assistance from animation algorithms.

Many of these techniques rely on artist-defined splines, interpolating between character poses in the timeline. One difficulty is that splines are not generalizable; a slow-moving bone requires a wide falloff compared to a faster animation. These splines need to be manually defined on a case by case basis—a cumbersome task for the artist. Secondly, splines always have the same smooth shape, and may not appear natural for realistic motion.

We seek to find a natural method for interpolating poses and pose edits in the timeline. To accomplish this, we find a measure for pose similarity. Poses can be represented as high dimensional vectors that describe a character's configuration. In the case of a skeleton, these can be concatenated positional or rotational data vectors. The pose space of this character is the space spanned by all possible configurations of the character controls.

Instead of relying on splines, we propose to let the data "speak for itself" by basing the falloff directly on the change of pose of the underlying motion. A data-driven falloff should have these characteristics:

- It should be proportional to the change in pose and correspond to distance on the manifold of movement.
- it should be smooth if and only if the change in pose is smooth.
- It should be reasonably efficient to compute, thereby allowing interactive edits.

Criterion three prohibits the use of solutions (such as the length of a geodesic on the manifold of poses) that require numerical optimization.

Unfortunately, defining a distance on poses has well known difficulties: Euclidean distances are not appropriate due to the rotational nature

of an articulated body. On the other hand, distances founded on concatenated rotational degrees of freedom (dofs) have the problem that major joints such as the shoulder have a much larger influence than the distal joint of a finger – a metric in joint angle space typically ignores both bone lengths and their connections. This problem can be solved by weighting each dof differently [40], though (depending on the underlying representation of rotations) these weights may need to be recomputed at each pose.

Manifold learning techniques find an embedding of movement in a low (typically two- or three-) dimensional Euclidean space that approximately preserves distances on the manifold. While these methods are ideal for visualizing the abstract character of a motion, they are not immediately helpful for defining falloffs: in general, it is not possible to embed a motion in a 2- or 3-dimensional space without distorting distances and creating a non-injective mapping. Just as well, Euclidean distances in the 2- or 3-dimensional ambient space correctly reflect distances on the manifold only for nearby points (Figure 2.9).

In this thesis we instead start from another idea popularized in the manifold learning community, specifically, that the graph Laplacian captures the intrinsic geometry of a manifold (in this case the manifold of body movement). With this in mind, we show that the Green's function of the Laplacian provides a natural similarity measure between poses.

Our data-driven falloff is somewhat analogous to the difference between a generic spline and the intelligent region selection available in popular image editing software such as Photoshop (e.g. algorithms such as [41, 42]), but applied to the animation domain. The result is intelligent interpolation that blends new poses into the existing motion curves, with the advantage of a thresholding dial for integrating the edits. For cyclical animations, the data-driven falloff can be recycled, with capabilities for powerful non-local edit propagation.

The remainder of this chapter compares several distance metrics for use in pose interpolation. Other dimensionality reduction techniques have

been employed for this purpose [43, 44]. However, this chapter will explain the improvements to motion capture editing gained by our novel contribution, a natural animation falloff distance that is given by Green's function of the pose space Laplacian.

3.2 Background and Related Work

3.2.1 Motion Editing

The simplest form of editing motion is blending a pose or animation sequence into nearby frames. Mocap data is converted to motion splines, where the animator can edit poses and blend the curves together with custom falloffs. Spline blending can result in awkward pose transitions with bad timing. This problem is often treated as a signal processing problem [45] or a constrained optimization problem.

Witkin and Kass [46] initially proposed spacetime constraints for keyframed animation editing. Their system takes spacetime constraints (e.g. foot placement, jump height, energy, timed targets) and finds the optimal solution for physically accurate animation, while animators tweak these parameters to their liking. Gleicher [47] introduced registration curves, which blend motion curves using time warping, local coordinate frames, and motion constraints. These blending, constraint, and time warping techniques have since been applied to motion capture editing [47, 48]. Gleicher [49] provides a useful comparison of constraint-based techniques in animation.

This paper discusses a technique to enhance the artist's traditional workflow, without resorting to an automated simulation. We do not make use of animation constraints in this paper, but their relevance is significant in the subject of motion editing.

3.2.2 Dimensionality Reduction for Motion Editing

In character animation, dimensionality reduction has been applied to an array of various applications. Tournier *et al.* [50] used dimensionality reduction in motion capture to compress and reconstruct motion graph signals to make high-dimensional mocap data structures more manageable. Pose space data tends to be high-dimensional, sparse, and non-linear, so

dimensionality reduction is a common tool for making such information more manageable.

Barbic *et al.* [51] apply low-dimensional motion and probabilistic PCA to segment motion capture databases into classifiable activities (e.g. drinking, sitting, walking). Lee *et al.* [52] take advantage of this classification technique to train behaviours for an AI-controlled avatar. They preprocessed motion data with PCA to emphasize similarities between motions, where a first order Markov process helps the AI decide which motion path to follow. Glardon *et al.* [39] use PCA to synthesize and interpolate new walk patterns from trained mocap data. Safanova *et al.* [53] adopts PCA classification to procedurally synthesize new motions between artist-configured poses. Their solution is an optimization problem over the reduced-dimension animation space using animation constraints, inverse kinematics, and data from similar behaviors in the low-dimensional space.

Seward *et al.* [43] suggest using a lower-dimensional manifold to resequence and correct animated character motions. They construct a temporal-spatial MDS distance metric that is sensitive to the orientation and timing of the joints. The authors make insightful developments for motion timing in MDS, but they found that the temporal preservation was often extraneous. As such, our focus will remain solely on the spatial aspect of our data set.

Seward *et al.* assess the L_2 distance metric with the Lee distance—a distance metric derived from a first-order Markov process—but found no conclusive results [43, 52]. The Lee distance shares similarities with our method, in which the random-walk Laplacian is a probability of following a given connected path in the data set.

Shin *et al.* [54] compare PCA, MDS, and isomaps for categorizing motion capture data. They apply these techniques to data visualization and motion editing. In their system, the user edits motion directly in the low-dimensional space through path sketching. However, sketching in the

manifold space limits artistic control over the poses.

Given these comparisons of the dimensionality reduction methods, we have chosen isomapping and diffusion mapping for studying our results. Isomapping performs equally to or better than PCA and MDS in the examined literature [43, 44, 54]. We will also compare Diffusion Mapping due to its manifold-awareness and intrinsic similarity to our method.

Most of these related works focus on heavily automated animation synthesis or constrained animation methods. Our paper draws on the same resources in dimensionality reduction, but offer the artist a tool for hand-crafted control over their animations. It is designed to fit into a character animator's traditional workflow of posing and keyframing.

3.3 Method

3.3.1 The Pose Laplacian

We must first represent the skeletal rig as a vector. For an articulated skeleton, we have chosen a quaternion representation.

Let x be a concatenated vector of all quaternion joints for a single pose. To calculate the naive distance between poses, we use the L_2 quaternion norm from Equation 2.4.

We define a connectivity matrix, K, to relate close poses to one another. We use a Gaussian kernel to filter out distant poses,

$$K_{i,j} = e^{\frac{d(x_i,x_j)}{2\sigma^2}}, \tag{3.1}$$

where σ is a user-defined constant that adjusts the width of the Gaussian. At this point, we have a matrix that defines each pose and its closest neighbors.

In a motion capture skeleton, poses have a higher probability to shift into a slightly varied pose rather than a vastly different one. A person's leg, for example, is not likely to suddenly invert itself. The animation manifold is as smooth and continuous as the captured human motion. Therefore, we construct a random-walk Laplacian to utilize this feature of probabilistic pose-connectivity and define the animation manifold.

To create the Laplacian, let n be a vector of constants which row-normalize K. We then diagonalize n into $\mathbf{D}$ Obtain the adjacency matrix A and degree matrix D_A by

$$\mathbf{A} = \mathbf{D}^{-1/2}\mathbf{K}\mathbf{D}^{-1/2}, \tag{3.2}$$

$$\mathbf{D}_A = \text{diag}(\text{rowSum}(\mathbf{A})). \tag{3.3}$$

This produces the Laplacian,

$$\mathbf{L} = \mathbf{D}_{\boldsymbol{A}} - \boldsymbol{A}. \tag{3.4}$$

Now that we have information about the animation manifold, we can use techniques from spectral analysis to parameterize the space. To this end, we obtain the eigenvalues and eigenvectors λ_k, ϕ_k of $\mathbf{L}$ through eigen-decomposition.

3.3.2 Green's function

The Green's function of the Laplacian acts as a "generalized covariance" and provides a natural measure of similarity between poses. In Appendix A there is a detailed overview of Green's function and its use in related fields. This thesis exploits its utility as a distance measure below. As an intuitive justification, note that in a setting with regular samples such as an image, the Laplacian matrix is the approximate inverse of a commonly used covariance function (see Appendix A). Since it is defined in terms of the eigenvectors of the Laplacian, it reflects the geometry of the manifold itself rather than the embedding space (assuming the discrete approximation is adequate).

Methods like isomapping or diffusion mapping will approximate the manifold surface, on which we can measure geodesic distances. For a falloff, we don't want a strict distance measure, but rather a weighting that emphasizes similar poses. For the similarity measure, we turn to the Green's function (see Appendix A for Green's function as a similarity measure.)

To obtain the manifold pose distance between poses x and y, apply Green's function,

$$d(x, y) = \sum_{k>0} \frac{\phi_k(x_i)\phi_k(x_j)}{\lambda_k}. \tag{3.5}$$

We form a matrix P of all pose distances,

$$P_{ij} = d(x_i, x_j). \tag{3.6}$$

3.3.3 Edited Pose Interpolation

Here we illustrate a simple method for applying the Green's function distance from equation (3.5) as an animation falloff. As mentioned previously, animators commonly use spline falloffs to interpolate poses in keyframe animation. We define a falloff weight ranging from 0 to 1, that effectively shifts a pose edit from *off* to *on*.

For a pose, p_i, we wish to make a change and edit the animation to target a new pose, p'_i. The difference between these poses is the exponential mapping of the delta-pose dp, given by

$$dp_i = \log(p'_i) - \log(p_i). \tag{3.7}$$

Points of inflection in the rows of the distance matrix,P_i, highlight transitions in skeletal behavior. We scale P_i between 0 and 1 across the edited region of the timeline. That is, the pose distance must be 0 at the edited pose $P_{i=j}$, and 1 at a sufficiently distant motion transition. Doing so enforces a smoothly tapered timing extent, but an artist may want interactive control over this scaling. For each pose in the edited sequence, we find a weight w_{ij}:

$$w_{ij} = \min(1 - P_{ij}, 0). \tag{3.8}$$

For j frames in the editing sequence, apply the weighted delta-pose:

$$p'_j = \exp(\log(p_j) + w_{ij} dp_i). \tag{3.9}$$

At p_i, the edited location, the weight is at a maximum and fully activates the delta-pose. Further down the timeline, the weight tapers off, eventually deactivating the delta-pose at 0. Once this is baked into the animation data, the animated result is our edited output.

3.4 Results

We compare the Green's function distance metric to two other manifold learning techniques: isomapping and diffusion mapping. Diffusion mapping has not been historically applied to animation, but yields results of comparable quality to MDS and isomaps. Diffusion mapping features an added benefit of interactively viewing the manifold at various diffusion times [38].

In these experiments, we process mocap data from the Carnegie Mellon University motion capture database. We have chosen to examine animated run cycles due to their comprehensible and periodic characteristics.

3.4.1 Dimensionality Reduction

Dimensionality reduction has been noted as an excellent visualization tool for animation manifolds [54]. To understand the behaviors of our distance evaluation, it is useful to compare the various manifold learning techniques.

Figure 2.9a shows a run-cycle embedded in a lower dimensional space. Human movement never perfectly repeats itself, but recurs closely enough that the motion is organized into similar regions of the embedded space.

The diffusion mapping shape Figure 2.9b features bulges and corners, revealing an emphasis on the local transitions in motion behavior. Here the four corners represent the four key poses in the run cycle: step, transition, and their mirrors.

3.4.2 Pose Distance

Using the normalized pose distance calculated from Equation 3.5, Figure 3.1 shows the distance from a control pose to every other pose in a 130-frame animation sequence. This is periodic motion, as an actor steps right-left-right-left and so on. Small dents, bends, and plateaus indicate the

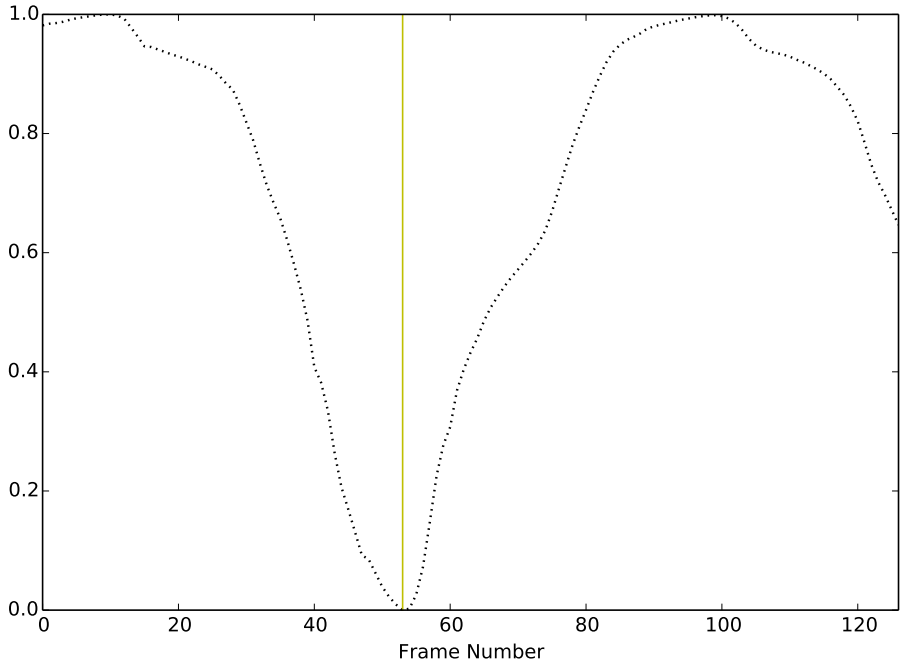


Figure 3.1: Green's function distance from pose 53 in a walk cycle. Compares distance against all poses in the timeline. We see that the pose distance is zero at the current frame *(yellow bar)*, as the pose is compared to itself.

prominent characteristics of the skeletal motion.

