공학박사학위논문

# Semantic Analysis for Human Motion Synthesis

# 사람 동작 생성을 위한 의미 분석

2017년 2월

서울대학교 대학원
전기.컴퓨터공학부
현 경 렬

# Abstract

One of main goals of computer-generated character animation is to reduce cost to create animated scenes. Using human motion in makes it easier to animate characters, so motion capture technology is used as a standard technique. However, it is difficult to get the desired motion because it requires a large space, high-performance cameras, actors, and a significant amount of work for post-processing. Data-driven character animation includes a set of techniques that make effective use of captured motion data.

In this thesis, I introduce methods that analyze the semantics of motion data to enhance the utilization of the data. To accomplish this, various techniques in other fields are integrated so that we can understand the semantics of a unit motion clip, the implicit structure of a motion sequence, and a natural description of movements. Based upon that understanding, we can generate new animation systems. The first animation system in this thesis allows the user to generate an animation of basketball play from the tactics board. In order to handle complex basketball rule that players must follow, we use context-free grammars for motion representation. Our *motion grammar* enables the user to define implicit/explicit rules of human behavior and generates valid movement of basketball players. Interactions between players or between players and the environment are represented with semantic rules, which results in plausible animation. When we compose motion sequences, we rely on motion corpus storing the prepared motion clips and the transition between them. It is important to construct good motion corpus to create natural and rich animations, but it requires the efforts of experts. We introduce a semi-supervised learning technique for automatic generation of motion corpus. Stacked autoencoders are used to find latent features for large amounts of motion capture data and the features are used to effectively discover worthwhile motion clips. The other animation system uses natural language processing technology to understand the meaning of the animated scene that the user wants to make. Specifically,

the script of an animated scene is used to synthesize the movements of characters. Like the sketch interface, scripts are very sparse input sources. Understanding motion allows the system to interpret abstract user input and generate scenes that meet user needs.

**keywords:** Computer Graphics, Character Animation, Data-driven Motion Synthesis, Motion Classification, Machine Learning

**Student Number:** 2008-21005

# Contents

# List of Figures

# 1

# Introduction

Character animation plays a crucial role in bringing virtual characters to life. In animated films, a character takes 24 poses per second situated by animators. Since pose settings are cumbersome tasks, a skilled animator composes a few seconds of character animation a week. To reduce the amount of tasks, keyframing methods have been proposed. They have helped animators by automatically generating the movements of characters between keyframe poses. However, sometimes they create unnatural movements of characters and still require a significant amount of tasks.

As a way for generating natural looking scenes, physically-based simulation has been used. It uses physical laws of the real world to simulate a virtual world. Since physical laws are inferred from observation of the real world, animation scenes synthesized with physically-based simulation are realistic therefore widely used to create scenes with fluids such as water and air. Nevertheless, to control objects in a desired way requires tedious parameter

tuning and considerable computation time for solving space-time optimization problems. It is much harder to control characters such as biped, quadruped because it yields too complicated problems to solve.

Another approach to bring verisimilitude to the virtual world is using the data observed in the real world. The motion data acquired in the real world shows valid movement of characters, so if you combine it carefully, you can get plausible results. This approach is called data-driven approach. A data-driven approach requires a sufficient amount of data and methods to utilize the data. A main source for collecting motion data is motion capture, a process that records the movement of people and objects. With the development of motion capture system, we can store information from the trajectories of joints to the minute vibrations of facial muscles. Along the improvement of capture technology, application methods have also been advanced. Studies on motion data enables us to cut, stitch and modify original motion data. We can adapt motion sequences to new environments and blend two motion fragments to generate motion in-between.

A general data-driven animation system takes user input and generates animated scenes based on user needs. We can decompose the system into four components; user interface, interpreter, synthesizer, and motion analyzer. Their relationship is shown in Figure 1.1.

**User interface** offers a variety of ways to express what the user wants and transforms user input into machine-readable motion representation. For example, a line drawn on the screen of a sketch interface is converted to a trajectory constraint that the character must follow and the handles of a motion editing system are translated as position constraints.

**Interpreter** forms solvable problems using prior knowledge. Since the motion represen-

tation generated through the user interface is incomplete and abstract, we should approximate the rest appropriately to synthesize an animation from it. For example, when drawing a character's trajectory, we implicitly expect him to show a natural gait and when editing a motion, we want the movement of the character not to be very different from the original motion. Motion graphs [38, 34] model motion corpus as a directed graph and transform motion synthesis problem into a problem of path finding on the graph.

**Synthesizer** composes animations with motion specifications provided by the interpreter. It solves the problem by using various optimizers. The result of solving the problem is a pose for each time of every character and place it in the environment to create a scene.

**Motion preprocessor** helps to generate the motion corpus. When the amount of capture motion data was small, manually created motion corpus was used. With the development of motion capture technology, obtaining motion data has been relatively easy so the demand for tools that can support motion corpus generation is growing. Motion preprocessor is not an integral part of the pipeline, but it reduces the amount of work required to create the system. For example, we observe the position of the foot to determine the walking cycle of locomotion. A match web [33] is a matrix that contains the distance of every pair of poses and from which we can find chains of similar motion clips.

This thesis introduces how to improve each component to create a new animation system. It is organized into three chapters and each of which focuses on the improvement of a single component.

**Motion Grammar**

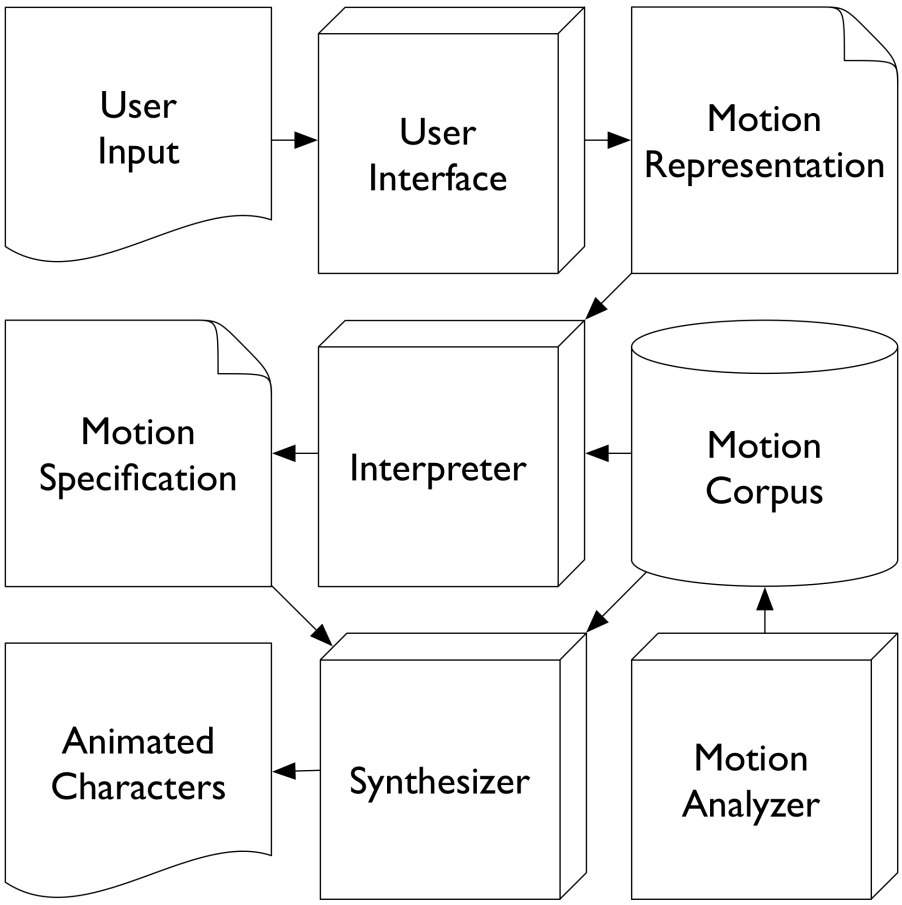


**Figure 1.1:** *A general data-driven animation pipeline. It consists of user interface, interpreter, synthesizer and motion analyzer.*

In Chapter 3, motion grammars are used to interpret the user's sketch on the tactics board. A motion grammar is a context-free grammar which is used to define structure of human movements. It can easily replaces formerly used simpler, finite-state machine based motion graphs. With the expressive power of context-free grammars, it can effectively deal with the *memory* of characters. When a computer animated film were just published, an animated scene consisting of a small number of characters and a simple environment is enough to fascinate people. With the development of technology, however, the demand for

splendid scenes has arisen and we need to deal with multiple characters and interactions between them. In order to deal with interactions between characters and surroundings, we define semantics feature which also fills a gap between context-free grammars for formal languages and motion grammars. Our multi-level MCMC (Markov Chain Monte Carlo) algorithm deals with the syntax, semantics, and spatiotemporal context of human motions to produce plausible, highly-structured, animated scenes. To demonstrate the expressive power of motion grammars, we built an animation system that produces an animation of basketball plays from drawings on a tactics board.

**Motion Embedding**

Motion corpus consists of unit motion clips which are processed for reuse. The process involves cropping and labeling. It is important to determine which frame to crop because we can concatenate two motion clips to form a motion sequence when their boundary poses are very similar. Labeling cropped motion clips allows the character to perform certain actions or to follow the given trajectory. Recent motion capture system allows us to obtain rich motion data easily, but it still takes a long time to process the captured motion data. Motion embedding is a method mapping the space of poses to a low-dimensional vector space in which we can effectively figure out relationship between poses. To achieve fully automatic system we construct stacked autoencoders. Training autoencoders requires no manual tasks and it is scalable. Effectiveness of our motion embedding technique is demonstrated in Chapter 4.

**Text to Animation**

Natural language is the most natural tool for communication. When we film movies, we

write a play script which contains the action and speech of characters and according to the script, actors or virtual characters play their role. In Chapter 5, we propose an automatic animation system which can generate animated characters directly from the text on a script. Like an actor has his or her own role in a play, a word in a sentence has its own role. Dependency parsers help us to understand semantics of words on a script and to translate them to actions of characters and their spatiotemporal constraints. Word embedding technique is used to find correspondence of strange words in scripts and motion clips in motion corpus.

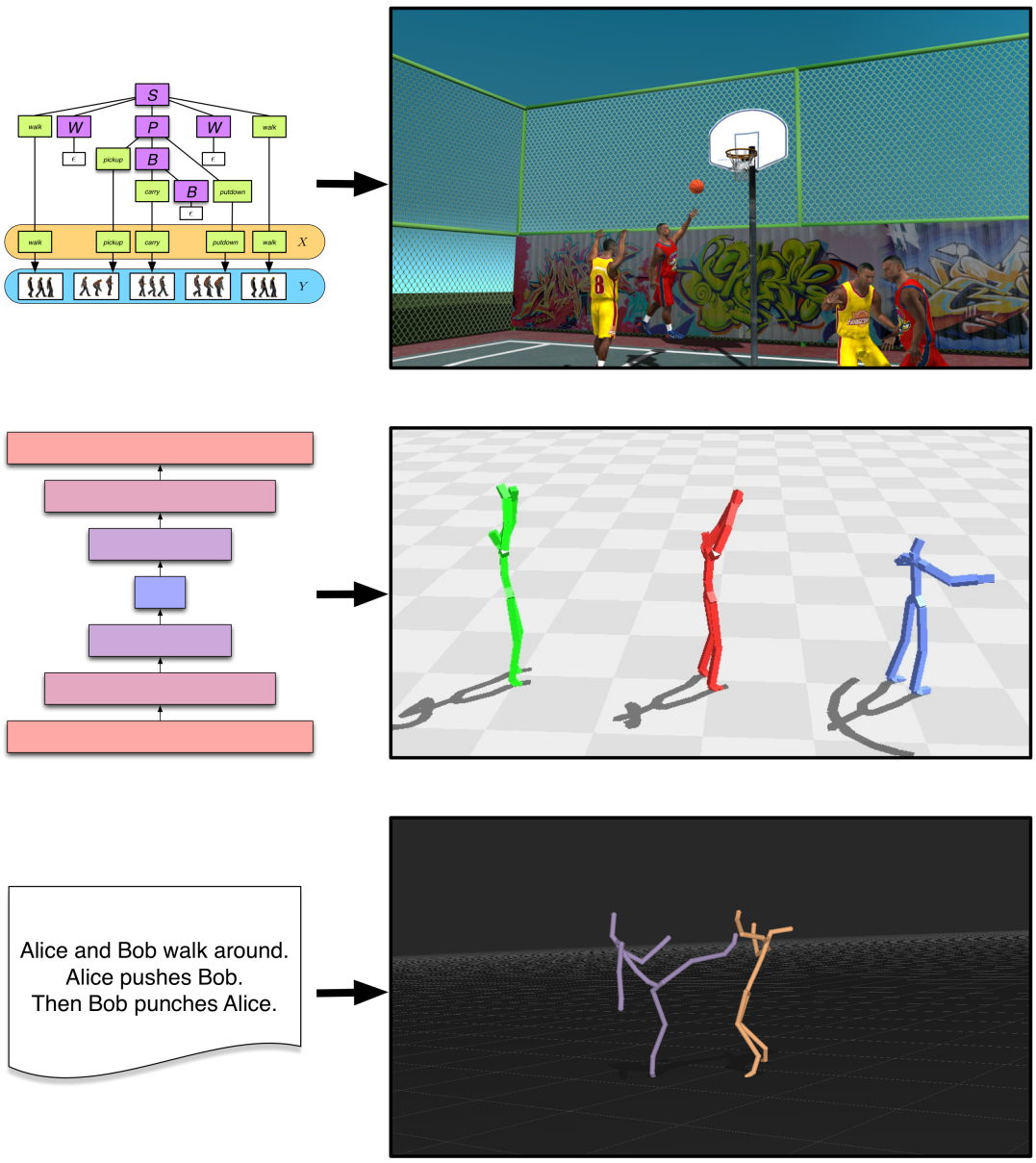


**Figure 1.2:** *Improving a component of the animation pipeline yields a new animation system. Formal grammar theory, machine learning and natural language processing are used to enhance the interpreter, the motion preprocessor, and the user interface, respectively.*

# 2

# Background

## 2.1 Representation of Human Movements

Movements of human are driven by more than 200 bones and 600 muscles in a complicated way and expressed by the surface of skins. More than dozens of thousands of vertices are necessary to represent precise movements of those surfaces. Since they yield too many degree of freedoms to deal with, we often use a small number of handles to manipulate them. One of the most used model is skeletal animation in which a character has a hierarchical structure of bones and a vertex weight map which determines the positions of vertices of the character with respect to the positions and orientations of the bones of the character. With the technique, we can model movements of a character as movement of its bones or joints. Once a hierarchy of bones and joints and their initial configuration is defined, a pose

of a character can be described as

$$\mathbf{p} = \{p, q_0, q_1, \cdots q_N\} \in \mathbb{P}$$

where $p$ is the position of the root joint and $q_i$ is the orientation of the $i$th joint, respectively. Since each joint other than the root joint is connected to their parent joint by a rigid bone, only the root joint has degree of freedoms for position.

Motion of a character is a continuous sequence of poses and we can recognize it as a mapping which maps time to a pose of the character.

$$\mathbf{m}(t) : \mathbb{T} \to \mathbb{P} = \{p(t), q_0(t), q_1(t), \cdots q_N(t)\}$$

where $\mathbb{T} \subset \mathbb{R}$ is a valid range of time on which a pose of the character is defined.

For notational convenience, we note $\mathbf{m}_t = \mathbf{m}(t)$. We are often interested in not the actual mapping but its sampled values (e.g., $\mathbf{m}_0, \mathbf{m}_1, \mathbf{m}_2, \ldots$).

Applying various operations on motion data allows us to generate a new motion sequence which adapts to a strange environment, performs a new task, and achieve a certain objective. First, a pose is very similar to a point in an affine space and we can define the pose displacement for two poses of the same character as

$$\mathbf{p}^\alpha - \mathbf{p}^\beta = \{p^\alpha - p^\beta, (q_0^\beta)^{-1} q_0^\alpha, (q_1^\beta)^{-1} q_1^\alpha, \cdots (q_N^\beta)^{-1} q_N^\alpha\}.$$

Then displacement vectors form the vector space $\mathscr{V}$ such that

$$\begin{aligned} &\text{for } \mathbf{v}^\alpha, \mathbf{v}^\beta, \mathbf{v} \in \mathscr{V}, c \in \mathbb{R} \\ &\mathbf{v}^\alpha + \mathbf{v}^\beta = \{p^\alpha + p^\beta, q_0^\alpha q_0^\beta, q_1^\alpha q_1^\beta, \cdots q_N^\alpha q_N^\beta\} \\ &c\mathbf{v} = \{c \cdot p^v, (q_0^v)^c, (q_1^v)^c, \cdots, (q_N^v)^c\} \end{aligned}$$

A pose displacement can be added to a pose to form the new pose.

$$\mathbf{p}+\mathbf{v}=\{p+p^v, q_0 q_0^v, q_1 q_1^v, \cdots q_N q_N^v\}$$

Operations above generally extends to be applied on motion data.

$$(\mathbf{m}^\alpha-\mathbf{m}^\beta)(t)=\mathbf{m}^\alpha(t)-\mathbf{m}^\beta(t)$$

$$(\mathbf{m}+\mathbf{v})(t)=\mathbf{m}(t)+\mathbf{v} \qquad \text{for } \mathbf{v}\in\mathscr{V}$$

Now we can introduce advanced applications.

**Motion Warping** is the process that modifies the give motion data a little so that it can satisfy desired objectives. For example, we can change the target of a kicking motion or the height of chair on which a character sits. Let $\mathbf{m},\mathbf{p}$ be a given motion and the target pose at the time $s$, respectively. Then, the warped motion

$$Warp(\mathbf{m},\mathbf{p},s)(t)=\mathbf{m}(t)+\big(w(t)\cdot(\mathbf{p}-\mathbf{m}(s))\big)$$

where $w(t):\mathbb{T}\to[0,1]$ is a smooth warping function such that $w(s)=1$ and $w(t)=0$ for $t\notin\mathscr{N}(s)$. For example,

$$w(t)=\begin{cases}\cos\left(\dfrac{\pi}{2}\cdot\dfrac{t-s}{\Delta}\right) & \text{if } |s-t|\le\Delta\\ 0 & \text{otherwise}\end{cases}$$

can be used as a warping function.

