

# Example-Based Approach For Real-Time Rigged Character Animation

Ruoqing Sun

Technische Universiteit Delft



casual



turn around



walk



forlone



jumping



surprised!



running forward



running back



# EXAMPLE-BASED APPROACH FOR REAL-TIME RIGGED CHARACTER ANIMATION

by

**Ruoqing Sun**

in partial fulfillment of the requirements for the degree of

**Master of Science**  
in Computer Science

at the Delft University of Technology,  
to be defended publicly on Wednesday November 15, 2017 at 12:30 PM.

Supervisor:	Dr. K. A. Hildebrandt	
Thesis committee:	Prof. Dr. E. Eisemann,	TU Delft
	Dr. K. A. Hildebrandt,	TU Delft
	Dr. Jun Wu,	TU Delft

An electronic version of this thesis is available at <http://repository.tudelft.nl/>.



## ABSTRACT

We present a system that allows users to generate animations for 3D rigged geometry easily and quickly using just a regular consumer Microsoft Kinect. The interactive system allows diverse meshes to be animated through the interaction of human body. Traditionally, creating expressive and convincing animation is a challenging and laborious. Our system is a real-time performance-based puppetry application for any rigged object, which makes the animation generation more playful. This paper describes the system in full, explaining the generation procedures, highlighting the technical contributions, and demonstrating many examples.



# CONTENTS

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Contribution . . . . .	2
1.3	Structure of the Document . . . . .	2
<b>2</b>	<b>Background</b>	<b>3</b>
2.1	Related Work . . . . .	3
2.1.1	Keyframing . . . . .	3
2.1.2	Performance animation . . . . .	4
2.1.3	Deformation Transfer . . . . .	5
2.2	Related Concepts . . . . .	5
2.2.1	Skinning . . . . .	5
2.2.2	Rigging. . . . .	6
2.2.3	Mesh Deformation. . . . .	6
2.2.4	Skeleton Subspace Deformation . . . . .	7
2.2.5	Microsoft Kinect . . . . .	7
<b>3</b>	<b>Overview of the System</b>	<b>11</b>
3.1	User Scenario . . . . .	11
3.2	System Design . . . . .	12
<b>4</b>	<b>Technologies</b>	<b>13</b>
4.1	Goal. . . . .	13
4.2	Required Operations . . . . .	13
4.3	Method . . . . .	13
4.4	Example-based Projection . . . . .	14
4.4.1	Linear Projection. . . . .	14
4.4.2	Positive Constrained Projection . . . . .	15
4.4.3	L1 Regularization . . . . .	15
4.5	Shape Interpolation. . . . .	16
4.5.1	Linear Blending . . . . .	16
4.5.2	Forward Kinematics . . . . .	16
4.6	Optimization . . . . .	18
<b>5</b>	<b>Experimental Results</b>	<b>19</b>
5.1	Example-based projection . . . . .	19
5.2	Shape Interpolation. . . . .	26
5.3	System . . . . .	27
5.4	Results . . . . .	29
<b>6</b>	<b>Discussion, Limitations and Future Work</b>	<b>31</b>
6.1	Discussion . . . . .	31
6.2	Limitations and Future Work . . . . .	32
<b>7</b>	<b>Conclusion</b>	<b>33</b>
<b>A</b>	<b>Appendix</b>	<b>35</b>
A.1	Experimental Poses . . . . .	35
	<b>Bibliography</b>	<b>37</b>





# 1

## INTRODUCTION

### 1.1. MOTIVATION

Animation is a powerful tool applied in video games and movies to give life to virtual characters. However, the process is quite time-consuming and requires that every user is experienced in the creation of computer animation. Imagine you would like to generate a 3D animation of a rigged object. You need to think about deformation, skinning, and key-framing. Even if you are experienced in the process, you might spend hours to days manipulating software such as Blender or Maya. Thanks to new consumer input devices, even if you are a novice user and have little knowledge in computer graphics, you have the opportunities to interactively animate virtual characters. For instance, Microsoft Kinect outputs the 3D positions of joints in human body skeleton. Based on traditional skeleton-based animation pipelines, we can map user's joint data to a rigged virtual character as long as the virtual character is humanoid, to control the movements of the character. However, cases are that often characters without human structure need to be manipulated. For example, horses can be rigged with skeletons, but they have a different body shape as humans. Ogre's arms are quite long thus it does not have the same body proportions as ordinary people. Characters like octopus even have a different number of limbs towards human, so even if a skeletal mapping could be defined, traditional methods through direct human skeletal control would be impossible. In these cases, mapping and transferring human motions towards arbitrary rigged characters would require considerable manual work, and with all those limitations, it's hard to map arbitrary motions.

In general, 3D animations are implemented with virtual articulated figures, which means, hierarchical limbs are defined within the body of virtual characters to enable the 3D model to deform locally. These ideas in computer graphics come directly from the field of robotics. Thus if we can have methods of semantically transferring human motions to virtual characters, we can use the similar way to control robot characters, no matter they are in the shape of human, animals or fictitious characters. Existing character control algorithms or systems cannot map human motions to arbitrary target motions, which we will categorize and discuss current methods in Section 2.1. But there should be possible to map a human movement to an octopus, ogre, a flour sack without explicitly specify all spacial correspondences. So our goal is to introduce a new option for synthesizing 3D animation: a system that enables animating 3D rigged geometry through human body interactions naturally in real-time.

Creating expressive and convincing animation is challenging for various reasons. We aim to animate virtual characters and let them behave in their native ways, while the user performs a broad range of continuous input poses. The large differences in the body proportions, structure topologies, and movement behaviors of humans from those of our target characters lead to several challenges that have not been sufficiently considered by the previous puppetry systems.

Firstly, *which part of target creature corresponds to a specific feature of human body?* There is no unique method to match human body part towards specific feature of a target object. For example, animals usually have a different body structure and number of joints from those of a human being. Thus designing mappings between a human body and rigged object that enable efficient and meaningful human-creature control is a non-trivial task.

Secondly, even if we have an intuitive and practical coupling of human body parts and target objects, ***how to generate natural and plausible creature animations from the input body motions of users?*** For example, an elastic bar has their patterns of stretching, shrinking and twisting that human bodies cannot possibly imitate. Furthermore, animals like horses with four legs have a different way of walking from human beings.

Finally, ***how to design the system to make it real-time?*** As a real-time application, the system has to generate desirable animations whenever receiving input body motions of Kinect. Thus we cannot use complex models or equations that are hard to solve in the application.

## 1.2. CONTRIBUTION

This paper proposes a real-time animation puppetry application for any kind of rigged object, especially non-humanoid characters. It enables intuitive pose reconstruction of arbitrary characters. And can run in real-time on common hardware. Most importantly, the application uses the human body as input and thus enables novice user to create interesting and personal 3D animations. The system requires some preprocessing steps to make pose couplings. After the registration process, equipped with a Microsoft Kinect, users easily create individual and expressive animation motions for virtual characters. Our main contributions include:

1. Firstly, we formulate a brand new example-based methodology to semantically transfer the body movements towards target character, to generate 3D rigged character animations in real-time. Compared to the previous applications, our system doesn't require the users input any previously generated motions into the system. Just based on some example poses, the users can generate unique animation in real-time for the target character;
2. Secondly, we present a new interactive animation system that allows any users to generate plausible animations from body position scanning data from Microsoft Kinect;
3. Furthermore, we apply several methods to create mesh animation and compare their results. Users can adjust several parameters for different types of mesh objects;
4. Finally, we demonstrate our results operating on input meshes of different morphologies, body proportions, number of degrees of freedom (DOF) and size. The system enables new possibilities for human-character animation previously unavailable to novice users.

## 1.3. STRUCTURE OF THE DOCUMENT

In this paper, we will first introduce some related works and concepts in the field of computer animation (see Section 2). Following this section, we will present the system design and user scenarios (see Section 3.1), where the readers can find how to interact with our system. Section 4 explains the technologies involved in this system. Section 5.1 evaluates our methods in the aspect of projection, interpolation, and the system in the whole. We will provide some examples to demonstrate the effectiveness of our system. In Section 6, we will discuss the limitations, future work and we will draw conclusions at last (see Section 7).

# 2

## BACKGROUND

### 2.1. RELATED WORK

The major dominant approaches for computer animation are keyframing and performance animation.

#### 2.1.1. KEYFRAMING

2D hand-drawn animation deals with a sequence of two-dimensional drawings that simulate motion. 3D computer animation applies the similar concept and involves creating a three-dimensional model on the computer. Figure 2.1 illustrates the process of creating character motion: filmmakers set keyframe poses and then let the computer generate the in-between frames[1]. With keyframing, the only keyframe poses need to be built. The huge number of DOFs can be carefully handled by users one after another. Current animation software like *Blender* and *Maya* contains complicated implementations of keyframing techniques, which create a huge barrier for novice users. Those applications and interfaces are quite powerful, but there are several drawbacks:

1. Making motions look organic and natural is a demanding task.
2. The adjusting process is quite time-consuming, even for professional animations.

Besides well-developed tools such as *Blender* and *Maya*, there are other interfaces and systems make use of the keyframing animation techniques. Most applications focus on making the posing process easier and quicker. HoloSketch is a virtual reality-based 3D geometry creation and manipulation tool with 3D user input[2]. Users can select virtual objects and apply 3D transformations to pose and animate them. The system allows novice users to manipulate the virtual character easily, but it only allows simply rotations and rotations to the whole objects instead of each part. So it is not possible to generate more complicated poses. [Li et al. \[3\]](#) and [Chao et al. \[4\]](#) enables animations to be create by sketching the hand-drawn keyframe postures of the articulated object. They allow users to define the required motion by sketching several motion strokes over a drawn character, which requires less effort and extends the users' expressiveness. But the system requires the ability to sketching, which is not friendly for novice users. Moreover, the system can only

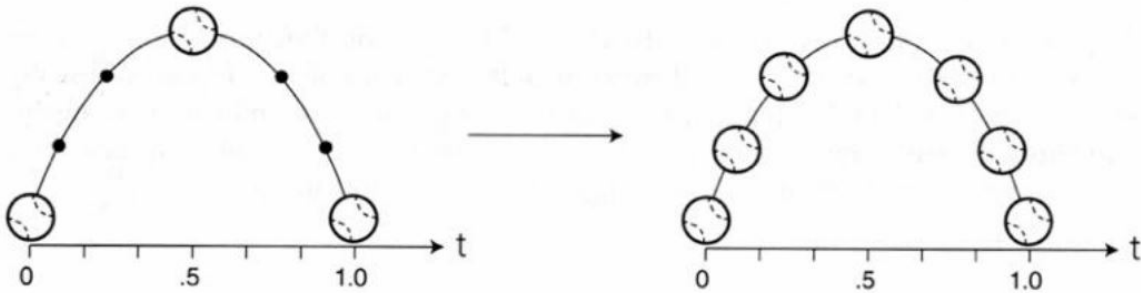


Figure 2.1: Keyframes and generated inbetween frames of a moving ball.



Figure 2.2: : Users wearing a few retro-reflective markers control the full-body motion of avatars by acting out the motion in front of two synchronized cameras. From left to right: walking, running, hopping, jumping, boxing, and Kendo (Japanese sword art).

apply motions from the motion clip database. Eitsuka and Hirakawa [5] presented a system for generating keyframes of virtual objects with a touch operation in AR-based 3D space. The manipulation of the virtual object is carried out by the user's finger in front of the camera on a mobile phone. Similar to HoloSketch[2], the users can only perform some limited transformations (scaling, translation, and rotation) to the whole objects due to the difficulty and limitations in AR-based control. Jacobson *et al.* [6] presented a physical modular input device that enables users to design the keyframe directly poses rather than using virtual interfaces by manipulating the degrees of freedom in composing parts. Single physical modular can be assembled and manipulated, making the system flexible and easy to control. Based on similar principle, stop motion is an animation technique that physically manipulates an object so that it appears to move on its own[7]. Filmmakers physically move the object with postures in small increments, creating the illusion of movement when playing the keyframes at a fast pace.

### 2.1.2. PERFORMANCE ANIMATION

In contrast to keyframing, performance animation systems rely on the user's physical timing and coordination skills[8]. One of the most well-known technique is motion capturing: filmmakers map a performer's motions towards a virtual character[9]. In most cases, direct mapping is applied to create realistic and natural human-like animations[10]. Figure 2.2 illustrates the process of generating the animation. With the movements of actors, the virtual character can perform various motions, such as walking, running, and hopping. There are mostly two drawbacks to motion capturing:

1. Direct mapping is applied to create realistic and natural human-like animations.
2. Only animations that are physically possible for a human can be generated by motion capturing. For example, it is not feasible for humans to perform stretching and squeezing.