In figure Figure 3.2, we see the pose distances for a non-periodic motion. This actor abruptly turns around in the middle of a walk. There appear to be points of transition in the motion, but the beginning and end of a falloff is not well defined from this distance measure. Inflection points and bends serve as good guidelines for falloff boundaries, but an artist may want customized control over the falloff scaling for blending their edit into this motion.

Poses for diffusion mapping and isomapping lie in an embedded space, in which the L_2 norm returns a manifold-preserving distance metric. Figure 3.3a depicts an overlaid comparison of all three distance metrics. Overall, the three methods detect the same animation characteristics and reveal

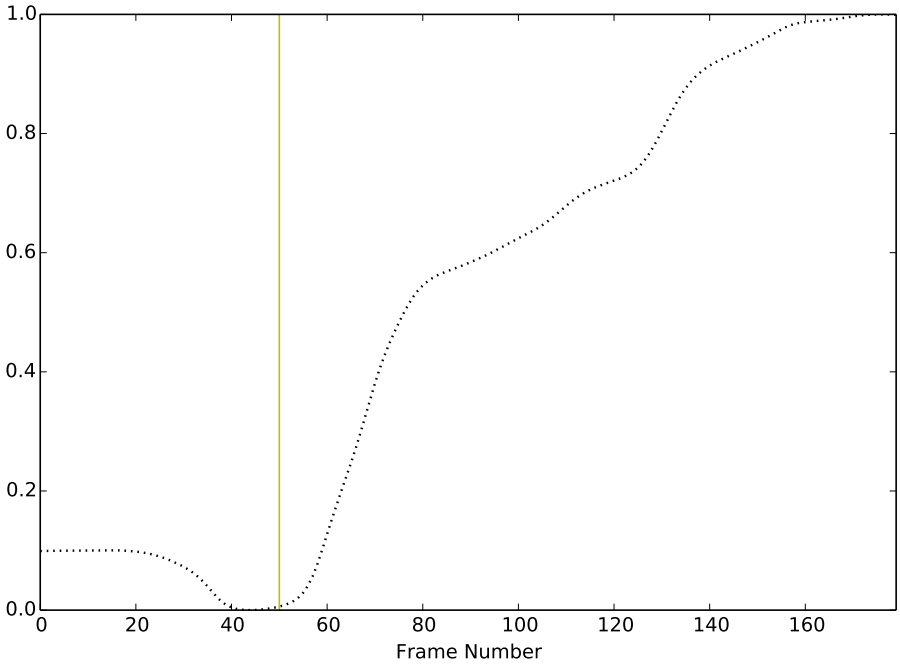


Figure 3.2: Green's function pose distance for a different animation. In this animation, a character is turning around mid-walk. It is non-periodic, so there are multiple phases to the action (indicated by bends): walk, stop, turn, and walk. The falloff near the control pose (yellow bar) breaks up the local movement into a smooth curve.

periodic motion behaviors across the entire timeline.

Figure 3.3b takes a closer look at the distance falloff near the edited pose. Zooming in, the local pose information is very rounded for Green's function, somewhat tapered in the diffusion map, and sharply cornered in the isomap. When animating the sharp isomap falloff, one will note a quick snap into and out of the edited pose. This effect is noted in the accompanying video results. A smooth falloff causes the animation to ease gently through the targeted pose, demonstrating the effectiveness of Green's function for animation interpolation.

3.4.3 Edited Animation

Animation characteristics can be inferred from the pose distance data. A heatmap of the Green's function distance matrix (equation 3.6) is given in Figure 3.4. This map visualizes the contours of the animation manifold. The main diagonal is constant, as $||p_{i=j}|| = 0$.

Along the main diagonal, the color of the band reveals the local falloff curve. The width of the band gives us insight into motion characteristics. For example, areas of constant thickness (frames 35, 60) indicate unchanging poses, which briefly happens between steps frozen in midair. At frame 100, a pinch occurs from the jolt experienced by planting a foot heavily on the ground. The falloff properly reflects this jolt, and edited poses will carry a similar feature. Noise at frame 19 distorts the graph, but only at that localized region. An edited pose will accumulate the same motion feature.

The periodic run-cycle shows its sinusoidal patterns along the x-axis, suggesting the periodic running motion. This repetition is what allows for propagated edits to similar poses throughout the timeline.

Figure 3.5 demonstrates the edit propagation in practice. For repetitive actions, a single edited pose and falloff animation can propagate to all instances of similar poses in the timeline. In Figure 3.5, the artist wishes for the character to lift their knees while running. With one simple edit, the entire run-cycle is changed.

The efficacy of the animated falloff can be observed in the accompanying results video. As seen in Figure 3.3b, the pose edits for Green's function are smoothly eased, and the edit adheres to the local manifold. However, large edits or physics-altering edits introduce inaccuracies in the animation. For example, editing a foot that is planted on the ground will produce the famous 'skating foot' if not treated as a constrained problem. (The skating effect is just as one would expect, the sliding of body parts on contact surfaces, such as the ground or hand-holds.)

As seen in Figure 3.6, the Green's function distance changes

significantly on a global scale for different values of σ. Local information near the edited frame is more stable, but a slight offset of 2-3 frames is sometimes introduced for bad values. σ should be chosen based on density of poses in the edited pose space. This parameter, as well as the falloff shape, should be exposed to the artist for tweaking.

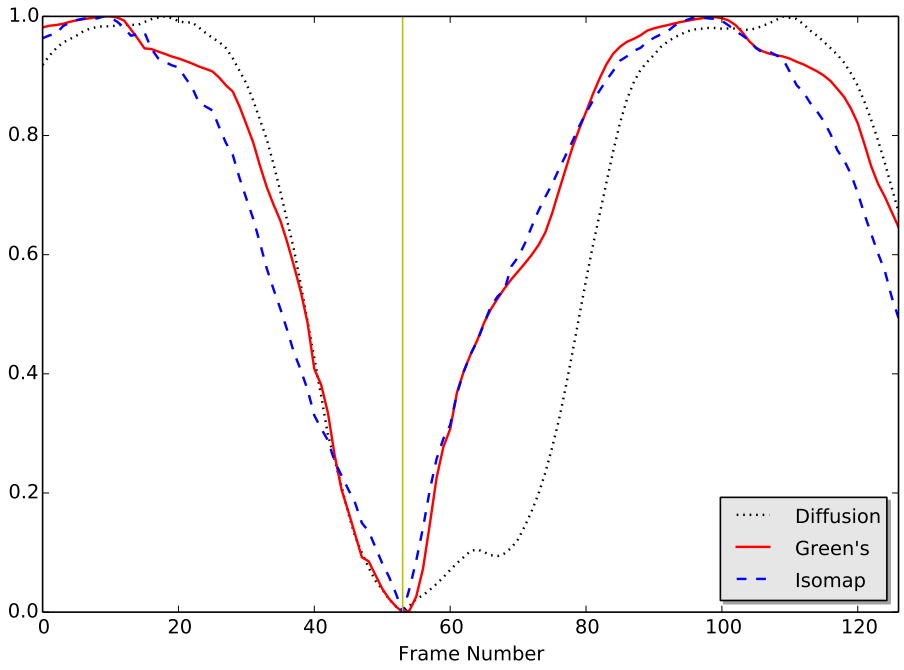
3.4.4 Evaluation

This pose falloff technique is designed for an artist using spline keyframes to animate a character, so we will compare our results to this method. In spline animation, the motion of a single joint will be presented in a graph of motion channels, one for each degree of freedom in the joint. After moving the joint to a new position, the spline is recalculated to incorporate the new data point, and the new falloff is used to interpolate the in-between poses.

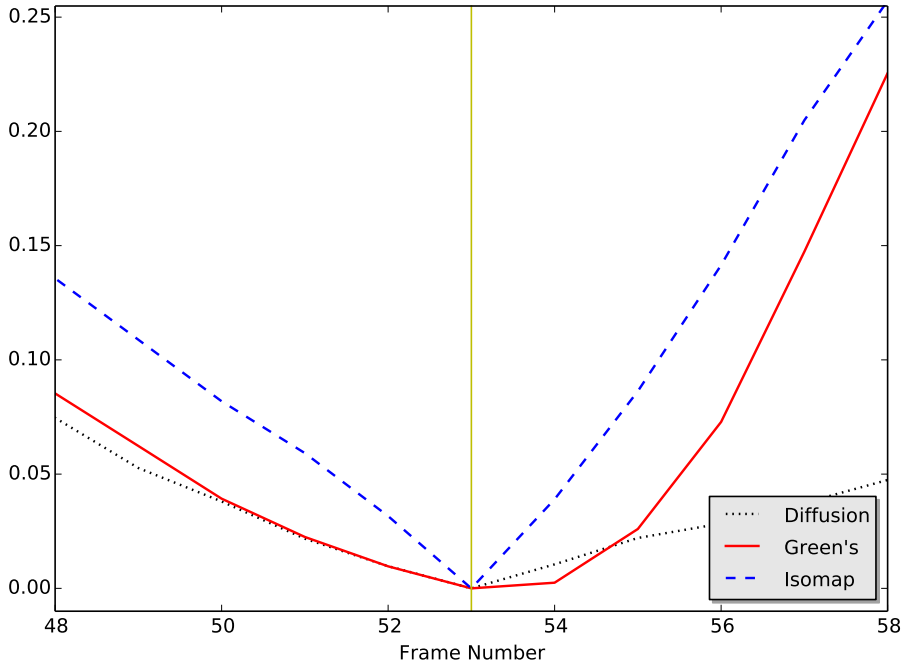
In Figure 3.7 we look at the vertical movement channel of the ankle in a walk cycle. In the first animation style, five keyframes were used to create the motion on an inverse-kinematics foot. In the second style, the animation was baked into a forward-kinematics motion capture format, with 40 keyframes (1 per frame). The two animations were each edited, forcing the character to lift her feet higher as she walks.

Figure 3.7 depicts four different motion graphs: the original keyframe animation, keyframe animation with an edit to raise the knees in the walk cycle, the baked 'mocap' version of the animation, and the Green's function falloff-edited motion.

Unsurprisingly, the original animations are nearly identical, with some minor detail loss attributed to the baking from an intricate IK rig to a simplified FK rig. For the edited animations, the spline falloff matches the Green's function falloff in both shape and timing extent. Editing a single pose of mocap data with Green's function falloff produces the same effect as editing a pose in a spline animation.



(a)



(b)

Figure 3.3: Comparison of Green's distance vs. diffusion distance (10 iterations) vs. isomapped distance for frame 53 in a run cycle. Figure 3.3b is a zoomed-in comparison of Figure 3.3a near the control pose. Note the smoothness of Green's function at this point.

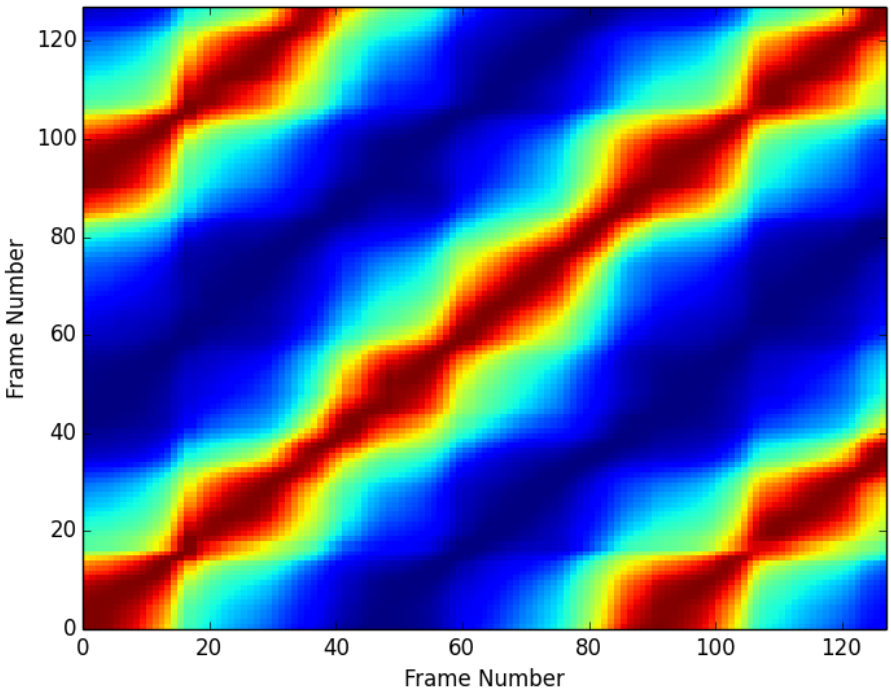


Figure 3.4: This map is pose distance vs. pose distance for all animation frames, describing the 3d surface from which Figure 3.1 is a single slice. Dark red indicates a minimum distance.

Figure 3.5: An artist edits the pose *only* at frame 20 *(left)*, lifting the knees higher. A propagated edit occurs automatically at frame 110 *(right)*, which is a similar pose in the run-cycle.

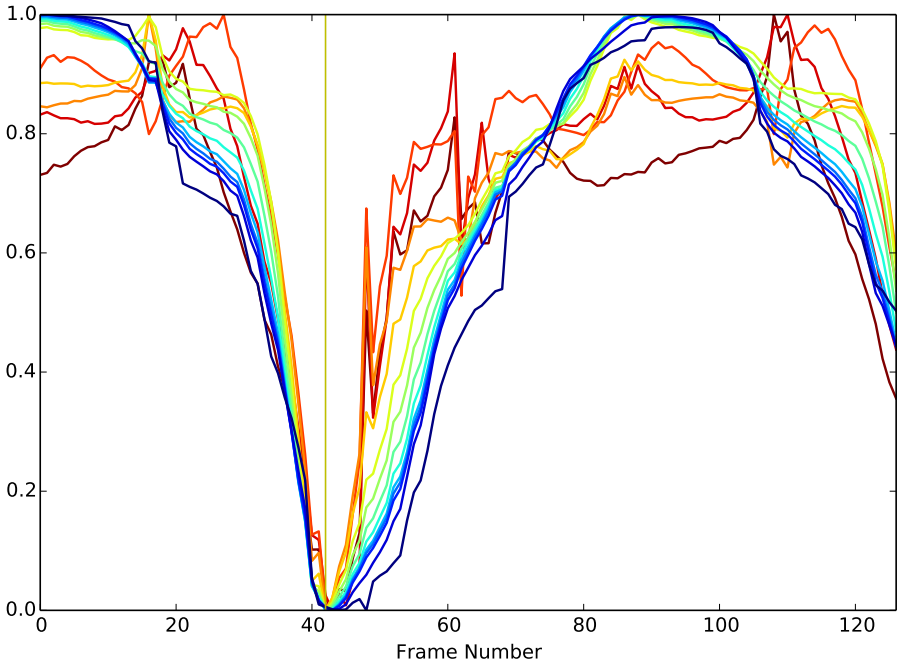


Figure 3.6: The Green's function distance from a control pose over several different sigma values (lower values are more blue). There are a wide range of choices for an artist, that may highlight different motion features.