**Motion Stitching** is concatenating two or more motion clips to form a long motion sequence. Assume we want to stitch two motions $\mathbf{m}^\alpha,\mathbf{m}^\beta$ where $\mathbf{m}^\alpha$ is defined on time $[0,s]$ and $\mathbf{m}^\alpha(s)\approx\mathbf{m}^\beta(0)$. Then, the stitched motion

$$Stitch(\mathbf{m}^\alpha,\mathbf{m}^\beta,s)(t)=\begin{cases}\mathbf{m}^\alpha(t) & \text{if } t\le s\\ \tilde{\mathbf{m}}^\beta(t-s) & \text{if } t>s\end{cases}$$

where $\tilde{\mathbf{m}}^{\beta} = Warp(\mathbf{m}^{\beta}, \mathbf{m}^{\alpha}(s), 0)$ is the warped motion to adjust their boundaries. Note that the boundaries of motion clips being stitched should be very similar or the stitched motion is very awkward.

## 2.2 Motion Annotation

Motion segmentation is the process that crops captured motion data into motion clips. It involves determining the boundaries and length of the motion clip, which is important when stitching motion clips. Decreasing the number of boundaries increases the length of the motion clip, making it too specialized and generally difficult to use. If the length of the motion clip is too short, the type of motion clip is too fragmented to control the character as desired. Motion classification is the task of categorizing cropped motion clips which makes it easier to find out which action a character should take when controlling a character. For example, we can jump a character by stitching a motion clip labeld ‘jump’ that matches the current state of the character.

Constructing a reliable motion corpus requires good quality in both motion segmentation and classification. Since they are fundamental requirement of character animation, there have been a lot of studies on developing automatic system. Principal component analysis have been widely to segment motion capture data [7]. There have been studies using hand-made features to automatically annotate motion. [6, 35]. Dynamic time warping have been used to the similarity between two motion clips. [33, 52, 51, 17].

These algorithms are based on comparing poses, that is, that two motion clips that share a

similar pose sequence are likely to be classified in the same class. Since it is necessary to construct a similarity matrix and find a path with a high similarity value, it does not scale to a very large motion database which has more than a million poses. In situations where public motion databases are available [1, 2], we need a scalable algorithm that can generate very large motion corpus utilizing a large amount of motion data

Unsupervised learning is to discover knowledge about the data. From unlabeled input data alone, it figures out intrinsic structure of data and yields meaningful results. It includes clustering such as k-means clustering, Gaussian mixture model(GMM), and latent Dirichlet allocation(LDA) and latent feature extractions such as principal component analysis(PCA), multidimensional scaling(MDS), and Gaussian process latent variable model(GPLVM). In computer animation field, PCA has been used widely to get low-dimensional feature vectors [3, 63]. GPLVM, a nonlinear generalization of PCA which can effectively learn input data, has been regarded as an alternative to PCA because of its expressive power [22, 67, 43]. Nevertheless, due to the the complexity of time and space, it can not be applied to large amounts of data.

## 2.3 Motion Grammars

Motion capture is a major source of realism in modern character animation. Data-driven approaches commonly segment motion capture data into short fragments and splice them in a novel sequence. Animation techniques based on motion capture can roughly be classified into three categories. A large class of data-driven methodologies edit individual fragments subject to new requirements. There exist another class of methodologies that focus on tem-

poral sequencing and spatial alignment of motion data in space and time. The first category of techniques tend to make smooth, continuous changes over individual motion fragments or a family of parameterized motions, while the problems in the second category are discrete and combinatorial. The last category of techniques tried to solve both continuous editing and discrete planning simultaneously.

The continuous editing of motion is usually formulated as constrained optimization, which minimizes the deviation from the original motion data subject to user-specified constraints and requirements. This formulation has been effective for a wide range of problems, such as retargeting motion to new characters [19], interactive manipulation [39], blending a family of similar motions [57], statistical modeling [50, 49], and incorporating physics-based objectives and constraints [46]. The idea has further been explored to deal with multiple interacting characters in the context of interactive manipulation [36, 32, 31].

The combinatorial planning of action sequences often requires an efficient data structure to store and search motion data. The most popular structure is a motion graph [38, 34], which is essentially a finite state machine encapsulating the connectivity among motion fragments. The concept of motion graphs has further been elaborated to cope with families of parameterized motions [60]. Good segmentation and clustering of motion fragments are key ingredients of building effective motion data structures [7, 8]. Provided that such a structure is built, synthesizing novel motion sequences entails combinatorial searching through the connectivity among motion fragments. Temporal sequencing of motion fragments has been addressed by using state-space search [38, 34, 58], dynamic programming [6], min-max search [62], and policy learning [48, 65]. State-space search methods are closely related to the path planning of three-dimensional characters in highly-constrained and dynamically-

changing environments [12, 42].

Naturally, there have been efforts to integrate continuous optimization and combinatorial planning into a single framework. Motion data and their connectivity can be projected into a single continuous configuration space via dimensionality reduction [59] and can be retrieved from the configuration space by using multivariate regression [41]. Combinatorial planning problems can be reformulated as continuous optimization in the projected space. Alternatively, careful design of user interfaces can allow two heterogeneous types (continuous and discrete) operations to occur seamlessly in interactive manipulation of motion data [32].

Animating multiple interacting characters is a very challenging problem because the computational complexity of naïve approaches scales exponentially with respect to the number of characters. Many recent studies exploited *motion patches* to address the problem. While Lee et al [40] originally invented motion patches to animate characters in complex virtual environments, subsequent studies adopted motion patches to deal with interactions among characters and coordinate their actions in the spatiotemporal domain [61, 28]. Won et al [69] presented a pre-visualization system that generates full-body fight scenes of multiple characters from a high-level graphical scene description. Our work also addresses motion synthesis with single and multiple characters, but tackles a new aspect of the problem with emphasis on understanding, formulating, and animating the structured behavior of individuals and their interactions using formal grammars.

Formal languages and context-free grammars have previously been explored in the computer graphics community for shape analysis and procedural geometric modeling [45, 10]

and in robotics for the control of robotic manipulators [13]. Grammar induction from human motion data has been used to understand the static structure of human movements [53, 24, 23]. To the best of our knowledge, we for the first time demonstrate structured, full-body motion synthesis for practical applications using motion grammars.

## 2.4 Natural Language Processing

Natural language processing has drawn attention from many researchers since the birth of the computer. It emerged with automatic translation and expanded to understanding meaning of human language and generating natural language. In order to derive semantics of a text, the machine should comprehend the syntax of a language and the usage of a huge number of words.

A part-of-speech is a grammatical category or a role of words in a sentence. For example traditional eight parts-of-speech are noun, verb, pronoun, preposition, adverb, conjunction, participle, and article. The list of parts-of-speech tags is not unique but varies from one corpus to another. To analyze the syntax of a language, recent corpus uses a more detailed set of tags; the Penn Treebank project [47] has 45 tags and the Brown corpus [16] brings 87 tags. There have been computational methods for assigning parts-of-speech to words. Rule-based methods [29] applies hand-written rules on a word and related words in a text. Learning tagged corpus allowed us to use stochastic methods [11, 14, 68] such as hidden Markov models, maximum entropy models. Recently, deep learning technology has produced impressive results [15, 44].

Natural language generation is the task of generating natural language from machine representation. It was used to produce weather forecasts from weather data [20], or to generate readable text based on sales data [4].Advancement of deep learning technology has affected also natural language generation. A trained recurrent neural network can be a text sequence generator which mimics the style of training data and a hand-writing system that follows the movement of the hand. [21]. Moreover, we can generate a natural language description of images [30, 66]. The reverse process, imagination from a text is also becoming trendy research topic [37]. Deep learning is also used to synthesize motion sequences [27], or control characters in a dynamic environment [54].

# 3

# Motion Grammar

The behavioral structure of human movements is imposed by multiple sources, such as rules, regulations, choreography, habits, and emotion. Our goal is to identify the behavioral structure in a specific application domain and create a novel sequence of movements that abide by structure-building rules. To do so, we exploit the ideas from formal language, such as rewriting rules and grammar parsing, and adapted those ideas to synthesize the three-dimensional animation of multiple characters. The structured motion synthesis using motion grammars is formulated in two layers. The upper layer is a symbolic description that relates the semantics of each individual's movements and the interaction among them. The lower layer provides spatial and temporal contexts to the animation. Our multi-level MCMC (Markov Chain Monte Carlo) algorithm deals with the syntax, semantics, and spatiotemporal context of human motion to produce highly-structured, animated scenes. The power and effectiveness of motion grammars are demonstrated in animating basketball games from drawings on a tactic board. Our system allows the user to position players

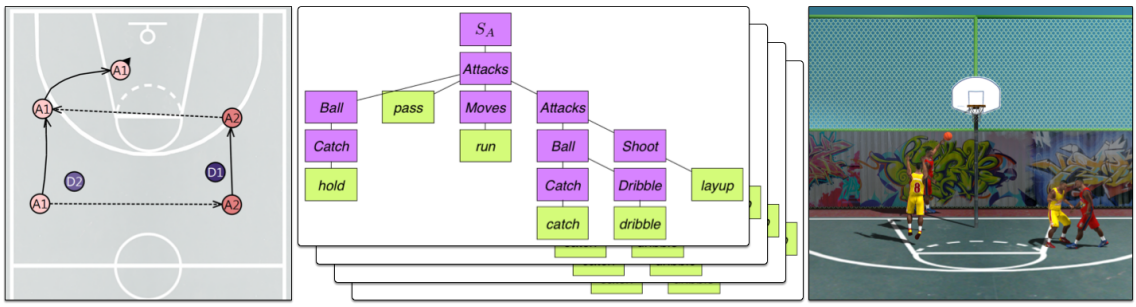


**Figure 3.1:** *Our motion grammar reconstructs a structurally-valid 3D animated scene from a sketch of the basketball tactics board.*

and draw out tactical plans, which are animated automatically in virtual environments with three-dimensional, full-body characters.

## 3.1 Overview

Natural human movements are often strongly structured. Behavioral structures may be imposed explicitly by rules, regulations, and choreography. Structures may arise implicitly from habits, cultural heritage, and emotion. For example, the rules of basketball dictate how many steps the player can take while holding the ball and prevent players from violations such as double dribbling. Basketball rules and regulations render highly-constrained structures of players' movements. Dance choreography is another example of structured human movements. Each category of dances has established the conventions of its form, motion, and rhythm. Dance choreographers dictate motion and form in a highly structured manner based on such conventions.

Motion capture technology is popular in computer graphics and thus large databases of

high-quality human motion data are readily available on the web. A common goal of data-driven animation research is to animate computer-generated characters by using a collection of canned motion data. Efforts for achieving this goal have developed a toolbox of data-driven techniques that range from low-level data manipulation to higher-level planning of a sequence of actions in virtual environments. Understanding the syntax and semantics of human movements would allow even higher-level control over the motion of animated characters and make it look plausible.

Our goal is to identify the behavioral structure of human movements and create a novel sequence of movements that abide by structure-building rules. To do so, we exploit formal language technologies, such as rewriting rules and grammar parsing, developed in computer science and computational linguistics. The underlying assumption is that behavioral structures can largely be formulated as context-free grammars. In graphics applications, human motion data have often been stored and maintained in motion graphs [38, 34], which are equivalent to finite state machines and regular grammars in terms of their expressive power. The motion graph maintains a collection of motion fragments and encodes the transitioning possibilities among motion fragments. Transitioning through the graph generates a sequence of character's actions. Context-free grammars are more expressive than regular grammars, so the use of context-free grammars allow us to uncover richer hierarchical structures, regularities, interactions, and patterns that could not be expressed in motion graphs. Motion grammars can be easily incorporated into existing animation systems that are equipped with motion graphs or finite state machines of character's actions.

The theory of formal language does not generalize easily to deal with human movements. Human motion data are high-dimensional, continuous, and time-series, whereas a formal

language is a string of discrete symbols. Human movements take place at specific spatial locations and specific time instances. The spatial and temporal contexts do not exist in formal languages. Our *motion grammar* is a set of context-free rewriting rules of quantized *motion symbols* and each rule is annotated with semantic context that may evaluate and trigger actions. We formulate a multi-character motion synthesis problem in two layers. The structure layer is a symbolic, tree-based description that relates the structure of each individual's movements and the interaction among them. The semantic layer provides spatial and temporal contexts to the animated scene and generates full-body animation. Our multi-level MCMC (Markov Chain Monte Carlo) algorithm deals with the syntax, semantics, and spatiotemporal context of human motion to produce plausible, highly-structured, animated scenes.

To demonstrate the power of motion grammars, we built an animation system that produces the animation of basketball plays from drawings on a tactics board. Most basketball coaches use clipboards to position players and draw out tactical plans. Our system reconstructs the full-body motion of multiple players dribbling, passing, and shooting balls. Motion grammars are used to describe basketball regulations and the behavioral patterns of offensive and defensive players.

## 3.2 Motion Grammar

The theory of formal languages is well-established in computer science. There exists a hierarchy of formal language classes. A set of languages generated by regular grammars form the smallest class. Context-free grammars generate a larger class of languages and

**Non-terminals :** $\{S, W, P, B\}$
**Terminals :** $\{walk, pickup, putdown, carry\}$

**Production Rules :**

$$
\begin{aligned}
S &\rightarrow walk^{\dagger}\ W\ P\ W\ walk^{\ddagger} \\
W &\rightarrow walk\ W \mid \varepsilon \\
P &\rightarrow pickup\ B\ putdown \\
B &\rightarrow carry\ B \mid \varepsilon
\end{aligned}
$$

**Semantic Rules :**

$$
\begin{aligned}
walk^{\dagger} &: f_{\text{pos}}(\text{start}, \mathbf{p}_0) \\
walk^{\ddagger} &: f_{\text{pos}}(\text{end}, \mathbf{p}_1) \\
pickup &: f_{\text{dir}}(\text{object}) \text{ and } f_{\text{dist}}(\text{object}, \leq 50cm)
\end{aligned}
$$

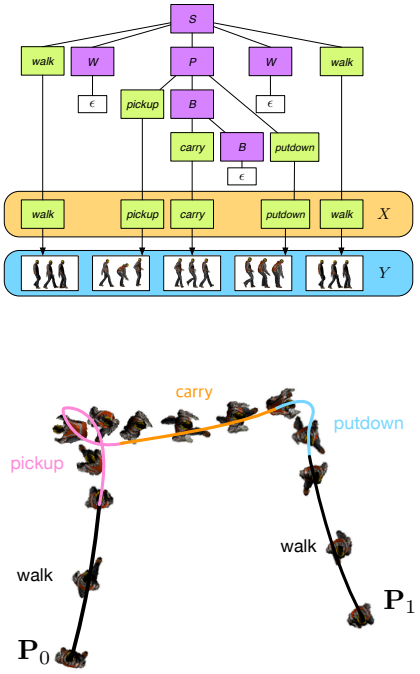


**Figure 3.2:** *A simple example. (Left) The motion grammar for carrying an object. (Right-top) A parse tree to generate an action string and a series of motion clips corresponding to the string. (Right-bottom) The motion sequence spliced and situated in the virtual environment.*

the class spanned by context-sensitive grammars is even larger. Regular grammars are simple, easy-to-implement and thus have frequently been used in many text processing applications. However, their expressive power is too limited to deal with the general structures of programming languages and natural languages. Context-sensitive grammars are the most powerful among them, but computing with context-sensitive grammars is extremely demanding. For example, parsing general context-sensitive grammars is NP-hard. Context-free grammars have long been considered a trade-off between two extremes and

still tremendously popular in designing programming languages and processing natural languages. Context-free grammars are not powerful enough to express the whole complexity of human movements and high-level behavioral patterns. However, we believe that context-free grammars capture a large portion of human behavioral structures that is significant enough to have practical uses.

The context-free grammar has a finite number of terminal and non-terminal symbols, and includes a set of rules which rewrites the original string of symbols. Context-free means that each individual rule replaces a single non-terminal symbol in the string with another string of terminal and non-terminal symbols. One of the non-terminal symbols serves as a starting symbol and fully expanding non-terminal symbols (until no non-terminals remain in the string) generates a string of terminal symbols. Parsing is a reverse process. It begins with a string of terminal symbols and searches for an ordering of rewriting rules that generates the input string. The result of parsing is a parsing tree. The internal nodes of the tree correspond to non-terminal symbols and its leaf nodes correspond to terminal symbols. The grammar is deterministic if any valid string has a unique parsing tree. Otherwise, the grammar is non-deterministic.

### 3.2.1 Instantiation, Semantics, and Plausibility

The motion grammar is a context-free grammar on motion sequences. Terminals are symbolic representations of unit actions, such as “a half cycle of walk” and “jump shoot”. Each non-terminal symbol has one or more production rules to substitute itself with series of symbols. For example, non-terminal symbol “Dribble” can produce an arbitrarily long

sequence of dribbling motions that may include many dribbling skills by recursively applying the production rules. Each terminal symbol is associated with many motion clips that perform a specific unit action. The multiplicity of motion clips includes spatial, temporal, and stylistic variations of acting out the action and thus provides rich connectivity between actions.

**Instantiation.** The action represented by a terminal symbol is *instantiated* in the virtual environment by picking a motion clip from many available choices associated with the symbol and locating the clip in the environment at a certain time. Instantiating a string of terminal symbols, which we call an *action string*, selects a sequence of associated motion clips. Splicing them and smoothing out the seams make an extended clip, which will be situated in the virtual environment. Let $X$ be an action string and $Y(X)$ be an extended motion clip instantiated by $X$. $X$ is a symbolic representation of the structure of a character's action, while $Y(X)$ is a concrete realization of the action in the spatiotemporal context. The actual form of $Y(X)$ is a sequence of a character's full-body poses varying over time.