Most of the recent techniques of performance animation aim to overcome these limitations by not applying direct mapping. *Creature Features* [11] focuses on non-humanoid creature animations. Users are tracked, and their movements are mapped onto the creature's movements. A combination of direct feature mapping and motion coupling enables the generation of natural creature motion, along with intuitive and expressive control. However, the system relies on some predefined animations to puppeteer and direct feature mapping remains a limitation to control creatures like spiders. Kim *et al.* [12] came up with their idea from traditional puppeteers who control devices such as rods or strings to move the body, head, limbs, and in some cases the mouth and eyes of puppets. The system uses a haptic input device and virtual physically simulated strings to mimic the process of traditional puppetry for synthesizing complex character animations. *KinÊtre* [13] aims at animating arbitrary object through deformation according to the user's performance using a regular consumer Kinect. However, in the registration phase, the user needs to physically move such that the rendered skeleton interpenetrates the mesh, which makes controlling four-leg figures impossible. Similarly, Vögele *et al.* [14] allows animate non-humanoid object from the motions of multiple actors. But these two systems cannot guarantee that the virtual object is animated naturally and expressively. *Creature Teacher*

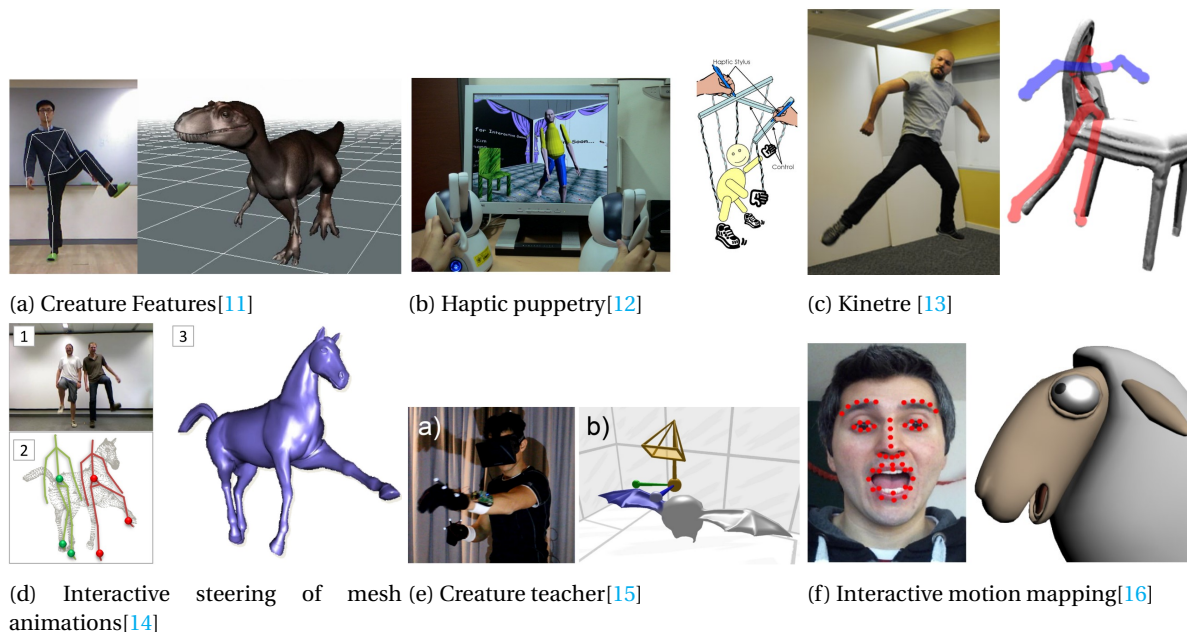


Figure 2.3: Examples of recent research in performance animation methods.

[15], with two custom-made pinch gloves and Oculus Rift, allows users to animate non-humanoid rigged objects through bimanual interaction paradigm. Users can select parts of the model with the left hand and then manipulate them with the other hand. But the system only tracks periodic movements. Thus non-cyclic animation cannot be synthesized by this method. Rhodin *et al.* [16] allow for puppeteering using arbitrary source motions, such as skeleton positions from Kinect and hand gestures from Leap Motion. Their source and target are not necessarily skeletal motions. Similar to our system, the user needs to define a small number of pose correspondences between source and target motion. But the system can only return the closest existing target pose in motion when given a new input pose. The input of the system includes motions from humans and also the virtual character. While performance animation can be advantageous regarding creation speed, several challenges still emerge. For example, input human motion must be physically possible for actors to perform. Otherwise predefined animations need to be deployed.

### 2.1.3. DEFORMATION TRANSFER

Deformation transfer, referring to transferring existing mesh deformation from one character to another, sometimes is another way to accelerate the process of synthesizing mesh animation. Under some circumstances, it is quite useful to preserve the semantic characteristics of the motion instead of its literal deformation. Baran *et al.* [17] enables semantic deformation transfer with a shape space that enables interpolation and projection. The system allows creating a correspondence between the shape spaces of the two characters. Users can generate new poses and animations for target characters without target motion as inputs. Ben-Chen *et al.* [18] proposed an automatic way to create a cage by wrapping the shape with many small cubes, which is still hard to manipulate for the user. Their system only enforces pure rotations on the medial axis of the shape, which restrict the ways for users to synthesize expressive animations. The system requires the target object to have similar topology as the source object to make it meaningful to transfer the motions. Moreover, some high-frequency details of the source deformation may be lost since the method projects the deformation into a low-dimensional linear subspace.

## 2.2. RELATED CONCEPTS

### 2.2.1. SKINNING

Skinning is the process of controlling the deformation of an object by controlling a set of primitives. The techniques of skinning are widely applied recently in the industry of video games and virtual animations. Direct methods compute deformations of virtual characters based on closed-form expressions. There is no process of optimization involved. Thus direct methods are popular because of its real-time performance[19].



Linear blend skinning (LBS) is well-known as the fundamental method for shape deformation applications. [Lewis et al.\[20\]](#) provided a full equation for LBS (mentioned as shape deformation and shape interpolation in this paper) as following formula:

$$v'_i = \sum_{j=1}^m w_{i,j} T_j v_i \quad (2.1)$$

Where  $v_i$  contains the rest position of vertex  $i$ . It is assumed that the connectivity of the mesh will not change during deformation. Only positions of vertices will be computed and modified. Then we store the value in  $v'_i$ .  $T_j \in \mathbb{R}^{3 \times 4}$  represents the transformations of bone  $j$  aligning its current position with the rest position.  $w_{i,j}$  stands for the influence of bone  $j$  to vertex  $i$ . Traditionally those weights are manually adjusted by designers, which is time-consuming. By default, the sum of weights of all handles for each vertex is one. At a high level, the equation takes a weighted average over the transformations of all bones as the conversion of each vertex. The method works well when the transformations at adjacent handles are similar. But issues would arise in the circumstances where the shifts of neighboring handles differ too much. That is because the combination of rigid transformations  $SO(3)$  is not linear. Thus the combined transformation we get from Equation 2.1 is not promised to be rigid. The phenomenon is called candy-wrapper artifact.

### 2.2.2. RIGGING

3D rigging is the process of creating a skeleton for a 3D mesh so that it can animate freely. Most commonly, 3D models are rigged before they can move because if character models don't have a rig, designers will waste a lot of time adjusting the positions of vertices. With the help of skeleton, we can calculate the influence of bones on each vertex. So the designers can move. Unsurprisingly, the usage of skeleton for animating characters has a long history in computer animation [21].

Traditionally, riggers construct a 3D rig as a hierarchy of line-segments, which forms a graphical tree. In this tree, nodes are named joints connecting two or more lines, which are called bones. The parent-child relationship is inherited as the skeleton is constructed. This parent-child tree structure can be used later to define a forward kinematics tree when calculating pose transformations from relative bone rotations.

At times, the rigging process can become technical and seem overwhelming. To accelerate this process, modern research has focused on automatically computing a skeletal tree given a 3D mesh [22]. In fact, a shape's topology and morphology imply the topology of its rig inside. There are mainly two principal questions in automatic skeleton construction: determining the skeletal topology and attaching the skeleton to the 3D mesh [23]. The topological problem focuses on how many bones make up the skeleton and how they are structured and connected. The attaching question asks where the skeleton is situated in space and how skeleton can control the mesh movement.

### 2.2.3. MESH DEFORMATION

Our system is related to a long series work on mesh and skeleton deformation. Applications like animation and special effects require the deformation and manipulation of complex geometric models. These models, like real-world geometry, often contain details at various scales. Controlling complex models with high resolutions can be difficult. So early preserving mesh deformation focus on multi-resolution techniques to reduce the complexity of controlling the meshes [24] [25]. They are applying algorithms combined with existing hierarchy methods to build a subdivision for meshes with arbitrary topology. These hierarchical techniques can decrease the complexity and preserve the details in mesh deformation through the technique of local shape control.

To achieve more global and complex deformation, many researchers switched to cast deformation problem as an energy minimization problem [26] [27]. Typically, the energy functions enable detail-preserving and contain position-constraint terms for direct manipulation. [26] reformulate the process of least squares fitting of the Euclidean geometry to the given Laplacian coordinates. In their fitting system, they compute transformation for each vertex, which is applied to its respective Laplacian coordinate. However, these existing techniques do not scale, as the optimizations involved are mostly nonlinear and require Gauss-Newton iterations, which is slow-converging.

Recent researchers focus on deformation via a mesh-based inverse kinematics. They provide the system with a series of example meshes to let the system learn the space of natural deformations to improve the efficiency by restricting the results to only constraint ones [28] [29]. Our system apply a different method for the similar problem: let the system deform the mesh every frame by learning from some example poses.

#### 2.2.4. SKELETON SUBSPACE DEFORMATION

Skeleton Subspace Deformation (SSD) have been applied in the graphics industry for a long time as an efficient and effective representation of animation [30] [31]. However, they achieve their success by restricting deformation to a particular subspace for efficiency. However, they may contain some artifacts like candy-wrapper artifacts as well as tedious tweaking of vertex weights. On the other hand, they each have advantages.

#### 2.2.5. MICROSOFT KINECT

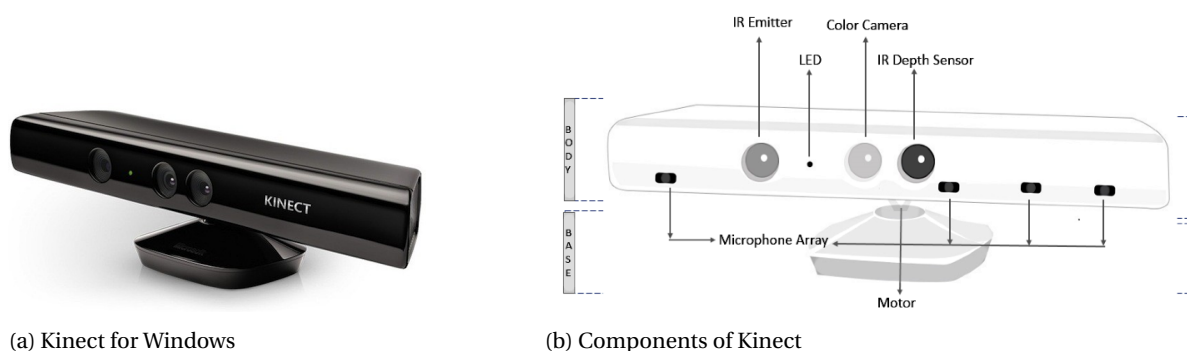


Figure 2.4: The exterior and internal functions of Kinect for Windows

Microsoft Kinect is a series of motion-detecting input device firstly developed by Microsoft in 2010. It allows users to interact with a computer without the help other game controller devices, just by motion and audio commands. The device is mainly designed for Xbox 360 and Windows PCs in developing interacting games.

#### COMPONENTS

Kinect is an aggregate device equipped with color cameras, depth sensors and a set of microphones, altogether in a horizontal box. A small motor works as the base, letting the device to be tilted in a horizontal direction. The Kinect contains four main components: a microphone array, an infrared emitter, an infrared receiver and a color receiver, illustrated in Figure 2.4b.

There is no central processing unit (CPU) contained in this device. Only a digital signal processor (DSP) processes the signal of the microphone array. So the Kinect driver can process data on the PC side. The driver can be installed on Windows 7, 8, Vista and 10, and runs on a 32- or 64-bit processor. You will need a least 2 GB of RAM and a dual-core 2.66-GHz or faster processor for the Kinect driver.

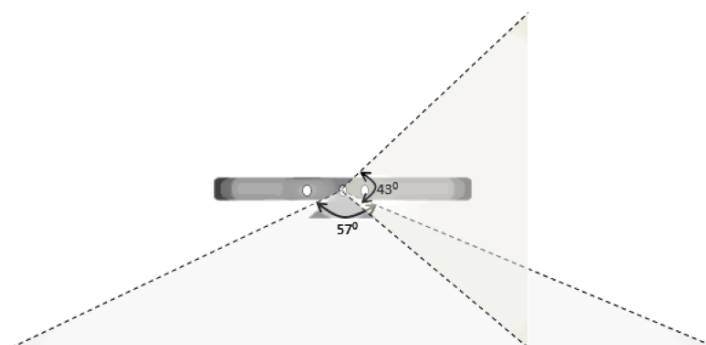


Figure 2.5: Range of Kinect Sensor

### THE COLOR CAMERA

The color camera inside is able to capture and stream the color video data. The camera can detect red, blue and green colors and create image frames. The range of the Kinect Sensor is 43 degrees in vertical direction and 57 degrees in the horizontal direction illustrated below. So Kinect can capture video streams using the following resolutions and frame rate:  $640 \times 480$  at 30 frames per second (FPS) using red, green, and blue (RGB) format,  $1280 \times 960$  at 12 FPS using RGB format, and  $640 \times 480$  at 15 FPS using YUV (or raw YUV) format.