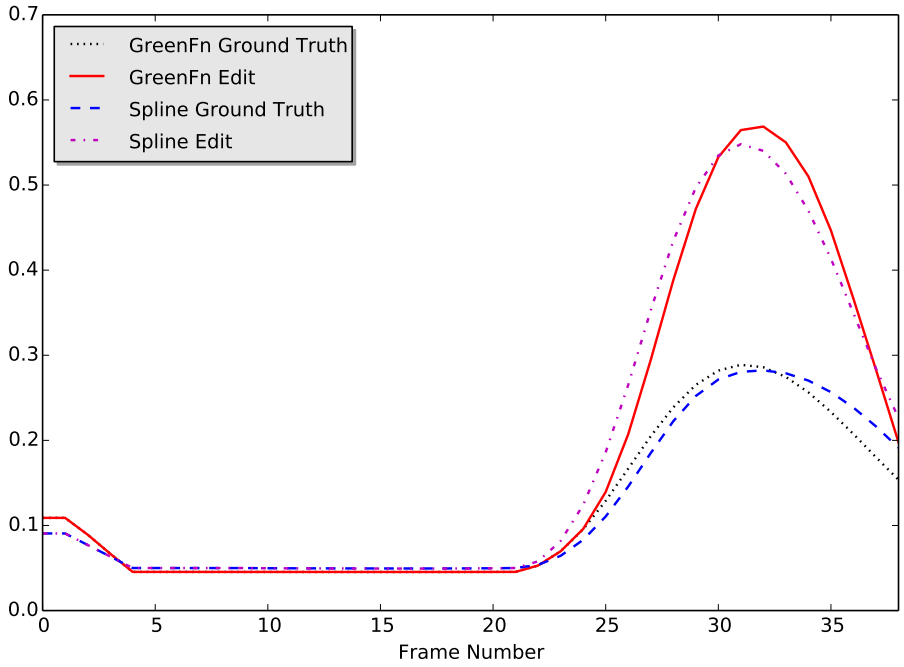


Figure 3.7: Motion Graphs: Vertical Position vs. time. This is the z motion channel of a foot in a keyframe-animated walk-cycle. A keyframe animation was created, and then edited to step higher. The original animation was converted to a motion capture format, where a similar edit was made using our method.

3.5 Chapter Summary

In this chapter we have introduced the Green's function distance for the animation manifold. We used this distance as a natural falloff for resequencing pose positions in a motion capture performance animation. Until now, this problem has been either solved tediously by hand with splines, or automated through physical simulation solvers. AI and constraint solutions are available through dimensionality reduction and manifold training, but such solvers remove the precision control that handcrafted artistry offers.

Our method proved to yield better falloff curves than diffusion mapping and isomapping, whose dimensionality reduction techniques have an established history in editing motion capture animation. Our method can be thought of as a data-driven mocap spline, treated to operate in the animator's usual workflow. We baked a keyframed spline-animated sequence into a motion capture format and tested the Green's function falloff directly against the spline method. The results were nearly identical, proving the accuracy of Green's function falloffs as motion capture splines.

Chapter 4

Non-local Pose Means for Motion Capture Noise Reduction

4.1 Introduction

4.1.1 What is noise?

Computer vision is prone to error. The algorithms and hardware are honed by engineers to precision detail, but there always exists a slight inaccuracy in even the most sophisticated of measurement devices. What appear to be smoothly executed gestures on the motion capture stage will actually contain small bumps, jitters, and outliers once measured. To eliminate this unwanted *noise* in our data, we turn to an area of research aptly named noise reduction, or *denoising*.

The predominant sources of noise in motion capture vary from system to system, but are often attributed to intrinsic hardware faults or physical and real-world limitations of the tracking setup. Two major sources of noise are observed in motion capture sensing:

Gaussian noise: This is a signal-independent noise that features a predictable normal distribution. A typical source of this noise is environmental, thermal, or electronic fluctuations in the sensor [55]. Gaussian noise

adds a jittery shake to every joint in the skeletal motion. One must be careful to not confuse Gaussian noise with high frequency motion detail, such as the tremors of a clenched fist.

Impulse noise: Sometimes called salt and pepper noise for its visual appearance in imagery, this can be caused by dust or mistracked markers [56]. This creates outlier joint orientations and discontinuities in otherwise smooth motion data.

Impulse noise is less predictable—automated processes cannot easily separate this systematic anomaly from the true motion data. Take, for instance, skeletal motion tracking. The body's underlying bone structure is rigid, but we cannot track bones directly—we must place markers on the skin and estimate the location of the bone. Deformation in the body tissue causes the surface markers to displace relative to the bone orientation.

In a mocap body suit with non-rigid velcro markers loosely attached [57], sharp movements may jostle and dislodge mocap markers out of place. Other frequent marker issues include marker occlusion, marker flipping, or false-positives (i.e. from shiny or reflective objects in an optical tracking space). These marker problems will cause outliers or missing data an animation sequence.

Some tracking solutions, such as the Microsoft Kinect [19] markerless skeletal tracking API, can estimate the underlying rigid structure of the skeleton. However, these are only estimates, exhibiting some degree of solver error from unknowns in the sensing data. This data is so volumous that data compression techniques are frequently employed for easy transferal or processing. When data is lost during the compression and decompression stages, it often resurfaces as noise or missing information [58]. Various motion tracking sensors will encounter different types of sensor-dependent noise, including Poisson noise, film grain, anisotropic noise, or uniform noise [56].

These are fundamental problems in computer vision, but the enormous variety of available motion tracking systems and noise conditions place

these concerns well beyond the scope of this thesis. This research aims to correct motion capture data itself, assuming the end user has no control over the recording environment.

For testing our results, we gather data provided by the Carnegie Mellon University motion capture database [21]. In this optical tracking system, the prevalent noise conditions are Gaussian and impulsive, while non-Gaussian sensor noise will contribute a reasonably small random error to the final motion tracking solution [59].

4.1.2 Processing Noise

Human motion denoising is difficult because the body's motion is articulated into a high number of degrees of freedom, with a hierarchical structure and detailed time-sensitive movements [60]. Each of these factors is challenging in their own right; when coupled together, slight changes in the data will create an implausibly unrealistic motion.

In basic image denoising techniques, such as a Gaussian low-pass filter, a blurring effect occurs to smooth over the noise details in an image. Blurring in the time-domain, however, means that the position of body parts change with respect to time. This is bad, and the blurring will alter the timing—and therefore the physics—of our motion. This physical shift will make a walking character appear as if they are slipping, sliding, or skating—introducing the classic 'skating foot' problem [61].

State of the art solutions in image denoising have made an imprint on motion denoising in recent decades [60, 61]. These methods adapt trends in machine learning techniques to train motion library databases for example-based denoising systems. These methods have proven to be extremely effective denoisers, but are not user-friendly. Additionally, extensive motion libraries are not readily accessible to most animators using commercial animation software, which are typically bundled with lightweight signal processing tools.

4.1.3 Research Method

Reiterating our objectives from the thesis introduction, this research aims to find a noise reduction solution that:

- preserves motion details with little blurring or skating,
- requires no external training data and relies only on the active animation,
- works at interactive computation speeds,
- reliably reduces noise for a variety of noisy conditions: uniform noise, gaussian noise, spikes,
- outperforms other denoising filters,
- and incorporates manifold learning, whereby the filter is aware of the pose space of our animation.

Most denoising techniques characteristically fit into one of two categories, using either local or non-local information from the noisy manifold. Local denoising techniques will compare the noisy sample to its spatially nearest neighbors. A non-local filter extends this comparison to any samples in the entire manifold—as long as they are similar signals (or in our case, poses) [61].

To fulfill our denoising objectives, this research proposes the novel adaptation of the non-local means (NLM or NLMeans) image denoising algorithm for motion capture noise reduction. Non-local means is an image processing technique that compares the averages of distant and local neighborhoods for data smoothing [62]. It smooths textures using this non-local information, and is able to preserve sharp edges. This detail preservation is highly important for motion capture—it will enforce spatio-temporal cohesion in the animation, preserving the 'texture' of the motion curves (a.k.a. high frequency motion details).

Our research contribution is the adaptation of NLMeans to mocap data, in which we average weighted pose-vectors using similar non-local features from the animation space. This novel application of non-local pose means (NLPM) retains the strengths of its image-based predecessor, providing a strong denoising method that does not rely on large motion databases.

In the following sections, we will identify related works in image and motion denoising, explain the non-local pose means method, and compare the NLPM results to standard benchmarks in this research domain.

4.2 Background and Related Work

4.2.1 Image Denoising Filters

Signal interference can appear in any measurement, so it is an area of interest in many academic fields, e.g. bio-medicine, astronomy, computer vision, audio engineering, radiometry, chemistry, or any field which analyzes empirically gathered numerical information. In Astronomy, for example, what once was believed to be static background noise in measurements of cosmic radiation actually contain information about the formation of our universe [63]. In medicine, a speck in an MRI biometric scan of the brain is important information for a brain surgeon, where the difference between a physical anomaly and magnetic field noise is crucial for patient diagnosis [64].

Consumer-grade computer sensing devices are recently popularized by low-cost products like the Microsoft Kinect [19], Thalmic Labs' Myo[1], and the Leap Motion [20] revolutionizing the way we interact with technology in our day-to-day lives. This increase in technological availability has garnered a large amount of attraction to the denoising problem [65]. Denoising is a is a well researched field, but its techniques are still being applied in new ways to pave the road for these exciting new computer vision technologies.

Of the many image denoising options available, the Gaussian filter ranks among the simplest and most commonly cited. The Gaussian low-pass image filter is expressed as:

$$G(x,y) = \frac{1}{\sqrt{2\pi\sigma}} e^{\left(\frac{-(x^2+y^2)}{2\sigma^2}\right)}, \tag{4.1}$$

where σ is standard deviation of the Gaussian curve, σ^2 describes the variance, and the Gaussian is situated in a square 2D kernel of size l. A Gaussian curve is infinite in size, so this cutoff point is user defined—by

[1] https://www.myo.com

Figure 4.1: Gaussian LPF denoising of an image. Shows the original image *(left)*, noisy image *(middle)*, and denoised image *(right)*.

convention it is set to encompass 3σ of the data.
The kernel size, $(-l \leq (x, y) \geq l)$, can then be determined by eliminating values less than 5 percent of the kernel maximum [66]. The convolution of an image with the Gaussian kernel has the effect of averaging a pixel with its neighbors. The Gaussian falloff determines the weight and extent of this averaging. This convolution can be implemented as a multiplication in the frequency domain by exploiting the properties of the Fast Fourier Transform (FFT) [67].

Despite this filter's prominence and widespread use, there are a number of issues. The present noise is smoothed, but in doing so the original data is distorted—blurring, edge displacement, and phantom edges appear in the filtered data [66]. Figure 4.1 shows an example of this quality loss, as Gaussian noise is added and removed from a photo. Due to this poor performance, the Gaussian filter is often used as a groundwork for more intricate denoising algorithms, but is seldom a solution in itself. Deng and Cahill [66] proposed one such improvement. Their adaptive Gaussian filtration partially combats the Gaussian's limitations by incorporating multiple variance values in pre-processing, and by partitioning the image to smooth sections of the image independently. A number of Gaussian filter adaptations like theirs exist, but they are numerous and only partially remedy Gaussian's limitations [56].

The Butterworth filter, first described by Stephen Butterworth in 1930,

is designed to have a flat-as-possible frequency response [68]. This 85-year-old filter excels at removing spikes from noisy data. It is useful for dust on a camera lens, or the common marker jumps noted in motion capture.

Perhaps inspired by the Gaussian FFT denoising, wavelet decomposition allows a signal to be reduced to a linear combination of wavelets– dozens of small impulse signals that, when combined, form a much more complicated and structured signal [69]. In image terms, we wish to remove seemingly random static and preserve high frequency details (hair strands, grass blades, sharp lines). In the wavelet domain, most of the image lies in only a few of the coefficients [69]. Denoising with wavelets becomes a practice of tweaking or thresholding unwanted coefficients to filter their respective contributions to the image.

One drawback to the wavelet thresholding method, however, is that all images will be decomposed differently, meaning wavelet coefficients will need to be tweaked for even slightly disparate images [70]. Automatic thresholding of the wavelet coefficients can be achieved through Bayesian statistics, placing a probabilistic belief distribution on each coefficient [71, 72]. This method is only useful for images that can be described using sparse coefficients, or for signals with a singular type of noise perturbation [71].

Kalman filters are another probability-based denoising method. The Kalman filter is a set of mathematical equations that also estimates noise using prior probabilistic information [73]. Unlike Bayesian wavelet thresholding, it is a linear predictive filter. It repeats this prediction process recursively, trying to minimize mean-squared-error as it progresses. It is often used for motion guidance systems in robotics, and has been applied to the denoising problem in DSP, images, gesture prediction, and motion capture [74].

In another approach to finding information about the underlying manifold beneath the noise, anisotropic diffusion [75] improves the Gaussian

model by incorporating edge detection mechanics and encourages blurring through diffusion in regions between hard lines. This diffusion is similar to the diffusion mapping seen in Chapters 2 and 3, in which the diffusion is a computationally expensive iterative process. Huang [76] applied the diffusion improvements as Porte [38], and performed eigendecomposition on the diffusion matrix to calculate diffusion time iterations with linear algorithmic complexity. Anisotropic Diffusion is a manifold-aware blurring technique, marking this algorithm as a strong candidate for the theme of this thesis and a good candidate for future work.