**Semantic Rules.** The animated *scene* includes one or more characters, each of which produces a course of actions $X_i$ and its instantiation $Y_i = Y(X_i)$. Situating and orchestrating them in the common environment requires careful coordination and synchronization. We use three types of semantic rules to orchestrate multiple characters. Each individual terminal symbol can be annotated with a set of semantic rules, which are inherited to motion instances derived from the symbol. The semantic rule of the first type locates a character at desired position and/or direction $(\mathbf{p}_0, \theta_0) \in R^2 \times R$ at time $t$ in the environment.

$$f_{\text{pos}} = \|\text{pos}(Y_i, t) - \mathbf{p}_0\|, \tag{3.1}$$

$$f_{\text{dir}} = \|\text{dir}(Y_i,t) - \theta_0\|, \tag{3.2}$$

where $\text{pos}(Y_i,t)$ and $\text{dir}(Y_i,t)$ are the position and facing direction, respectively, of the root segment (pelvis) projected on the ground surface at the $t$-th frame of $Y_i$. The rule of the second type synchronizes the action of two (or more) characters.

$$f_{\text{synch}} = \big|(t_j - t_i) - \Delta t\big|. \tag{3.3}$$

For example, if one character passes a ball at frame $t_i$ so that the other character catches the ball at frame $t_j$, $\Delta t$ is the estimated time of the ball flying between the characters. The last type coordinates and aligns the motion of a character with respect to other characters and environment objects.

$$f_{\text{dist}} = \begin{cases} 0, & \text{if } d_{\text{near}} \le \|\text{pos}(Y_i,t) - \mathbf{p}_0\| \le d_{\text{far}}, \\ 1, & \text{otherwise.} \end{cases} \tag{3.4}$$

$$f_{\text{line}} = \text{dist}\big(\text{pos}(Y_i,t), \text{line}(\mathbf{p}_1,\mathbf{p}_2)\big). \tag{3.5}$$

For example, the distance rule $f_{\text{dist}}$ encourages a defensive player to stay near, but not too close to an offensive player. The line rule $f_{\text{line}}$ locates the defensive player on the line between the offensive player and the goal rim.

**Plausibility.** The structural plausibility of an action string $X_i$ is:

$$P(X_i) \propto g_{\text{parse}} \cdot \exp\big(-\frac{g_{\text{user}}}{w_1}\big), \tag{3.6}$$

where $g_{\text{parse}}$ is a binary boolean function, of which value is 1 if the string $X_i$ has a parse tree and 0 otherwise. $g_{\text{user}}$ is a user-control term, which we will discuss in Section 3.3, and $w_1$

is its weight value. The semantic plausibility of action instances $Y = \{Y_i\}$ is:

$$P(Y) \propto \exp\big(-\sum_{k \in \mathscr{S}(Y)} \frac{f_k}{c_k} - \frac{g_{\text{smooth}}}{w_2} - \frac{g_{\text{collision}}}{w_3} - \frac{g_{\text{effort}}}{w_4}\big), \tag{3.7}$$

where $\mathscr{S}(Y)$ is a set of semantic rules associated with $Y$, $c_k$'s are weight values, and $g_{\text{smooth}}$ is the smoothness of motion concatenation in $Y$. The smoothness is measured by the weighted sum of pose and velocity mismatches at the boundaries of two consecutive motion clips [38]. $g_{\text{collision}}$ is a binary boolean function that penalizes interpenetration between characters. The function value is 1 if there is any collision, and 0 otherwise. $g_{\text{effort}}$ is an optional term to choose a better animation among many plausible animation samples based on a secondary goal, which is either the total distance the characters travelled or their traveling time in the animation.

### 3.2.2 A Simple Example

A simple example grammar is given in Figure 3.2(left). The motion grammar consists of a set of non-terminal symbols $\{S, W, P, B\}$, a set of terminal symbols $\{walk, pickup, putdown, carry\}$, a starting symbol $S$, production rules, and semantic rules. The animation from the grammar shows a character starting to walk from $\mathbf{p}_0 \in R^2$ towards an object, picking it up, carrying it, putting it down, and walking towards the target location $\mathbf{p}_1 \in R^2$. The production rules generate a valid sequence (in other words, structure) of actions, while the semantic rules provide spatial context to the motion sequence. The semantic rules specify the start and target locations, and also declare a condition that the character can pick up an object if it faces the object and within $50cm$ from the object. Note that the grammar does not specify the position of the object to pick up. Its position may be provided by the user to control the

scenario.

The production rules expand the parse tree $\big($Figure 3.2(middle)$\big)$ to generate an action, which describes how many steps the character will take to get to the object, how many steps it will take while carrying the object, and how many steps it will take again to get to the target location after putting down the object. A series of motion clips instantiated from the action string describe the details of individual walking steps, such as stride length and steering angle. Many walking motion clips with different strides and steering angles are available in our motion repertoire, and thus the character's moving path depends on both the structure (e.g., the number of steps) of the action string and its instantiation (e.g., a series of motion clips with different stride lengths and steering angles). The structure and its instantiation interact with each other in a way that the character may either take a few steps with long stride or more steps with shorter stride.

Our multi-level MCMC algorithm in Section 3.4 samples the space of plausible animations with respect to the production and semantic rules. The most plausible animation from the samples will be chosen $\big($Figure 3.2(right)$\big)$. The final step is motion editing to remove any mismatches in character coordination and synchronization. The animation is a collage of canned motion clips, so motion clips may not fit precisely with each other and in the virtual environment. We use a Laplacian motion editing technique [32] to get rid of any residual mismatches, foot sliding, and interpenetration.

## 3.3 Basketball Tactics Board

Although our motion grammar is a general method that generalizes easily to deal with various applications, this paper focuses on a single application domain (i.e., basketball) to discuss the grammar-based modeling process in-depth. The basketball play is highly-structured and the players move in a highly-coordinated manner. While the structures of basketball plays are derived from basketball rules and regulations, the semantic rules emerge from the game context that includes game strategies, coordinated tactics, and adversarial interaction between offensive and defensive players. We found that the motion grammar is an effective tool for describing the structure and semantics of basketball plays (see Appendix for the motion grammar with production and semantic rules).

Given the motion grammar for basketball plays, we can generate three-dimensional, full-body basketball animation from a diagram sketch on the tactics drawing board (Figure 3.3). Since the diagram is simple, sparse, and abstractive, reconstructing 3D animation from the diagram is an ill-conditioned problem. The motion grammar is a key to the reconstruction of structurally-valid, semantically-meaningful plays from a sparse sketch. Our sketch-based interface for the tactics board includes five elements:

**Circle:** The circle indicates the location of a player. Offensive players are shown in red-tones and defensive players are shown in blue-tones.

**Solid arrow:** A solid arrow from a circle to the goal rim indicates a player shooting a basketball.

**Solid curve with arrow:** A solid curve with an arrow end describes a moving path of a

player who may walk, run, or dribble along the path.

**Dashed line with arrow:** A dashed line with an arrow end describes one player passing a basketball to the other player.

**Zigzag line:** The zigzag lines indicate feint moves and rapid pivot turns.

**T-end:** A moving path with a T-end indicates an offensive player setting a screen to block a defensive player.

The tactics diagram in Figure 3.3 implicates the structure and semantics of the play; the offensive player $A2$ outside the three-point line runs toward the free throw line to catch a pass from player $A1$, and dribbles the ball toward the goal rim to make a shoot. In the meantime, player $A3$ sets a screen to block the defensive player $D2$. A regular expression $\hat{X} = (walk|run)^*(catch)(dribble)^*(shoot)$ describes the role of the player $A2$ in the tactics. Any action string $X$ fits to the diagram if the regular expression $\hat{X}$ matches a substring of $X$. Here, the substring may not be continuous in $X$. The actual computation is simple. We drop the asterisk symbols from the expression $\hat{X}$ and then the following equation evaluates how well $X$ matches the description of an individual player.

$$g_{\text{user}} = \text{length}(\hat{X}) - \text{LCS}(X, \hat{X}), \tag{3.8}$$

where $\text{LCS}(X, \hat{X})$ is the length of the longest common subsequence of $X$ and $\hat{X}$.

The diagram also generates a set of auxiliary semantic rules. The circles locate the characters at desired positions ($f_{\text{pos}}$). The direction of a character when shooting or passing a ball is also implied in the diagram ($f_{\text{dir}}$). Synchronization rules ($f_{\text{synch}}$) are set between reciprocal actions, for example, “passing a ball” and “catching a ball”, or “setting a screen”

and "pushing against the screen". Although it is not clearly specified in the diagram who is holding the ball at the beginning of the animation, the ball holder can be inferred by tracing the diagram backward from either shoot or pass elements. The diagram is infeasible if it has more than one ball holders or has a link traversing backward in time.

## 3.4 Motion Synthesis

The basketball tactics diagram implicates basketball plays governed by a motion grammar $\mathscr{G}$ and auxiliary semantic rules $\{f_k\}$ derived from the diagram. An animated basketball play is a tuple $Z = \big(\{X_i\},\{Y_i\},\{\mathscr{T}_i\}\big)$, where $X_i$ is an action string of the $i$-th player, $Y_i$ is an instance of $X_i$, and $\mathscr{T}_i$ is a parse tree of $X_i$. A set of play scenes form a probability distribution, and a play scene is likely to emerge with probability $P(Z) \propto P(X)P(Y)$ in Equation (3.6) and (3.7), where $X = \{X_i\}$ and $Y = \{Y_i\}$. Conceptually, synthesizing an animated scene from a diagram is to choose the most probable scene from the distribution with respect to our plausibility measures. The probability distribution of animated scenes is high-dimensional and very complex. The dimensionality of an animated scene includes the complexity of full-body poses, the dimension of time, and the coordination of multiple characters. Therefore, searching the most probable scene in such a high-dimensional space is computationally demanding.

We formulate the scene reconstruction in the framework of Markov Chain Monte Carlo (MCMC) methods, which are a class of algorithms for sampling from a high-dimensional probability distribution from which direct sampling is difficult. Specifically, the Metropolis-Hastings algorithm is an MCMC method for obtaining a sequence of random samples that

approximate the target probability distribution. We modified and generalized the Metropolis-Hastings algorithm to cope with the complexity of animated scenes, including the hierarchical structure of parse trees, the symbolic description of actions, and their instances in the virtual environment. Our formulation separates the structure and semantics of basketball plays into two layers. The structure layer is symbolic and tree-structured, while the semantic layer is continuous and spatiotemporal. Our novel multi-level MCMC algorithm can deal with the heterogeneous (structure vs semantics, symbolic vs continuous, and space vs time) nature of the scene reconstruction problem.

The Metropolis-Hastings algorithm begins with an arbitrary initial sample $Z^{(0)}$ and performs random walk to generates a Markov chain of random samples $\{Z^{(1)}, Z^{(2)}, \cdots\}$. The random walk requires a proposal density function $Q(Z';Z)$, which suggests a new proposal $Z'$ given the previous sample $Z$. The proposal is accepted with the ratio

$$\alpha_{Z\to Z'} = \min\left(1, \frac{P(Z')Q(Z;Z')}{P(Z)Q(Z';Z)}\right). \tag{3.9}$$

If the new sample is rejected, the next sample is the previous sample $Z$. Repeating this process generates a chain of samples one by one. Theoretically, the chain of samples converge to the target probability distribution with an arbitrary proposal density function. Selecting $Q$, however, influences heavily the computational performance of the algorithm in practice. Therefore, defining an effective proposal density function is a key to the application of an MCMC algorithm.

**Random Walks** We introduce two types of random jumps to generate a Markov chain. The first type $Q_I(Y';Y)$ substitutes a motion clip with another motion clip in $Y$, leaving the action strings $X$ and its parse trees $\mathscr{T}$ remain unchanged. Note that the scene may have

multiple characters, each of which has an action string, its instance, and a parse tree. The proposal density $Q_I$ selects one of the characters and picks one motion clip among a series of the character's motion clips. A new motion clip is instantiated randomly from the same motion symbol that generated the original motion clip. The probability of suggesting a particular motion clip for substitution is proportional to the error induced by the motion clip in $-\log\big(P(Y)\big)$ from Equation (3.7). The motion clip will be more likely to be chosen if it violates semantic rules, involves in a collision, or the connection to its previous or next motion clip is not smooth.

The second type $Q_S(\mathscr{T}';\mathscr{T})$ makes a larger, structural change to the scene by replacing a subtree of a parse tree $\mathscr{T}_i \in \mathscr{T}$ with a new subtree. The root of any subtree is a non-terminal symbol $T$. We generate a new subtree randomly by applying production rules recursively starting from the root symbol and updating action strings accordingly. Motion clips are re-instantiated from new action symbols and therefore $Y$ is changed as well. The scope of the change varies depending on the size of the subtree. The change can be as little as replacing a single action symbol with another, and as big as replacing the whole scene with a random scene. The proposal density should make a good balance among jumps of various sizes. The probability of suggesting a particular subtree for substitution is proportional to the error induced by the subtree. The error $E(T)$ is defined recursively:

$$E(T) = \Big( \prod_{T' \in \text{child}(T)} E(T') \Big)^{\frac{1}{n+1}} \tag{3.10}$$

which is the geometric mean of errors of the child nodes, and $n$ is the number of child nodes. The rationale of using the geometric mean is as follows. If the errors in the child nodes are equally high or equally low, the parent node is equally likely to be suggested for substitution as its descendants. If error-prone nodes and errorless nodes are mixed in the

descendants, the parent node is less likely to be suggested than its descendants. In this case, the proposal density tends to suggest error-prone descendant nodes selectively rather than make a big jump of substituting the parent node. We used the $(n+1)$-th root instead of the $n$-th root to favor small jumps moderately over larger jumps.

**Multi-level MCMC Algorithm.** Our algorithm begins with initial scene $Z^{(0)}$ that is structurally-plausible (i.e., $P(X^{(0)}) > 0$ in Equation (3.6)). It means that the action strings have valid parse trees and match the input tactics diagram. The algorithm has two nested loops. The outer loop is for structural jumps, while the inner loop is for optimizing the instantiation of the scene. The structural jumps are allowed only between structurally-plausible configurations (line 3–6). The semantic-layer MCMC instantiates a new scene from the structure suggestion $X'$ and optimizes the scene with respect to semantic rules and quality measures in Equation (3.7) (line 7–16). random() is a function drawing a random number in $[0,1)$. The structure suggestion is either accepted or rejected based on a structure-layer MCMC method (line 17–23).

**Bootstrapping.** The bootstrapping is a process of finding a structurally-plausible initial configuration $(X^{(0)}, \mathscr{T}^{(0)})$ before our multi-level MCMC algorithm starts. Since action strings are always derived by using production rules and parse trees, $X^{(0)}$ is valid if $g_{\text{user}} = 0$. The bootstrapping is yet another MCMC algorithm that searches for a valid configuration with probability $P(X)$ and proposal density $Q_S(\mathscr{T}'; \mathscr{T})$.

**Parallel Tempering.** Although MCMC algorithms are theoretically guaranteed to converge even in high dimensional space, the more difficult problem is to make it converge faster. The speed of convergence is closely related to determining proper size of jump pro-

**Algorithm 1** Multi-level MCMC

$Z^{(0)} \leftarrow (X^{(0)}, Y^{(0)}, \mathscr{T}^{(0)})$

**for** $i \leftarrow 1$ **to** $N$ **do**

  **repeat**

    $\mathscr{T}' \sim Q_S(\mathscr{T}'; \mathscr{T}^{(i-1)})$

    $X' \leftarrow \text{string}(\mathscr{T}')$

  **until** $P(X') > 0$

  $\bar{Y}^{(0)} \leftarrow \text{instantiate}(X')$

  **for** $j \leftarrow 1$ **to** $M$ **do**

    $Y' \sim Q_I(Y'; \bar{Y}^{(j-1)})$

    $\beta \leftarrow \min(1, \frac{P(Y')Q_I(\bar{Y}^{(j-1)}; Y')}{P(\bar{Y}^{(j-1)})Q_I(Y'; \bar{Y}^{(j-1)})})$

    $$\bar{Y}^{(j)} \leftarrow \begin{cases} Y' & \text{if random}() < \beta \\ \bar{Y}^{(j-1)} & \text{otherwise} \end{cases}$$

  **end for**

  $Z' \leftarrow (X', \bar{Y}^{(M)}, \mathscr{T}')$

  $\alpha = \min(1, \frac{P(Z')Q_S(\mathscr{T}^{(i-1)}; \mathscr{T}')}{P(Z^{(i-1)})Q_S(\mathscr{T}'; \mathscr{T}^{(i-1)})})$

  $$Z^{(i)} \leftarrow \begin{cases} Z' & \text{if random}() < \alpha \\ Z^{(i-1)} & \text{otherwise} \end{cases}$$

**end for**

**return** $\{Z^{(1)}, \ldots Z^{(N)}\}$

| Action | Clips | Action | Clips | Action | Clips |
|---|---|---|---|---|---|
| screen | 9 | hold | 6 | feint | 10 |
| block | 20 | guard | 121 | push | 9 |
| pass | 41 | fake_shot | 11 | pivot_l | 10 |
| pivot_r | 14 | dribble | 146 | shoot | 27 |
| catch | 41 | walk | 64 | run | 72 |
| layup | 15 | | | Total | 598 |

**Table 3.1:** *The number of motion clips for each action symbol*

posals especially with complex structures such as motion grammars. When jumps are too big, proposals are easily rejected and thus new samples are rarely accepted. Using small jumps only, on the other hand, leads to trapping in local extrema. Instead of going through laborious parameter tuning, we can run different chains simultaneously using a parallel tempering scheme [18]. We generate multiple copies of Markov chains with target distributions $P_n(Z) \propto P^{1/T_n}(Z)$ of different temperatures. The chain of high temperatures allows big jumps, while the chain of low temperature tends to search samples near local extrema with small jumps. An important feature of parallel tempering is exchanging samples at different temperatures based on the Metropolis criterion.

$$\alpha_{n \leftrightarrow n+1} = \min\left(1, \frac{P_n(Z_{n+1})P_{n+1}(Z_n)}{P_n(Z_n)P_{n+1}(Z_{n+1})}\right) \tag{3.11}$$

This swapping process gradually sends good samples to cool chains to stabilize the optimization process and bad examples to hot chains to explore new possibilities more aggressively. The simultaneous generation of multiple chains is easily implemented on multi-core/multi-thread architectures to parallelize the computation.