### IR EMITTER AND IR DEPTH SENSOR

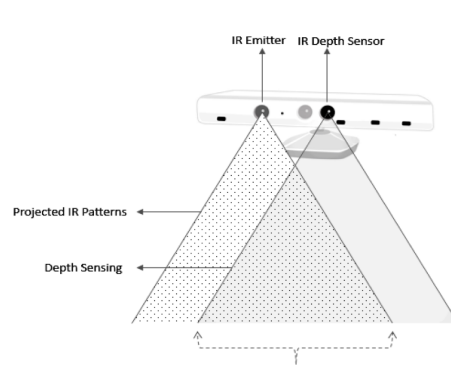


Figure 2.6: Demonstration of how the depth sensor works

The depth sensor for Kinect consists of an IR emitter and IR depth sensor. The position of the emitter and the sensor on the Kinect device is shown in Figure 2.6. The IR emitter would constantly emit infrared light and spread “pseudo-random dot” which is invisible to us over everything within reachable range. The IR depth sensor would capture those dots and convert them into depth information by calculating the distance between the sensor and the dots. All resolutions use a frame rate of 30 FPS. The resolutions supported are  $640 \times 480$ ,  $320 \times 240$  and  $80 \times 60$ .

### LIMITS

The optical lenses, the base of the sensor, have some limitations. Figure 2.7 shows the range where the sensor can work smoothly. Horizontal viewing angle is  $57^\circ$ . Vertical viewing angle is  $43^\circ$ . User distance (standard mode) is from 1.2m to 4m. User distance (near mode) is from 0.4m to 3m. Depth range is 400mm to 8000mm. Temperature range is 5 to 35 degrees Celsius (41 to 95 degrees Fahrenheit).

### SKELETON TRACKING

Microsoft Kinect can track the skeleton position of the user within its working range in real time. Figure 2.8 shows the 20 joints whose 3D coordinate positions can be tracked by the sensor. A tracked skeleton provides detailed information about the skeleton, including orientation and 3D coordinates for each skeleton joint. In this project, we make use of only the skeleton information when 20 joints can all be successfully tracked.



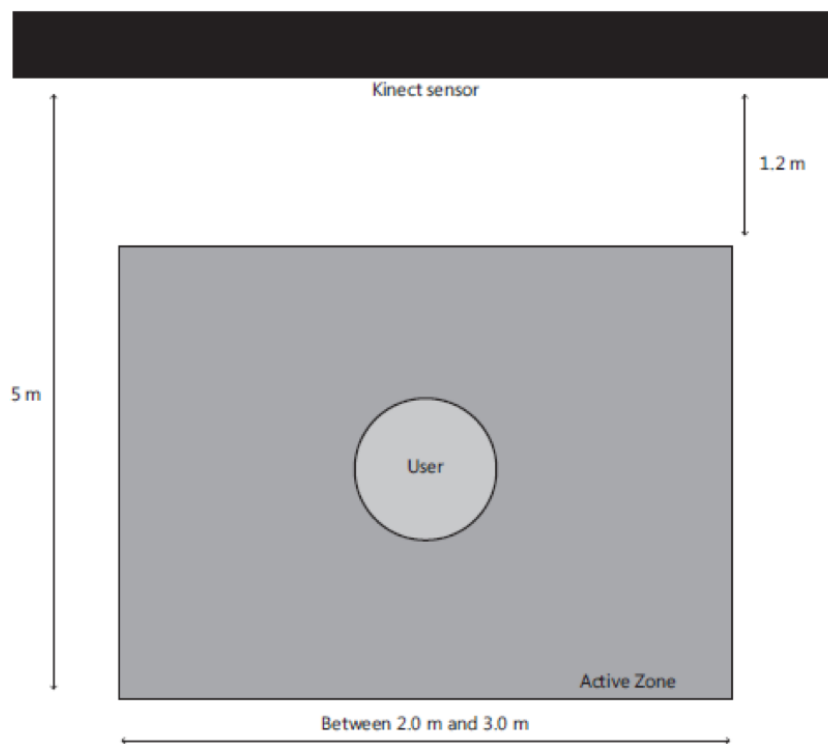


Figure 2.7: The scanner limit for Kinect

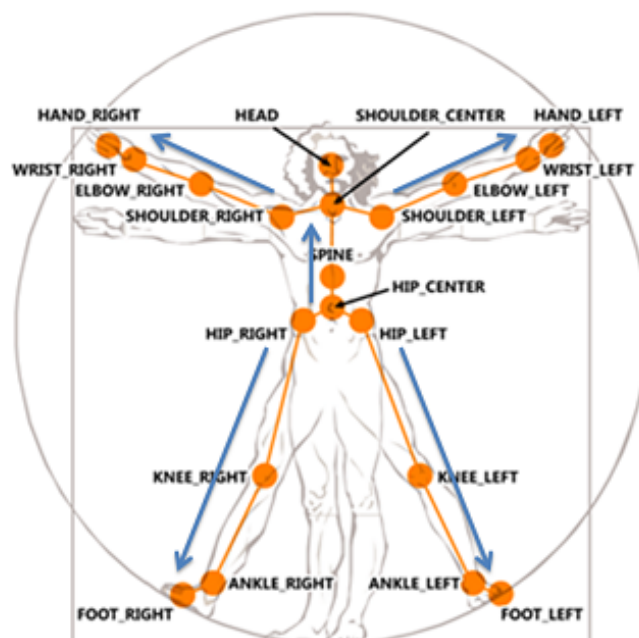


Figure 2.8: Kinect skeleton position



# 3

## OVERVIEW OF THE SYSTEM

### 3.1. USER SCENARIO

Before coming to the details of our system architecture, we begin with a motivating scenario, to illustrate the user experience and let the readers have a better overview of the project. Since our system is intended for applications and games whose content creation is essential, giving users the ability to express individualism is the critical point. In this example of user scenario, we will animate a flour sack. Figure 3.1 depicts the process of standing on the hand animation. The user can add individual detail to the users since the animation is performance-based. Imaging the user would like to animate an existing rigged 3D geometry, a flour sack, into a 3D game. Following are the steps for the user to go through to synthesize the mesh.

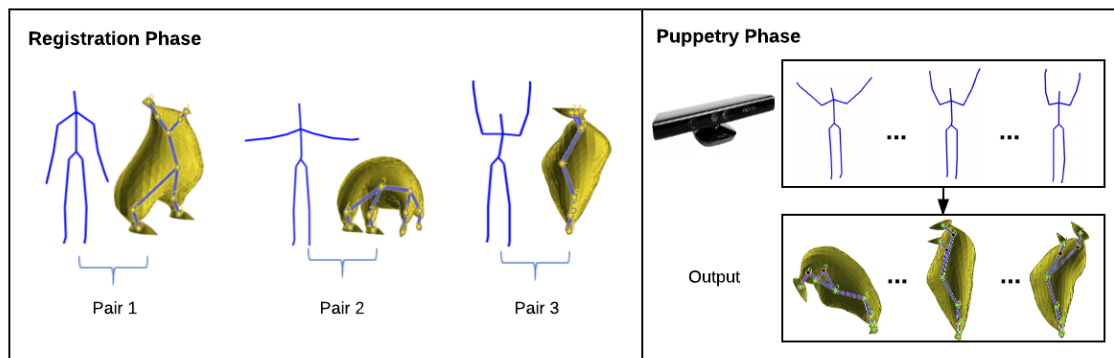


Figure 3.1: User scenario for using the system. On the left is the registration phase, where users can make some couplings of standard poses from the source object to the target object. On the right is the puppetry phase where the user can continuously give pose input to puppetry phase and the system will generate animations for target object in real-time.

**Step 1.** He picks up a Kinect camera and place it at a certain height where it can image his full body. At this point, the system tracks a human skeleton. The user can physically move such that the system tracks the desired poses of the rendered skeleton.

**Step 2.** The user can then select the key poses they want to sample with the right character key poses.

**Step 3.** The user manually generate a key pose for the mesh. Press "register" button and pair the key poses of the source and the target object.

**Step 4.** After repeating step 2 and step 3 a couple of times, we have completed the *registration phase* and mapped several couplings of key poses.

**Step 5.** The last step is the *puppetry phase*, where the user can physically move in front of the camera to puppeteer the virtual character. The system will automatically translate his motion into realistic deformations of the previously static mesh.

Our method enables motions of the human body to transfer onto the mesh in real-time. The system provides a natural way for users to perform playful and unique animations of arbitrary rigged mesh that they want to bring to life in their 3D applications.

### 3.2. SYSTEM DESIGN

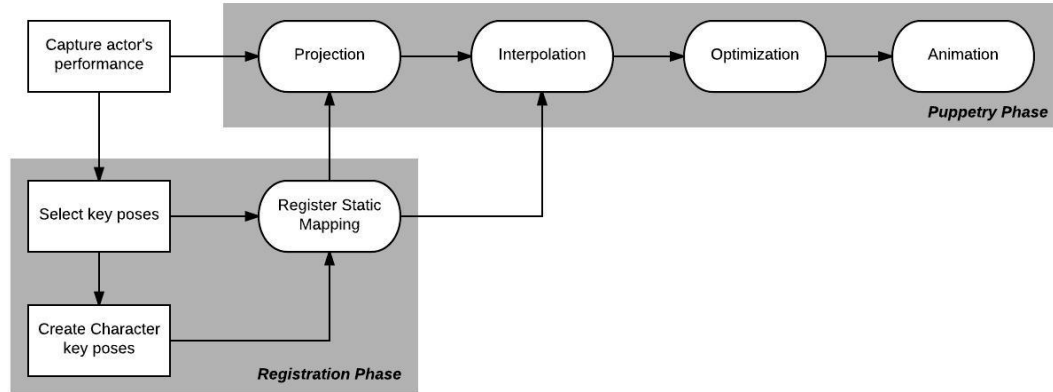


Figure 3.2: Overview of the System. Lower square is the procedure of registration phase. The upper right square is the process of puppetry phase. The rounded rectangle represents procedures that can be automatically processed by the system. The rectangle stands for the operations required from users.

Figure 3.2 illustrates the overview of our animation generation process. The manual blocks indicate the manual process, while the rounded rectangles represent automatic operations by the system. Our system can render both the human skeleton and articulated object side by side.

There are two phases of our system. One is registration phase, where the system learns static coupling between the source and target character. The other part is the puppetry phase, where the system generates the real-time animations. In the Registration Phase, we firstly capture the motions of an actor or actress performing the key poses they would like to behave. Then the actor selects a few key poses among the captured motion sequences as stand source postures. It is recommended that the poses should cover the space of the poses that would appear in the later captured motions. The last task for the users is to create some character pose corresponding to each of the selected poses for the source object. These couplings of crucial poses implicitly define the correspondence between the body parts of the human and character models, even if they have different topology and morphology. The system puts down the skeleton vertex position human key poses and maps them towards the linear and angular momenta of the character pose. In the puppetry phase, when the new input motion data comes in from Kinect, the system project the new pose into the stand pose space and generates the weight vector(Section 4.4). We will then obtain the interpolated poses, in either the form of transformation matrix or linear and angular momenta, for the target character(Section 4.5).

# 4

## TECHNOLOGIES

In this chapter, we will introduce the technologies that we apply to this project.

### 4.1. GOAL

Our source motion  $\mathbf{X}(t) = (\mathbf{x}_1, \dots, \mathbf{x}_N)$  is a 3D point sequence over time  $t$ , from tracking device Microsoft Kinect. Target character motions  $\mathbf{Y}(t) = (\mathbf{y}_1, \dots, \mathbf{y}_N)$  are mesh deformation sequences. The goal of our project is find the mapping  $\mathcal{M}$  that is able to transfer the source motion into the target motion:

$$\mathbf{X}(t) \xRightarrow{\mathcal{M}} \mathbf{Y}(t) \quad (4.1)$$

### 4.2. REQUIRED OPERATIONS

The proper choice of shape space is essential for deformation transfer between humans and virtual creature. Some existing mesh representations can satisfy our requirements[17]. We summarize some of them into an intuitive representation that enables semantic deformation transfer.

Two maps define a shape space for a specific mesh connectivity: an encoding map  $C : \mathbb{R}^{3n} \rightarrow \mathbb{R}^m$  that takes 3D vertex positions and outputs a coordinate vector; a reconstruction map  $C^{-1}$  that generates 3D vertex positions from a coordinate vector. The reconstruction of an encoding map  $C^{-1}$  should return the original 3D vertex positions. The deformation transfer relies on two primary operations in shape space:

- **Interpolation:** Given  $n$  example base poses  $\mathbf{p}_1, \dots, \mathbf{p}_n$  and  $n$  weights  $w_1, \dots, w_n$ , such that  $\sum_i w_i = 1$ , compute  $\sum_i w_i \mathbf{p}_i$  to reconstruct combination of the poses.
- **Projection:** Given  $n$  example base poses  $\mathbf{p}_1, \dots, \mathbf{p}_n$  and another input pose  $\mathbf{x}$ , compute  $n$  weights  $w_1, \dots, w_n$  that is able to reconstruct the input base pose  $\mathbf{x}$  by interpolating weighted base poses  $\sum_i w_i \mathbf{p}_i$ .