Lastly, example-based machine learning offers a different perspective on denoising. A number of techniques rely on example image libraries, and decompose images into a linear combination of overcomplete basis vectors that describe the images in the library [77]. This method, called sparse coding, is a popular trend in machine learning and non-local denoising. Yan et al. [61] performs such a technique on wavelet-decomposed images with impressive noise reduction results. Our research avoids these example-based methods, in preference of lightweight solutions that require no external data training. We cannot assume that an animator has access to large motion capture libraries. With the large range of possible motions in human movement, we cannot even guarantee that a library will contain similar motions to any given animation.

4.2.2 Motion Denoising

Many researchers have applied the aforementioned image filters to motion capture, which have become common benchmarks for comparison in this research area. In motion denoising, researchers have come to prefer systems aware of the spatio-temporal motion characteristics, such as dynamical systems.

The previously-mentioned Kalman filter is a type of dynamical system, in which a spatial and time dependent system of multiple states, driven

by variables hidden to the observer [78]. Linear dynamic systems (LDS) model observed measurements as noisy linear projections that evolve via a low-dimensional dynamic process—such as a Markov process, in which the state evolution of a system is dependent on a finite set of previous states [79, 80]. Temporal lag is a problem in dynamical systems, as a time delay will offset the data [9, 58]

Lately, data-driven methods have been noted as the most successful denoising techniques [58, 60]. Lou and Chai's example-based method employs singular value decomposition (SVD) to construct filter bases, iterating over the solutions to optimize noise elimination [60]. It denoises extremely well, but only when the database contains very similar 'families' of motion [58]. Actions from different spatio-temporal patterns lead to poor filter bases.

To compensate for this limitation, Xiao et al. [58] adopt ideas from sparse coding. While techniques such as PCA allow us to learn a complete set of basis vectors efficiently, sparse coding returns an over-complete set of basis vectors to represent input. The advantage of having an over-complete basis is that our basis vectors are better able to capture structures and patterns inherent in the input data [81].

Xiao et al. [58] and Feng et al. [9] break the pose up into partitions, called poselets, to segment motion learning into smaller and more manageable chunks. Their initial results outperform the previously mentioned methods, but rely on pre-cleaned motion input libraries to function properly.

Animation Software Denoising

Raw motion capture data is usually directly applied to skeletons inside major animation software, where it is then processed. Occasionally, the tracking system software will offer some form of data processing, especially where compression and decompression is required [58], but typically the editing is left to the animator. Blender, Maya, Motion Builder

[82], and other major animation software titles often rely on simple signal processing kits and are seldom bundled with extensive denoising libraries.

Blender features a simple weighted moving means function filter for smoothing bumps in motion curves. This tool is very handy for eliminating basic noise. Although there is little documentation on their method, conceptually it is similar to the averaging scheme in NLMeans. We will compare Blender's results to our own in the evaluation section.

Maya is packaged only with a Euler filter for correcting Euler gimbal lock or angle flipping. Motion Builder is a software exclusively for developing character motion, containing the most comprehensive collection of animation data tools: Gaussian, Butterworth, smoothing, resampling, and other rudimentary signal processing functions are available for minor edits.

4.2.3 Non-local Means

To locally denoise a pixel in an image, one simple strategy is to take the average pixel value in its local neighborhood, blending the noisy pixel into its neighbors, as in the Gaussian filter above. This naive approach leads to spotty and blurred images, where the noise is simply spread around the neighborhood.

To further improve the texture recovery, it helps to define non-local neighborhoods of similar textures from elsewhere in the image. Take, for instance, a noisy picture of a brick building. Various elements and features of the scene will be repeated throughout the image. One can use textural information from distant bricks to recover degraded pixel detail for nearby bricks [62].

To denoise a pixel $p(x, y)$, its pixel neighborhood—or pixel patch— is compared to all other patches q in a learning region of the image. The pixel patches are weighted based on similarity to the original noisy patch. This step allows us to find all of the most relevant texture patterns in the

image. The mean pixel value is computed from these weights. Figure 4.2 illustrates the concept of non-local comparisons in the learning region [62].

Given an image M, target noisy pixel patch p, and learning window I of radius $r < min(width(I), height(I))$, we create a library of pixel patches for image I q of radius $1 < r_q < r$ for every pixel in the learning window. Then non-local means can be expressed as:

$$p_{filtered}(x, y) = \frac{1}{C(p)} \sum_{q \in I} q(x, y) f(p, q), \tag{4.2}$$

where (x, y) are pixel coordinates and $C(p)$ is the sum of the Gaussian weights,

$$f(p, q) = e^{-\frac{|q-p|^2}{2\sigma^2}}. \tag{4.3}$$

Put simply, the output pixel is a normalized sum of the most similar pixel patches in a learning window. When $p = q$, the Gaussian weight is equal to 1. To prevent a pixel from weighting itself too highly, we instead assign it the maximum weight of the other pixel patches [83].

Related Research for NLM

Later research [84] has categorized NLM as a semi-local filter, rather than a truly non-local one. Results are dependent on input image structure, but the best output is usually constrained within a smaller learning window. As the training area tends toward non-locality by expanding to the size of the image, MSE declines for many practical examples of pictures. This phenomenon is due to the large number of small weights contained within the oversized training window [84]. Too many weights, although insignificant, lead to the averaging of dissimilar patches.

Conversely, in periodic images where patterns repeat themselves at a larger scale, increasing the radius of the learning window has a profound positive impact on mean-squared error (MSE). When we later apply this technique to motion capture, human motion can be both periodic and non-

periodic, so it is necessary to choose a locality measure for case-by-case denoising.

Figure 4.3: Non-local means denoising of an image. Denoised output image *(left)*, from the badly noised input image *(right)*.

Figure 4.3 shows non-local means correcting a slightly corrupted image. There is no visible blurring, and all of the noise appears to be eliminated.

Goossens et al. [85] remark that NLMeans is not able to compete with the recent trends in sparse-coding methods. It is for this reason, perhaps, that NLMeans has been overlooked in motion capture denoising. Goossens observes that NLMeans is the first iteration of a Jacobi algorithm, or the diagonal normalized steepest descent algorithm, and offer improvements from this method. Their NLM method is able to produce comparable results to sparse coding solutions.

Goossens et al. also criticize the algorithm for its $O(n^4)$ complexity, which increases exponentially given in large 2D image arrays. In a later paper, these authors accelerate the algorithm by reducing the enormous number of weight computations [86]. Other research enhancements to NLMeans include FFT-inspired acceleration [87], integration with the Laplacian Pyramid [87], and a GPU-based implementation [88].

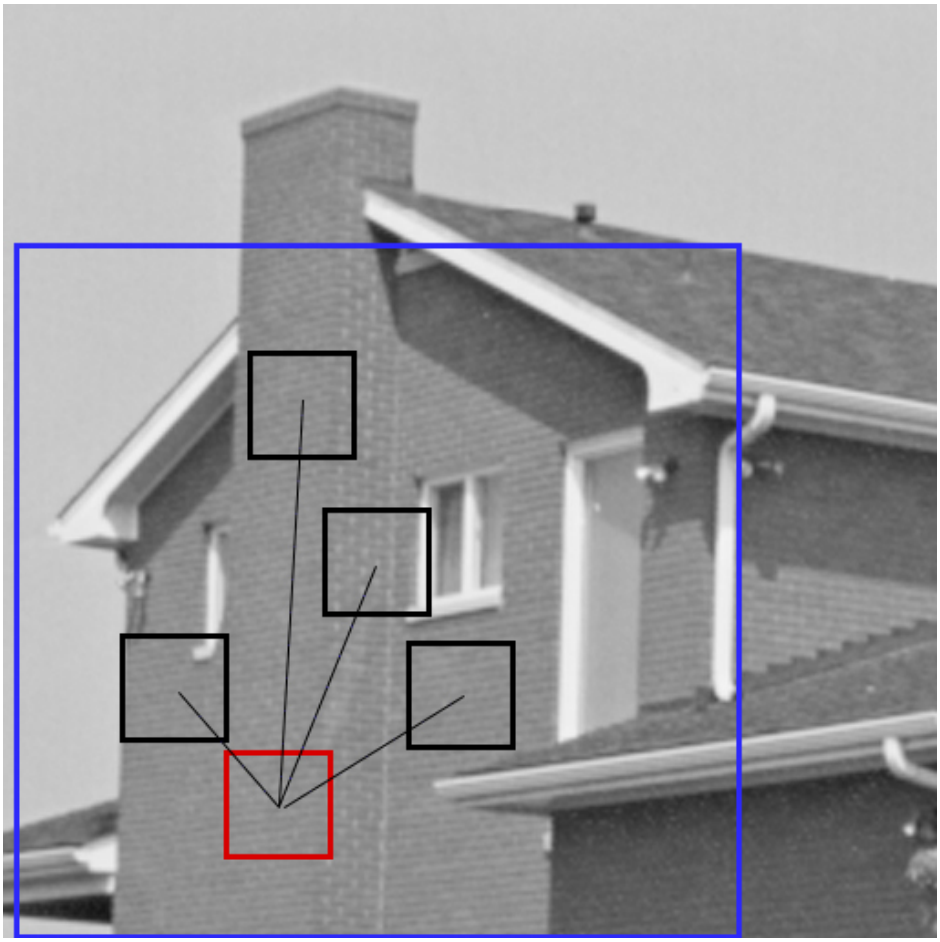


Figure 4.2: An example of the non-local means algorithm. A noisy pixel *(center of red square)* is denoised by comparing its local patch to other similar patches of pixels *(black squares)* inside the learning radius *(blue square)*.

4.3 Non-local Pose Means

This section details our contribution to motion capture denoising using the non-local means algorithm. We explain our method for a novel adaptation of the image processing technique to process n-dimensional rotation vectors.

4.3.1 Pre-processing Poses

NLMeans linearly interpolates data by averaging points via distance weights. This works well for vectorized patches of pixels, but special consideration is needed for averaging rotational poses. Therefore, we initialize all pose vectors as concatenated unit quaternions, and exponentially map them into log-space via Chapter 2's exponential mapping (equation 2.6). This allows linear operation on quaternion pose data.

Much like the image-based NLM, we begin by pre-processing the motion capture poses into 'patches' of pose vectors. The patch radius r_p determines the number of included poses from the animation timeline, forming a patch of size $(2r_p + 1)$. The patch vector is expressed as:

$$\mathbf{P}(t) = \begin{bmatrix} p(-r_p) & \dots & p(-2) & p(-1) & p(0) & p(1) & p(2) & \dots & p(r_p) \end{bmatrix}^T \quad (4.4)$$

for pose $p(t)$ and patch $\mathbf{P}(t)$ at time t in the mocap timeline. When we calculate weights for points beyond the edges of the data set, we optionally employ mirroring, Neumann (data boundary gradient is held constant), or Dirichlet (values are assumed constant) boundary conditions [89]. The choice of boundary condition should depend on the nature of the motion bounds.

This patch calculation step can be implemented as a method at runtime, but we have found that pre-processing the pose patches improves computation speeds of the algorithm.

4.3.2 The Non-local Pose Mean Filter

Locality on the Manifold or Time Domain

The choice of learning window determines the type of locality our algorithm covers. A direct implementation of non-local means would place the learning window bounds as the k-nearest pose frames: $p(t-k) < p(t) < p(t+k)$ for learning window $\mathbf{T}$ of radius k frames.

Alternatively, we can build on the pose space manifold learning discussed in Chapter 3 and choose a learning window that exists for k-nearest neighbors in the animation space. The manifold method averages poses in a truly non-local scope, but the former option is limited to semi-local information. Much like images, in the case of periodic actions, the more non-local solution will offer the optimal denoising. If the k-nearest patches have a large variance, however, the noise reduction will falter.

The Pose Mean

The remainder of our algorithm follows the original NLMeans scheme. Instead of a weighted-average pixel, we will find the average pose determined by distance weights. This average pose represents either the average collection of spherical joint rotations or the average pose on the animation manifold. To denoise a pose corresponding to the patch $\mathbf{P}(t)$, we calculate the Gaussian weights for all other patches $\mathbf{Q}(t)$:

$$f(p,q) = e^{-\frac{|\mathbf{Q}-\mathbf{P}|^2}{2\sigma^2}}. \tag{4.5}$$

The distance evaluation may be performed by equation 2.4 or the L_2 norm in the exponentially mapped space. A further manifold-based improvement might even utilize the Green's distance falloff instead of the Gaussian weights for refined manifold feature detection, but we leave this task for future work.

The poses can be averaged via their weights,

$$\mathbf{P}_{filtered}(x, y) = \frac{1}{C(\mathbf{P})} \sum_{\mathbf{Q} \in \mathbf{T}} \mathbf{Q}(t) f(\mathbf{P}, \mathbf{Q}), \tag{4.6}$$

once again taking care to set $\mathbf{P}(t)$'s weight to the maximum weight of its neighbors.

4.4 Results and Evaluation

Here we present the results of our method, and evaluate the NLPM noise reduction quality against some of the methods discussed in the related works section.

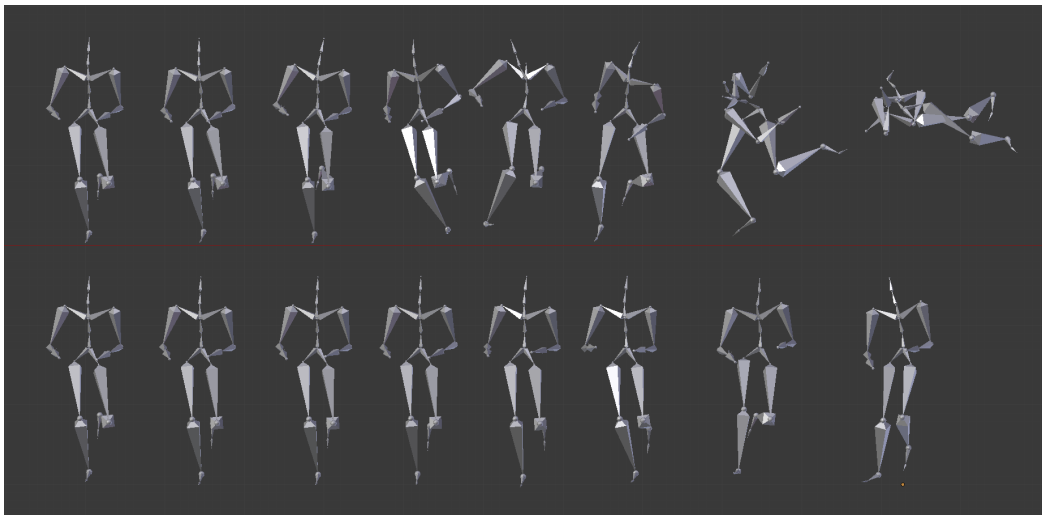


Figure 4.4: Non-local means applied to noisy motion capture data. Top: Original data, with increasing degrees of Gaussian noise from left to right (ranging from 1σ to 100σ). Bottom: The recovered motion for each respective animation, after applying non-local means.