## 3.5 Results

Our motion grammar for basketball plays is given in Appendix. The grammar has two sets of production rules; one for offensive players and the other for defensive players. The grammar has 19 non-terminal symbols and 17 terminal symbols. The offensive players start with non-terminal symbol $S_A$ to expand their action strings, while the defensive players start with $S_D$. Although we use two starting symbols in the example, the grammar can be adapted for individual player positions with a slight modification. Our basketball grammar is deterministic and in the LR(1) class of grammars. It means that any action string can be parsed sequentially by descending through productions recursively, picking the next production to expand using a single token of lookahead without backtracking. Even though our MCMC algorithm does not require deterministic grammars, it is convenient to have a deterministic parser when we want to use simpler motion synthesis algorithms based on A*-search, dynamic programming, or reinforcement learning.

The motion grammar can be easily incorporated into existing animation systems that are equipped with the motion graph or the finite state machine of the character's action. The motion grammar facilitates data-driven motion analysis and synthesis in many ways.

**Validation:** Given any motion data, the grammar parser can check if the motion is feasible with respect to the grammar governing its behavior. Grammar parsing reveals the hierarchical process how a sequence of actions are structured.

**Action Suggestion:** Assuming that a sequence of actions have already been performed, the parser can suggest a set of structurally-valid, plausible candidates for the next

| Diagram | # of characters | # of motion instances | # of animation frames | Computation time (second) | |
|---|---|---|---|---|---|
| | | | | Outer loop | Inner loop |
| Pass and Shoot | 4 | 33 | 179 | 25 | 217 |
| Screen | 5 | 18 | 132 | 28 | 215 |
| Backdoor | 4 | 20 | 136 | 17 | 160 |
| Give and Go | 6 | 36 | 158 | 45 | 309 |
| Double Team | 4 | 22 | 170 | 25 | 244 |
| Passing Drills | 3 | 34 | 235 | 103 | 485 |
| Weave Pass | 3 | 26 | 245 | 86 | 436 |

**Table 3.2:** *Results and performance.*

action. This allows us to choose a series of actions one-by-one sequentially as if we use a motion graph. It means that the motion grammar can replace a motion graph seamlessly in existing animation systems.

**Interactive Editing:** Interactive manipulation of motion data entails continuous deformation of motion paths and large deformation often leads to structural changes on a sequence of actions [32]. Discrete, structural editing of motion data may introduce a new action in the middle of the sequence to cope with user manipulation, remove an action from the sequence, or reorder the actions to form a novel sequence. Motion grammars can parse valid structural changes and suggest new plausible sequences.

**Motion Planning and Synthesis:** Given a collection of motion fragments, there exist a number of motion planning and synthesis algorithms that can generate a sequence of motion fragments by splicing them together to satisfy user-specified goals and constraints. The motion grammar can facilitate any synthesis problem that entails splicing of actions.

**Motion Data.** We collected about 100 minutes' worth of basketball motion data, which captured two to four players simultaneously in each session. We segmented and labelled raw motion data manually to identify 598 motion clips, each of which is associated with a terminal symbol (Table 3.1). The motion clips are annotated with context information, including keyframes where important events occur (e.g., the moment of releasing a ball for shooting/passing and the moment of catching a ball), reciprocal relations between motion clips (e.g., pass and catch, shoot and block), and the relative pose of interacting players (e.g., the position and direction of a catcher relative to the passer, and the position and direction of a shooter relative to the goal rim). Our motion data do not include hand motion. The gaze direction, hand shapes, and ball trajectories are added to match the scene context in the post-processing phase.

**Tactics Diagram.** A number of offensive and defensive tactics/drill plans are available in basketball textbooks and coaching guides in the form of diagram sketches. We reproduce 3D animated scenes for representative diagram sketches (Figure 3.4) including offensive tactics (e.g., Pass and Shoot, Backdoor, Screen, and Give and Go), defensive tactics (e.g., Double Team), and drills (e.g., Weave Pass). Our algorithm works well with any diagram sketches as long as a vocabulary of actions are readily available. The interface system is very easy to use. The user can sketch an arbitrary diagram using a set of predefined elements (e.g., location, move, pass, shoot, screen, and feint). Then, the system checks the validity of the diagram and generates a 3D animated scene in a fully automatic manner.

**Performance.** Reconstructing play scenes from scratch took 3 to 10 minutes depending on the number of characters, the complexity of interaction among them, and the length of the animation (Table 3.2). Performance statistics are measured on a desktop PC equipped

with an Intel Core i7-4820K CPU (8 cores, 3.7 GHz) and and 32 GB main memory, except for the 20-core parallel tempering example, which runs on another machine with slower Intel Xeon E7-4870 CPUs (2.4 GHz). The semantic layer sampling (the inner loop in the algorithm) takes more computation than the structure-layer sampling (the outer loop). The computation times for bootstrapping and motion editing in the post-processing phase are negligible comparing to the MCMC sampling. Parallel tempering effectively parallelizes the sampling procedure. As shown in Figure 3.6, the benefits of parallel tempering are twofold. It achieves a significant performance gain over single-chain sampling and, more importantly, finds a better solution effectively avoiding local extrema. The 6-core tempering converges to a better solution than a single Markov chain, and the 20-core tempering converges to a even better solution.

The optimization performance based on MCMC sampling depends heavily on the choice of the initial configuration. If the user changes the input diagram slightly, the animation can be updated incrementally by taking the previous result as the initial configuration for the subsequent optimization. Our algorithm allows the input diagram to be manipulated interactively and updates the corresponding animation at interactive rates with up to two players. The incremental update for a small change is in average two orders of magnitude faster than the full reconstruction from scratch, so it takes only several seconds to update the example scenes incrementally.

## 3.6 Discussion

We have presented the motion grammar as a general method for designing the behavior model of characters and generating animated scenes from a simple sketch. The rigorous formulation of the behavior model made it possible to synthesize the coordination and interaction among multiple characters, which are structurally-valid as well as semantically-meaningful. Even though our focus has been on animating basketball plays in this paper, our approach could be easily extended to deal with other types of scenes where there is a requirement for the structured behavioral patterns and the coordination of multiple interacting characters, such as sports, dancing, and social interactions.

Our motion grammar is a subset of the comprehensive grammar for basketball plays, since our motion grammar focused on modeling scenarios that are likely to appear in tactics plans. For example, consider a scenario that a ball is loose on the court and players compete for the ball. Such a scenario can occur in real basketball plays, but it is unlikely to consider such an unusual scenario in a tactics plan. An advantage of our grammar-based approach is its flexibility and scalability. Even though our motion grammar is not comprehensive in the present form, it is easy to add new production and semantic rules incrementally to deal with a larger domain of basketball plays, situations, and scenarios.

The most time-consuming part of the grammar-based modeling is motion data segmentation and labelling. Although there have been significant technical advances recently in data-driven animation, automating motion segmentation and labelling are still challenging. Many game developer companies have already built domain-specific motion databases and finite state machines for character animation using a number of labelled motion clips. The

motion grammar can be built on top of those readily-available motion databases to have more structure than a finite state machine can offer.

Recent basketball video games provide rich details of characters' motion using large, high-quality motion databases captured from professional players. The character animation algorithms in video games are usually simple and very efficient. The efficiency comes at a cost of compromising the quality of animation. Typically, game characters are allowed to slide or change its position or direction suddenly in an implausible manner. Game developers design finite state machines and character control mechanisms very carefully to make such artifacts less obtrusive. The goal of this work is different than what game developers are pursuing. Our MCMC algorithm is a general solution method to address the problem without compromising structural and semantic constraints.

Won et al [69] also addressed the animation of tightly-coupled multiple interacting characters from a sparse, high-level description, though their generate-and-rank method is quite different from ours and their problem domain (scripts-based description for fight scenes) is also different from our basketball tactics animation. Technically, the generate-and-rank method based on a motion graph and uniform random sampling is complementary to our motion grammars, which would provide more structures and better descriptions on their fight scenes. Uniform random sampling for motion synthesis can be replaced with our multi-level MCMC sampling to achieve significant performance improvements. Conversely, their generate-and-rank method can be a valuable supplement to our MCMC algorithm, which generates a chain of samples. Although generated samples are the best samples with respect to the plausibility measure, they are similar to each other due to the Markov property of random walks. The generate-and-rank algorithm would add diversity

to the system by providing the user with multiple distinct representative samples. Our motion grammar also improves the flexibility of description. While the relative position and orientation of characters participating in each interaction event are fixed in their work, our motion grammar allows the relationships among characters to be flexibly described in rules so that a range (from tightly-coupled to loosely coupled) of interactions can be dealt with in our work.

An interesting direction for future research is inferring motion grammars automatically from a corpus of motion data. In computational linguistics, grammar induction has been an active research topic to learn a formal grammar from a corpus of texts. Grammar induction is also called grammatical inference. Learning weak structures and habitual patterns from loosely organized dance choreography and idling pedestrians seems feasible [53]. However, learning stronger structures, such as basketball rules, directly from a database of basketball motions would be challenging even with state-of-the-art grammar induction algorithms. Probably, semantic reasoning and extra negative samples (violating basketball rules) would allow us to infer motion grammars from canned motion data.

# Basketball Grammar

**Starting symbols :** $\{S_A, S_D\}$

**Non-terminals :** $\{S_A, Moves, Move, Attacks,$
$Ball, Shoot, Catch, Feint, Runs,$
$Pivot, PivotL, PivotR, Fake,$
$Fakes, Dribble, Dribbles,$
$S_D, Defenses, Defense\}$

**Terminals :** $\{run, walk, idle, screen, hold, catch,$
$feint, pivot\_left, pivot\_right,$
$dribble, pass, shoot, layup,$
$fake\_shot, guard, block, push\}$

**Production Rules :**

| **Attack** | | |
|---|---|---|
| $S_A$ | $\rightarrow$ | $Moves\ Attacks$ |
| $S_A$ | $\rightarrow$ | $Attacks$ |
| $Moves$ | $\rightarrow$ | $Move \mid Move\ Moves$ |
| $Move$ | $\rightarrow$ | $run \mid walk \mid idle \mid screen$ |
| **Ball play** | | |
| $Attacks$ | $\rightarrow$ | $Ball\ pass\ Moves\ Attacks$ |
| $Attacks$ | $\rightarrow$ | $Ball\ Shoot \mid \varepsilon$ |
| $Ball$ | $\rightarrow$ | $Catch\ Pivot\ Dribble$ |

| | | |
|---|---|---|
| **Catch** | | |
| $Catch$ | $\rightarrow$ | $hold \mid catch \mid Feint\ catch$ |
| $Feint$ | $\rightarrow$ | $feint\ Runs$ |
| $Runs$ | $\rightarrow$ | $\varepsilon \mid run\ Runs$ |
| **Pivot** | | |
| $Pivot$ | $\rightarrow$ | $\varepsilon \mid PivotL\ Fake \mid PivotR\ Fake$ |
| $PivotL$ | $\rightarrow$ | $pivot\_left \mid pivot\_left\ PivotL$ |
| $PivotR$ | $\rightarrow$ | $pivot\_right \mid pivot\_right\ PivotR$ |
| **Fake** | | |
| $Fake$ | $\rightarrow$ | $\varepsilon \mid Fakes$ |
| $Fakes$ | $\rightarrow$ | $fake\_shot \mid fake\_shot\ Fakes$ |
| **Dribble** | | |
| $Dribble$ | $\rightarrow$ | $\varepsilon \mid Dribbles\ Pivot$ |
| $Dribbles$ | $\rightarrow$ | $dribble \mid dribble\ Dribbles$ |
| **Shoot** | | |
| $Shoot$ | $\rightarrow$ | $layup \mid shoot$ |
| **Defense** | | |
| $S_D$ | $\rightarrow$ | $Defenses$ |
| $Defenses$ | $\rightarrow$ | $Defense \mid Defense\ Defenses$ |
| $Defense$ | $\rightarrow$ | $run^{\dagger} \mid walk^{\dagger} \mid guard$ |
| $Defense$ | $\rightarrow$ | $block \mid push$ |

**Semantic Rules :**

$$
\begin{aligned}
pass: \quad & f_{\text{dir}}(\text{receiver}), \\
& f_{\text{synch}}(\text{receiver}, catch, \Delta t) \\
layup: \quad & f_{\text{dir}}(\text{rim}) \text{ and } f_{\text{dist}}(\text{rim}, \leq 2.5m) \\
shoot: \quad & f_{\text{dir}}(\text{rim}) \text{ and } f_{\text{dist}}(\text{rim}, > 1.5m) \\
feint: \quad & f_{\text{dist}}(\text{opponent}, < 1m) \\
fake\_shot: \quad & f_{\text{dist}}(\text{opponent}, < 1m) \\
screen: \quad & f_{\text{dir}}(\text{receiver}), \\
& f_{\text{synch}}(\text{opponent}, push, 0) \\
run^{\dagger}, walk^{\dagger}: \quad & f_{\text{dir}}(\text{opponent}),\ f_{\text{line}}(\text{opponent}, \text{rim}) \\
guard: \quad & f_{\text{dist}}(\text{opponent}, > 50cm \text{ and } \leq 2m) \\
block: \quad & f_{\text{dir}}(\text{opponent}), \\
& f_{\text{line}}(\text{opponent}, \text{rim}), \\
& f_{\text{synch}}(\text{opponent}, \{shoot, fake\_shot\}, 0)
\end{aligned}
$$