### 4.3. METHOD

A shape space that helps interpolation and projection facilitates semantic deformation transfer with the simple method stated in Algorithm 1. Together the projection and interpolation cover a linear mapping from the source shape space towards the target shape space. Using the following equation, we firstly encode the source motion  $\mathbf{X}(t)$  into source shape space to get the weight vector  $[w_1, \dots, w_n]$ . And the target motion  $\mathbf{Y}(t)$  then comes from the interpolation of example base poses in the target shape spaces:

$$\mathbf{X}(t) \xRightarrow{C_{src}} [w_1, \dots, w_n] \xRightarrow{C_{tgt}^{-1}} \mathbf{Y}(t) \quad (4.2)$$

**Algorithm 1** Motion Puppetry

- 1: Given  $n$  pairs of example poses, encode them into the source and target shape spaces using  $C_{src}$  and  $C_{tgt}$ .
- 2: Given a new source pose, encode it into the source shape space and use projection to express it as an affine combination of the source example poses with weights  $w_1, w_2, \dots, w_n$ .
- 3: Use these weights to interpolate corresponding target example poses in the target shape spaces and use  $C_{tgt}^{-1}$  to reconstruct the resulting pose.

However, this method sometimes have disadvantages. For example, the global transformations will affect the projection result (We will discuss this problem more in Section 5.1). We therefore separate global motions from character pose, so that we can eliminate the effects of global transformations. We parametrize a motion  $\mathbf{X}(t) = (\mathbf{x}_1, \dots, \mathbf{x}_N)$  into character pose feature vectors  $\mathbf{X}^{pose}(t) = (\mathbf{x}_1^{pose}, \dots, \mathbf{x}_N^{pose})$ , global translations  $\mathbf{X}^{trans}(t) = (\mathbf{x}_1^{trans}, \dots, \mathbf{x}_N^{trans})$ , and global rotations  $\mathbf{X}^{rot}(t) = (\mathbf{x}_1^{rot}, \dots, \mathbf{x}_N^{rot})$ . In this project, we use the orientation of spine as the orientation of the actor.

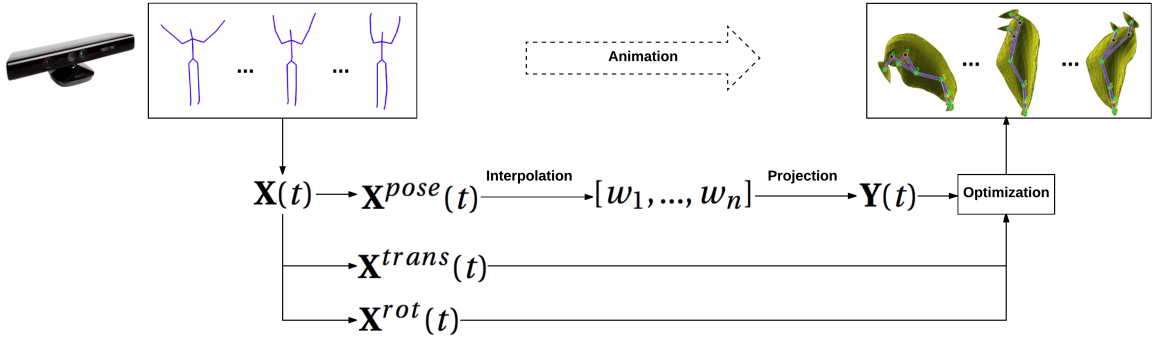


Figure 4.1: This image illustrate the our method in the puppetry phase, where we get current input pose  $\mathbf{x}(t)$  from kinect and output the animations  $\mathbf{y}(t)$  to the system.

#### 4.4. EXAMPLE-BASED PROJECTION

The projection process starts by obtaining the newest coordinates that correspond to a new human pose. The system will automatically project the pose into source shape space and generate  $n$  weights  $\mathbf{w} = [w_1, \dots, w_n]$ . We apply three methods to solve this projection problem: quadratic programming, positive constraint minimization and  $L_1$  norm.

##### 4.4.1. LINEAR PROJECTION

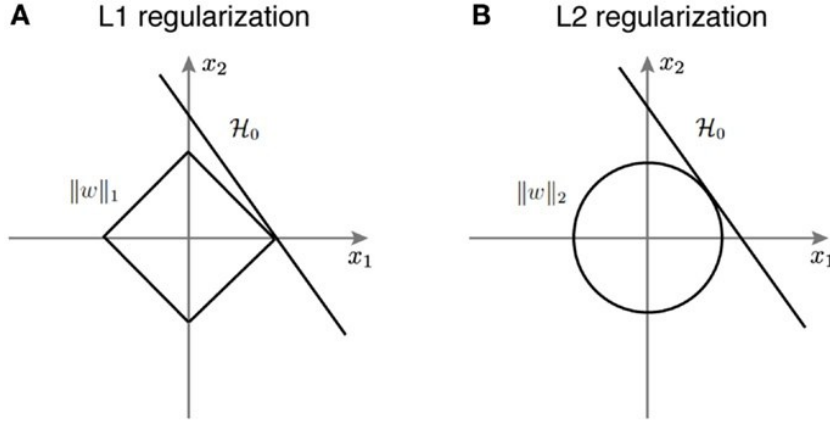
The first method we applied is quadratic programming. For a new human pose  $\mathbf{p}$ , we search for the best weight vector  $\mathbf{w} = [w_1, \dots, w_n]$  that can reconstruct a pose that is closest to the original pose. Ideally, the reconstructed pose should be identical to the original pose. Thus here we apply the  $L_2$  norm here to minimize the errors.

$$\begin{aligned} \min_{w_i} \|\mathbf{p} - \sum_{i=0}^n w_i P_i\|^2 \\ \text{while } \sum_{i=0}^n w_i = 1 \end{aligned} \quad (4.3)$$

The solution can be calculated through the following equation [32]:

$$\begin{bmatrix} \mathbf{w} \\ \lambda \end{bmatrix} = \begin{bmatrix} G & -A^T \\ A & 0 \end{bmatrix}^{-1} \times \begin{bmatrix} c \\ 1 \end{bmatrix} \quad (4.4)$$

where  $\mathbf{w} \in \mathbb{R}^n$  is the weight vector in the form of  $[w_1, w_2, \dots, w_n]^T$ .  $G \in \mathbb{R}^{n \times n}$  is an outer product matrix equal to  $\mathbf{xx}^T$ .  $c \in \mathbb{R}^n$  equals to  $\mathbf{x}^T \mathbf{p}$ .  $A \in \mathbb{R}^n$  is a unit matrix. We can compute outer product matrix  $G$  and the

Figure 4.2: Visualization about the reason of sparsity for adding  $L_1$  norm

inverse matrix of  $\begin{bmatrix} G & -A^T \\ A & 0 \end{bmatrix}$  for projection in advance in the source shape space. By pre-calculating the necessary inverse matrix, the whole process can be solved in real-time.

#### 4.4.2. POSITIVE CONSTRAINED PROJECTION

The second method we apply here is the convex optimization of a quadratic function with positive constraints. Sometimes we might want to eliminate the effect of extrapolation. So besides the method of convex optimization of a quadratic function with equality constraint, we add an inequality constraint  $w_i \geq 0$  to allow only positive weights in the solution.

$$\begin{aligned} \min_{w_i} & \left\| \mathbf{p} - \sum_{i=0}^n w_i \mathbf{p}_i \right\|^2 \\ \text{while } & \sum_{i=0}^n w_i = 1, w_i \geq 0 \end{aligned} \quad (4.5)$$

Compared to the unconstrained quadratic programming problem, the constrained problem is more difficult to solve. In this paper, we use MOSEK optimization software to solve this problem. We will introduce the implementations in detail in Section 5.4.

#### 4.4.3. L1 REGULARIZATION

In fact, sometimes we will be bothered by weights that are not sparse enough to choose the actual relevant poses. In practice, we prefer weights that can semantically project the latent pose rather than the weight vector that can reproduce the original pose accurately. Sparsity in the quadratic optimization problem can be enforced by adding a  $L_1$  regularization form into the quadratic energy. Figure 4.2 illustrates the reason why  $L_1$  can bring sparsity into the system.  $H_0$  is the constraint, which in our system represents the partition and unity relationship of the weight vector. The intersection of the intersection point is the solution to the system. In Figure 4.2 B  $L_2$  is a convex and  $H_0$  is tangential to the  $L_2$  norm. So the solution is involved with both  $x_0$  and  $x_1$ , which is not sparse. However, in Figure 4.2 A, because of the attributes of  $L_1$  norm, the intersection points will either be on  $x_1$  axis or  $x_2$  axis, which enforce sparsity in the results.

In this method, we decide to add a  $L_1$  regularization  $\lambda \|w\|_{L_1}$  into the minimization energy. In Equation 4.6,  $\lambda$  is the parameter to balance the influence of  $L_1$  norm and the quadratic energy. It allows extrapolation but adds sparsity into the selection of weight vectors.

$$\begin{aligned} \min_{w_i} & \left\| \mathbf{p} - \sum_{i=0}^n w_i \mathbf{p}_i \right\|^2 + \lambda \|w\|_{L_1} \\ \text{while } & \sum_{i=0}^n w_i = 1 \end{aligned} \quad (4.6)$$

We cannot find existing libraries to solve this problem. Thus we implement the solution provided by previous researchers [33] [34]. We firstly put everything into matrix form. Then Equation 4.6 is equal to:

$$\begin{aligned} \min_{\mathbf{w}} f(\mathbf{w}) &= \mathbf{w}^T Q \mathbf{w} + q^T \mathbf{w} + \lambda \|\mathbf{w}\|_{L_1} \\ \mathbf{1}^T \mathbf{w} &= 1 \\ \mathbf{w}^+, \mathbf{w}^- &\geq 0 \end{aligned} \quad (4.7)$$

Next, we bound the  $L_1$  norm of  $\mathbf{w}$  to  $\mathbf{w}^+$  and  $\mathbf{w}^-$ , where  $\mathbf{w} = \mathbf{w}^+ - \mathbf{w}^-$ . As every element is positive, the  $L_1$  norm of  $\mathbf{w}^+$  and  $\mathbf{w}^-$  is  $l(\mathbf{w}^+, \mathbf{w}^-) = \mathbf{1}^T \mathbf{w}^+ + \mathbf{1}^T \mathbf{w}^-$ . Then:

$$\begin{aligned} \|\mathbf{w}\|_{L_1} &= \|\mathbf{w}^+ - \mathbf{w}^-\|_{L_1} \leq l_1(\mathbf{w}^+, \mathbf{w}^-) = \mathbf{w}^+ + \mathbf{w}^- \\ \min_{\mathbf{w}} f(\mathbf{w}) &= \mathbf{w}^T Q \mathbf{w} + q^T \mathbf{w} + \lambda \|\mathbf{w}\|_{L_1} \\ &\leq (\mathbf{w}^+ - \mathbf{w}^-)^T Q (\mathbf{w}^+ - \mathbf{w}^-) + q^T (\mathbf{w}^+ - \mathbf{w}^-) + \lambda \mathbf{1}^T (\mathbf{w}^+ + \mathbf{w}^-) \end{aligned} \quad (4.8)$$

Thus we can get the optimized weight vector  $\mathbf{w}$  by solving the constrained quadratic minimization problems for  $\mathbf{w}^+$  and  $\mathbf{w}^-$ . The matrix form for the translated problem becomes as follows:

$$\begin{aligned} \min_{\mathbf{w}^+, \mathbf{w}^-} f(\mathbf{w}^+, \mathbf{w}^-) &= \begin{bmatrix} \mathbf{w}^+ \\ \mathbf{w}^- \end{bmatrix}^T M \begin{bmatrix} \mathbf{w}^+ \\ \mathbf{w}^- \end{bmatrix} + n^T \begin{bmatrix} \mathbf{w}^+ \\ \mathbf{w}^- \end{bmatrix} \\ &\begin{bmatrix} \mathbf{w}^+ \\ \mathbf{w}^- \end{bmatrix}^T \begin{bmatrix} \mathbf{1} \\ -\mathbf{1} \end{bmatrix} = 1 \\ &\mathbf{w}^+, \mathbf{w}^- \geq 0 \end{aligned} \quad (4.9)$$

The problem then is transformed into a quadratic optimization problem with inequality constraint. So we can solve the equations by using the same method as that of Equation 4.5.

## 4.5. SHAPE INTERPOLATION

Given such a weight vector from the projection procedure, we will reconstruct the vertex positions for the target object in this subsection.

### 4.5.1. LINEAR BLENDING

The linear blending method is the basic algorithm for skeletal shape deformation. Linear blending computes the reconstruct vertex positions  $\mathbf{v}'$  according to the following equation[31]:

$$\mathbf{V}' = \mathbf{M} \left( \sum_{i=0}^k w_i \mathbf{T}_i \right) \quad (4.10)$$

Where  $\mathbf{V}' \in \mathbb{R}^{n \times d}$  is the matrix whose rows are the reconstructed vertex positions,  $k$  is the number of key poses,  $\mathbf{M} \in \mathbb{R}^{n \times (d+1)m}$  is the matrix combining rest-pose vertex positions  $v_j$  with the vertex weights  $w_p(v_j)$ ,  $m$  is the number of handles, and  $\mathbf{T}_i \in \mathbb{R}^{(d+1)m \times d}$  stacks transposed transformation matrices for each handle. Transformation matrix for handle  $j$   $\mathbf{T}_{ij} \in SE(3)$  is a rigid body transformation. Linear blending works well when each blended transformations  $\mathbf{T}_i$  is not too much different the others. Issues might arise if we need to blend transformations which differ too much. Besides, the blended transformation is not guaranteed to be a rigid body transformation.