4.4.1 Non-local Pose Means

Figure 4.4 highlights our method's noise reduction capabilities for light to drastic jitters in a skeleton. The top row of animated rigs feature a range of Gaussian additive noise, added individually to each joint in the animated skeletal rig. Non-local means can recover up to $\sigma = 1$ of normally-distributed rotational noise while retaining the original motion. Beyond this, even in the most extreme case of noise degradation the original motion can be identified, although some distortion is inevitably irrecoverable.

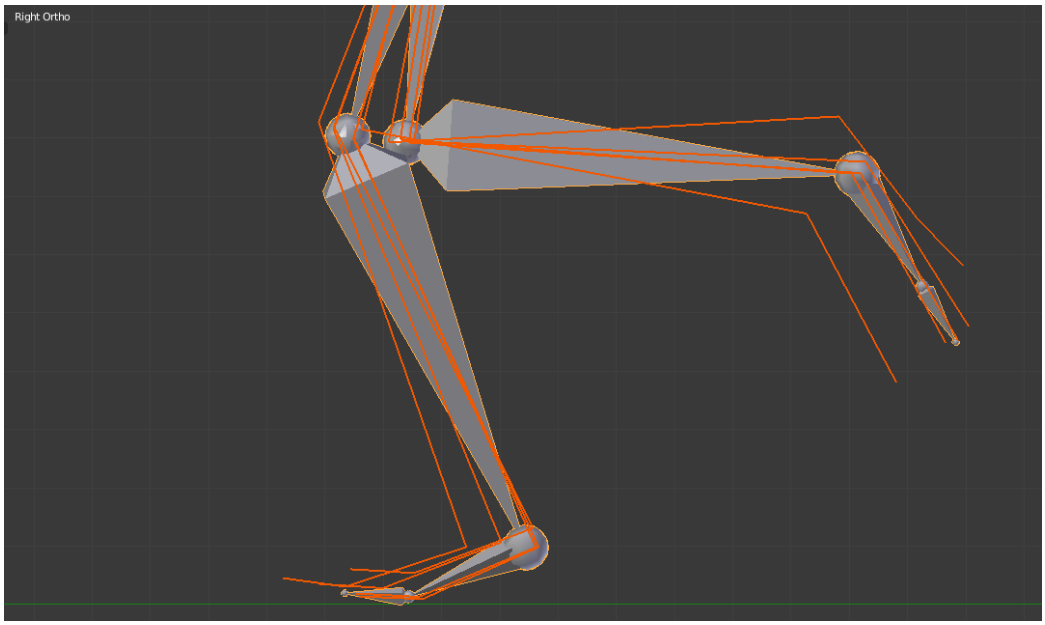


Figure 4.5: This figure shows the NLPM's susceptibility to angle offsets for denoised skeletons *(orange wireframes)* compared to the ground truth *(grey rig)*. For low to medium input noise, there is no noticeable slipping. For medium to high noise, an angular offset is detected, but Figure 4.6 shows that the foot placement remains constant during its contact with the floor.

One concern for algorithms that edit motion capture data is the ability to retain the physics and timing constraints of the original motion. Figure 4.5 shows the variation of pose averaging for different amplitudes of noise, at the precise moment when the foot stomps on the floor. At low noise intervals, the skeletal configuration is consistent with the original. At high levels of noise, an angular offset is introduced. This commonly causes slipping and hand/foot skating in animation. However, Figure 4.6 shows the foot's contact with the ground over a 20-frame step interval. In both low and high noise conditions, the foot contact remains solid, despite this angle offset. This is due to the NLPM's spatio-temporal gradient preservation, which preserves 'hard edges' in the motion curves. Although the values of the angles changed, the gradient information did not.

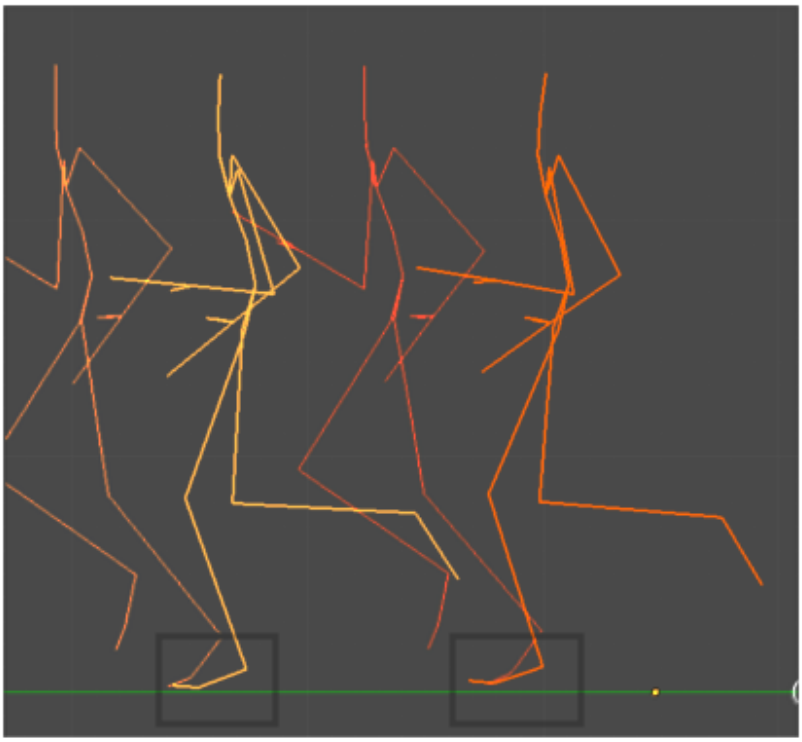


Figure 4.6: This figure shows the NLPM's resistance to foot skating. For medium noise (*left*) and high noise (*right*) the foot remains planted in the same location over 20 frames in a stepping animation. There is no noticeable skating effect, unlike many other algorithms, although we do note an overall slight change in skeletal angles.

A Note on Rotational Noise

The Gaussian noise introduced by the tracking system is experienced as displacement noise—that is, perturbations in the spatial domain that confuse the true marker positions. The skeleton is later inferred from these joint marker positions and converted into an angle-driven FK animated rig. When we add noise to quaternions in log-space, a small value for σ produces enormous joint displacement.

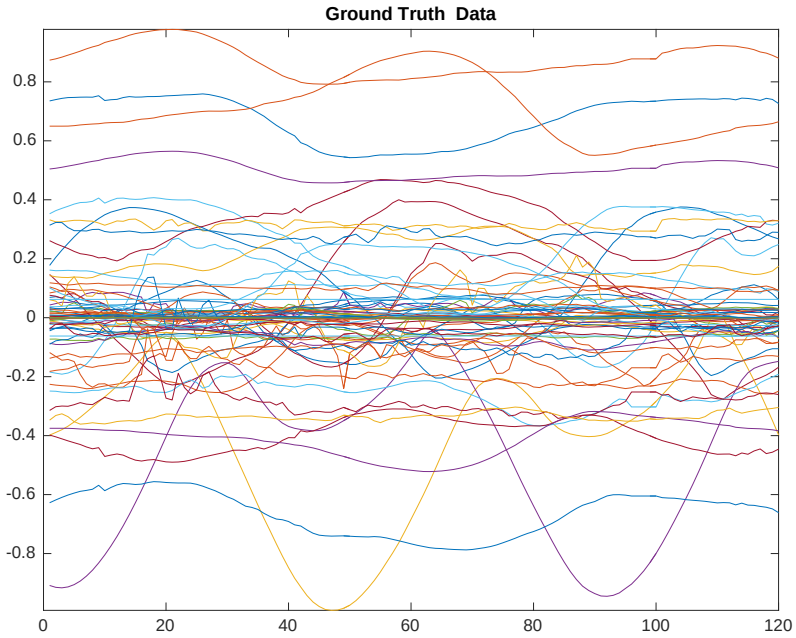
Due to the effect of hierarchical joint importance and bone length, discussed in Chapter 3.1, a single σ value produces erratic compounding

noise variations for every bone in the skeleton. This does not affect the individual dof signals, whose parent joint rotations have been ignored, nor does it affect the denoising table results in Table 4.1. This only impacts the visual distribution of noise in the output animation and the incredibly tiny size of sigma (which exists in exponentially mapped quaternion space). These denoising comparisons are still valid, but future studies should take caution and devise a means to add experimental noise in the spatial domain to simulate realistic motion noise.

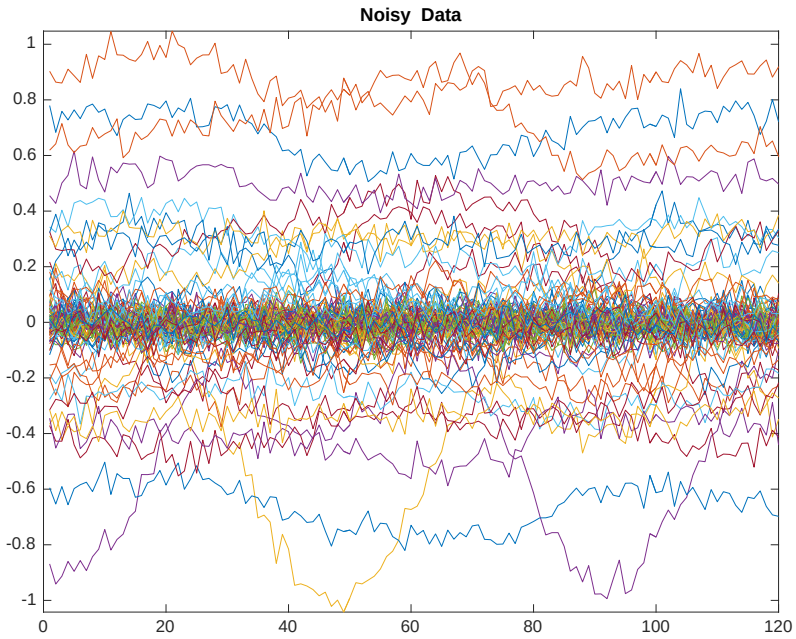
4.4.2 NLPM Benchmark Comparisons

Figures 4.8a-4.10a show the ground truth data, the noisy motion data, and the NLPM denoised output. The output looks much like the original, with a single sharp bend for all curves on the far right side of the graph. This is due to a poor choice of boundary condition, where mirroring causes abnormal weight distributions. Generally, the Dirichlet condition for holding assuming the gradient remains constant works best.

In Figures 4.10b-4.12b, a single motion channel is isolated from these dense figures, and we compare the NLPM result to ground truth and noise signals. We can note some loss of precision, and a small degree of temporal shifting (e.g. Figure 4.10b at frame 90).

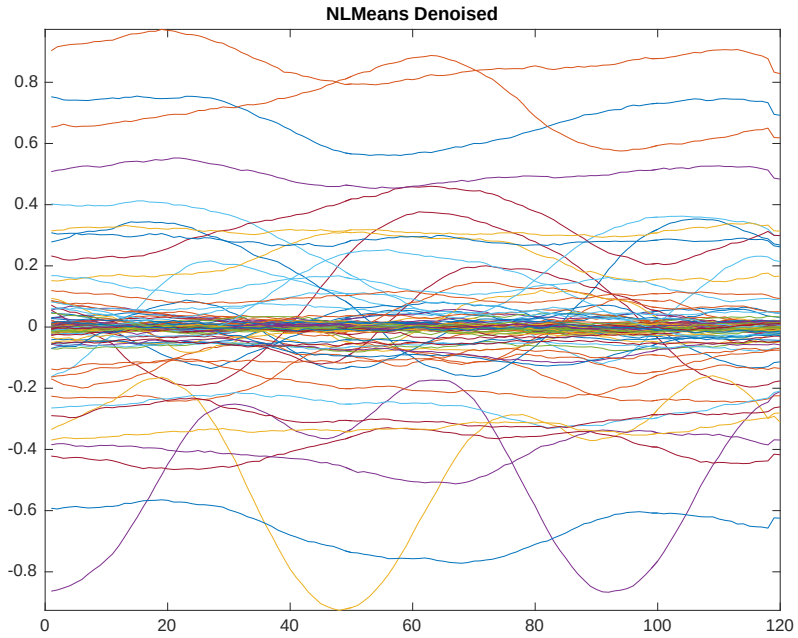


(a) Ground Truth curves for all rotations in the motion graph

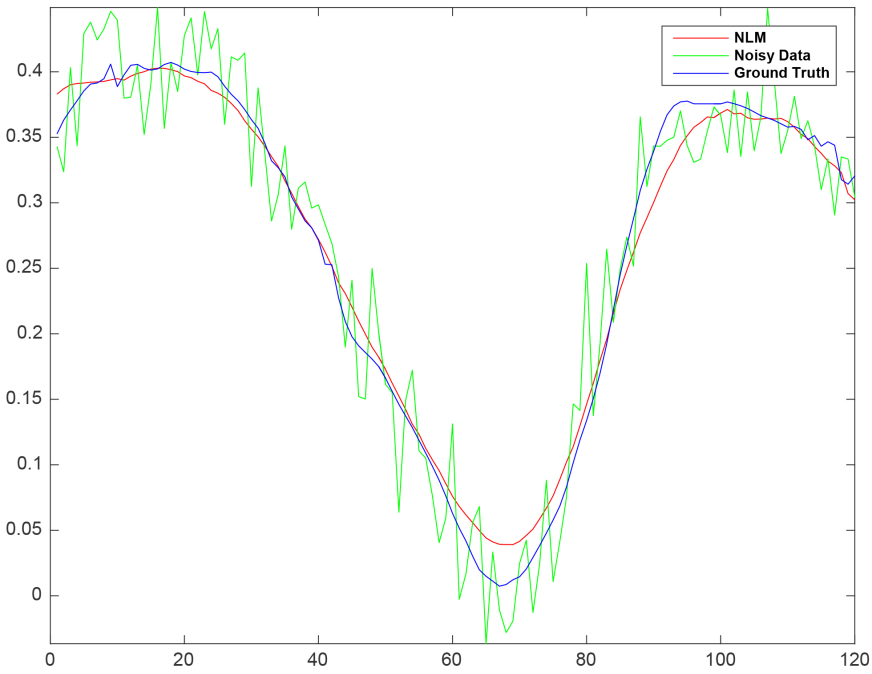


(b) Added Gaussian noise to motion curves (rotational $\sigma = .4$) for all rotations in the motion graph