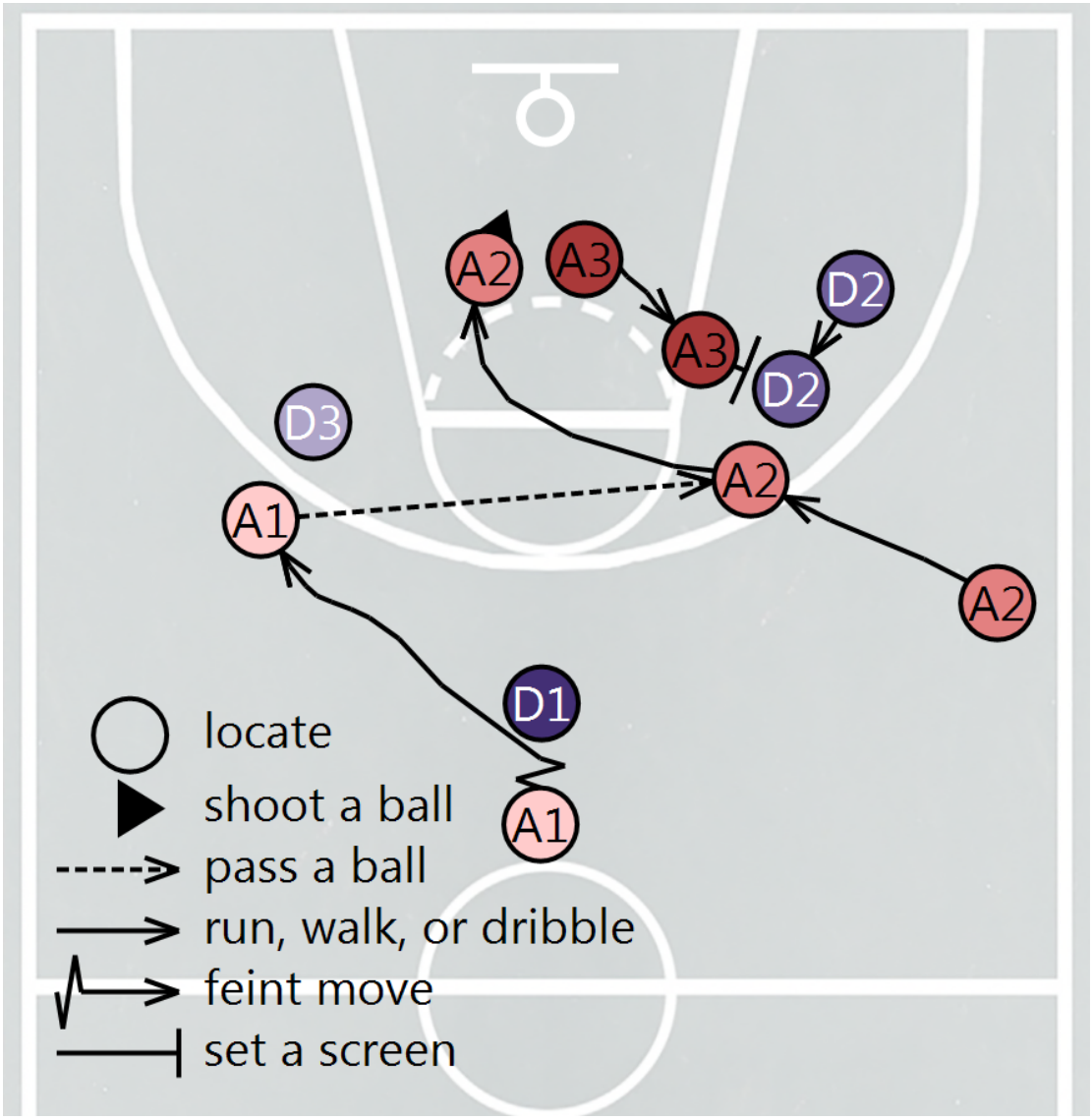


**Figure 3.3:** *A sketch-based interface for drawing basketball tactics diagrams*

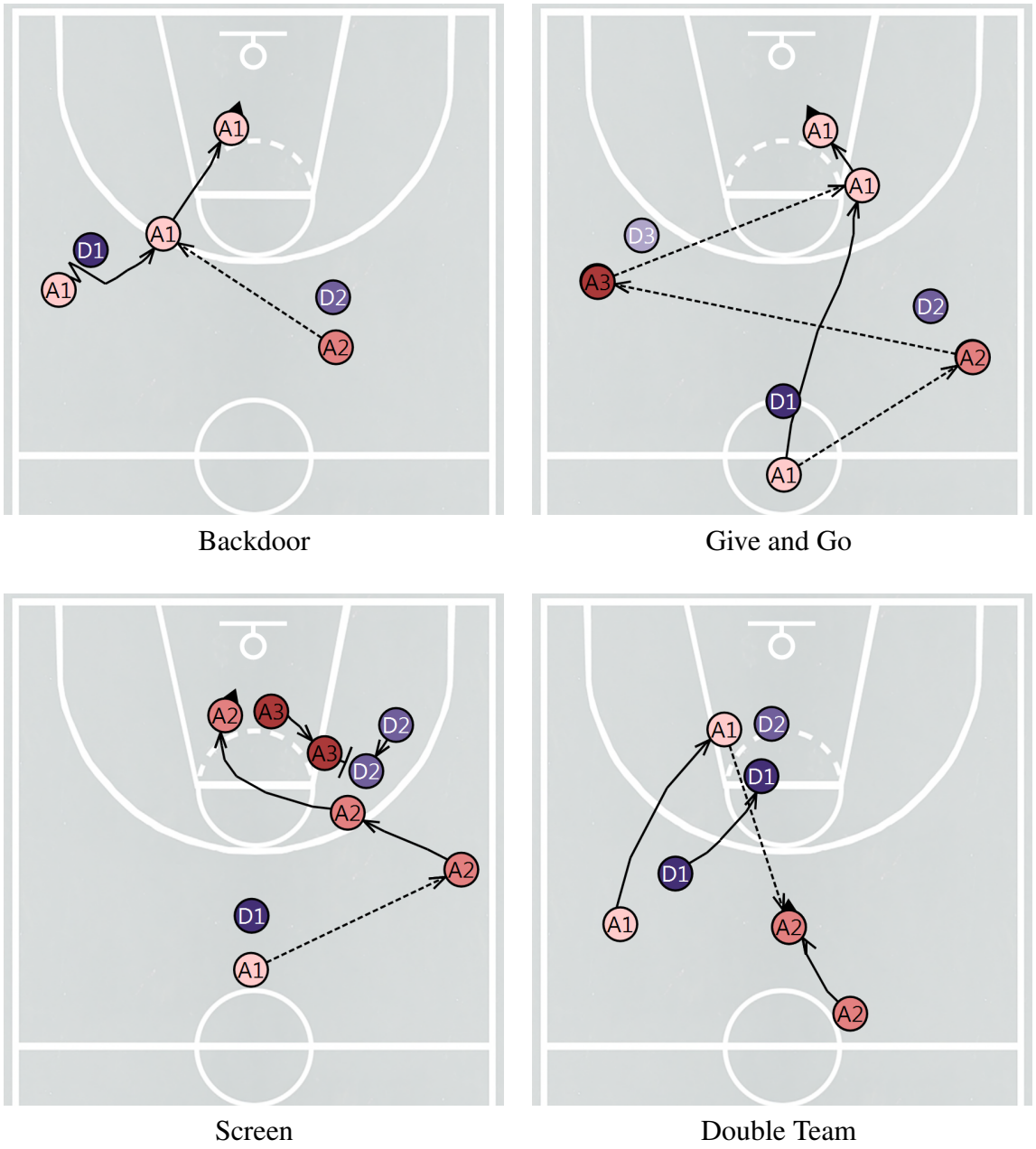


**Figure 3.4:** *Diagram sketches*

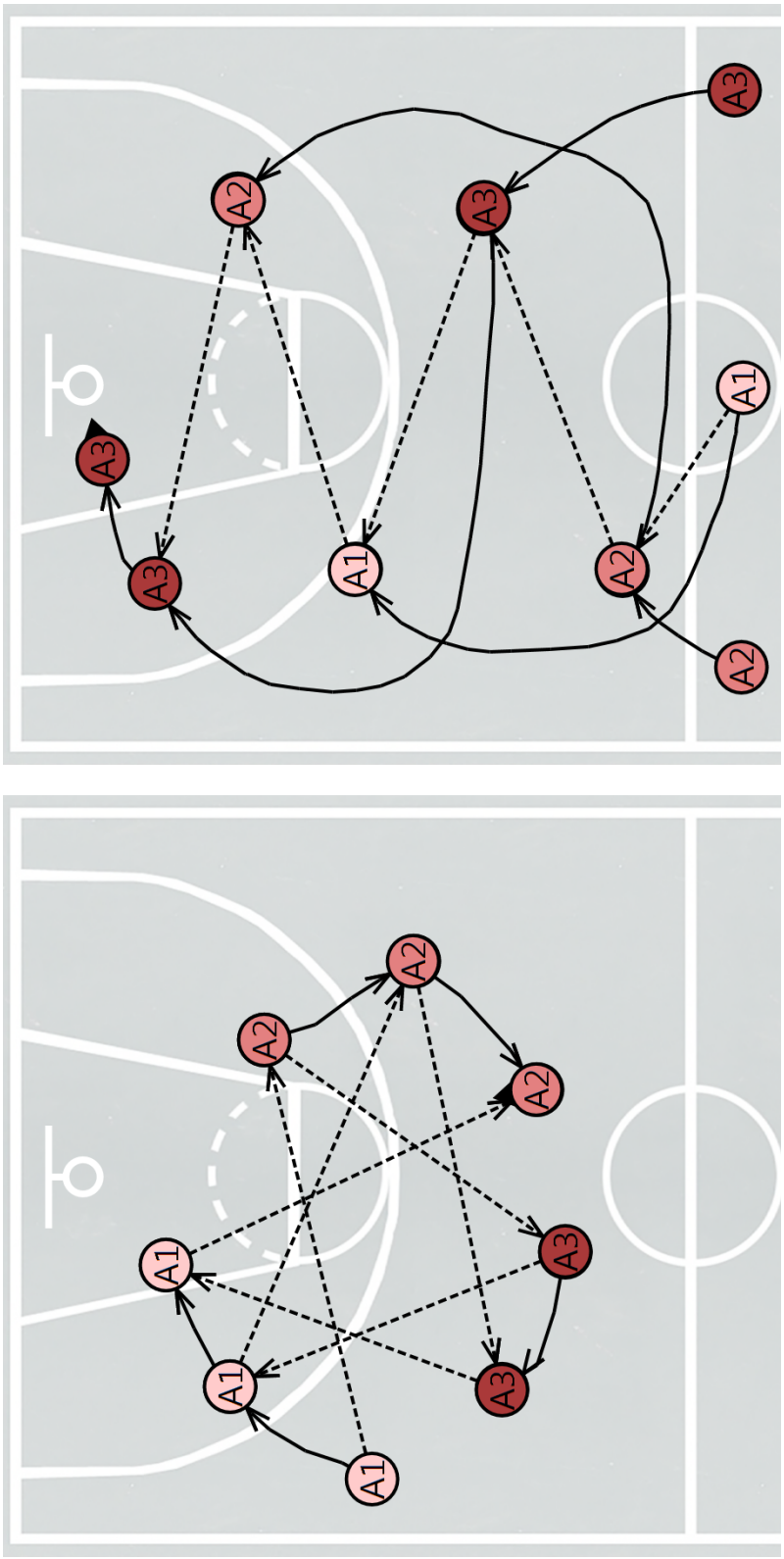


**Figure 3.5:** *Diagram sketches (continued). Complex interactions between characters can be represented easily in our system.*

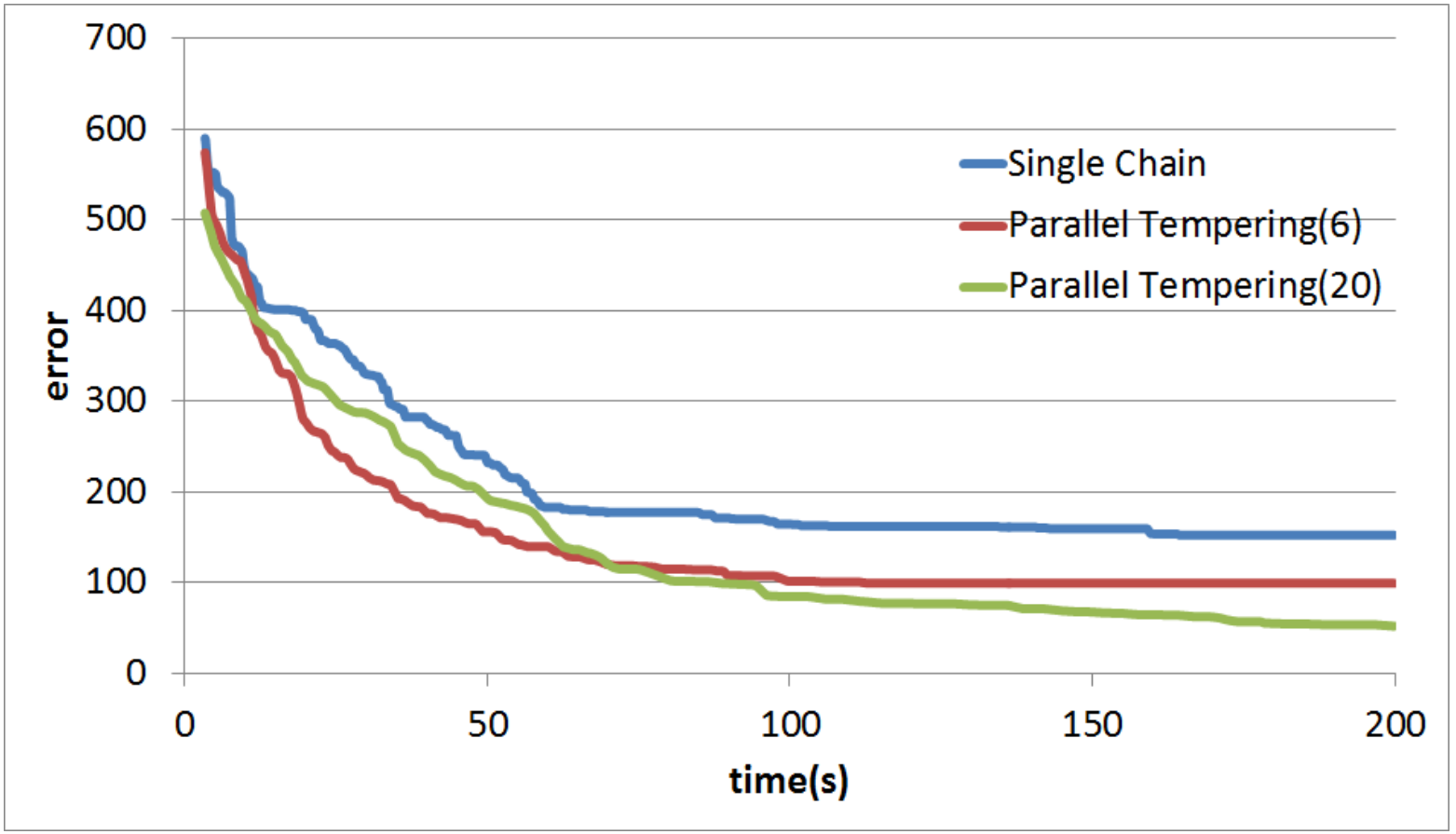


**Figure 3.6:** *A single Markov chain vs parallel tempering. The X-axis is the computation time in seconds, and the Y-axis is the error* $-\log(P(Y))$*. The performance is averaged over 10 trials for each of a single chain, 6-core tempering, and 20-core tempering.*

# 4

# Motion Embedding

This chapter introduces a motion embedding technique that can be used effectively for constructing motion corpus. With the advancement of motion capture technology, we can easily get the desired motion data. However, motion analysis that generates reusable motion corpus still focuses on locomotion and does not scale for applying to dynamic motion data. Analyzing, segmenting and classifying static motion like locomotion is relatively easy and well studied. Dealing with dynamic motion, however, still involves a large number of manual tasks. We introduce stacked autoencoders to analyze large amounts of dynamic motion data and find efficient and meaningful representation of original data for increasing reusability.

## 4.1 Overview

Motion data is a record of human action consisting of consecutive poses of a person. In order to reduce the complexity of human poses, a pose is represented by dozens of joints. Nevertheless, the motion data is high-dimensional because it is a time series data of poses. To describe high-dimensional data, we need to use non-linear, non-trivial representations.

Motion corpus is a collection of canned motion data. It stores labeled segmented motion clips to find the proper motion clip in a given context. Moreover, it considers connections between motion clips to make smooth animation of a character. Motion capture technology allows us to own a large amount of motion data, but segmenting and classifying motion data for reusing is an difficult task that requires expertise in motion data processing.

Semi-supervised learning is a machine learning technique that uses unlabeled data and labeled data together to train a model. It is used when there is a large amount of unlabeled data and it is not easy to obtain a large amount of labeled data. First, unsupervised learning is performed to figure out hidden structure of unlabeled data, and then a small number of labeled data is used to train the model. Using a stacked autoencoder, we build a semi-automatic motion corpus generator that discovers meaningful motion clips automatically.

To ensure the usefulness of features learned from the stacked autoencoder, we compare their expressive power with principal components. We can see that only a few parameters can effectively represent the key characteristics of specific motion data. Using the learned features, our feedforward network proposes candidate motion clips with corresponding labels. The effectiveness of our system is demonstrated with labeled motion clips automatically

discovered by the system.

## 4.2 Motion Data

The character we used in the experiment consists of 25 joints including the root joint. The position or orientation of joints can be used as input data. For pose reconstruction, we use local orientation of each joint except for root as the input data and represent the orientation as $3 \times 3$ matrix that is 216-dimensional input vector. Matrix representation limits input vector in [-1, 1] and the dimension of the input(216) greater than the degree of freedom of a pose(72) helps regularization. We train a stacked autoencoder with about 44 thousands frames of basketball motion data. Compared to locomotion, basketball motion is much more dynamic and has more distinct poses.

Traditionally, we use the position and the velocity of end effectors as features for segmentation. For example, a boundary pose of locomotion is a state in which both feet are fully in contact with the ground. Motion classification is the process determining the class of motion clips. It measures similarity between two motion clips and makes similar motion clips belong to the same class.

For segmentation and classification the velocity of end effectors should be considered. Instead of calculating velocity of end effectors, we use the vector that concatenate 5 consecutive poses.

## 4.3 Autoencoders

Artificial neural network is a universal function approximator consisting of layers of neurons. Neuron is a primitive unit that activates other neurons where the output activation is determined with given input activations. Neurons in the first layer(input layer) get input and outputs activation to the neurons in the next layer and neurons in the layer pass on activations to the next layer and so on. We can get output from the activations of the last layer(output layer). If the network does not contain cycle, the network is called feedforward neural network.

Autoencoder[25] is an artificial neural network that is used to find efficient encoding of the given data. It consists of an encoder $\phi : \mathscr{X} \to \mathscr{Z}$, and a decoder$\psi : \mathscr{Z} \to \mathscr{X}$, and find representation such that

$$\min_{\psi,\phi} \|X - \psi\big(\phi(X)\big)\|$$

It learns from data to yield compressed representation of a set of data. Main characteristic of autoencoder compared to general neural network is that the numbers of nodes of input layer and output layer are the same and the network does not predict output value $y$ but reconstruct $x$ i.e., the input value. If we have a hidden layer which has fewer nodes than the input layer, the hidden layer contains compressed representation i.e., *code* of inputs. When the hidden layer has more nodes than the input layer, one can expect identity mapping is learned. However useful features can be still learned from such autoencoders.

### 4.3.1 Stacked autoencoders

When linear activations or a single hidden layer is used, the optimal solution is similar to PCA(Principal Component Analysis), which is limited to represent linear transformation of bases or principal components. To overcome this we use stacked autoencoders, which has deep hidden layers where each hidden layer is an autoencoder. Figure 4.1 shows our neural network hierarchy. While it has deep hidden layers, we can train each layer independently since part of it is also autoencoder. When a network is small, backpropagation algorithm works well in training especially the network is shallow. In order to train large networks, we can use dropout method[26] or pre-train networks with restricted Boltzmann machine(RBM).

## 4.4 Motion Corpus

Feature vectors obtained from the stacked autoencoder are used to construct a motion corpus. For segmentation and classification, we prepare each feed-forward neural network. The segmentation network is to find the boundary poses and proposes candidate motion clips and the classification network classifies the class of a motion clip.

### 4.4.1 Training

We train the segmentation network such that

$$f_s(\mathbf{m}_i) = \sin \frac{i}{N}\pi$$

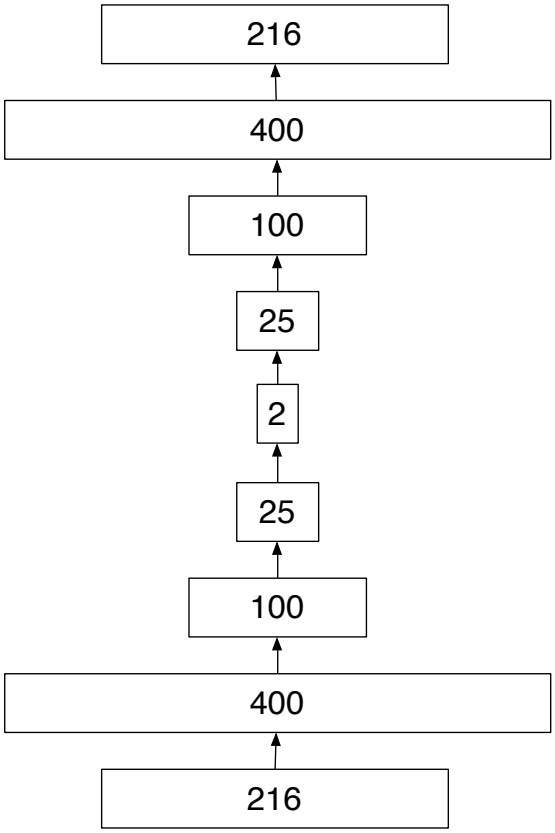


**Figure 4.1:** *Neural network hierarchy used in the experiment.*

where $i$ is the frame index within the labeled motion clip and $N$ is the length of the motion clip. That is, $f_s(p) = 0$ if a pose $p$ is boundary.