### 4.5.2. FORWARD KINEMATICS

As the system aims to generate real-time animation for an articulated object, we can make use of the hierarchical modeling of rigged characters. In fact, much of the ideas concerning the animation of hierarchies in computer graphics comes from Robotics. The articulated models consist of the rigid part connected by joints that can rotate around a fixed point of a link. Figure 4.3 illustrates the movements of hands for an articulated character. When we want to raise the hand of the character to touch the ball, we firstly rotate the shoulder, then rotate the elbow, which is the child of the joint shoulder. That is how the forward kinematics works: it can describe the positions of the body parts as a function of the joint angles rather than rigid transformation matrix[35].



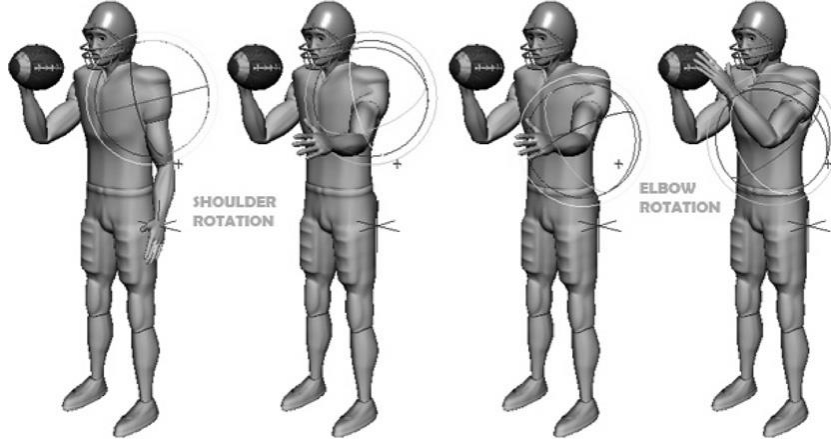


Figure 4.3: Sample sequence of forward kinematic specification of joint rotations.

By applying forward kinematics, we can animate the character by specifying the joint rotation angles relative to its parent node as a function of time. In the registration phase, we register the relative rotation and relative translations of each joint towards its parent node for each key poses. During the shape interpolation process, we calculate the linearly blend the relative rotations and translations for each joint, using the following equation:

$$\begin{aligned}\theta'_j &= \sum_{i=0}^k w_i \theta_{ij} \\ \mathbf{T}'_j &= \sum_{i=0}^k w_i \mathbf{T}_{ij}\end{aligned}\tag{4.11}$$

After the interpolation, We can aggregate the transformation matrix for each joint, which is a matrix composition of all joint transformation between the joint and the root of the hierarchy. The affine transformation matrix  $affine(j)$  for each joint  $j$  can be calculated as follows:

$$\begin{aligned}\mathbf{affine}(j) &= \mathbf{affine}(\mathbf{parent}(j)) \mathbf{T}_j \mathbf{R}(\theta_j), \text{ while } j \text{ is leaf node} \\ \mathbf{affine}(j) &= \mathbf{T}_j \mathbf{R}(\theta_j), \text{ while } j \text{ is parent node}\end{aligned}\tag{4.12}$$

The affine transformation at each joint is calculated by compositing all the transformations up the hierarchy to the root node. After that, we can stack all the affine matrix into transformation matrix  $T$  and apply the linear blending equation similar to Equation 4.10 to reconstruct vertex positions  $\mathbf{v}'$ :

$$\mathbf{V}' = \mathbf{M} \cdot \mathbf{T}\tag{4.13}$$

Forward kinematics guarantee that the input transformation matrix for linear blending must be rigid. Artifacts like candy wrapper or exaggeration would be avoided.

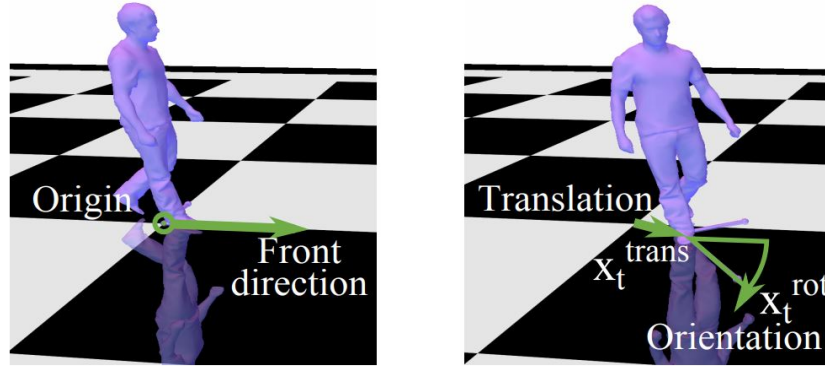


Figure 4.4: Global translation  $\mathbf{x}^{trans}(t)$  and rotation  $\mathbf{x}^{rot}(t)$  of a general pose (right) extracted in relation to the rest pose (left)

#### 4.6. OPTIMIZATION

For each actor pose in the puppetry phase  $\mathbf{x}(t)$ , we estimate the global position and orientation of the actor's spine. Figure 4.4 illustrates the procedures how we extract the global translation  $\mathbf{x}^{trans}(t)$  and the global rotation  $\mathbf{x}^{rot}(t)$  for each frame [16]. Global motion is represented by 3 degrees of freedom: a translation vector  $\mathbf{x}^{trans}(t) \in \mathbb{R}^3$  and a rotation angle  $\mathbf{x}^{rot}(t) \in \mathbb{R}$

# 5

## EXPERIMENTAL RESULTS

In this chapter, we evaluate the effectiveness of our approach in a number of steps. Firstly, we will evaluate different methods for example-based projection. Next, we will provide some examples to compare the forward kinematics and linear blending in shape interpolation. Then we will compare our system with the previous methodologies for performance-based animation.

### 5.1. EXAMPLE-BASED PROJECTION

In this sections, we show the examples and statistics of weight generation and evaluate the results of applying a linear projection, positive-constrained projection, and l1 regularization.

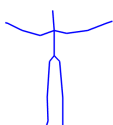
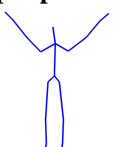

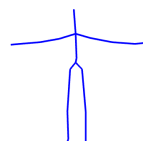

		Input poses		
				
		1	2	3
Base Poses				
(1)		0.76	0.41	0.15
(2)		0.24	0.59	0.85

Table 5.1: Project three input poses into two-base-pose shape space with linear projection

During the puppetry phase, Microsoft Kinect continuously inputs real-time skeleton positions of the actor into the system. The system treats this real-time input data as an interpolated result of base positions achieved during the registration phase. Then we can calculate the weight vectors for interpolated shape reconstruction in the next step. We first try to solve this problem with quadratic programming.

Table 5.1 shows the weights of three input pose when we have two base poses. The three input poses are intuitively interpolated pose from the base poses. The weights demonstrate the extents of interpolation and are consistent with human perception. Next, we test what happens if we add a third pose that may interfere with the first poses to the system.


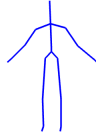
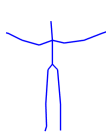
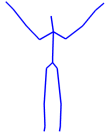

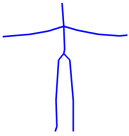


		Input poses				
						
Base Poses		1	2	3	4	5
(1)		0.24	0.52	0.99	0.80	0.40
(2)		-0.04	-0.06	0.19	0.50	0.80
(3)		0.80	0.54	-0.18	-0.30	-0.20

Table 5.2: Project five input poses into three-base-pose shape space with linear projection


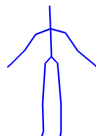
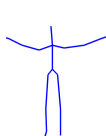
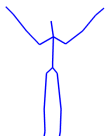
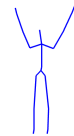
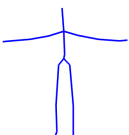


		Input poses				
						
Base Poses		1	2	3	4	5
(1)		0.16	0.42	0.76	0.41	0.15
(2)		-	-	0.24	0.59	0.85
(3)		0.84	0.58	-	-	-

Table 5.3: Project five input poses into three-base-pose shape space with positive-constrained projection.

Table 5.2 presents the results with an additional base pose three. The last three input pose is the same as the in Table 5.1. As a result, we see that compared to the setting with just two poses, the weights completely change and negative weights appear. The negative weights seem undesired here as the input poses are chosen to be inbetween poses of the input poses.

To eliminate negative weights, we apply a positivity constraint to the weights (see Equation 4.5). Table 5.3 presents updated results from the identical input data in Table 5.2. The weights of the last three input pose are the same as those in Table 5.1. The method successfully enables the system to consider the effect of

relevant base pose only.


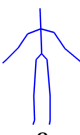
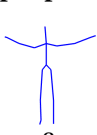
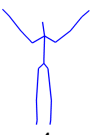

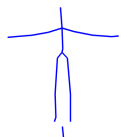








		Input poses				
						
		1	2	3	4	5
Base Poses						
						
	(1)	0.17	0.42	1.00	1.00	1.00
						
	(2)	0.83	0.58	-	-	-








Table 5.4: Project five input poses into two-base-pose shape space with positive constrained projection.

The above example illustrates our observation that the positivity helps us in dealing with the construction of weights for input poses that are inbetween the base poses. What if the input poses exceed the boundary of base poses? Table 5.4 demonstrates the case where the base poses are "arms down" and "arms straight" and the input poses include "arms up" poses. Explicitly, the input poses 3, 4, and 5 are not inbetween poses of the input poses. The results show that the poses 3, 4 and 5 are approximated as the base pose 1. The reason is that with the restriction to positive weights, the system cannot extrapolate beyond convex combinations of the base poses. Hence, extrapolated poses like the "arms up" poses cannot be represented. This can be a desirable feature for some applications, but can also be a several limitations in other scenarios.








(a)  $\lambda = 0.1$ 

		Input poses				
						
		1	2	3	4	5
Base Poses						
	(1) 	0.17	0.42	1.12	1.40	1.45
	(2) 	0.83	0.58	-0.12	-0.40	-0.45

(b)  $\lambda = 0.2$ 

		Input poses				
						
		1	2	3	4	5
Base Poses						
	(1) 	0.17	0.42	1.00	1.24	1.29
	(2) 	0.83	0.58	-	-0.24	-0.29

(c)  $\lambda = 0.3$ 

		Input poses				
						
		1	2	3	4	5
Base Poses						
	(1) 	0.17	0.42	1.00	1.08	1.13
	(2) 	0.83	0.58	-	-0.08	-0.13

(d)  $\lambda = 0.4$ 








		Input poses				
						
		1	2	3	4	5
Base Poses						
	(1) 	0.17	0.42	1.00	1.00	1.00
	(2) 	0.83	0.58	-	-	-

Table 5.5: Project five input poses into two base-pose through l1 regularization with different value of  $\lambda$

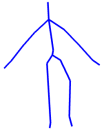







Input Pose	Methods	Pattern	Weights for Standard Poses				
			Base a	Base b	Base c	Base d	Base e
	Linear		0.08	0.54	-0.15	-0.10	0.64
	L1		-	0.37	-0.05	-	0.68
	Positive		0.01	0.29	-	-	0.70
	Linear		-0.70	0.66	0.52	-0.13	0.64
	L1		-0.14	0.51	0.54	-0.13	0.23
	Positive		-	0.42	0.55	-	0.03

Table 5.6: The two input poses contains movement in both arms and legs. We compare the results of projection by using linear projection, L1 regularization and positive constrained projection.

An alternative way to regularize the system is to add a sparsity enforcing prior. This type of regularization allows for extrapolation while avoiding that too many poses influence the result. Explicitly, we introduce  $L_1$  norm into the optimization energy as a sparsity enforcing term. The  $L_1$  term receives a weight  $\lambda$ . The higher the weight, the sparser we expect the solution to be. To explore the effects of the parameter  $\lambda$ , we assign  $\lambda$  with several values.

Table 5.5 shows the generated with  $\lambda$  equals from 0.1 to 0.4. We can tell that the Table 5.5(d) is a pure interpolation. The result is identical to that in Table 5.4. When  $\lambda$  equals to 0.1, the result shows extrapolation which is consistent with our perception. The effect of extrapolation decreases with larger and larger  $\lambda$ .  $\lambda$  can control the extent of extrapolation in the result. Thus we believe  $\lambda$  is quite essential in controlling the properties in the reconstruction process because it can adjust the elasticity in the weight generation. We believe this method is an in-between method for quadratic programming and positive constraint.  $\lambda$  is the control stick. We will discuss more the effects of interpolation and extrapolation for mesh reconstruction in the next section.

We have discussed effects of the three methods in the weight generation of simple input poses. How would they behave when we have more complicated input data? Table 5.6 demonstrate the case when leg poses are involved. In this case, there are five standard poses: three for moving hands and two for moving legs. From the perspective of human perception, the first input pose is a combination of base pose b and e. The quadratic programming can give us a similar result, but it brings negative weights for irrelevant base pose c and d.  $L_1$  and positive constraint both provide satisfactory results, with large weights in base pose b and e. Based on human cognition, the second input pose should be a combination of base pose b, c, and e. The  $L_1$  behaves the best and can recognize the main poses. The quadratic programming behaves with quite large negative weight in base a to eliminate the stretching effect of base b and c. The positive constraint recognizes the wrong leg because it does not negative weights.