Figure 4.7: Results

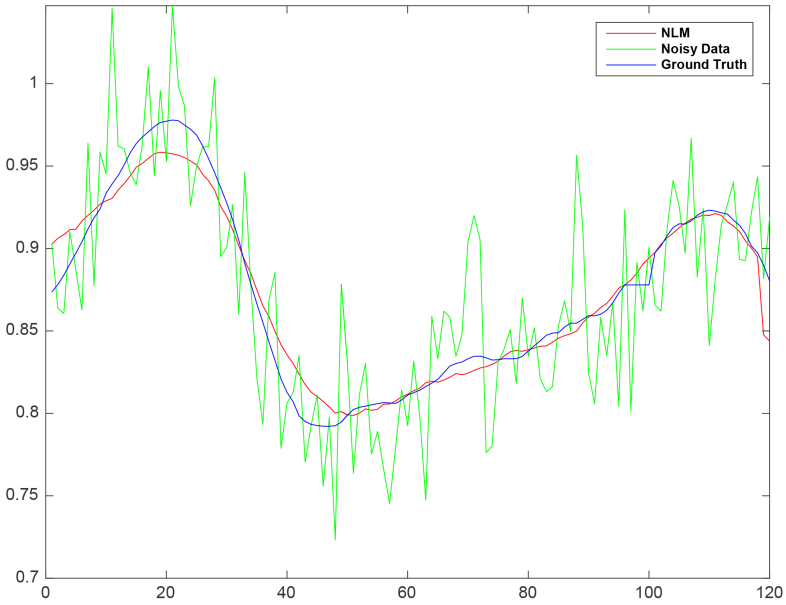


(a) Non-local pose means solution for all rotations in the motion graph, using noisy curves from Figure 4.8b as input.

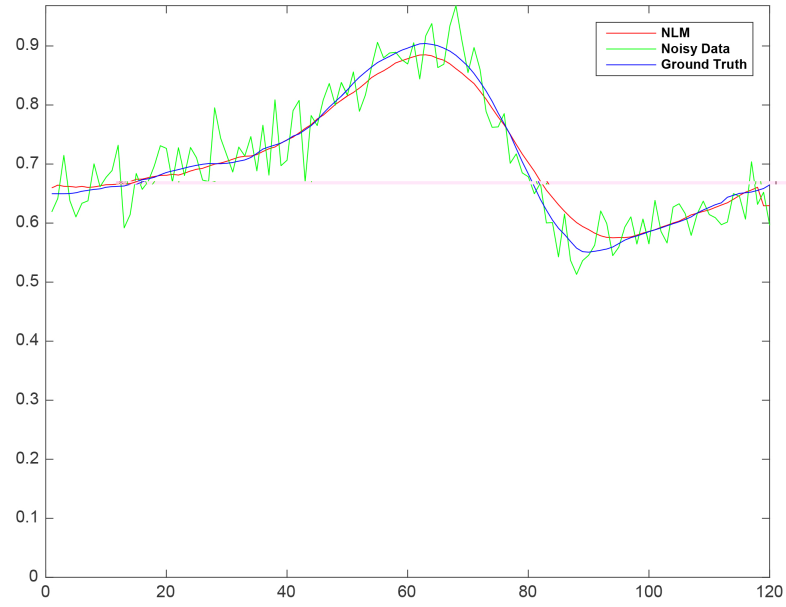


(b) NLPM vs. Ground Truth vs. Noisy data for a single channel of joint rotation data.

Figure 4.9: Results

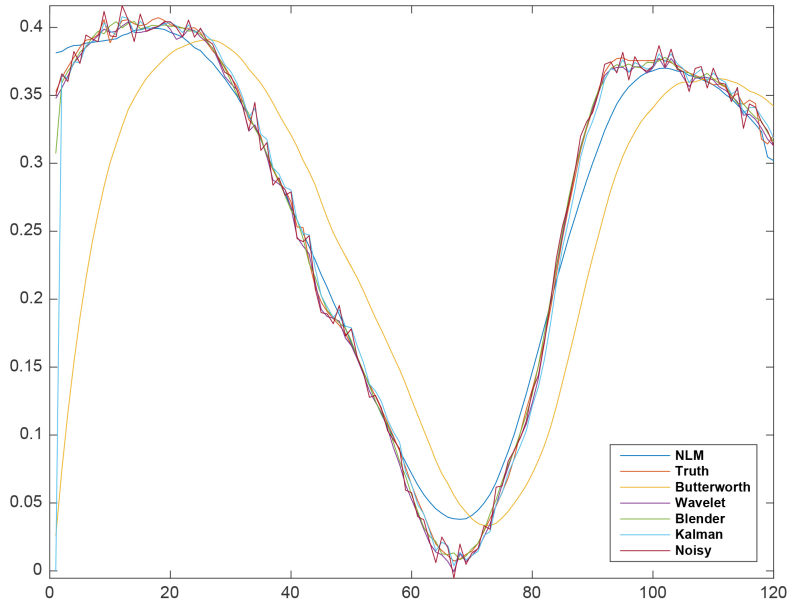


(a) NLPM vs. Ground Truth vs. Noisy data for a single channel of joint rotation data.

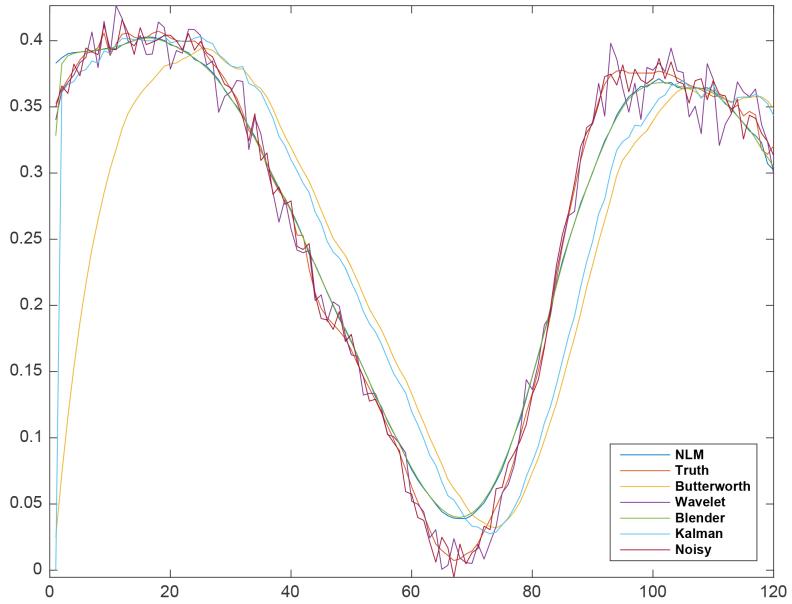


(b) NLPM vs. Ground Truth vs. Noisy data for a single channel of joint rotation data.

Figure 4.11: Results



(a) An overlaid comparison of all denoising methods from Table 4.1, for the light noise condition (15σ).



(b) An overlaid comparison of all denoising methods from Table 4.1, for the medium noise condition (30σ).

Figure 4.13: Results

PSNR, when compared to ground truth

	σ	15σ	30σ	50σ	75σ	Impulse
Gaussian LPF	73.89	73.87	73.76	73.33	72.02	73.61
Kalman	80.46	80.42	78.50	73.83	–	80.01
Blender WMM	93.39	**92.53**	84.99	84.40	78.99	**88.61**
Wavelet	**93.76**	67.01	–	–	–	82.59
Butterworth	82.21	82.14	81.69	80.30	77.21	81.39
Our method	85.58	85.52	**85.29**	**84.68**	**82.46**	85.18

Table 4.1: PSNR Noise Elimination Result Comparison. The optimal result in each column has been emboldened or excluded where smooth data was not possible. $\sigma = .008$, added in quaternion log-space.

Table 4.1 compares the peak signal to noise ratios (PSNRs) of the common motion denoising techniques that we discovered in the literature review. PSNR is related to the mean squared error, but describes the We add normally-distributed noise to the quaternion poses of the CMU library motion capture data and compare the mean-squared error of the denoised output to the original ground truth. For each denoising method, care was taken to find the optimal input parameters for the smoothest result, without sacrificing motion detail. We used Matlab signal processing libraries for Gaussian, wavelet thresholding, and Butterworth denoising. The python Kalman filter implementation was borrowed from Bishop and Welch [73], where it was optimized specifically for motion denoising research.

We denoised a running character from the CMU library and applied NLPM with a learning window of 21 poses, a patch size of 11 poses, and a Gaussian width ($\sigma = .008$). In every noise category, our method produced superior denoised-signals that most resembled the ground truth data. As noise degradation consumes the original data, PSNR slowly and steadily declines in our method. In the most extreme noise, the animated output appears smooth, but the physical actions are visibly different. At 50σ,

smoothed trembles appear for the more prominent bones. In the other noise conditions, it is difficult to visibly spot any difference between the denoised animations.

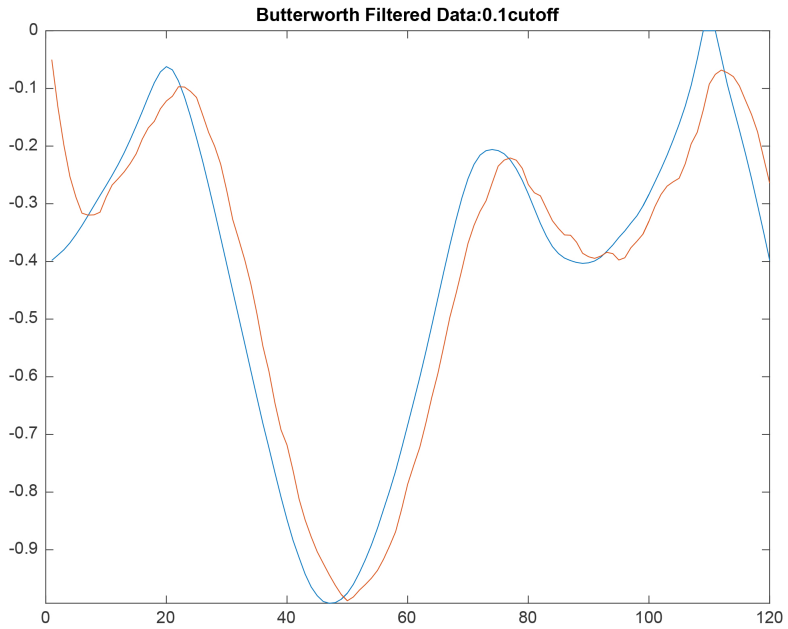


Figure 4.15: Butterworth *(blue)* vs. ground truth *(red)*. The Butterworth filter is capable of smooth results, but only after introducing a time-delay. A similar effect occurs in dynamical filters, like the Kalman filter. The Butterworth filter has a problem with edge boundaries, and sometimes is too smooth, losing important detail.

The Butterworth filter perform surprisingly well. The results in Table 4.1 are not indicative of this performance, as an offset develops for increasing cutoff values. Figure 4.15 shows the offset from the ground-truth data. The same effect can be observed in the Kalman figure, whose time-delay is a well-known problem in dynamical systems. Kalman eventually destabilizes, and cannot handle large quantities of noise, where Butterworth's PSNR slowly diminishes. Butterworth is especially noted for its ability to filter out peaks and outliers and bests most of the other methods in these results. The Butterworth and Kalman filters compete with NLPM

for light noise conditions, but our method shows the better PSNR in every listed category.

Wavelet thresholding produced the best smoothing for the smallest noise amplitudes. At every other noise interval, however, the smoothing was ineffective, or it warped the motion unrecognizably. A PSNR comparison does not represent a meaningful measure in either of these cases. A different threshold condition may perhaps yield more conclusive results, but we were unable to draw a consistent denoising for hundreds of thresholds. This variability makes wavelets a difficult, unintuitive, and unpredictable denoising tool for an artist. As mentioned in the related literature, statistical thresholding algorithms exist that may aid in this process, but for now we will dismiss wavelets as an unwieldy noise reduction technique.

Blender's weighted moving means algorithm performs equally well as or better than NLPM for small levels of noise degradation, but worsens as noise increases. Our input parameters were optimized for medium to high-levels of noise; with further experimentation, NLPM may be able to match Blender's performance in the lower noise conditions. Regardless, the strong performance of NLPM against professional-grade software is indicative of our method's limited but promising success.

Figures 4.14b and 4.14b depict an overlaid comparison of the best denoising methods, for two different noise conditions. Between these two graphs, NLPM shows the best consistency and fewest artifacts. Blender's method declines in quality, as it derails from the ground truth between the two graphs. The minimum of NLPM does not reach the same minimum as the ground truth in this curve; this is an example of too many dissimilar poses watering down the result with their accumulated weights. A better non-local manifold parameterization may correct this, but we must leave this optimization for future work.

4.5 Chapter Summary

In this chapter, we have examined image and signal filters for noisy data reduction. We identified previously implemented denoising techniques and their applications to motion capture processing. We found that a recent trend employs the use of unsupervised learning tools to denoise motion using extensive motion capture data libraries. Sparse coding techniques are drawn from to optimize the denoising process. Because few people have access to the resources required for these solutions, we sought to model our solution after untrained denoising filters.

A number of these methods have previously been adapted to suit motion capture data, such as Gaussian, wavelet, Kalman, and Butterworth filters. NLMeans image denoising, to our knowledge, had not yet been applied to motion capture data.

Our adaptation of NLMeans to mocap required the preprocessing of pose vectors, such that they could be averaged within the exponential mapping or a manifold space. We then applied non-local averages, weighted by quaternion pose distance, to denoise the data.