The classification network learns one-hot encoding of motion class.

$$f_c(p)_i = \begin{cases} 1 & \text{if } p \text{ belongs to the class } i \\ 0 & \text{otherwise} \end{cases}$$

Two networks share the feature learned from the stacked autoencoder. During the training, the weights for generating feature is fixed. Fine-tuning the networks give poor results.

When the amount of labeled data is small, overfitting easily occurs. Thanks to features

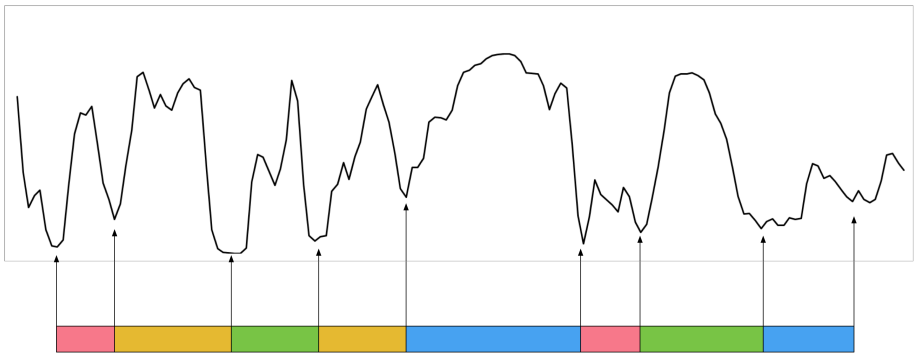

**Figure 4.2:** *Finding candidate motion clips. A sequences of consecutive poses between two local minima is a candidate.*

learned from the stacked autoencoder, we can avoid overfitting. Figure 4.4 shows the validation error during training. Although the fine-tuned neural network converges faster, our features allow us to obtain more accuracy for the validation set.

### 4.4.2 Finding Motion Clips

In order to find a candidate motion clip, we first search for boundary pose. Poses that are local minima of segmentation become candidate boundaries. (See Figure 4.2) Consecutive poses between two boundary poses form a candidate motion clip. Once a candidate motion clip is proposed, its class is determined based on the results of classification network.

## 4.5 Results

**Reconstruction of a single pose**

| | |
|---|---|
| Stacked autoencoders (2 nodes) | 0.388566 |
| PCA (2 components) | 0.775257 |
| PCA (6 components) | 0.417552 |
| PCA (7 components) | 0.357886 |
| PCA (10 components) | 0.255322 |

**Table 4.1:** *Reconstruction error of autoencoder and PCA.*

For comparison, we perform PCA on the same data and use 2 principal components to represent data. We also obtain 2-dimensional feature vector using stacked autoencoders. Table 4.1 shows reconstruction error which is calculated with mean squared error. We can notice that 2 coefficients of the stacked autoencoder can cover about 6 principal components. Stacked autoencoders are not only good for reconstruction but also good for classification. Figure 4.5 shows the scatter plot of the two-dimensional feature of the labeled motion. Please note that training data includes unlabeled data such as pivot, feint and dribble. We can notice that autoencoders can separate different actions well with only 2-dimensional feature and similar actions (shoot and pass) have close relationship in feature space.

**Motion Indexing** For unsupervised training, 179309 frames of basketball motion are used without labels and 14331 frames of labeled motion data are used to train the segmentation and classification network. Our algorithm automatically discovers meaningful motion clips. Dribbling and shooting motion discovered through our algorithm are shown in Figure 4.6.

## 4.6 Discussion

We introduced a powerful dimension reduction technique that can be applied to raw motion capture data including dynamic motion data and generate motion corpus automatically. It was possible to find meaningful features by applying stacked autoencoders to the motion data that has characteristics different from the data that was handled with deep learning. The features are then used to segment and classify motion data. With a small number of training data, our system can automatically find meaningful motion clips automatically.

We capture motion data every day in the world, and the amount of motion data is increasing day by day. If we can generate motion corpus automatically from a large motion database and find the motion clip we need easily, it will help us to increase the usefulness of animation systems. In a situation where a large amount of motion data can be obtained from public motion databases [1, 2], our corpus generator will provide endless source of motion corpus.

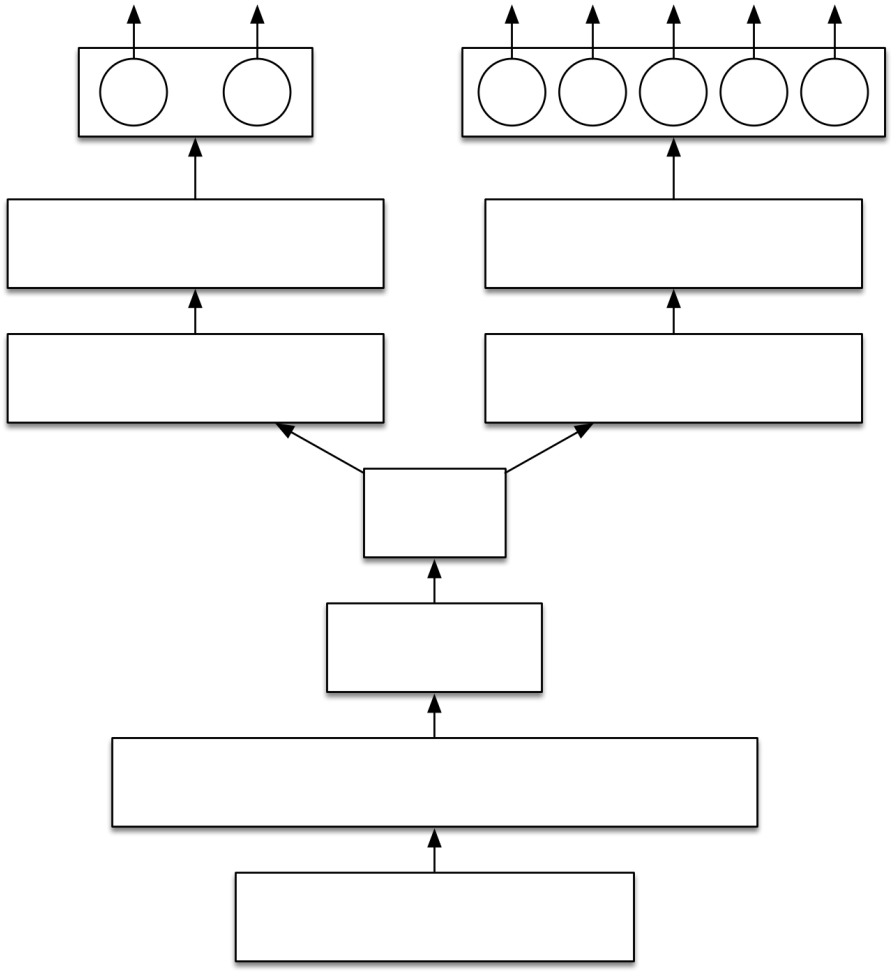

**Figure 4.3:** *Feedforward neural networks for segmentation and classification. Two networks share the feature learned from autoencoder.*

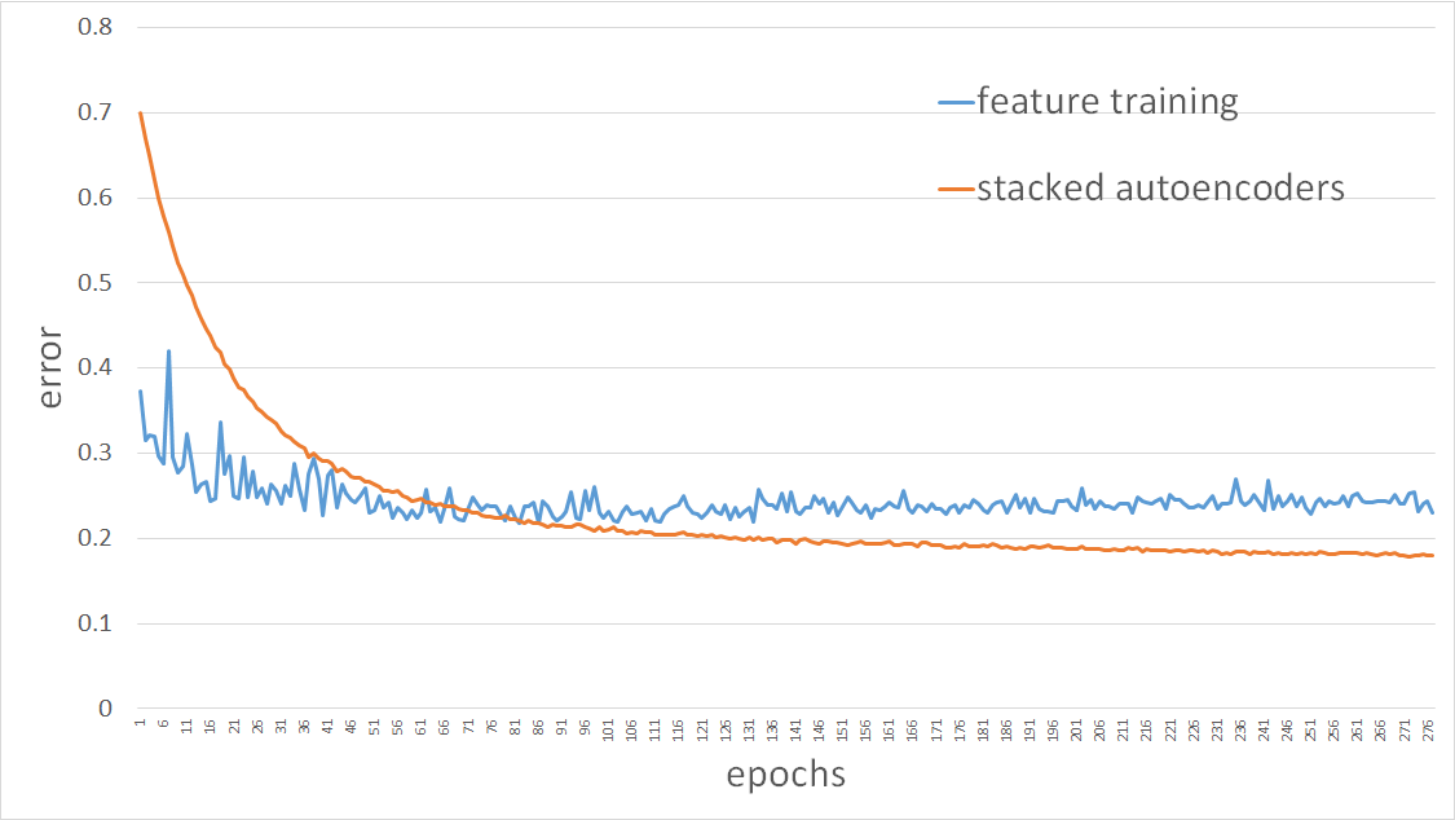


**Figure 4.4:** *Segmentation error of validation data.*

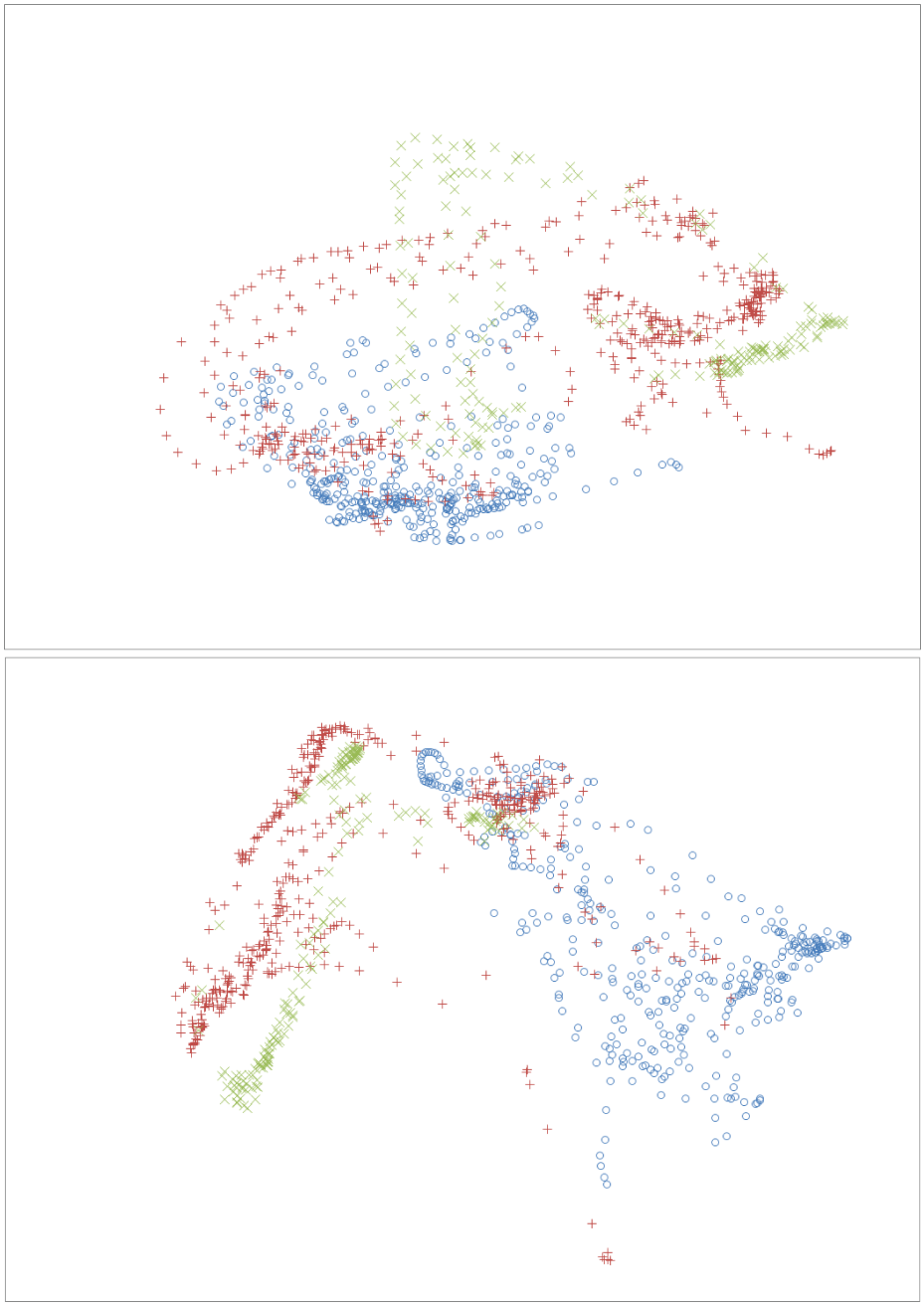

**Figure 4.5:** *Comparison between PCA and stacked autoencoders. Catch, shoot, and pass are marked as green x, red cross and blue circle, respectively. (top) PCA (bottom) Stacked autoencoders*

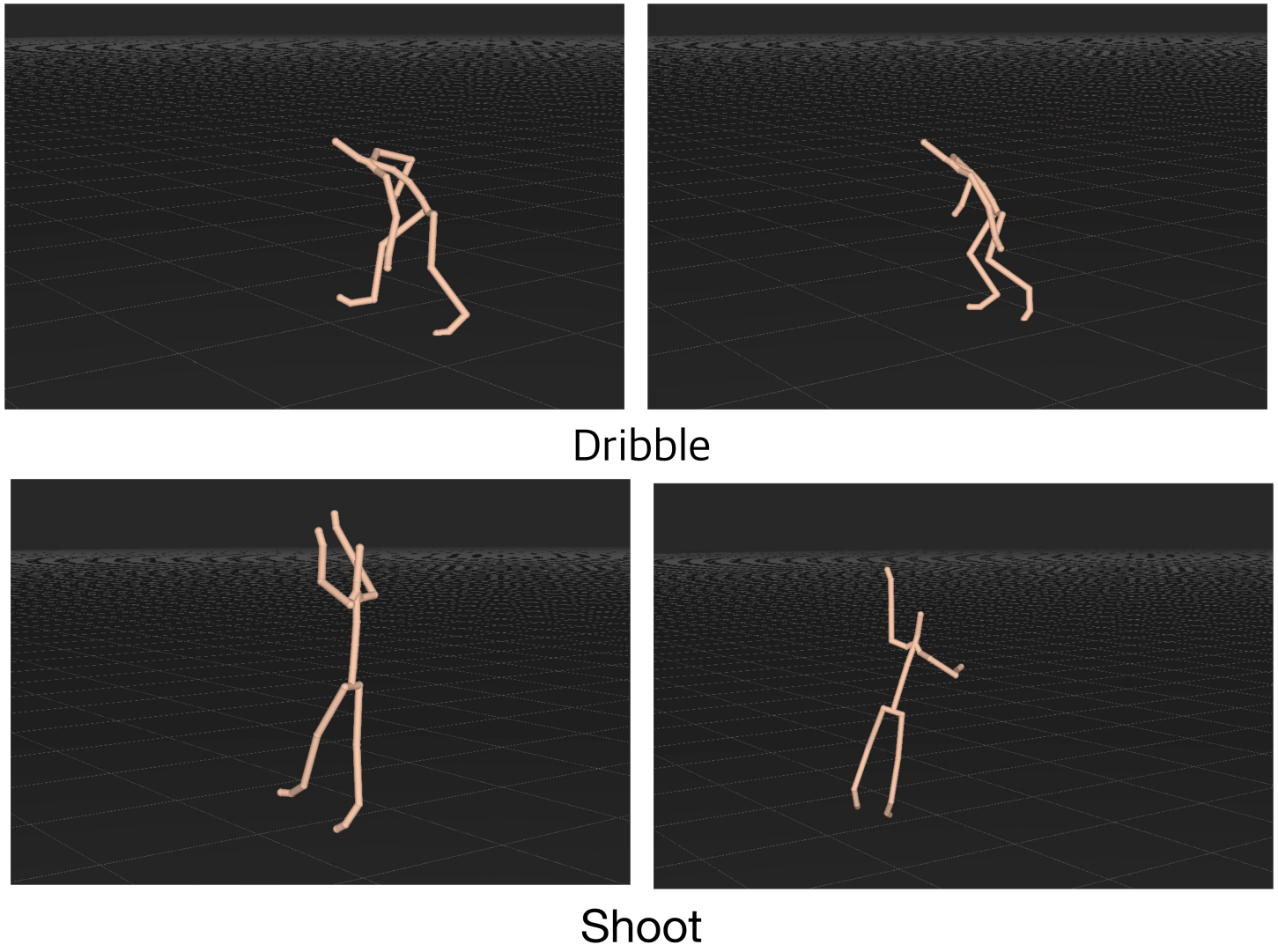


**Figure 4.6:** *Motion clips found through our algorithm. Dribble and shoot motions are automatically discovered.*

# 5

# Text to Animation

In this chapter, we demonstrate an application which based upon understanding on human behavior. Movement of characters can be described in many ways. A high-level description of animated characters is using natural languages. Natural language processing techniques are used to figure out semantics of sentences. Then a MCMC based optimization process is performed to synthesize plausible animates scenes. Our system allows the user to write scripts to describe actions of characters, their surroundings, and relationships between them, then from which animated scenes with full-body characters interacting each others are automatically generated.