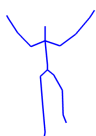



Input Pose	Methods	Pattern	Weights for Standard Poses				
			Base a	Base b	Base c	Base d	Base e
	Linear		0.09	0.20	-0.04	1.04	-0.29
	L1		-	0.06	-	0.94	-
	Positive		-	0.06	-	0.94	-
	Linear		-0.68	0.32	0.63	1.01	-0.29
	L1		-0.23	-0.11	0.68	0.69	-0.25
	Positive		-	-	0.69	0.31	-

Table 5.7: Based on table 5.6, we shift the coordinates of two inputs poses 0.5 in the x direction. This table compares the influence of translation in the three methods.



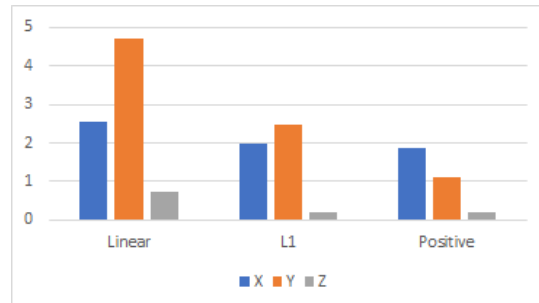
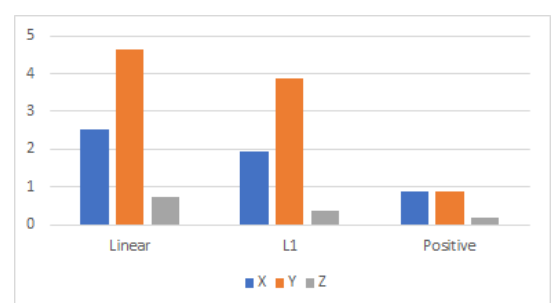
						
	Linear	L1	Positive	Linear	L1	Positive
x	2.53	1.98	1.88	2.54	1.93	0.90
y	4.70	2.48	1.12	4.65	3.89	0.90
z	0.72	0.22	0.22	0.72	0.39	0.19

Table 5.8: This table shows the differences in weights when we shift the input coordinates 0.5 in x, y, z direction respectively.



(a) Input Pose 1



(b) Input Pose 2

Figure 5.1: This figure illustrate the influence in weight generation when we shift the coordinates in x, y, z direction respectively.



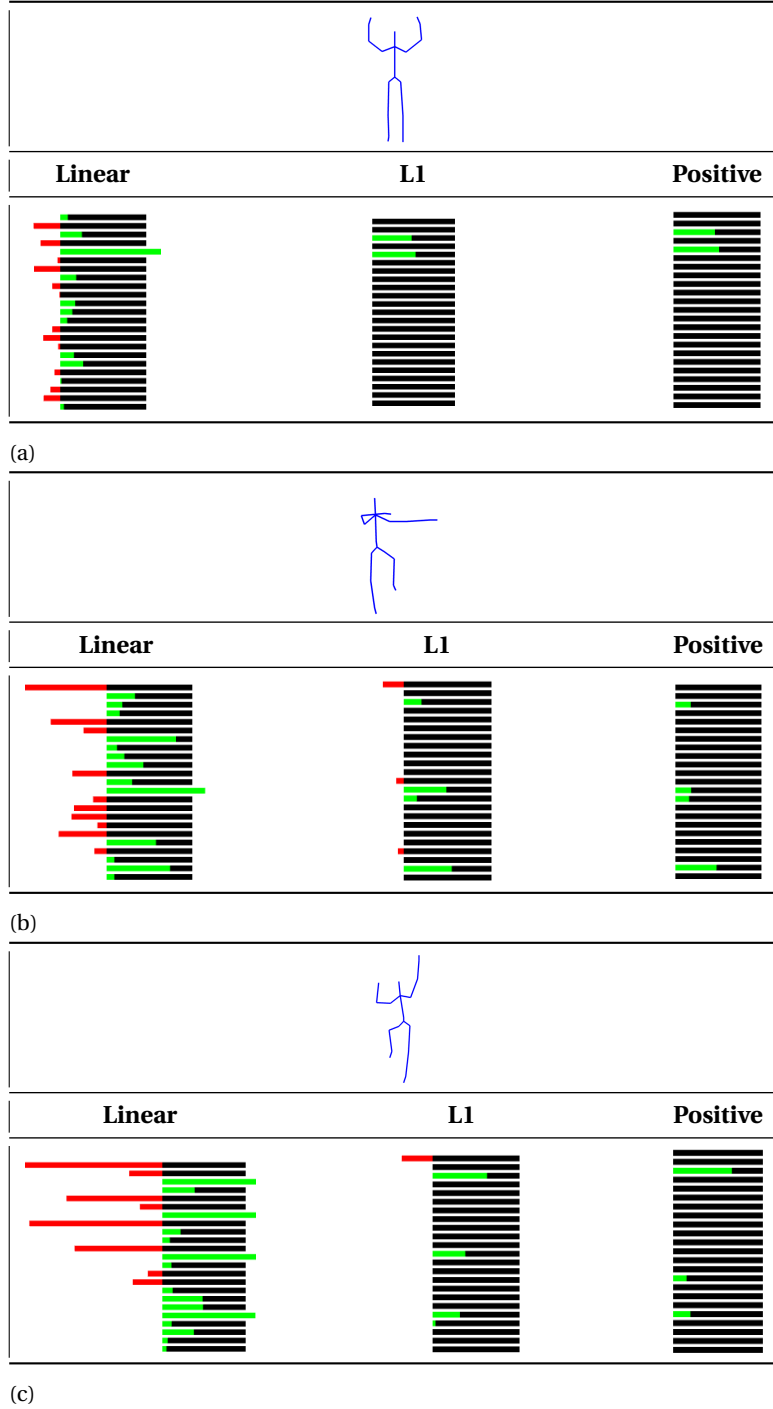


Table 5.9: This table illustrates the projection when we take all 23 base poses in Appendix A into consideration. The visualization shows the weight generated by linear projection, l1 regularization and positive constrained projection respectively.

Apart from the influences of applying different methods, there are also other factors that we need to add into consideration. In the case demonstrated in Table 5.7, we shift all the vertexes in the input poses in Table 5.6 0.5 to the right.  $L_1$  is the most stable method as it still generates similar result as in Table 5.6. Quadratic programming is the most unstable method, and the result is seriously influenced by the shifting. The system cannot recognize the relevant input poses. Table 5.8 shows the influences of weight generation for the three methods when we shift the input pose by 0.5 towards the direction x, y, and z. The result is visualized in the bar chart in Figure 5.1. 5.1a visualizes the shifting result of example 1, and the shifting result of example 2 is on the right. We can tell from the graph that the shift in Y direction affect the result most, and the quadratic









methods are quite unstable that shifting in all direction will result in largest difference for this method.

The above examples are based on 5 standard poses at most. In fact, we have generated 23 example base poses in the project. You can view all of them in Section A. Table 5.9 shows the result when we have 23 base poses. From the three examples, we can find the quadratic programming involve most of the base poses in the final weights. This method is not ideal when there are more than three base cases. Either  $L_1$  regularization or positive constraints can be applied in this case. Quadratic programming with the positive constraint is the most stable method so far, but it may give wrong answers when extrapolation happens.

## 5.2. SHAPE INTERPOLATION

In this section, we report the results of shape deformation by interpolation and extrapolation.

After the procedure of shape space projection, we will get a weight vector for interpolation and extrapolation. In this section, we will reconstruct vertex positions of a virtual object based on different weight vectors. Table 5.10 shows the reconstructed poses of an ogre when we apply different methods. Reconstructed results based on forward kinematics is in the first row. Linear blending results are on the second row. The third column and the fifth column are the base poses. Inbetween is the interpolated poses. We can tell that both methods work well here but for the extrapolation, but linear blending will stretch the hands of the ogre to make it looks abnormal when negative weights are involved. In the first column when the weight is  $(-1.0, 2.0)$ , the forward kinematics will cross the hands above the ogre's head. So although we input large negative weights into the system, we can still get intuitive poses based on this method.

<b>Forward Kinematics</b>				
<b>Linear Blending</b>				
<b>Weights</b>	$(-1.0, 2.0)$	$(-0.5, 1.5)$	$(0.0, 1.0)$	$(0.5, 0.5)$


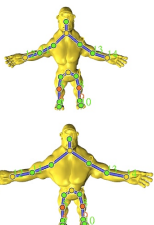
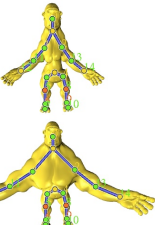
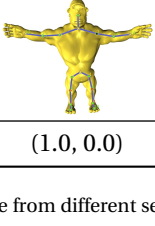
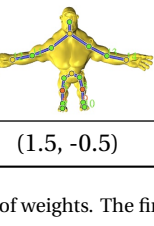
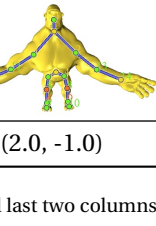




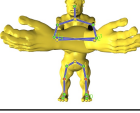



<b>Forward Kinematics</b>			
<b>Linear Blending</b>			
<b>Weights</b>	$(1.0, 0.0)$	$(1.5, -0.5)$	$(2.0, -1.0)$

Table 5.10: Reconstructed poses of ogre from different setting of weights. The first two and last two columns are extrapolation. The rest columns are interpolation.

Table 5.2 demonstrates another example when the base pose combination aims to help the ogre to clap hands. In the first column and the last column, the hands of ogre stretch out tremendously when applying direct interpolation. Even when the negative weight is not too large ( $W_1 = -0.5$ ), the hands of the ogre looks bigger indirect method compared with forward kinematics.

Up to now, We might have the impression that negative weights will bring inconvenience for shape interpolation. Should we eliminate it from the weight generation by using only positive weight constraint? Are there any advantages of using negative weights? We make another experiments as demonstrated in Table 5.12. The third one (normal pose) and the fifth one (compressing the bar) are the bases poses. When the first weight is negative, the bar is stretching out, and when the second weight is negative, the bar is compressed more. That effect makes the bar looks elastic and vivid. So extrapolation has its advantages of adding elasticity to the object.

<b>Forward Kinematics</b>				
<b>Linear Blending</b>				
<b>Weights</b>	(-1.0, 2.0)	(-0.5, 1.5)	(0.0, 1.0)	(0.5, 0.5)


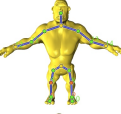
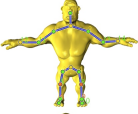


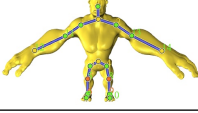
<b>Forward Kinematics</b>			
<b>Linear Blending</b>			
<b>Weights</b>	(1.0, 0.0)	(1.5, -0.5)	(2.0, -1.0)

Table 5.11: Reconstructed poses of ogre from different setting of weights. The first two and last two columns are extrapolation. The rest columns are interpolation.

In the last example, the base poses bar involves only translations in the joint, what if we add twisting into the base pose? Table 5.13 demonstrates this process. The first three columns are an interpolation, where the two methods behave all well. From the fourth column to the right, large negative weights are presented. With linear blending, the bar is not behaving twisting after the boundary of the base pose. On the contrary, one side of the bar exaggerates in size. In the first row, we apply forward kinematics. The bar keeps twisting when the negative weights are larger.

### 5.3. SYSTEM

In this section, we compare our result with previous puppetry approaches. We compare our method for mesh puppetry with previous approaches based on some criteria that are important for puppetry (see Table 5.14). These criteria include the following aspects:

1. The type of target characters supported by the system. Whether they system can puppeteer non-humanoid creatures.
2. Whether the system runs in real-time.
3. Whether the system can synthesize motion with a natural creature style.
4. Whether the system support dynamics in the motion generation, not only purely select existing motion from the library.
5. Whether the system can generate human-impossible motion.

Similar to Creature Features [11], our method is aimed at animating non-humanoid creatures. However, they require the users to initially collect several pairs of motions from both actors and virtual objects to trigger the direct feature mapping between the source and target. In other words, users need to import extra motions from elsewhere to generate their animation here, which raise a barrier for normal users. The Kim *et al.* [12] propose to control the characters through traditional marionette control as natural interfaces. Thus the users need a large amount of practice to synthesize motions with natural creature style. KinÊtre [13] claims to be able to animate arbitrary objects. However, they deform the mesh according to actors' real-time performance to let them imitate the motion of humans. So their animation is not natural if the target object has different body structure of human body. For example, when they want to animate a four-leg chair, two of the chair leg cannot move because humans only have two legs. Similarly, Interactive Steering [14] separates the mesh

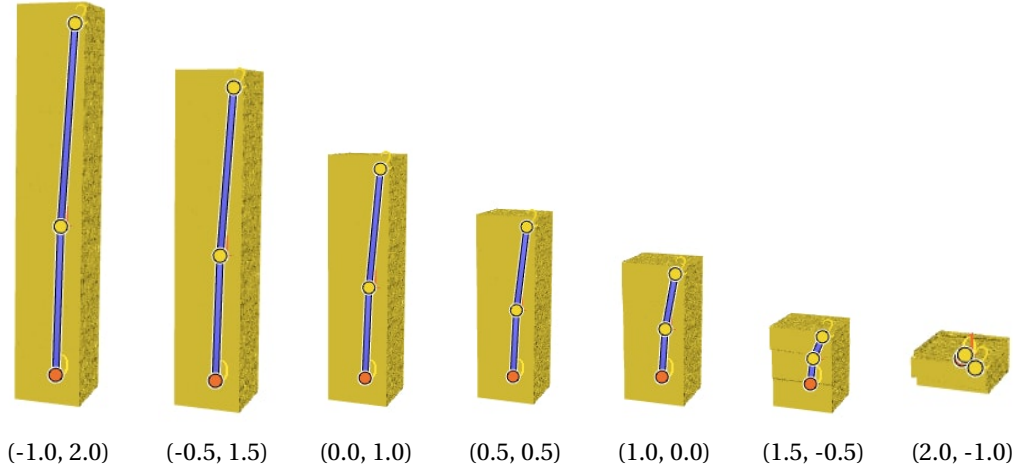


Table 5.12: This figure illustrates the advantages of negative weights in reconstruction. The bar can stretch out and squeeze in when extrapolation is involved.