The results of NLPM were compellingly smoothed animation curves. Surprisingly, Blender's weighted moving means algorithm bested NLPM for low degrees of noise. When noise interference increases, our algorithm better adapted to the noise, creating a stronger result. For the remaining denoising methods, the NLPM denoising yielded the smoothest results without compromising character motion.

Chapter 5

Discussion and Conclusion

This research proposed that the Green's function falloff distance is an effective interpolant for inserting edits into mocap data. Additionally, we used non-local means denoising as a smoothing filter for raw motion capture data. Both of these techniques draw on information from the animation manifold in the space of character poses.

In this chapter, we present a review of these techniques and a summary of their effectiveness at manipulating motion capture data. We discuss the Limitations of the Green's function falloff and NLPM as well as ideas for future work that can address these flaws or expand the research scope.

5.1 Summary

This thesis introduced a new spline-like falloff interpolant and denoising method for editing animation.

To accomplish this, Chapter 2 investigated animation workflows for creating animations with spline-based motion graphs. We learned useful mathematics for processing poses with linear techniques. Chapter 2 also presented various machine learning and dimensionality reduction techniques to define a low-dimensional animation manifold within the character pose space.

In Chapter 3, we looked at popular methods for editing motion capture animation and discovered the role of dimensionality reduction methods in motion editing. Our subsequent contribution defined a novel measure for pose similarity by using Green's function in a pose-space manifold. This pose-similarity doubles as pose distance, from which we defined falloff weights for pose interpolation. This method was implemented in Blender, where we edit motion capture data by reposing a character. The algorithm handles the falloff in a data-driven manner, returning an artist-editable falloff spline for pose interpolation.

We compared our manifold distance, the Green's function distance, to metrics in isomapping and diffusion mapping. Green's distance falloff outperformed the others for seamless animation blending. We showed that our falloff curve is smoother than those obtained with the isomap and diffusion map methods, while still retaining the benefits of a manifold learning method. In some cases, diffusion maps have similar qualities to the Green's function falloff. However, our method creates consistent falloff graphs from fewer user input parameters. For repeated motion activities, we showed that motion edits can be generated and propagated throughout an entire animation sequence automatically.

To satisfy our initial research objective of creating an editing tool inspired by conventional animation workflows, we compared our method to traditional spline-based keyframe animation. The Green's function method produced the same falloff curve as the spline-based animation edit in Blender, interpolating the animated edits in a nearly identical manner. Artists are accustomed to viewing and editing spline-like curves, making our solution useful and convenient. In fact, this method is complementary to most editing algorithms in animation software, which rely on artist-defined falloff splines. Incorporating our method in this software should be simple, as the default distance measure can be replaced by our Green's function distance.

In Chapter 4, we examined methods for post-processing motion data

for noise reduction. Our novel application of non-local means image denoising is used to smooth motion capture data. Our method differs from image denoising in that we are smoothing a collection of hundreds of inter-dependent spatio-temporal skeletal joint dofs. We altered the algorithm to allow quaternion pose interpolation in the animation manifold or quaternion log-space.

We compared our results against other non-example based methods, and found promising, yet mixed, results. In conditions of low noise, the best results were produced by wavelet thresholding and Blender's weighted moving means algorithm. However, our method offered consistent results for a variety of noise samples, and offered stronger denoising capabilities than Blender or wavelets for large amounts of normally-distributed noise.

5.2 Limitations

The largest drawback of the Green's function is the unpredictability of σ as an input parameter. Poor choices of σ can lead to a timing offset, where edited animations slip a few frames out of synchronization. Green's function is not a pure distance measure, but rather a measurement of pose arrangement on the manifold. This is valuable for treating it as a falloff, but care should be taken in parameterization.

The σ parameter should be exposed to the artist and varied experimentally based on the local and global density of the pose space. An overly narrow σ does not capture the relation between similar poses, resulting in abrupt falloff curves. An overly broad σ views all poses as similar, and thus cannot localize edits.

The complexity of the Green's function algorithm scales by $O(n^3)$. Working with a typical movie shot of few hundred frames is nearly instantaneous. A C++ implementation exploiting the sparseness of the Laplacian would improve interaction times.

Noise in sparse pose samples may lead to a poorly defined manifold, resulting in unreliable falloffs. However, using a non-local pose means denoising should alleviate this concern.

Our proposed solution for noise reduction boasts successful results, however it still leaves clear room for improvement. Chief among our concerns is the performance for low-noise conditions, in which our algorithm was bested by Blender. We believe that this is a case of non-optimized input parameters or improper choice of locality. The NLPM algorithm has three different user input values: patch size, training window size, and a σ for pose distance threshold. This is not nearly as cumbersome as some denoising filters, such as the input-heavy Kalman filter, but slight changes in NLPM input parameters tend to have a large impact on filter quality.

Many denoising techniques from the literature review also address missing marker recovery, a common challenge in processing motion capture data. Our method is able to recover a few frames of missing data through averaging, but cannot yet handle extensive recovery.

Another concern in applying NLPM is the choice of locality. A fully-non local pose estimation that uses a large training window usually results in a drooping-skeleton effect. The many poses blend together, and the skeleton drifts toward the average pose of the sample set when too many weights are calculated (even if most are significantly small). One way to eliminate this problem is to use k-nearest neighbors on the manifold, instead of k-nearest frames in the training window. For periodic motions, this manifold-based NLPM will function well. For non-periodic motions, a semi-local solution will work much better. However, most motion is somewhere between these two poles; only trial and error will yield optimal results.

5.3 Future work

The promising results from this research suggest that further development is a worthy cause. These methods are not production ready, and should naturally undergo further research, development, and testing.

The ultimate test of our method is live interaction with artists. Feedback and usage statistics from production artists in the game or film industries is invaluable for research of this nature, and will help guide the direction of future research.

Motion Capture Splines

In Chapter 3, we presented a mathematical formulation for a *motion capture spline*, derived from the Green's function of the pose Laplacian. The primary future work for this research is the development of a motion capture editing system with a fully integrated motion capture spline workflow. Based on elements from this research, the envisioned system should meet the following criteria:

- It should continually update the Pose Laplacian as the animator updates the animation.
- The Green's function pose distance graph should be interactively viewable.
- Multiple spline editing methods should be available to the artist, who can then directly edit the Green's function falloff for tweaks or custom interpolation.
- It should display inflection points in the pose distance curve, so the artist can interactively choose how their edits blend back into the animation.
- It should allow the editing of a sequence of poses, rather than just a single frame.

- The system should be able to read from motion repositories, for data-driven manifolds learning using mocap libraries.
- The system should use dimensionality reduction for animation manifold visualization.
- And lastly, this system should be integrated in common animation environments, like Maya or Blender.

The most challenging point in this list is the requirement for animation sequence editing. In our system, many different pose edits can be made before recalculating the Laplacian. However, if edits are very close together on the manifold, there will be an issue of overlapping falloffs.

Instead of editing a single pose, an artist may want to edit several at once. In this case, animation splines replace the falloff between poses, but the first and final pose still use our data-driven falloff to blend into the overall animation. However, edit propagation then becomes nontrivial, as our algorithm does not currently make any temporal considerations for comparing sequences of poses—it only compares individual poses. The propagated animation timing will likely be squished or stretched, since timing can differ slightly among similar gestures. Overcoming these challenges would establish a robust animation editing system.

Non-Local Pose Means

Noise occurs in a variety of ways, for an infinite number of skeletal motion and construction possibilities. It is difficult to foresee every complication that will arise in raw signal data, which is why the denoising field is such an active area of research. A number of minor enhancements to NLPM can address the limitations of our system, leaving room for many future works:

- Our method should be compared against hundreds of animation samples for robust evaluation. Our method works great with overly

noisy signals for the chosen mocap samples. Improving the denoising for smaller amplitude noise may be a matter of fine-tuning parameters by exploring best practices with different mocap data sets.

- The manifold-based methods for non-local means should be analyzed with large motion databases and compared to other example-based methods.

- Related to the question of locality, experimentation for partitioned skeletal subsets of joints should be carried out. The pose-distance metric in NLPM calculates weights based on the distance between poses in the training window. It is possible that including the entire skeleton as opposed to skeletal partitions (*poselets or partlets*) will have an impact on denoising results.

- Goossens et al. [85] points out that the NLM is a first-order Jacobi operation, and they suggest improvements that allow the method to compete with sparse coding techniques. In a future work, NLPM should make these same improvements, and a study should be conducted comparing our method against mocap sparse coding.

- As we pointed out in Chapter 4, adding rotational interference creates unpredictable positional noise amplitudes across the entire skeleton. Future studies should add noise in the spatial domain, instead of the rotational domain, to conduct experiments on data that better represents real-life noise situations.

- Related to the previous note, experiments should be conducted on mocap data that features multiple sources of noise interference. The configuration of many denoising algorithms are, in our experience, dependent on a noise signal with a singular σ value, whereas ours is an acceptable level of stability for various noise conditions. This could prove further that our method is more effective at dealing with real-life mocap data.

- NLPM-based missing marker recovery should be explored for mocap data that is broken or missing large segments of motion curves. Many of the other denoising algorithms offer data recovery solutions.

- Future work could seek to incorporate our findings from Green's function pose distance in NLPM. Using Green's function distance, or another weighting metric, instead of a Gaussian weighting could address the limitations for non-local schemes featuring large training windows.

- Many mocap denoising algorithms are specially treated to handle spatio-temporal awareness. Our method has little awareness of time-domain constraints, but is designed to preserve hard edges in this domain. Regardless, future work could explore alternate methods of enforcing physics constraints.

5.4 Conclusion

Modern animation software is disorienting in its enormity and complexity. New users are often at a total loss in their first time encountering Maya, and advanced users only ever use a small fraction of its available tools. To complete any one task in animation, there are dozens of built-in tools, hundreds of downloadable plug-ins, and even more research papers for completing the task—and despite all of this, most large production companies still hire in-house programmers to write custom software.

Each of these tools was designed with questions in mind: How do we allow for more creative control? How do we make it cheaper, faster, prettier, or more realistic?

Many researchers have attempted to delegate, assist, and even automate the animation job for artists, approaching the problem from countless angles, applying techniques from the far reaches of abstract geometry or artificial intelligence programming to find a solution.

In this thesis we drew on dimensionality reduction, manifold learning, and image processing techniques to solve a few specific problems. Our data-driven splines bridged the gap between the classical animator and the motion capture technician, so that an animator can mould a live actor's performance just as they would a digital character. Our non-local pose means denoising technique attempts to make mocap processing tools lightweight and reliable. Hopefully in doing so, we have made it slightly easier for an animator to bring fantasy to life on the big screen.

Chapter 6

Appendix A

Green's function as a similarity measure

The Green's function of the Laplacian is not widely used as a similarity measure in computer graphics, but it has appeared in other fields. In geostatistics the Green's function of the Laplacian has been used as a generalized covariance [90]. In machine learning, the Green's function of the Laplacian is know to be a kernel, and kernels are often employed to summarize the similarity between data points.

The use of the Green's function of the Laplacian as a similarity measure can be intuitively motivated by considering the case of regularly sampled signals, such as audio signals or images. In this case, the Laplacian matrix has Toeplitz structure with the stencil $-1, 2, -1$ shifted by one place in each subsequent row. For $\rho \neq 1$ but near 1, the matrix

$$\frac{1}{1-\rho^2}\begin{bmatrix} 1 & -\rho & 0 & 0 & \dots \\ -\rho & 1+\rho^2 & -\rho & 0 & \dots \\ 0 & -\rho & 1+\rho^2 & -\rho & \dots \\ & & \dots & & \end{bmatrix}$$

is an approximate Laplacian. This matrix is known to be the inverse of the Kac-Murdock-Szego matrix, $[C_{i,j}] = \rho^{|i-j|}$ (this relationship appears in the literature on the Discrete Cosine Transform, where $C_{i,j} = \rho^{|i-j|}$ has been used as a generic covariance matrix for images [91]).

The Green's function of the graph Laplacian also appears in several expressions for distances on graphs. For example the resistance distance [92] between two nodes i, j is

$$r_{ij} = G(i,i) + G(j,j) - 2G(i,j)$$

where the Green's function acts as a similarity ($G(i,j)$ small means the distance is large). This distance also has a well-known interpretation as the expected length of a random walk from node i to j and back [93].

Bibliography

[1] Bureau of Labor Statistics, U.S. Department of Labor, "Occupational outlook handbook," 2014. [Online]. Available: http://www.bls.gov/ooh/arts-and-design/multimedia-artists-and-animators.htm. Accessed 21- Feb- 2016.

[2] L. M. Tanco and A. Hilton, "Realistic Synthesis of Novel Human Movements from a Database of Motion Capture Examples," 2000.

[3] O. Johnston and F. Thomas, "The illusion of life: Disney animation," 1995.

[4] J. Lasseter, "Princples of Traditional Animation Applied to 3D Computer animation," *Computer Graphics, volume 21, Number 4*, 1987.

[5] A. Menache, *Understanding Motion Capture for Computer Animation, Second Edition (Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann, 2010.

[6] "Syntel," Blender Foundation — Durian Project, 2013. [Online]. Available: https://www.flickr.com/photos/communitymag/. Flickr. Licensed Under CC BY 3.0. [Accessed 20- Feb- 2016].

[7] Y. Freedman, "Is It Real... or is it Motion Capture?: The Battle To Redefine Animation in the Age of Digital Performance," 2012.

[8] T. Allison, "More than a Man in a Monkey Suit: Andy Serkis, Motion Capture, and Digital Realism," 2011.

[9] Y. Feng, M. Ji, J. Xiao, X. Yang, J. J. Zhang, Y. Zhuang, and X. Li, "Mining Spatial-Temporal Patterns and Structural Sparsity for Human Motion Data Denoising," *IEEE Transactions on Cybernetics*, vol. 45, no. 12, pp. 2693–2706, 2014.

[10] R. Lai, P. Yuen, and K. Lee, "Motion capture data completion and denoising by singular value thresholding," *Proc. Eurographics Association*, pp. 1–4, 2011.

[11] L. Herda, P. Fua, R. Plänkers, R. Boulic, and D. Thalmann, "Skeleton-based motion capture for robust reconstruction of human motion," in *Computer Animation 2000. Proceedings*, pp. 77–83, IEEE, 2000.