## 5.1 Overview

When we create character animations, we describe movements of characters in many ways. Animators controls their characters by setting keyframe poses. Sketch interfaces have been introduced to provide an intuitive interface. To deal with multiple characters, we have to involve interactions between characters that are hardly represented in the sketch interface. We choose natural language as tool for communication between humans and computers.

Our goal is to provide a high-level user interface that uses natural language for describing character animation and create appropriate motion sequences that abide by the text. To do so, we employ natural language processing techniques, such as part-of-speech tagging and dependency parsing which allow us to understand meaning of the text and locate characters in the right place.

Many filmmakers have recently used previsualization to save budget and time. Previsualization visualizes the complex scene in a movie before shooting, allowing the director to design the scene and the actor to understand the shape of the final scene. The script is prepared at the stage of creating previsualization and our system can automatically generate the previsualization from the script. The author can also edit the script and see the results of the previsualization at the same time.

We use a stochastic approach to synthesize character animation from a script. Once our dependency parser finds the role of each word, each character has its own action chain. Actions in the chain are associated to represent interactions between characters. In order to generate scene, our system assign the corresponding motion clip to each action such

that motion clips with a label similar to the given action word are likely to be selected. Spatiotemporal constraints coming from the script, the plausibility of behavior are taken into account when creating the scene. To get the desired scene, we define probability distribution of the scene space and sample from the distribution using MCMC (Markov Chain Monter Carlo).

## 5.2 Understanding Semantics

In order to synthesize animated scenes from a script, the first step is to understand sentences in the script. Understanding a sentence is to figure out roles for each word in the sentence. Part of speech(POS) tagging is the process of labeling a word in a text as a category of words such as noun, verb, adjective, adverb based on it and related words in the text. Consider the below six words sentence.

| Alice | punched | Bob | at | the | restaurant |
|---|---|---|---|---|---|
| NOUN | VERB | NOUN | ADP | DET | NOUN |
| NNP | VBD | NNP | IN | DT | NN |

The tags directly below the sentence are coarse-level POS tags that encode the basic grammar category and tags in the last line are fine-level POS tags which allow us to distinguish further details. For example, ‘Alice’, ‘Bob’, and ‘restaurant’ have the same NOUN tag at the coarse-level, but at the fine-level ‘Alice’ and ‘Bob’ are proper nouns(NNP). Although, the POS tags provide information about what happened(VERB) in the scene, they don’t tell who did it or who was the object. For example, we can not distinguish who punched in the above sentence with only POS tags. It is necessary to figure out grammatical relationships

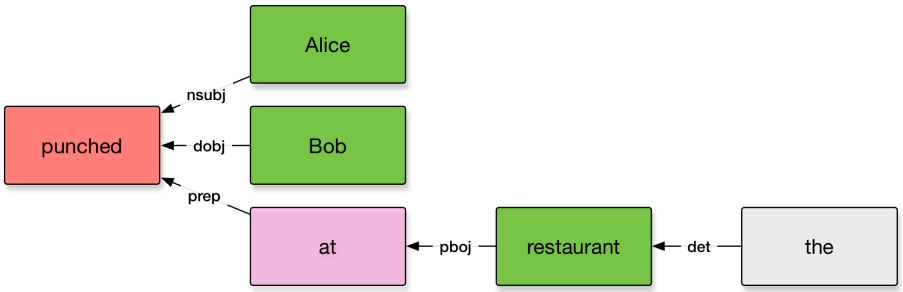


**Figure 5.1:** *Dependency graph for the example sentence*

between words to comprehend meaning of the sentence. A dependency parser generates a directed graph of word with annotated arrows. Each arrow represents the role of the word such as subject, object, conjunct.

Figure 5.1 shows the dependency graph for the above sentence. With the dependency graph, we can notice that 'Alice' is the nominal subject(*nsubj*), 'Bob' is the direct object(*dobj*) and 'restaurant' is the object of the preposition(*pobj*) of 'at' which is the preposition(*prep*) of the root 'punched'. Now we can prepare a *punch* motion for Alice and a *punched* motion for Bob, synchronize two motions, and add a spatial constraint to locate them on a restaurant.

## 5.3 Action Chains

Figure 5.2 shows our workflow. To generate animated scenes from a script, we first generate an action chain for each character. An action chain is a sequence of actions obtained from the dependency parser. If a single verb in the script represents an interaction, it can contribute to more than one action chain.

We formulate the problem as a sampling problem from the conditional distribution $P(X|Y)$

where $X \in \mathscr{X}$ is a scene, $Y$ is the given script and the conditional probability distribution is calculated as factorization of factors.

$$P(X|Y) \propto \prod f_i(X,Y)$$

### 5.3.1 Word Embedding

In scripts a wide variety of words are used to represent characters. For example, Shakespeare used 31,534 different words in his writings. Moreover, kinds of motion clips vary for motion databases so it is very difficult to map a word which will appear on a script to a motion clip in motion database. There are also words that can be mapped to more than one motion clip. For example, 'hit' can be either mapped to 'punch' or 'kick'.

To alleviate this problem, we evaluate similarity between words using pre-trained global vectors for word representation(GloVe) [55] to find the corresponding motion clip. Global vectors are trained such that their dot product equals the logarithm of the words' probability of co-occurrence on corpora. In order to achieve a many-to-many relationship, we introduce a stochastic approach that maps words to a probability mass function of motion clips. So the likelihood that a verb $w$ is used in the script to describe a motion $m$ is defined using softmax function,

$$P(w|m) = \frac{\exp\left(\frac{\mathbf{m}^T\mathbf{w}}{\tau}\right)}{\sum_{\tilde{\mathbf{m}}\in\mathbb{M}} \exp\left(\frac{\tilde{\mathbf{m}}^T\mathbf{w}}{\tau}\right)}$$

where $\mathbf{w}$ and $\mathbf{m}$ are vector representations of words $w$ and $m$, respectively and $\tau > 0$ is the temperature which control uniformity of the distribution. Then the posterior probability is

| | push | run | walk | punch | kick | point | bump |
|---|---|---|---|---|---|---|---|
| jog | 0.3577 | 0.3716 | 0.5707 | 0.2181 | 0.2808 | 0.2067 | 0.3127 |
| touch | 0.4560 | 0.2726 | 0.2726 | 0.3548 | 0.3389 | 0.3632 | 0.3756 |
| crush | 0.4127 | 0.2368 | 0.2120 | 0.4063 | 0.3398 | 0.2253 | 0.3837 |
| knock | 0.5059 | 0.4235 | 0.3855 | 0.5417 | 0.5680 | 0.3738 | 0.4855 |
| bounce | 0.4559 | 0.3192 | 0.2741 | 0.3405 | 0.4435 | 0.3371 | 0.5194 |

**Table 5.1:** *Cosine similarity between word vectors*

given as

$$P(m|w) = \frac{P(w|m)P(m)}{P(w)} = \frac{P(w|m)P(m)}{\sum_{\tilde{m}} P(w|\tilde{m})P(\tilde{m})}$$

where $P(c)$ is the prior belief that reflects our preferences. By setting $P(c)$ appropriately, we can adjust the characteristics of each character such as aggressiveness and activeness.

Table 5.1 shows similarities between word vectors, for example the word *jog* has a high probability of being translated into *walk* and sometimes *run* is used.

The word factor that measures the correspondence between motions in the scene $X$ and actions in the script $Y$ is defined as

$$f_{word}(X,Y) = \prod_{w \in \mathscr{A}(Y)} P(m|w)$$

where $\mathscr{A}(Y)$ are the set of actions in the script $Y$ and $m$ is the corresponding motion of $w$ in the scene $X$.

### 5.3.2 Motion Plausibility

When motions of the characters are determined, we plot motion sequences of characters according to the constraints that occurs in the action chains. Motion editing [32] is per-

formed to satisfy those constraints and errors returned from the procedure can be used to measure plausibility of the scene. However, motion editing takes considerable time and hinders generating many enough scenes to compare. Instead of measuring plausibility of the scene with motion editing, we approximate the error for quick computation.

**Local Alignment** If there are multiple paths between two interactions, they should be synchronized. Let $\mathbf{m}, \mathbf{m}$ be motions for two paths and $\tau, \tau'$ be length of each motion, respectively. Then for each pair of paths,

$$\delta(c) = w_v \|\mathbf{v} - \mathbf{v}'\| + w_r |\theta - \theta'| + w_t |\tau - \tau'|$$

where $v, \theta$ are the translation and rotation of the transform of $\mathbf{m}(\tau) - \mathbf{m}(0)$, respectively. The local alignment factor is defined as negative exponent of the error sum.

$$f_{local}(X) = \exp\Big(-\sum_c \delta(c)\Big) = \prod_c \exp\Big(-\delta(c)\Big)$$

**Global Alignment** Pairs of a preposition(*prep*) and an object of the preposition(*pboj*) gives global constraints. For example, the phrase 'at the restaurant' situates a motion at the restaurant. When we deal with global constraints, it is hard to We only consider two *adjacent* actions with global constraints to approximate error due to global constraints. First, we compose the graph in which a node is an action with global constraints and an edge is is the shortest paths between two nodes. Two actions are adjacent if the edge between two actions belongs to the minimum spanning tree of the graph. The approximate error for an edge is defined as

$$\eta(e) = w_v \|\mathbf{v} - \mathbf{v}'\| + w_r |\theta - \theta'|$$

where $\mathbf{v}, \theta$ are the translation and rotation the transform of the path and $\mathbf{v}', \theta'$ are the translation and rotation of the transform between two global constraints. The global alignment

factor is the negative exponent of the approximate error sum.

$$f_{global}(X,Y) = \exp\Big(-\sum_{e}\eta(e)\Big) = \prod_{e}\exp\Big(-\eta(c)\Big)$$

## 5.4 Scene Generation

By sampling from the probability distribution defined in section 5.3, we can generate animated scenes. However, sampling is not straightforward because the probability distribution is high-dimensional and the normalization constant is intractable. We use multi-level MCMC used in Chapter 3 to sample scenes.

The Metropolis-Hastings algorithm begins with an arbitrary sample $X_0$ and at each time propose a new sample $X'$ dependent on the current sample $X'$ is accepted with the probability

$$\alpha = \min\left(1, \frac{P(X')Q(X;X')}{P(X)Q(X';X)}\right)$$

where $Q(X';X)$ is the probability that $X'$ is proposed when $X$ is the current sample. If the new sample $X'$ is accepted, it is the next sample, otherwise the current sample is the next sample.

**Random walks** We propose two types of random jumps to create a Markov chain. The first type $Q_W(X';X)$ tries to change the content of the scene by replacing the label of a motion clip in $X$. The motion clip is selected randomly and the proposal density function is defined as

$$Q_W(X';X) \propto f_{local}(X')f_{word}(X')$$

. Since $f_{local}$ is defined per cycle, when calculating $Q_W$, we only need to check the error of cycles including the motion clip.

Since global features cost .. we only consider local error and word embeddings. Please note that dropping global error out affect but convergence speed.

The second type $Q_M(X';X)$ substitutes a motion clip with another motion clip in $Y$. The jump takes a random motion clip which is either an action motion or a transition motion and replaces it with another motion clip with the same label. Since we do not change the type of motion clip, the proposal density is defined as

$$Q_M(X';X) \propto f_{local}(X')$$

## 5.5 Results

Figure 5.3 shows the results of our work. Our algorithm effectively generates scenes from a script.

We use NLTK [9] and Gensim [56] to deal with natural languages. SyntaxNet [5] is used as a dependency parser.

## 5.6 Discussion

We have presented a pipeline that transforms motion describing text into animated scenes. Although we demonstrated our system with English, the methods used in the pipeline are

language-independent and can be applied to other languages.

One possible future direction is to obtain natural language description of animated characters. In the case of images, there were studies that generate natural language descriptions [30, 66]. The stochastic approach of suggesting words from motion annotation techniques and writing sentences from the words probably allows us to generate storytellers that can describe animated scenes in natural language.

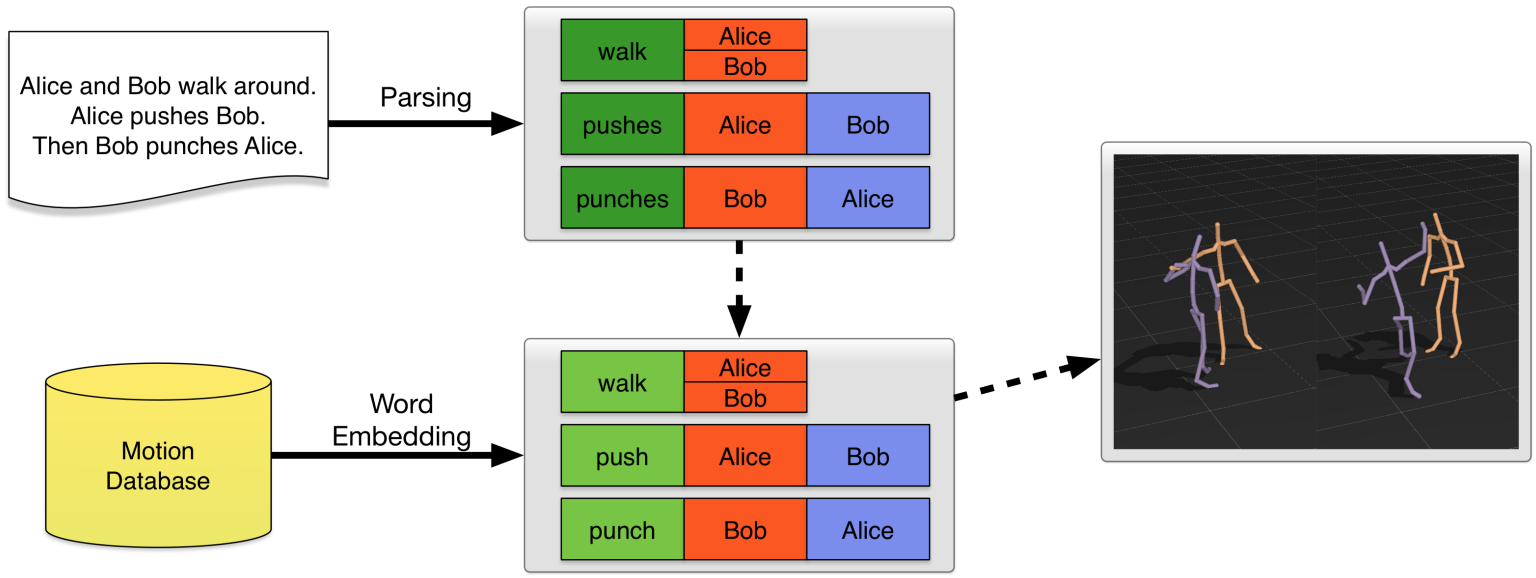


**Figure 5.2:** *Workflow of our system. The action chains for characters extracted from the dependency parser synthesize animated scenes for which word embedding is used to find correspondence*

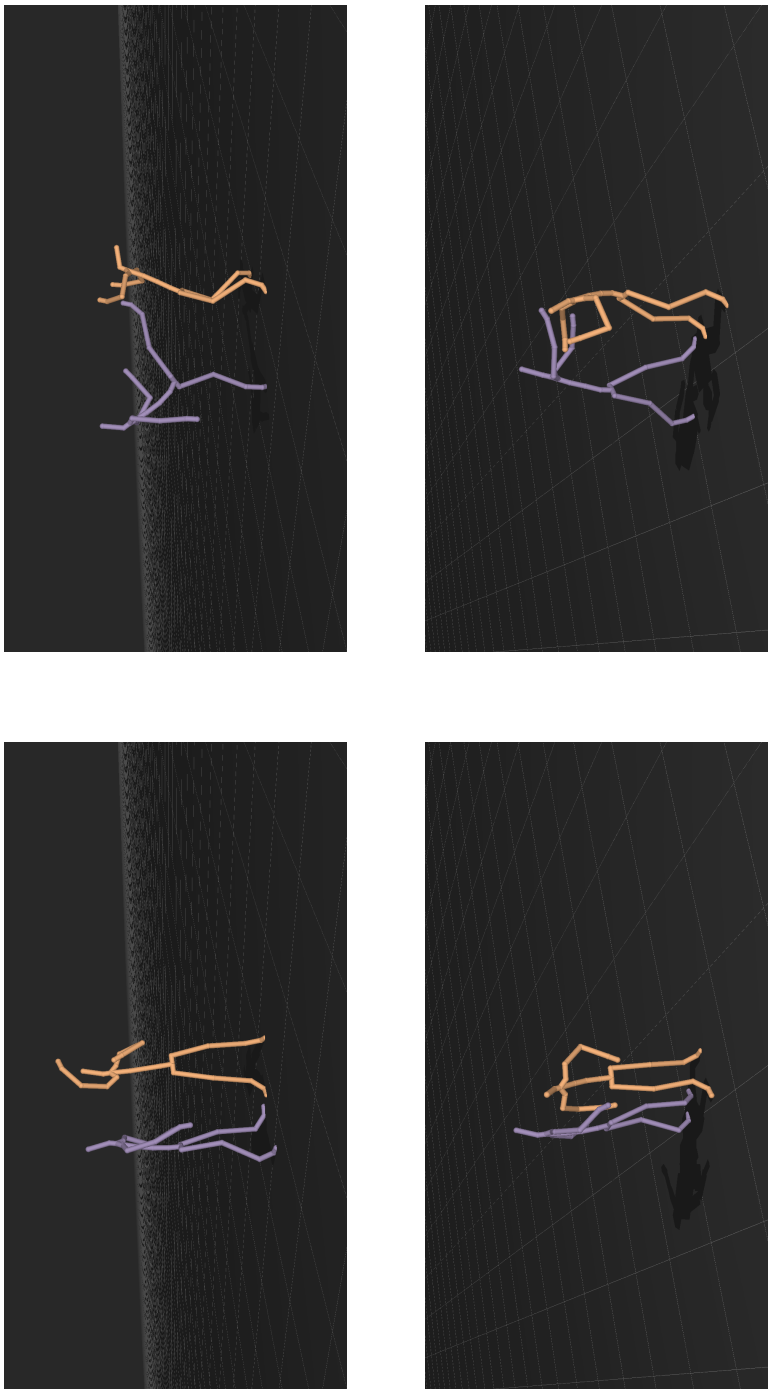

**Figure 5.3:** *Result Scenes.*

# 6

# Conclusion

Through this thesis, I introduce how to break up the data-driven animation pipeline and improve each component to make new animation systems. The methods used to enhance are different for each component; natural language processing for user interface, formal language for representation, and machine learning for preprocessing. Since different fields formulate different types of problems, it is not straightforward to import tools from other field and apply them to create a new system. It is an effective way for resolving problems to break down the process components, analyze the component to be improved, and enhance components by introducing appropriate tools. The proposed pipeline is not limited to data-driven character animation, but can also be applied to computer graphics applications. A computer graphics application that interacts with the user includes user interface, interpreter, solver, and preprocessor. When solving computer graphics problems, a strategy that focuses on a single component and improves it will be effective.