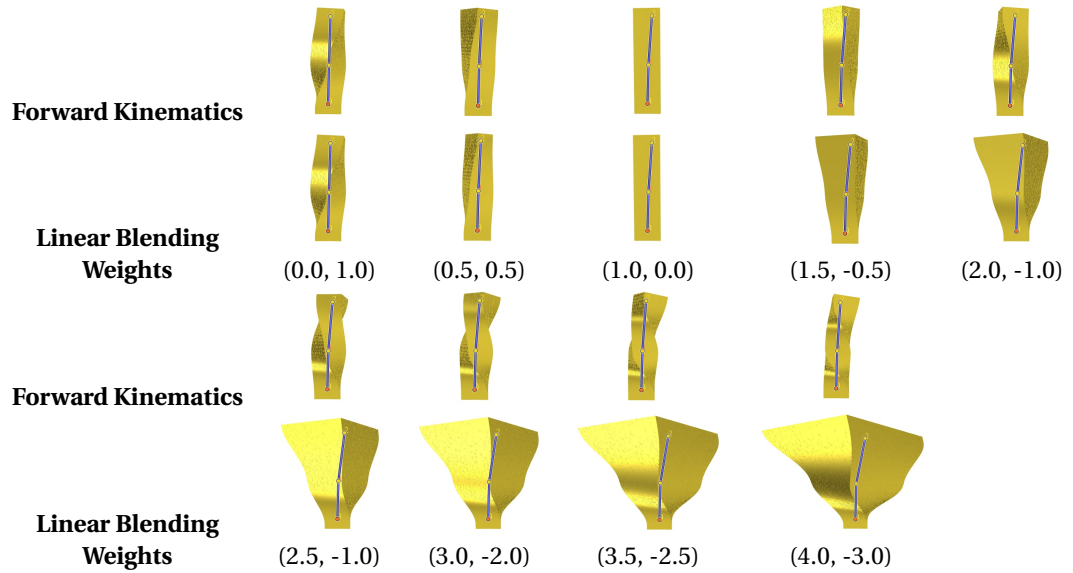


Table 5.13: We compare the reconstructed results of squeezing bars between forward kinematics and linear blending.

into limbs and body parts and let the users animate them separately according to their performance in real-time. So multiple users can animate a single virtual object at the same time. However, making a natural animation becomes even harder as several people need to perform certain actions concurrently, which is difficult to control. Creature Teacher [15] only allows periodic movements, and the users have to generate the animation with several rounds of manipulation, so they cannot output the animation in real-time. Similar to our method, Interactive Motion Mapping [16] learns the mapping relationship between source and target characters by registering multiple pairs of poses from the actor and the virtual object. Different from our method, the system also requires the input of several clips of animations for generating animation in the puppetry phase.

Our method has the advantages over the direct feature mapping when we consider the aim of creating natural motions for non-human character puppetry. In the registration phase, direct feature mapping requires the user to specify the number of feature pairs, which is sometimes a difficult process. Because non-humanoid characters often have a different body structure with humans. So sometimes the system even leaves some DOFs of the target object blank, and thus some parts of the target object cannot move. This phenomenon lets the target object behave weird and unnatural. Besides, when we apply direct mapping to non-

	Ours	Creature Features	Haptic Puppetry	KinÊtre	Interactive Steering	Creature Teacher	Motion Mapping
Human-Human	✓	✓	✓	✓	✓	✓	✓
Human-Creature	✓	✓	✓	✓	✓	✓	✓
Real-Time	✓	✓	✓	✓	✓	✗	✓
Natural Creature Style	✓	✓	-	✗	✗	✓	✓
Dynamics	✓	✗	✓	✓	✓	✓	✗
Human Impossible Motion	✓	✓	✓	✗	✗	✓	✓
Extra Motion Independent	✓	✗	✓	✓	✓	✓	✗

Table 5.14: Comparison with previous puppetry methods: Creature Features[11], Haptic Puppetry[12], KinÊtre[13], Interactive Steering[14], Creature Teacher[15], Motion Mapping[16]

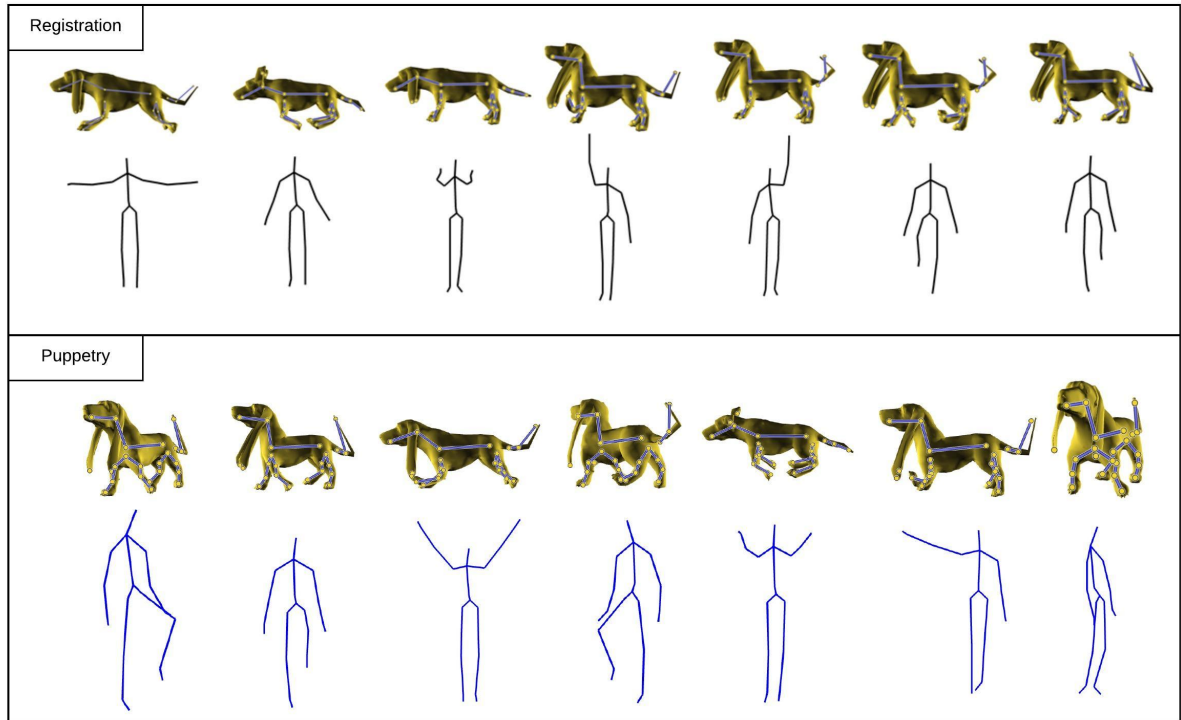
humanoid characters, there are several kinds of motions that the user cannot perform, such as squeezing, stretching and twisting, which is in contrast with the initial aim of the performance animation: to animate meshes with human-possible body motions. Moreover, our system does not require any motion inputs from the character side. So the users don't have to spend extra time in other systems to create desirable motion clips beforehand. They can directly come to our system and generate their first animation in real-time and with fun.

## 5.4. RESULTS

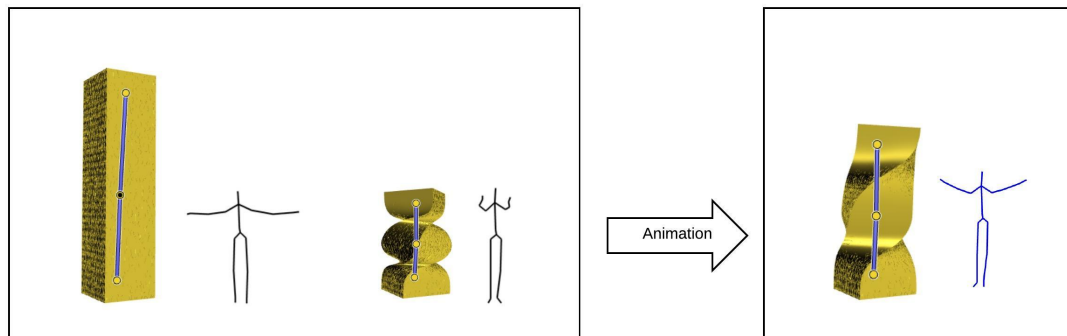
In this section, we will display some demo animations for various meshes. Figure 5.2 presents two examples generated by our system.

Figure 5.2a presents animation generation process of a dog. It has a different body topology from human beings. We want to generate the movement of walking, running, waving a paw and moving the tail. In the registration phase, we make seven pose couplings between the user and the character. The "arm straight" and "arm front" poses control the running of the dog. "Left arm up" controls the paw waving. "Right arm up" controls the tail moving. Human walking represents the dog walking. In the puppetry phase, we can tell that the system can generate different animations interactively according to the user's performance. When the user turns left and walk, the dog will turn correspondingly and walk. When the user moves from "arm straight" to "arm front", the dog will run. In the meanwhile, the dog's ears will also move up and down. We can tell that the system can differentiate different input poses from Kinect side and generate useful and playful animations for non-humanoid characters.

Figure 5.2b demonstrates a case when users control an object with fewer DOFs compared to the human body. The task is generating squeezing animation for a simple bar. Traditionally, it is impossible for users to animate squeezing and twisting. But our system can animate the bar with just the arm movement. In the registration phase, we make two couplings of example poses from the user and the bar. For the second example pose of the bar, we add twisting effect to the squeezing pose. So in the puppetry phase, when the user moves from "arm straight" to "arm front", the bar will have a cartoonish effect of twisting and squeezing simultaneously. We can conclude that our system can animate object with arbitrary topology and let them behave human-impossible motions.



(a) Animated Result of a dog. The dog can move according to not only the input pose of user, but also the user's orientation



(b) The animated result of a simple bar. With only two standard poses registered, the user can let the bar squeeze and twist based on the movement of arms

Figure 5.2: Some animations generated by our system.

# 6

## DISCUSSION, LIMITATIONS AND FUTURE WORK

In this chapter, we will discuss our findings and limitations and suggest some future work.

### 6.1. DISCUSSION

Let's recap the challenges we have mentioned in the field of mesh puppetry:

- **How to match arbitrary target creature to human body?**

We eliminate the traditional way of direct feature mapping, which is not able to control non-humanoid creatures. We propose to project the human skeleton information into a low-degree shape space and reconstruct the target deformation by interpolation. So we can control any 3D rigged mesh, despite morphology and topology.

- **How to generate natural and plausible creature animations from the input body motions of users?**

By using motion transfer, we transform this question into generating the best weights for mesh deformation in reconstruction. We apply three methods here: quadratic programming, quadratic programming with positive constraints, quadratic programming with  $L_1$  regularization. During the evaluation process, we use multiple examples with different numbers and types of base poses. We find that quadratic programming performs better when there is a limited number of base poses. Positive constraints will perform better when input motion is within the boundary of example poses. So it can not expect extrapolation in the result, which keeps the rigidity in the shape space. However, when the input motion is crossing the shape space boundary, the animation will be stuck. Because the projection returns the same result for the extrapolation.  $L_1$  regularization can select sparse result, which is suitable when there are a lot of base poses in the shape space. At the same time, it allows extrapolation. There is parameter  $\lambda$  in this method, which balances the influence of  $L_1$  regularization and quadratic programming. In our experiments, we can find that when  $\lambda$  is larger, the result is closer to the result from positive constraints. And naturally, when  $\lambda$  is closer to zero, the result is more like quadratic programming. So this method can work as a compromise between the other two methods, and the  $\lambda$  is the control stick to adjust the tolerance of extrapolation.

Is it necessary to allow extrapolation in the pose reconstruction? Is it better if we just keep the rigidity in the shape space and eliminate all the negative weights? Actually, extrapolation is very useful. When an object is moved, the movement emphasizes any rigidity in the object. In real life, only the most rigid shapes (such as chairs, dishes, and pans) remain so during motion. Anything composed of living flesh or more elastic materials such as rubber, no matter how bony, will show considerable movement in its shape during an action. For example, a face, whether chewing, smiling, talking, or just showing a change of expression, is alive with changing shapes in the cheeks, the lips, and the eyes. The movements and expressions will exaggerate if the character is cartoonish. So with extrapolation, the designers won't have headaches about choosing the boundary key poses because the system can exaggerate them a little bit when the user have poses with larger movement. This uncertainty increases the continuity and the enjoyability of the system.