[12] C. Dean, J. Lewis, A. Chalmers, and J. Kim, "Motion splines: Manifold pose distance for intelligent motion capture editing," Publication forthcoming.

[13] C. Dean and J. Lewis, "Non-local pose means for motion capture noise reduction," Publication forthcoming.

[14] The Blender Foundation, "Blender 3d," v2.74. 2016.

[15] Autodesk, Inc., "Autodesk maya," version 2016.

[16] C. Cabrera, *An Essential Introduction to Maya Character Rigging with DVD*. Focal Press, 2008.

[17] "Beyond: Two souls - motion capture," Community Mag., 2013. Available: https://www.flickr.com/photos/communitymag/. Flickr. Licensed Under CC BY-NC 2.0. [Accessed 20- Feb- 2016].

[18] N. Rico., "Motion capture suit.," 2007. [Online]. Available: https://www.flickr.com/photos/nrico/. Flickr. Licensed Under CC BY-NC-ND 2.0. [Accessed 20- Feb- 2016].

[19] Z. Zhang, "Microsoft kinect sensor and its effect," *IEEE Multimedia*, vol. 19, no. 2, pp. 4–10, 2012.

[20] Leap Motion, Inc., "Leap motion," 2010.

[21] C. M. U. G. Lab, "Cmu motion capture library database," [Online.] Available: mocap.cs.cmu.edu. Accessed 7- January- 2016.

[22] J. McCarthy, "An introduction to theoretical kinematics mit press," *Cambridge, Mass*, 1990.

[23] J. P. Lewis, M. Cordner, and N. Fong, "Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 165–172, ACM Press/Addison-Wesley Publishing Co., 2000.

[24] J. Diebel, "Representing attitude: Euler angles, unit quaternions, and rotation vectors," 2006.

[25] E. W. Weisstein., "Quaternion," in *MathWorld–A Wolfram Web Resource*, [Online]. Available: http://mathworld.wolfram.com/quaternion.html. Accessed: 20- February- 2016.

[26] F. Gissen, "Quaternion distance metrics," in *The ryg blog*, [Online]. Available: https://fgiesen.wordpress.com/. Accessed: 16- January- 2016.

[27] K. Shoemake, "Animating rotation with quaternion curves," in *ACM SIGGRAPH computer graphics*, vol. 19, pp. 245–254, ACM, 1985.

[28] F. S. Grassia, "Practical parameterization of rotations using the exponential map.," in *Journal of graphics tools, 3(3), 29-48.*, 1998.

[29] R. H. Bartels, J. C. Beatty, and B. A. Barsky, *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann Pub, 1987.

[30] M. M. Stanisic, *Mechanisms and Machines: Kinematics, Dynamics, and Synthesis*. CL Engineering, 2014.

[31] B. Choe, H. Lee, and H.-S. Ko, "Performance-driven muscle-based facial animation," *The Journal of Visualization and Computer Animation*, vol. 12, no. 2, pp. 67–79, 2001.

[32] Q. Li and Z. Deng, "Orthogonal-blendshape-based editing system for facial motion capture data," *Computer Graphics and Applications, IEEE*, vol. 28, no. 6, pp. 76–82, 2008.

[33] J. Lewis and K.-i. Anjyo, "Direct manipulation blendshapes," *IEEE Computer Graphics and Applications*, no. 4, pp. 42–50, 2010.

[34] H. Mllerland, E. Loosch, and P. H. Erfurt, "Functional variability and an equifinal path of movement during targeted throwing."

[35] E. W. Weisstein., "Laplacian," in *MathWorld–A Wolfram Web Resource*, [Online]. Available: http://mathworld.wolfram.com/Laplacian.html. Accessed: 20- February- 2016.

[36] M. Hein, J. Audibert, and U. von Luxburg, "Graph laplacians and their convergence on random neighborhood graphs," *Journal of Machine Learning Research*, vol. 8, pp. 1325–1368, 2007.

[37] H. J. Shin and J. Lee, "Motion synthesis and editing in low-dimensional spaces," *Computer Animation and Virtual Worlds*, vol. 17, no. 3-4, pp. 219–227, 2006.

[38] J. de la Porte, B. M. Herbst, W. Hereman, and S. J. van der Walt, "An introduction to diffusion maps," in *Pattern Recognition Association of South Africa*, 2008.

[39] P. Glardon, R. Boulic, and D. Thalmann, "PCA-based walking engine using motion capture data.," in *Computer Graphics International. Proceedings (pp. 292-298)*, 2004.

[40] J. Wang and B. Bodenheimer, "An evaluation of a cost metric for selecting transitions between motion segments," in *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03, (Aire-la-Ville, Switzerland), pp. 232–238, Eurographics Association, 2003.

[41] E. N. Mortensen and W. A. Barrett, "Intelligent scissors for image composition," in *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, (New York, NY, USA), pp. 191–198, ACM, 1995.

[42] M. Gleicher, "Image snapping," in *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, (New York, NY, USA), pp. 183–190, ACM, 1995.

[43] A. E. Seward and B. Bodenheimer, "Using nonlinear dimensionality reduction in 3d figure animation," in *ACM Southeast Conference*, 2005.

[44] S. Hauberg and K. S. Pedersen, "Spatial measures between human poses for classification and understanding," in *Articulated Motion and Deformable Objects*, 2012.

[45] A. Bruderlin and L. Williams, "Motion signal processing.," in *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques (pp. 97-104)*, 1995.

[46] A. Witkin and M. Kass, "Spacetime constraints.," in *ACM Siggraph Computer Graphics (Vol. 22, No. 4, pp. 159-168)*, 1988.

[47] M. Gleicher, "Motion editing with spacetime constraints.," in *Proceedings of the 1997 symposium on Interactive 3D graphics (pp. 139-ff)*, 1997.

[48] C. K. Liu and Z. Popovic, "Synthesis of complex dynamic character motion from simple animations.," in *ACM Transactions on Graphics (TOG) (Vol. 21, No. 3, pp. 408-416).*, 2002.

[49] M. Gleicher, "Comparing constraint-based motion editing methods.," in *Graphical models, 63(2), 107-134*, 2001.

[50] M. Tournier, X. Wu, N. Courty, E. Arnaud, and L. Reveret, "Motion compression using principal geodesics analysis," in *Computer Graphics Forum 28(2) (2009) 355364*, 2009.

[51] J. Barbic, A. Safonova, J. Y. Pan, C. Faloutsos, J. K. Hodgins, and N. S. Pollard, "Segmenting motion capture data into distinct behaviors.," in *Proceedings of Graphics Interface 2004 (pp. 185-194). Canadian Human-Computer Communications Society*, 2004.

[52] J. Lee, J. Chai, P. Reitsma, J. K. Hodgins, and N. S. Pollard, "Interactive control of avatars animated with human motion data," in *ACM Transactions on Graphics 21, 3 (July 2002), 491500. ISSN 0730-0301 (Proceedings of ACM SIGGRAPH 2002)*, 2002.

[53] A. Safonova, J. K. Hodgins, and N. S. Pollard, "Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces.," in *ACM Transactions on Graphics (TOG) (Vol. 23, No. 3, pp. 514-521)*, 2004.

[54] H. J. Shin and J. Lee, "Motion synthesis and editing in low-dimensional spaces.," in *Computer Animation and Virtual Worlds, 17(34), 219-227*, 2006.

[55] F. Luisier, T. Blu, and M. Unser, "Image denoising in mixed poissongaussian noise," *IEEE Transactions on Image Processing*, 2011.

[56] R. Verma and J. Ali, "A Comparative Study of Various Types of Image Noise and Efficient Noise Removal Techniques," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 10, pp. 2277–128, 2013.

[57] D. J. Sturman, "A Brief History of Motion Capture for Computer Character Animation,"

[58] J. Xiao, Y. Feng, M. Ji, X. Yang, J. J. Zhang, and Y. Zhuang, "Sparse motion bases selection for human motion denoising," *Signal Processing*, 2015.

[59] R. Khadem, C. C. Yeh, M. Sadeghi-Tehrani, M. R. Bax, J. A. Johnson, J. N. Welch, E. P. E. Wilkinson, and R. Shahidi, "Comparative Tracking Error Analysis of Five Different Optical Tracking Systems," *Comp Aid Surg*, vol. 5, pp. 98–107, 2000.

[60] H. Lou and J. Chai, "Example-based human motion denoising," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 5, pp. 870–879, 2010.

[61] R. Yan, L. Shao, and Y. Liu, "Nonlocal hierarchical dictionary learning using wavelets for image denoising.," *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, vol. 22, no. 12, pp. 4689–98, 2013.

[62] A. Buades and B. Coll, "A Non-local Algorithm for Image Denoising," *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, no. 0, pp. 60–65, 2005.

[63] S. Vaseghi in *Advanced Digital Signal Processing and Noise Reduction.*, (Chichester, West Sussex), pp. 35–50, John Wiley, 2000.

[64] I. Lemahieu, "Technique for Medical Imaging," no. February 2016, 2002.

[65] I. Akhter, T. Simon, S. Khan, I. Matthews, and Y. Sheikh, "Bilinear spatiotemporal basis models," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 2, p. 17, 2012.

[66] G. Deng and L. Cahill, "An adaptive Gaussian filter for noise reduction and edge detection," *IEEE Conference Record Nuclear Science Symposium and Medical Imaging Conference*, no. JANUARY 1993, pp. 1615–1619, 1993.

[67] H. J. Blinchikoff and A. I. Zverev, "Filtering in the time and frequency domains," 1986.

[68] S. Butterworth, "Experimental wireless and the wireless engineer," *Wireless Eng*, vol. 7, p. 536, 1930.

[69] S. Mallat, "A wavelet tour of signal processing," pp. 391–444, 1999.

[70] M. Forouzanfar, H. A. Moghaddam, and S. Ghadimi, "Locally adaptive multiscale bayesian method for image denoising based on bivariate normal inverse gaussian distributions," *International Journal of Wavelets, Multiresolution and Information Processing*, vol. 6, no. 04, pp. 653–664, 2008.

[71] A. K. Seghouane, "An adaptive bayesian wavelet thresholding approach to multifractal signal denoising," in *Proceedings of the 6th Nordic Signal Processing Symposium-NORSIG*, vol. 2004, Citeseer, 2004.

[72] S. G. Chang, B. Yu, and M. Vetterli, "Adaptive wavelet thresholding for image denoising and compression," *Image Processing, IEEE Transactions on*, vol. 9, no. 9, pp. 1532–1546, 2000.

[73] G. Welch and G. Bishop, "An introduction to the kalman filter," 1995.

[74] R. Hyndman and R. D. Snyder, "Kalman filter," 2001.

[75] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 12, no. 7, pp. 629–639, 1990.

[76] W. Huang, "Computation of eigenvalue problems with anisotropic diffusion operators," in *Proceedings of the international conference on numerical analysis and aplied mathematics 2014 (ICNAAM-2014).*, vol. 1648, p. 020008, AIP Publishing, 2015.

[77] A. Danielyan, V. Katkovnik, and K. Egiazarian, "Bm3d frames and variational image deblurring," *Image Processing, IEEE Transactions on*, vol. 21, no. 4, pp. 1715–1728, 2012.

[78] J. M. Wang, D. J. Fleet, and A. Hertzmann, "Gaussian process dynamical models for human motion," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 30, no. 2, pp. 283–298, 2008.

[79] L. Li, J. Mccann, N. Pollard, C. Faloutsos, L. Li, J. Mccann, N. Pollard, and C. Faloutsos, "Bolero: A principled technique for including bone length constraints in motion capture occlusion filling," in *In Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2010.

[80] L. Li, J. Mccann, N. Pollard, and C. Faloutsos, "Dynammo: Mining and summarization of coevolving sequences with missing values."

[81] B. A. Olshausen and D. J. Fieldt, "Sparse coding with an overcomplete basis set: a strategy employed by v1," *Vision Research*, vol. 37, pp. 3311–3325, 1997.

[82] Autodesk, Inc., "Autodesk motionbuilder," version 2016.

[83] D. Raghuvanshi, S. Hasan, and M. Agrawal, "Analysing Image Denoising using Non Local Means Algorithm," *International Journal of Computer Applications*, vol. 56, no. 13, pp. 7–11, 2012.

[84] J. Salmon, "On two parameters for denoising with non-local means," *Signal Processing Letters, IEEE*, vol. 17, no. 3, pp. 269–272, 2010.

[85] B. Goossens, H. Luong, A. Pizurica, and W. Philips, "An improved non-local denoising algorithm," *2008 International Workshop on Local and Non-Local Approximation in Image Processing*, 2008.

[86] A. Dauwe, B. Goossens, H. Luong, and W. Philips, "A fast non-local image denoising algorithm," *Proceedings of SPIE - The International Society for Optical Engineering*, 2008.

[87] Y.-L. Liu, J. Wang, X. Chen, Y.-W. Guo, and Q.-S. Peng, "A robust and fast non-local means algorithm for image denoising," *Journal of Computer Science and Technology*, vol. 23, no. 2, pp. 270–279, 2008.

[88] K. Huang, D. Zhang, and K. Wang, "Non-local means denoising algorithm accelerated by gpu," in *Sixth International Symposium on Multispectral Image Processing and Pattern Recognition*, pp. 749711–749711, International Society for Optics and Photonics, 2009.

[89] S. B. Gueye, K. Talla, and C. Mbow, "Solution of 1d poisson equation with neumann-dirichlet and dirichlet-neumann boundary conditions, using the finite difference method," *Journal of Electromagnetic Analysis and Applications*, vol. 6, no. 10, p. 309, 2014.

[90] P. K. Kitanidis, "Generalized covariance functions in estimation," *Mathematical Geology*, vol. 25, no. 5, pp. 525–540.

[91] K. Rao and P. Yip, *Discrete Cosine Transform: Algorithms, Advantages, Applications*. Academic Press, 1990.

[92] C. Ding, H. D. Simon, R. Jin, and T. Li, "A learning framework using green's function and kernel regularization with application to recommender system," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 260–269, ACM, 2007.

[93] A. K. Chandra, P. Raghavan, W. L. Ruzzo, and R. Smolensky, "The electrical resistance of a graph captures its commute and cover times," in *Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing*, STOC '89, (New York, NY, USA), pp. 574–586, ACM, 1989.