Because the grammar of a language is fixed, manually-written rules can approximate the syntax of the language. However, since the motion grammar is context-dependent, the grammar can change depending on the situation and it is even harder to define implicit rules. Recently, unsupervised learning techniques have been attempted to train models from huge corpus to generate natural language [64, 21]. They used more than hundreds of millions of characters or words to train the models. We have a lot of motion data, but currently captured motion data is not enough to train a model that automatically derives the grammar from the given data, and since motion data is high-dimensional, continuous data, much more data is required to train the model. Fortunately, as the amount of motion capture data is increasing day by day, and the method of processing time series data is evolving day by day, we will achieve automatic or semi-automatic grammar induction in the near future.

We can get words of the motion of characters by motion corpus generation in Chapter 4. By adding words that represent the spatial and temporal relationships between characters or characters and the environment, we can get a set of words that describes the animation scene. Arranging these words in a grammatical order is a natural language description of the scene. Studying storybooks of animated movies or scripts of taken films will lead to successful generation of natural description.

# Bibliography


[1] Cmu graphics lab motion capture database. http://mocap.cs.cmu.edu/. Accessed: 2015 June.

[2] Snu motion db. http://mrl.snu.ac.kr/ mdb/. Accessed: 2015 June.

[3] Marc Alexa and Wolfgang Muller. Representing animations by principal components. In *Computer Graphics Forum*, volume 19, pages 411–418. Citeseer, 2000.

[4] Tej Anand and Gary Kahn. Making sense of gigabytes: A system for knowledge-based market analysis. In *Proceedings of the Fourth Conference on Innovative Applications of Artificial Intelligence*, IAAI'92, pages 57–69. AAAI Press, 1992.

[5] Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. Globally normalized transition-based neural networks. *CoRR*, abs/1603.06042, 2016.

[6] Okan Arikan, David A. Forsyth, and James F. O'Brien. Motion synthesis from annotations. *ACM Transactions on Graphics (SIGGRAPH)*, 22(3):402–408, 2003.

[7] Jernej Barbič, Alla Safonova, Jia-Yu Pan, Christos Faloutsos, Jessica K. Hodgins,

and Nancy S. Pollard. Segmenting motion capture data into distinct behaviors. In *Proceedings of Graphics Interface 2004*, GI '04, pages 185–194, 2004.

[8] P. Beaudoin, S. Coros, M. van de Panne, and P. Poulin. Motion-motif graphs. In *Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer Animation*, pages 117–126, 2008.

[9] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, 2009.

[10] Martin Bokeloh, Michael Wand, and Hans-Peter Seidel. A connection between partial symmetry and inverse procedural modeling. *ACM Transactions on Graphics (SIGGRAPH)*, 29(4):104:1–104:10, 2010.

[11] Thorsten Brants. Tnt: a statistical part-of-speech tagger. In *Proceedings of the sixth conference on Applied natural language processing*, pages 224–231. Association for Computational Linguistics, 2000.

[12] Myung Geol Choi, Manmyung Kim, Kyung Lyul Hyun, and Jehee Lee. Deformable motion: Squeezing into cluttered environments. *Computer Graphics Forum (Eurographics)*, 30(2):445–453, 2011.

[13] N. Dantam and M. Stilman. The motion grammar: Analysis of a linguistic method for robot control. *IEEE Transactions on Robotics*, 29(3):704–718, 2013.

[14] Steven J DeRose. Grammatical category disambiguation by statistical optimization. *Computational linguistics*, 14(1):31–39, 1988.

[15] Cícero Nogueira dos Santos and Bianca Zadrozny. Learning character-level representations for part-of-speech tagging. In *ICML*, pages 1818–1826, 2014.

[16] W. Nelson Francis. A tagged corpus – problems and prospects. In Sidney Greenbaum, Geoffrey Leech, and Jan Svartvik, editors, *Studies in English linguistics for Randolph Quirk*, pages 192–209. Longman, London and New York, 1979.

[17] Ada Wai-Chee Fu, Eamonn Keogh, Leo Yung Lau, Chotirat Ann Ratanamahatana, and Raymond Chi-Wing Wong. Scaling and time warping in time series querying. *The VLDB Journal*, 17(4):899–921, July 2008.

[18] Charles J Geyer. Markov chain monte carlo maximum likelihood. *Proceedings of the 23rd Symposium on the Interface: Computing Science and Statistics*, pages 156–163, 1991.

[19] Michael Gleicher. Retargetting motion to new characters. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 33–42, 1998.

[20] Eli Goldberg, Norbert Driedger, and Richard I Kittredge. Using natural-language processing to produce weather forecasts. *IEEE Expert*, 9(2):45–53, 1994.

[21] Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

[22] Keith Grochow, Steven L. Martin, Aaron Hertzmann, and Zoran Popović. Style-based inverse kinematics. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 522–531, New York, NY, USA, 2004. ACM.

[23] G. Guerra-Filho and Y. Aloimonos. A language for human action. *Computer*, 40(5):42–51, 2007.

[24] Gutemberg Guerra-Filho and Yiannis Aloimonos. The syntax of human actions and interactions. *Journal of Neurolinguistics*, (5):500–514, 2010.

[25] G E Hinton and R R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006.

[26] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[27] Daniel Holden, Jun Saito, and Taku Komura. A deep learning framework for character motion synthesis and editing. *ACM Trans. Graph.*, 35(4):138:1–138:11, July 2016.

[28] Kyunglyul Hyun, Manmyung Kim, Youngseok Hwang, and Jehee Lee. Tiling motion patches. *IEEE Transactions on Visualization and Computer Graphics*, 19(11):1923–1934, 2013.

[29] Fred Karlsson, Atro Voutilainen, Juha Heikkilae, and Arto Anttila. *Constraint Grammar: a language-independent system for parsing unrestricted text*, volume 4. Walter de Gruyter, 1995.

[30] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3128–3137, 2015.

[31] Jongmin Kim, Yeongho Seol, Taesoo Kwon, and Jehee Lee. Interactive manipulation of large-scale crowd animation. *ACM Transactions on Graphics (SIGGRAPH)*, 33(4):83:1–83:10, 2014.

[32] Manmyung Kim, Kyunglyul Hyun, Jongmin Kim, and Jehee Lee. Synchronized multi-character motion editing. *ACM Transactions on Graphics (SIGGRAPH)*, 28(3):79:1–79:9, 2009.

[33] Lucas Kovar and Michael Gleicher. Automated extraction and parameterization of motions in large data sets. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 559–568, New York, NY, USA, 2004. ACM.

[34] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. *ACM Transactions on Graphics (SIGGRAPH)*, 21(3):473–482, 2002.

[35] Björn Krüger, Jochen Tautges, Andreas Weber, and Arno Zinke. Fast local and global similarity searches in large motion capture databases. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, pages 1–10, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[36] Taesoo Kwon, Kang Hoon Lee, Jehee Lee, and Shigeo Takahashi. Group motion editing. *ACM Transactions on Graphics (SIGGRAPH)*, 27(3):80:1–80:8, 2008.

[37] Angeliki Lazaridou, Dat Tien Nguyen, Raffaella Bernardi, and Marco Baroni. Unveiling the dreams of word embeddings: Towards language-driven image generation. *arXiv preprint arXiv:1506.03500*, 2015.

[38] Jehee Lee, Jinxiang Chai, Paul S. A. Reitsma, Jessica K. Hodgins, and Nancy S. Pol-

lard. Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics (SIGGRAPH)*, 21(3):491–500, 2002.

[39] Jehee Lee and Sung Yong Shin. A hierarchical approach to interactive motion editing for human-like figures. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pages 39–48, 1999.

[40] Kang Hoon Lee, Myung Geol Choi, and Jehee Lee. Motion Patches: Building blocks for virtual environments annotated with motion data. *ACM Transactions on Graphics (SIGGRAPH)*, 25(3):898–906, 2006.

[41] Yongjoon Lee, Kevin Wampler, Gilbert Bernstein, Jovan Popović, and Zoran Popović. Motion fields for interactive character locomotion. *ACM Transactions on Graphics (SIGGRAPH ASIA)*, 29:138:1–138:8, 2010.

[42] Sergey Levine, Yongjoon Lee, Vladlen Koltun, and Zoran Popović. Space-time planning with parameterized locomotion controllers. *ACM Transactions on Graphics*, 30:23:1–23:11, 2011.

[43] Sergey Levine, Jack M. Wang, Alexis Haraux, Zoran Popović, and Vladlen Koltun. Continuous character control with low-dimensional embeddings. *ACM Trans. Graph.*, 31(4):28:1–28:10, July 2012.

[44] Wang Ling, Tiago Luís, Luís Marujo, Ramón Fernández Astudillo, Silvio Amir, Chris Dyer, Alan W. Black, and Isabel Trancoso. Finding function in form: Compositional character models for open vocabulary word representation. *CoRR*, abs/1508.02096, 2015.

[45] Markus Lipp, Peter Wonka, and Michael Wimmer. Interactive visual editing of grammars for procedural architecture. *ACM Transactions on Graphics (SIGGRAPH)*, 27(3):102:1–102:10, 2008.

[46] C. Karen Liu, Aaron Hertzmann, and Zoran Popović. Learning physics-based motion style with nonlinear inverse optimization. *ACM Transactions on Graphics (SIGGRAPH)*, 24(3):1071–1081, 2005.

[47] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[48] James McCann and Nancy Pollard. Responsive characters from motion fragments. *ACM Transactions on Graphics (SIGGRAPH)*, 26(3), 2007.

[49] Jianyuan Min and Jinxiang Chai. Motion graphs++: A compact generative model for semantic motion analysis and synthesis. *ACM Transactions on Graphics (SIGGRAPH ASIA)*, 31(6):153:1–153:12, 2012.

[50] Jianyuan Min, Yen-Lin Chen, and Jinxiang Chai. Interactive generation of human animation with deformable motion models. *ACM Transactions on Graphics*, 29:9:1–9:12, 2009.

[51] Meinard Müller. *Information Retrieval for Music and Motion*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[52] Meinard Müller and Tido Röder. Motion templates for automatic classification and retrieval of motion capture data. In *Proceedings of the 2006 ACM SIG-*

*GRAPH/Eurographics Symposium on Computer Animation*, SCA '06, pages 137–146, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[53] Jong Pil Park, Kang Hoon Lee, and Jehee Lee. Finding syntactic structures from human motion data. *Computer Graphics Forum*, 30, 2011.

[54] Xue Bin Peng, Glen Berseth, and Michiel van de Panne. Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Trans. Graph.*, 35(4):81:1–81:12, July 2016.

[55] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[56] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA.

[57] C. Rose, M. F. Cohen, and B. Bodenheimer. Verbs and Adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications*, 18:32–40, 1998.

[58] Alla Safonova and Jessica K. Hodgins. Construction and optimal search of interpolated motion graphs. *ACM Transactions on Graphics (SIGGRAPH)*, 26(3), 2007.

[59] Hyun Joon Shin and Jehee Lee. Motion synthesis and editing in low-dimensional spaces. *Computer Animation and Virtual Worlds*, 17(3-4):219–227, 2006.

[60] Hyun Joon Shin and Hyun Seok Oh. Fat graphs: constructing an interactive charac-

ter with continuous controls. In *Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer Animation*, pages 291–298, 2006.

[61] Hubert P. H. Shum, Taku Komura, Masashi Shiraishi, and Shuntaro Yamazaki. Interaction patches for multi-character animation. *ACM Transactions on Graphics (SIGGRAPH ASIA)*, 27(5):114:1–114:8, 2008.

[62] Hubert P. H. Shum, Taku Komura, and Shuntaro Yamazaki. Simulating multiple character interactions with collaborative and adversarial goals. *IEEE Transactions on Visualization and Computer Graphics*, 18(5):741–752, May 2012.

[63] Peter-Pike Sloan, Jesse Hall, John Hart, and John Snyder. Clustered principal components for precomputed radiance transfer. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, pages 382–391, New York, NY, USA, 2003. ACM.

[64] Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024, 2011.

[65] Adrien Treuille, Yongjoon Lee, and Zoran Popović. Near-optimal character animation with continuous control. *ACM Transactions on Graphics (SIGGRAPH)*, 26(3), 2007.

[66] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156–3164, 2015.

[67] J.M. Wang, D.J. Fleet, and A. Hertzmann. Gaussian process dynamical models for

human motion. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(2):283–298, Feb 2008.

[68] Ralph Weischedel, Richard Schwartz, Jeff Palmucci, Marie Meteer, and Lance Ramshaw. Coping with ambiguity and unknown words through probabilistic models. *Computational linguistics*, 19(2):361–382, 1993.

[69] Jungdam Won, Kyungho Lee, Carol O'Sullivan, Jessica K. Hodgins, and Jehee Lee. Generating and ranking diverse multi-character interactions. *ACM Transactions on Graphics (SIGGRAPH ASIA)*, 33(6):219:1–219:12, 2014.

# 초 록

캐릭터 애니메이션 분야의 가장 큰 주요 목표중 하나는 애니메이션을 만들기 위한 비용을 줄이는 것이다. 사람의 움직임을 이용하면 보다 쉽게 캐릭터를 움직일 수 있어 동작 캡쳐 기술은 표준적인 기술로 사용된다. 그러나 사람의 동작을 기록하기 위해서는 고가의 카메라들을 다수 갖춘 스튜디오, 동작을 수행할 연기자가 필요하며 기록된 동작의 후처리도 필요하기 때문에 원하는 동작을 구하는 것은 쉬운 일이 아니다. 데이터 기반 캐릭터 애니메이션은 기록된 사람의 동작 데이터를 보다 잘 활용하기 위한 다양한 기법들을 일컫는다.

이 논문에서는 동작 데이터의 활용도를 높히기 위하여 동작의 의미를 분석하고 이를 이용해 새로운 애니메이션 시스템을 만드는 방법들을 소개한다. 최소의 단위가 되는 단위 동작의 의미를 파악하거나, 단위 동작들이 연결되는 구조적인 의미를 파악하기도 하고 사용자가 동작을 서술하는 의미를 파악하기도 한다. 동작의 의미에 대한 폭 넓은 이해를 위해 다양한 분야의 최신 기술들을 고루 사용하였다. 먼저 일련의 동작의 의미를 파악하기 위하여 먼저 형식 문법의 일종인 문맥 자유 문법을 이용한다. 문맥 자유 문법의 일종인 동작 문법을 정의하여 농구 선수의 동작을 선언하고 이를 이용해 농구 작전판 위에 그린 스케치로부터 농구 장면을 만들어낼 수 있다. 농구 선수들이 따라야 하는 복잡한 농구의 규칙을 정의하여 항상 올바른 애니메이션을 생성할 뿐만 아니라, 선수들 사이의 또는 선수와 환경 사이의 상호작용을 효과적으로 표현할 수 있어 자연스러운 장면을 만

들어낼 수 있다. 캐릭터의 동작을 구성할 때는 잘 정제되어 있는 단위 동작들과 동작들을 연결할 수 있게 해주는 동작 뭉치를 이용한다. 동작 뭉치를 잘 구성하는 것은 자연스럽고 풍부한 애니메이션을 만드는데 중요한 일이지만, 숙련된 인력의 고된 노력이 필요한 일이다. 캡쳐된 동작 데이터로부터 자동으로 동작 뭉치를 생성할 수 있는 준지도학습 방법을 소개한다. 여러층의 오토인코더를 이용해 동작 데이터의 의미를 파악하고 사용자가 입력한 적은 갯수의 단위 동작들로부터 비슷한 단위 동작들을 자동으로 찾아내어 동작 뭉치를 생성하게 된다. 그리고 사용자가 만들고 싶어하는 동작의 의미를 이해하기 위하여 자연어 처리 기술을 이용한다. 구체적으로 애니메이션 장면을 서술한 대본을 이해하여 캐릭터들의 움직임을 합성한다. 스케치 인터페이스처럼 대본은 굉장히 추상적인 입력 수단이지만, 동작에 대한 폭 넓은 이해는 추상적인 입력을 잘 이해하여 사용자가 원하는 애니메이션 장면을 생성할 수 있게 한다.

**주요어:** 컴퓨터 그래픽스, 캐릭터 애니메이션, 데이터 기반 모션 생성, 모션 분류, 기계학습

**학번:** 2008-21005