- **How to design the system to make it real-time?**

To achieve the real-time performance, we choose to apply three methods that can be linearly solved. Although the method with  $L_1$  minimization is not quadratic programming, we apply solving procedure written in paper [34] that can transform the problem into quadratic programming. So the solving time is very small, and the animation can be generated in real-time. Therefore, by using our system, Users can produce continuous motions comfortably in real-time.

## 6.2. LIMITATIONS AND FUTURE WORK

Our puppetry system has several limitations. An inherent limitation of pose-to-pose mapping is that accurate control of all degrees of freedoms is a non-trivial task. When the number of DOFs is quite large, it will take a long time to adjust a natural key pose for the target character. Jacobson *et al.* have proposed a method to deform a shape with only a subset of DOFs are specified. It keeps the rigidity of the target mesh by minimizing a rigidity energy to automatically solve the transformation matrix for other handles. Inverse Kinematics can be applied to transform the absolute transformation information back into the relative rotations.

Using the Kinect as an input device also restricts the level of control that can be applied by the users. For example, it is not very sensitive to track the leg movement of the user. The arrange of accepting of full-body information is limited. So is the optimization step for generating mesh animation. If the full body information is not tracked, our system will pause until the users step back completely in the camera range.

Another limitation is that in the projection procedure, the movements of unrelated parts will sometimes affect each other. For example, an arm raising pose will affect a leg raising pose because in the leg pose we also take down the arm information. In a later study, we can focus on split the shape space into several parts: leg movement, arm movements, and body movements. Projection can be applied individually in the three shape space, which will not interfere each other[14].

One interesting extension of our work is in the optimization step. We only apply the basic optimization method to rotate and translate the target shape according to users' movements and orientation. Other optimization criteria can be added to the system in the future. For example, overlapping in the body parts in the target are not allowed. We can add contact constraints to this step. What's more, physical simulations can increase the realism of the animation. For example, a fat man's belly will shake, when he quickly turns around.

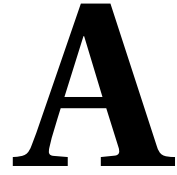


# 7

## CONCLUSION

We have presented a puppetry approach that is motivated by the recent advances in motion detecting devices and the increase in applications that require the users to control a virtual character through their body performance. Existing puppetry application mainly focus on human or humanoid target characters and the extension to non-humanoid characters for novice user in real-time is not attempted, despite the broad range of recent applications. Our approach produces natural and continuous motions for this kind of non-human-like characters comfortably and enjoyably in real-time, despite differences in body structures. Therefore, we sincerely expect that our approach will continuously contribute to other puppetry applications in the field of 3D computer games and interactive media in the future.





## APPENDIX

### A.1. EXPERIMENTAL POSES

In this section, we will list all the experimental base poses as the reference for the researchers and users who are interested in the related projects.

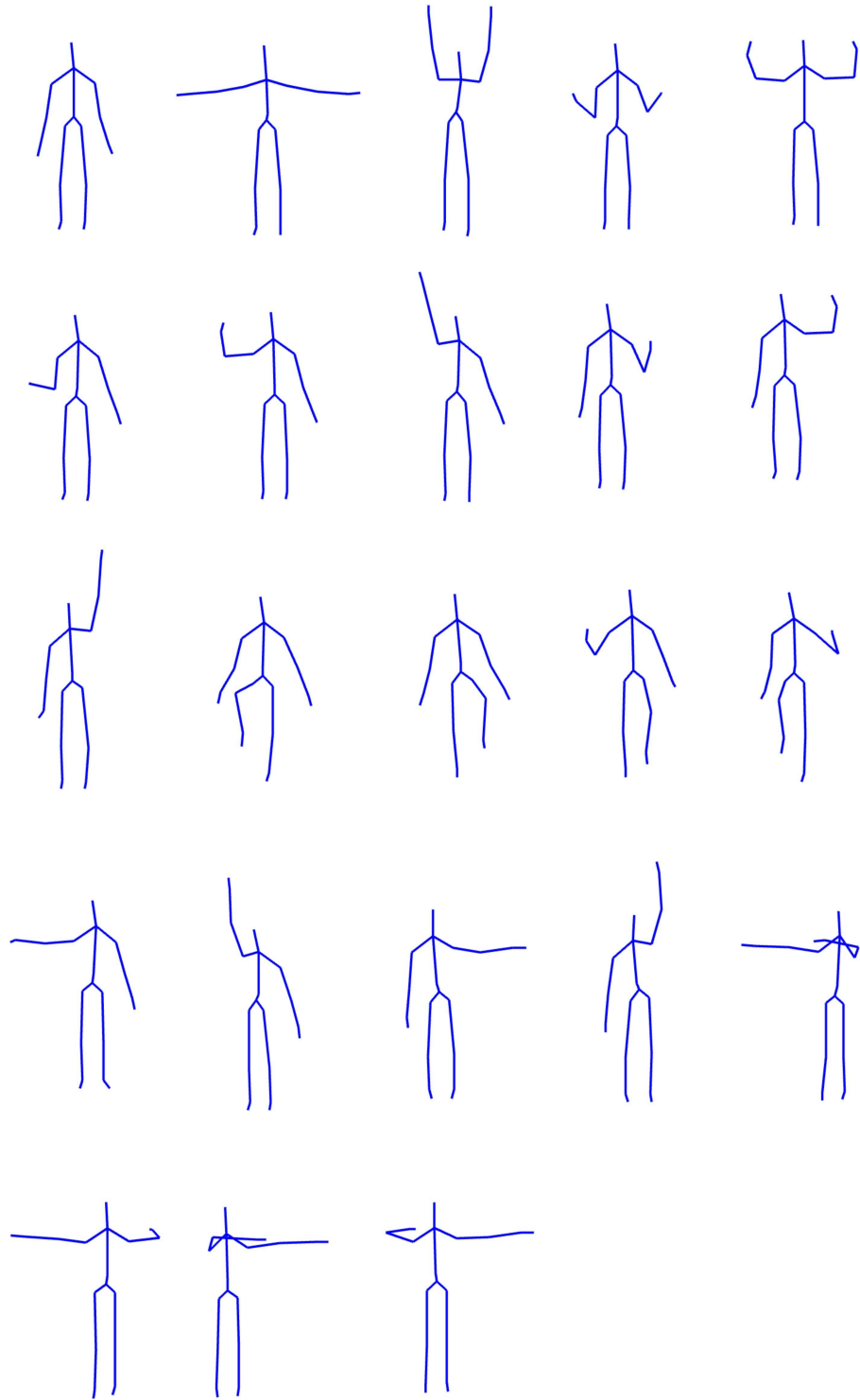


Figure A.1: 23 example base poses that we use in this application

## BIBLIOGRAPHY

- [1] J. Lasseter, *Principles of traditional animation applied to 3d computer animation*, in *ACM Siggraph Computer Graphics*, Vol. 21 (ACM, 1987) pp. 35–44.
- [2] M. F. Deering, *Holosketch: a virtual reality sketching/animation tool*, *ACM Transactions on Computer-Human Interaction (TOCHI)* **2**, 220 (1995).
- [3] Q. L. Li, W. D. Geng, T. Yu, X. J. Shen, N. Lau, and G. Yu, *Motionmaster: authoring and choreographing kung-fu motions by sketch drawings*, in *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Eurographics Association, 2006) pp. 233–241.
- [4] M.-W. Chao, C.-H. Lin, J. Assa, and T.-Y. Lee, *Human motion retrieval from hand-drawn sketch*, *IEEE Transactions on Visualization and Computer Graphics* **18**, 729 (2012).
- [5] M. Eitsuka and M. Hirakawa, *Authoring animations of virtual objects in augmented reality-based 3d space*, in *Advanced Applied Informatics (IIAIAI), 2013 IAI International Conference on* (IEEE, 2013) pp. 256–261.
- [6] A. Jacobson, D. Panozzo, O. Glauser, C. Pradalier, O. Hilliges, and O. Sorkine-Hornung, *Tangible and modular input device for character articulation*, *ACM Transactions on Graphics (TOG)* **33**, 82 (2014).
- [7] A. Thomas and N. Tufano, *Stop motion animation*, *DIY Media: Creating, Sharing, and Learning with New Technologies*, 161 (2010).
- [8] P. Nogueira, *Motion capture fundamentals*, Faculty of Engineering, University of Porto (2011).
- [9] D. J. Sturman, *Computer puppetry*, *IEEE Computer Graphics and Applications* **18**, 38 (1998).
- [10] J. Chai and J. K. Hodgins, *Performance animation from low-dimensional control signals*, in *ACM Transactions on Graphics (TOG)*, Vol. 24 (ACM, 2005) pp. 686–696.
- [11] Y. Seol, C. O’Sullivan, and J. Lee, *Creature features: online motion puppetry for non-human characters*, in *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (ACM, 2013) pp. 213–221.
- [12] S. Kim, X. Zhang, and Y. J. Kim, *Haptic puppetry for interactive games*, in *International Conference on Technologies for E-Learning and Digital Entertainment* (Springer, 2006) pp. 1292–1302.
- [13] J. Chen, S. Izadi, and A. Fitzgibbon, *Kin tre: animating the world with the human body*, in *Proceedings of the 25th annual ACM symposium on User interface software and technology* (ACM, 2012) pp. 435–444.
- [14] A. V gele, M. Hermann, B. Kr ger, and R. Klein, *Interactive steering of mesh animations*, in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Eurographics Association, 2012) pp. 53–58.
- [15] A. Fender, J. M ller, and D. Lindlbauer, *Creature teacher: A performance-based animation system for creating cyclic movements*, in *Proceedings of the 3rd ACM Symposium on Spatial User Interaction* (ACM, 2015) pp. 113–122.
- [16] H. Rhodin, J. Tompkin, K. In Kim, K. Varanasi, H.-P. Seidel, and C. Theobalt, *Interactive motion mapping for real-time character control*, in *Computer Graphics Forum*, Vol. 33 (Wiley Online Library, 2014) pp. 273–282.
- [17] I. Baran, D. Vlastic, E. Grinspun, and J. Popovi , *Semantic deformation transfer*, in *ACM Transactions on Graphics (TOG)*, Vol. 28 (ACM, 2009) p. 36.

- [18] M. Ben-Chen, O. Weber, and C. Gotsman, *Spatial deformation transfer*, in *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (ACM, 2009) pp. 67–74.
- [19] L. Kavan, *Part i: direct skinning methods and deformation primitives*, in *ACM SIGGRAPH* (2014) pp. 1–11.
- [20] 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* (ACM Press/Addison-Wesley Publishing Co., 2000) pp. 165–172.
- [21] N. Magnenat-Thalmann, R. Laperriere, and D. Thalmann, *Joint-dependent local deformations for hand animation and object grasping*, in *In Proceedings on Graphics interface'88* (Citeseer, 1988).
- [22] I. Baran and J. Popović, *Automatic rigging and animation of 3d characters*, in *ACM Transactions on Graphics (TOG)*, Vol. 26 (ACM, 2007) p. 72.
- [23] A. Jacobson, Z. Deng, L. Kavan, and J. Lewis, *Skinning: Real-time shape deformation*, in *ACM SIGGRAPH*, Vol. 22 (2014).
- [24] D. Zorin, P. Schröder, and W. Sweldens, *Interactive multiresolution mesh editing*, in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (ACM Press/Addison-Wesley Publishing Co., 1997) pp. 259–268.
- [25] I. Guskov, W. Sweldens, and P. Schröder, *Multiresolution signal processing for meshes*, in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (ACM Press/Addison-Wesley Publishing Co., 1999) pp. 325–334.
- [26] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H.-P. Seidel, *Laplacian surface editing*, in *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing* (ACM, 2004) pp. 175–184.
- [27] Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, and H.-Y. Shum, *Mesh editing with poisson-based gradient field manipulation*, in *ACM Transactions on Graphics (TOG)*, Vol. 23 (ACM, 2004) pp. 644–651.
- [28] R. W. Sumner, M. Zwicker, C. Gotsman, and J. Popović, *Mesh-based inverse kinematics*, *ACM transactions on graphics (TOG)* **24**, 488 (2005).
- [29] O. Sorkine and M. Alexa, *As-rigid-as-possible surface modeling*, in *Symposium on Geometry processing*, Vol. 4 (2007).
- [30] X. Shi, K. Zhou, Y. Tong, M. Desbrun, H. Bao, and B. Guo, *Mesh puppetry: cascading optimization of mesh deformation with inverse kinematics*, in *ACM Transactions on Graphics (TOG)*, Vol. 26 (ACM, 2007) p. 81.
- [31] A. Jacobson, I. Baran, L. Kavan, J. Popović, and O. Sorkine, *Fast automatic skinning transformations*, *ACM Transactions on Graphics (TOG)* **31**, 77 (2012).
- [32] S. J. Wright and J. Nocedal, *Numerical optimization*, Springer Science **35**, 7 (1999).
- [33] R. Tibshirani, *Regression shrinkage and selection via the lasso*, *Journal of the Royal Statistical Society. Series B (Methodological)* , 267 (1996).
- [34] C. Brandt and K. Hildebrandt, *Compressed vibration modes of elastic bodies*, *Computer Aided Geometric Design* **52**, 297 (2017).
- [35] R. Parent, *Computer animation: algorithms and techniques* (Newnes, 2012).